Air Force Institute of Technology AFIT Scholar

Theses and Dissertations

Student Graduate Works

9-18-2014

Optimal Autonomous Spacecraft Resiliency Maneuvers Using Metaheuristics

Daniel J. Showalter

Follow this and additional works at: https://scholar.afit.edu/etd

Recommended Citation

Showalter, Daniel J., "Optimal Autonomous Spacecraft Resiliency Maneuvers Using Metaheuristics" (2014). *Theses and Dissertations*. 560. https://scholar.afit.edu/etd/560

This Dissertation is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.



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

DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

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

DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

OPTIMAL AUTONOMOUS SPACECRAFT RESILIENCY MANEUVERS USING **METAHEURISTICS**

DISSERTATION

Daniel J. Showalter, BS, MS Captain, USAF

Approved:

/signed/

Jonathan T. Black, PhD (Chairman)

/signed/

Darryl K. Ahner, PhD (Member)

/signed/

Raymond R. Hill, PhD (Member)

/signed/

William E. Wiesel, PhD (Member)

Accepted:

A.B. Badiru

/signed/

Dean, Graduate School of Engineering and Management

15 Aug 2014

Date

29 Jul 2014 Date

29 Jul 2014

Date

29 Jul 2014 Date

29 Jul 2014

Date

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 deltavelocity. 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

]	Page
Ab	stract							• •	••			•••		•		•	•	• •	•	iv
De	dicati	on														•			•	v
Ac	know	ledgmer	nts													•			•	vi
Tał	ole of	Content	S													•			•	vii
Lis	t of F	igures .														•			•	xi
Lis	t of T	ables .														•			•	xiii
Lis	t of S	ymbols														•			•	XV
Lis	t of A	cronym	s													•			•	xx
I.	Intro	duction																	•	1
	1.1 1.2 1.3 1.4	Motiva Backgr Resear Docum	tion ound ch Objecti ent Previe	ves	· · · ·	· · · · · ·	· · · ·	 	 	· · · · · · · · · · · · · · · · · · ·	· · · · · ·	· · · ·	· · · ·		· · · ·		• • •	 	•	1 3 4 5
II.	Back	ground														•			•	7
	2.12.22.3	Coordi 2.1.1 2.1.2 2.1.3 2.1.4 Orbital 2.2.1 2.2.2 Literatu 2.3.1	nate Fram Geocentr Perifocal Topocent Local Ve Mechanic Equation Equation are Review Enabling 2.3.1.1 2.3.1.2	es ic Equa Coordi rtic Hor rtical, L cs s of Mc s of Mc s of Mc v Techni Gauss Shape	toria nate izon cocal otion otion ques ' Prol -Base	I Coor Fram Coor Hori in Ge in Pe blem	ordina dina zont cocer rifoc	ate I ate Fr al Co ntric cal an 	Fram Fram came coord Equ nd N	 ne e linat Joda ajec	 e F1 rial 1 I Fr 	· · · · · · · · · · · · · · · · · ·	· · · · · · · · · · · · · · · · · ·		· · · · · · · · · · · · · · · · · ·		•••••••••	· · · · · · · · · · · ·		7 7 8 10 12 13 13 14 16 16 16 17
			2.3.1.3	Time-	Fixed	l Maı	neuv	ers i	n Re	elati	ve (Drbi	ts						•	19

	2.3.2 Optimization Techniques	20
	2.3.2.1 Direct and Indirect Techniques	20
	2.3.2.2 Evolutionary Algorithms and Metaheuristics	22
	2.3.2.2.1 Particle Swarm Optimization	22
	2.3.2.2.2 Genetic Algorithms	26
	2.3.2.2.3 Other Evolutionary Algorithms	28
	2.3.2.2.4 Constrained Optimization with Evolutionary	
	Algorithms	28
	2.3.3 Spacecraft Trajectory Optimization Research	30
	2.3.3.1 Evolutionary Algorithms in Trajectory Optimization	30
	2.3.3.2 Hybrid Optimal Control	31
2.4	Summary	34
III. Res	ponsive Theater Maneuvers via Particle Swarm Optimization	35
	r	
3.1	Abstract	35
3.2	Nomenclature	37
3.3	Introduction	39
3.4	Background	39
3.5	Methodology	42
3.6	Responsive Theater Maneuvers	44
	3.6.1 Single Pass Maneuvers	44
	3.6.2 Mulitple-Pass Maneuvers	47
3.7	Numerical Results	48
	3.7.1 Comparison of Optimization Tools for Single Pass RTM Problem .	48
	3.7.2 Single Pass Results	51
	3.7.3 <i>n</i> -Pass RTM	54
	3.7.3.1 Double Pass RTM	54
	3.7.3.2 Triple Pass RTM	56
3.8	Conclusion	57
3.9	Appendix	58
IV. Low	Thrust Responsive Theater Maneuvers Using Particle Swarm Optimization	
and	Direct Collocation	62
4.1	Abstract	62
4.2	Nomenclature	63
4.3	Introduction	64
4.4	Background	65
4.5	Methodology	67
	4.5.1 Responsive Theater Maneuvers	67

Page

		4.5.2 Shape-Based Approximation Method Applied to Responsive The-							
		ater Maneuvers	59						
		4.5.3 Optimal Low-Thrust Responsive Theater Maneuvers	1						
	4.6	Analysis	2						
		4.6.1 Single Pass Responsive Theater Maneuvers	2						
		4.6.2 Double Pass Responsive Theater Maneuvers	7						
		4.6.3 Triple Pass Responsive Theater Maneuvers	30						
		4631 Triple-Pass Low-Thrust responsive theater maneuvers							
		(RTMs)	80						
		4632 Triple-Pass Multiple-Revolution Impulsive RTMs	24						
		4.6.3.2 Comparison of Triple-Pass Low-Thrust and Impulsive	7						
		RTMc	25						
	17		26						
	4.7	Appendix: Coordinate Transformation Matrices	20						
	4.0	Appendix. Coordinate Transformation Matrices)7						
v	Onti	nal Coostationary Transfer Managura with Cooperative En route Inspec							
۷.	Opin tion 1	Jaing Hybrid Optimal Control	0						
	tion		0						
	51	Abstract)U						
	5.1	Nomenelature	νΟ 11						
	5.2 5.2	Motivation	יי רו						
	5.5 5 4		'Z						
	5.4		<i>1</i> 4						
	5.5	Geostationary Iransfer Maneuver with En-route Inspection	8						
		5.5.1 Cooperative Inspection Boundary Conditions	18						
		5.5.2 Cooperative Inspection Segment)()						
		5.5.3 Impulsive Transfer to Target)2						
		5.5.4 Impulsive Transfer to GEO Segment)3						
		5.5.5 GTMEI as a Hybrid Optimal Control Problem)3						
	5.6	Analysis)5						
		5.6.1 Three Target Problem)5						
		5.6.1.1 Three Target Enumeration)6						
		5.6.1.2 Three Target Hybrid Optimization)9						
		5.6.2 Fifteen Target Problem	4						
	5.7	Conclusions	7						
VI	. Conc	lusions and Contributions	20						
	6.1	Impulsive Responsive Theater Maneuvers	20						
	6.2	Continuous Thrust Responsive Theater Maneuvers							
	6.3	Geosynchronous Transfer Maneuvers with Cooperative En-Route Inspection 12	21						
	6.4	Overall Conclusion	22						
	6.5	Assumptions and Limitations	23						

Pag	ge
6.6 Areas for Future Work	24
Appendix A: Derivation of Spherical Equations of Motion	26
Appendix B: Equations of Motion in the Local Vertical, Local Horizontal Frame 12	28
Appendix C: Design of Experiments on Particle Swarm Optimization Parameters 13	30
Appendix D: Code for Impulsive Responsive Theater Maneuvers	37
Appendix E: Code for Low Thrust Responsive Theater Maneuvers	24
Appendix F: Code for Geostationary Transfer Maneuvers	56
Bibliography	54

List of Figures

Figu	re	Page
2.1	Geocentric equatorial coordinate frame	8
2.2	Perifocal coordinate frame	9
2.3	Spherical coordinate definitions in perifocal coordinate frame	10
2.4	Topocentric horizon coordinate frame	11
2.5	Local vertical, local horizontal coordinate frame	12
2.6	Thrust acceleration vector	15
3.1	Single pass RTM intercept geometry	47
3.2	Response surface for single pass RTM with $a_e = 150 \text{ km}$ and $b_e = 15 \text{ km} \dots$	49
3.3	Optimal design variables and constraints for single pass RTM as functions of	
	exclusion ellipse size $(r_0 = 6800 \text{ km}) \dots \dots$	52
3.4	Optimal solution for single pass RTM maneuver with $a_e = 150$ km, $b_e = 15$ km	53
3.5	Optimal design variables and constraints for double pass RTM as functions of	
	exclusion ellipse size $(r_0 = 6800 \text{ km}) \dots \dots$	55
3.6	Optimal design variables and constraints for triple pass RTM as functions of	
	exclusion ellipse size $(r_0 = 6800 \text{ km}) \dots \dots$	57
4.1	Exclusion ellipse orientation with respect to γ_1	71
4.2	Exclusion zone and exclusion ellipse arrival conditions for $r_0 = 6800$ km,	
	$a_e = 150 \text{ km} \dots \dots$	75
4.3	Optimal control necessary conditions for $r_0 = 6800$ km, $a_e = 150$ km	75
4.4	Comparison of impulsive and low-thrust single pass RTMs	77
4.5	Optimal control necessary conditions for double-pass RTM $r_0 = 6800$ km,	
	$a_e = 150 \text{ km} \dots \dots$	80
4.6	Comparison of impulsive and low-thrust double pass RTMs	81

Figure

Page

4.7	Optimal control necessary conditions for triple-pass RTM $r_0 = 6800$ km,
	$a_e = 150 \text{ km} \dots \dots$
4.8	Comparison of impulsive and low-thrust triple pass RTMs
5.1	Segments of the geostationary transfer maneuver with cooperative en-route
	inspection (GTMEI) problem
5.2	Target pass times for the three target GTMEI
5.3	Characterization of three target categorical variable space
5.4	Path of chaser corresponding to the optimal three target GTMEI
5.5	Bi-level algorithm performance data for three target problem
5.6	Target pass times for the fifteen target GTMEI
5.7	Characterization of fifteen target categorical variable space
5.8	Fifteen target performance data

List of Tables

Table	e	Page
3.1	Comparison of optimization algorithms in single pass RTM problem	. 50
3.2	Optimal cost of single pass RTM problem for varying exclusion ellipse sizes .	. 52
3.3	Optimal cost of double pass RTM problem for varying exclusion ellipse sizes	. 55
3.4	Optimal cost of triple pass RTM problem for varying exclusion ellipse sizes .	. 57
3.5	Optimal single pass RTM for various exclusion ellipse sizes	. 59
3.6	Optimal double pass RTM for various exclusion ellipse sizes	. 60
3.7	Optimal triple pass RTM for various exclusion ellipse sizes	. 61
4.1	PSO settings	. 73
4.2	Optimal cost in m/s of low-thrust single pass RTMs	. 76
4.3	Optimal cost in m/s of low-thrust double pass RTMs	. 79
4.4	Optimal cost in m/s of low-thrust triple pass RTMs	. 83
4.5	Optimal cost in m/s of impulsive triple pass RTMs	. 85
5.1	Initial conditions of the chaser and targets	. 106
5.2	Inner-loop PSO settings	. 107
5.3	Best three target costs found by enumeration	. 108
5.4	Outer-loop optimization routines	. 110
5.5	Lowest cost solution for three target problem found by each bi-level algorithm	. 111
5.6	Bi-level algorithm performance comparison	. 113
5.7	Target satellites' initial conditions	. 115
5.8	Best/worst solution for each target-pass combination converged upon by the PF	Pi 116
C .1	Performance data for initial set of design of experiments (DOE) bounds	. 131
C.2	Performance data for second set of DOE bounds on two parameter study	. 132
C.3	Performance data for third set of DOE bounds on two parameter study	. 133

Tabl	e	Page
C.4	Performance data for fourth set of DOE bounds on two parameter study	. 134
C.5	Performance data for fifth set of DOE bounds on two parameter study	. 135

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
<i>c</i> ₂	swarm social parameter
\boldsymbol{g}_{best}	global best position in the solution space
\boldsymbol{g}_k	unit vector perpendicular to v_k and 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
lbest	neighborhood best position in the solution space
l _{GEO}	true longitude at epoch of the arrival location on the geostationary orbit, rad
т	number of particles in the swarm
n	number of design variables in the nonlinear function to be optimized

XV

Symbol Definition

N _{rev}	minimum number of orbital revolutions required for multiple revolution
	impulsive maneuver
Р	period of the initial orbit, sec
p _{best}	particle best position in the solution space
R_{a_k}	orbit apogee radius after the k^{th} maneuver, km
\mathbf{r}_{CYL}^{c}	position vector of the chaser in the cylinder coordinate frame, km
\mathbf{r}_{IJK}^{c}	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
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}_{IJK}^m	inertial position vector of the <i>m</i> 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
\mathbf{r}_{RSW}^c	position vector of the chaser in the local vertical, local horizontal coordinate
	frame, <i>km</i>
r ₀	initial position vector, km
R_\oplus	radius of the earth, km
S	Solution space encompassing all <i>n</i> design variables
t_k	expected k^{th} time of entry into exclusion zone, sec
t_f	final time of GTMEI scenario, sec
t_{enter}^k	entry time of the <i>m</i> th target's <i>k</i> th pass over the ground site, <i>sec</i>

Symbol Definition

t_{exit}^k	exit time of the <i>m</i> th target's <i>k</i> th pass over the ground site, <i>sec</i>
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
<i>t</i> 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}_{IJK}^{c}	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
$\boldsymbol{v}_{k_t^-},\boldsymbol{v}_{k_t^+}$	velocity vectors at the instant just before and just after the k^{th} impulse, km/sec
$\boldsymbol{V}_{p}\left(s ight)$	<i>n</i> -dimensional velocity vector of the p^{th} particle at the s^{th} iteration
v_{max}^i, v_{min}^i	upper and lower bounds on the velocity of the i^{th} design variable
\boldsymbol{v}_0	initial velocity vector, km/sec
\boldsymbol{v}_{RSW}^c	velocity vector of the chaser in the local vertical, local horizontal coordinate
	frame, <i>km</i>
$X_{p}\left(s ight)$	<i>n</i> -dimensional position vector of the p^{th} particle at the s^{th} iteration
x_{max}^{i}	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,

 β angle measured from the primary axis to the cylinder in the local vertical,

local horizontal frame, rad

local horizontal frame, rad

 ΔV_k velocity vector of the k^{th} maneuver, km/sec

Symbol	Definition
ΔV_k	cost of the k^{th} maneuver, m/sec
$oldsymbol{\gamma}_{k_f}$	pre-maneuver flight path angle for the k^{th} maneuver, rad
$oldsymbol{\gamma}_{k_t}$	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_{max}^{c}	maximum allowable elevation angle of the chaser with respect to the ground
	site, <i>rad</i>
ϵ^{g}_{min}	minimum elevation angle required by ground site for line-of-sight contact with
	the <i>m</i> th target, <i>rad</i>
ϵ^m	elevation angle of the target satellite, rad
η	thrust pointing angle, rad
$ heta_k$	angle defining position of spacecraft on the k^{th} exclusion ellipse, rad
μ	Earth's gravitational parameter, km^3/sec^2
<i>V_{enter}</i>	true anomaly of the spacecraft as it enters the latitude band of the exclusion zone
ρ_{IJK}	vector from the ground site to the target in the inertial coordinate frame, km
$ ho_{RSW}$	vector from the ground site to the target in the local vertical, local horizontal
	coordinate frame, <i>km</i>
$ ho_{SEZ}$	vector from the ground site to the target in the topocentric horizon coordinate
	frame, <i>km</i>
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
$\boldsymbol{\psi}_k^*$	actual angle traveled by the spacecraft in the orbit plane during the k^{th}
	maneuver, rad
ω_\oplus	rotation rate of the earth, rad

xviii

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
СР	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 longterm 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 righthanded 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 r and velocity v vectors shown in Equation 2.1.

$$\mathbf{r} = x\hat{I} + y\hat{J} + z\hat{K}$$

$$\mathbf{v} = v_x\hat{I} + v_y\hat{J} + v_z\hat{K}$$
(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 \mathbf{r}_{IJK} in the IJK frame to a corresponding vector \mathbf{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. *R*1 and *R*3 are rotation matrices about the first and third axes, respectively.

$$\boldsymbol{r}_{PQW} = R3(\omega)R1(inc)R3(\Omega)\boldsymbol{r}_{IJK}$$
(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 ω_{\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. *R*2 and *R*3 are rotation matrices about the second and third axes, respectively. Figure 2.4 depicts the SEZ coordinate frame.

$$\mathbf{r}_{SEZ} = R2 \left(\pi/2 - \Phi \right) R3 \left(\Lambda + \omega_{\oplus} t_{\hat{l}} \right) \mathbf{r}_{IJK}$$
(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 \mathbf{r}_{PQW} in the PQW frame into a vector \mathbf{r}_{RSW} in the RSW frame, where ν is the true anomaly. Figure 2.5 shows the RSW frame.

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



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, r and v, respectively. In the IJK frame, r and v take the form shown in Equation 2.5.

$$\mathbf{r} = x\hat{I} + y\hat{J} + z\hat{K}$$

$$\mathbf{v} = v_x\hat{I} + v_y\hat{J} + v_z\hat{K}$$
(2.5)

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

$$\begin{bmatrix} \dot{\boldsymbol{r}} \\ \dot{\boldsymbol{v}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ -\frac{\mu}{|\boldsymbol{r}|^3} \boldsymbol{r} \end{bmatrix}$$
(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 ΔV 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 ΔV , which is equal to the difference between the velocity vector at the instant after the maneuver, v^+ , and the velocity vector at the instant prior to the maneuver, v^- .

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

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, 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, A_T . The thrust acceleration vector is defined by its magnitude, A_T and the angle η measured from local horizontal to 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.

$$\dot{r} = V_r$$

$$\dot{\psi} = \frac{V_{\psi}}{r}$$

$$\dot{V}_r = \frac{V_{\psi}^2}{r} - \frac{\mu}{r^2} + A_T \sin \eta$$

$$\dot{V}_{\psi} = \frac{-V_{\psi}V_r}{r} + A_T \cos \eta$$
(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.

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

$$a = \frac{1}{r_i} \quad b = \frac{\tan \gamma_i}{r_i} \quad c = \frac{1}{2r_i} \left(\frac{\mu}{r_i^3 \psi_i^2} - 1 \right)$$
(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 .

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

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

$$\int_{0}^{t_{f}} \mathrm{d}t = \int_{0}^{\psi_{f}} \sqrt{\frac{r(\psi)^{4}}{\mu}} \left[1/r(\psi) + 2c + 6d\psi + 12e\psi^{2} + 20f\psi^{3} + 30g\psi^{4} \right] \mathrm{d}\psi \qquad (2.15)$$

[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.

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

where

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

$$\dot{\psi} = \frac{\mu}{r^4} \frac{1}{(1/r + 2c + 6d\psi + 12e\psi^2 + 20f\psi^3 + 30g\psi^4)}$$
(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, r_i and r_f , respectively, are defined in Equations 2.20 and 2.21. The time-of-flight is t_{if} seconds.

$$\boldsymbol{r}_i = x_i \hat{\boldsymbol{R}} + y_i \hat{\boldsymbol{S}} + z_i \hat{\boldsymbol{W}}$$
(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{v}_i and \tilde{v}_f , respectively, given r_i , 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, $\tilde{r}_i = r_i$ and $\tilde{r}_f = r_f$.

Equations 2.22 and 2.23 show \tilde{v}_i and \tilde{v}_f , respectively, as functions of \tilde{r}_i , \tilde{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 $\Delta \tilde{y} = \tilde{y}_f - \tilde{y}_i$.

$$\begin{bmatrix} \dot{\tilde{x}}_{i} \\ \dot{\tilde{y}}_{i} \\ \dot{\tilde{z}}_{i} \end{bmatrix} = 2\pi \begin{bmatrix} \frac{-4\tilde{S}+6\pi\tilde{T}\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{4\tilde{S}-6\pi\tilde{T}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{-2+2\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} \\ \frac{-14+12\pi\tilde{T}\tilde{S}+14\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{2-2\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{\tilde{S}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} \\ 0 & -\frac{\tilde{C}}{\tilde{S}} & 0 & \frac{1}{\tilde{S}} & 0 \end{bmatrix} \begin{bmatrix} \tilde{x}_{i} \\ \tilde{z}_{i} \\ \tilde{z}_{i} \\ \tilde{z}_{f} \\ \Delta \tilde{y} \end{bmatrix}$$
(2.22)

$$\begin{bmatrix} \dot{\tilde{x}}_{f} \\ \dot{\tilde{y}}_{f} \\ \dot{\tilde{z}}_{f} \end{bmatrix} = 2\pi \begin{bmatrix} \frac{-4\tilde{S} + 6\pi\tilde{T}}{8 - 6\pi\tilde{T}\tilde{S} - 8\tilde{C}} & 0 & \frac{4\tilde{S} - 6\pi\tilde{T}\tilde{C}}{8 - 6\pi\tilde{T}\tilde{S} - 8\tilde{C}} & 0 & \frac{2 - 2\tilde{C}}{8 - 6\pi\tilde{T}\tilde{S} - 8\tilde{C}} \\ \frac{2 - 2\tilde{C}}{8 - 6\pi\tilde{T}\tilde{S} - 8\tilde{C}} & 0 & \frac{-14 + 12\pi\tilde{T}\tilde{S} + 14\tilde{C}}{8 - 6\pi\tilde{T}\tilde{S} - 8\tilde{C}} & 0 & \frac{\tilde{S}}{8 - 6\pi\tilde{T}\tilde{S} - 8\tilde{C}} \\ 0 & -\frac{1}{\tilde{S}} & 0 & \frac{\tilde{C}}{\tilde{S}} & 0 \end{bmatrix} \begin{bmatrix} \tilde{x}_{i} \\ \tilde{z}_{i} \\ \tilde{z}_{i} \\ \tilde{x}_{f} \\ \tilde{z}_{f} \\ \Delta \tilde{y} \end{bmatrix}$$
(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 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 S according to a uniform random distribution. The p^{th} particle's position X_p and velocity V_p vectors in S take the forms shown in Equation 2.24.

$$X_{p} = \left[x_{p}^{1}, x_{p}^{2}, \dots, x_{p}^{n}\right]$$

$$V_{p} = \left[v_{p}^{1}, v_{p}^{2}, \dots, v_{p}^{n}\right]$$
(2.24)

The bounds on each component in X_p match the bounds in S corresponding to that component. That is, the *i*th dimension of each particle's position vector is bounded by x_{min}^i and x_{max}^i . 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 $v_{min}^i = -v_{max}^i$.

$$\begin{aligned} x_{min}^{i} &\leq x_{p}^{i} &\leq x_{max}^{i} \\ v_{min}^{i} &\leq v_{p}^{i} &\leq v_{max}^{i} \end{aligned}$$
 (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 S to the best position visited by swarm (g_{best}) and the best position ever visited by that specific particle (p_{best}). Each particle's position in 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 (p_{best}) while the social parameter influences particle velocity towards (g_{best}). The variables z_1 and z_1 are stochastic parameters uniformly distributed between zero and one.

$$V_{p}(s) = V_{p}(s-1) + c_{1}z_{1}\left(\boldsymbol{p}_{best} - \boldsymbol{X}_{p}(s-1)\right) + c_{2}z_{2}\left(\boldsymbol{g}_{best} - \boldsymbol{X}_{p}(s-1)\right)$$
(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.

$$X_p(s) = X_p(s-1) + V_p(s)$$
 (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 I_{best} is set equal to $X^p(s)$. The velocity update for the local PSO variant used in this research is shown in Equation 2.28.

$$\boldsymbol{V}_{p}(s) = \boldsymbol{V}_{p}(s-1) + c_{1}z_{1}\left(\boldsymbol{p}_{best} - \boldsymbol{X}_{p}(s-1)\right) + c_{2}z_{2}\left(\boldsymbol{l}_{best} - \boldsymbol{X}_{p}(s-1)\right)$$
(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

$$\boldsymbol{V}_{p}(s) = w \boldsymbol{V}_{p}(s-1) + c_{1} z_{1} \left(\boldsymbol{p}_{best} - \boldsymbol{X}_{p}(s-1) \right) + c_{2} z_{2} \left(\boldsymbol{g}_{best} - \boldsymbol{X}_{p}(s-1) \right)$$
(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 S at early iterations and exploitation of promising neighborhoods in 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 *S*. The constriction factor is defined in Equation 2.30, where $\phi = c_1 + c_2$

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

The corresponding velocity update equation is shown in Equation 2.31.

$$V_{p}(s) = \chi \left[V_{p}(s-1) + c_{1}z_{1} \left(\boldsymbol{p}_{best} - \boldsymbol{X}_{p}(s-1) \right) + c_{2}z_{2} \left(\boldsymbol{g}_{best} - \boldsymbol{X}_{p}(s-1) \right) \right]$$
(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 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 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 outerloop 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 solarelectric 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 bilevel 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	Nomenclat	ure		
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	
g best		=	global best position in the solution space	
\boldsymbol{g}_k		=	unit vector perpendicular to v_k and 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	
l _{best}		=	neighborhood best position in the solution space	
т		=	number of particles in the swarm	
n		=	number of design variables in the nonlinear function to be optimized	
Р		=	period of the initial orbit, sec	
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, <i>km</i>	
\boldsymbol{r}_k^*		=	actual position vector of spacecraft at k^{th} time of entry into exclusion	
			zone, <i>km</i>	

$\boldsymbol{r}_{k_t^-}$	=	position vector at the instant just before the k^{th} impulse, km		
r ₀	=	initial position vector, km		
S	=	Solution space encompassing all <i>n</i> 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		
$\boldsymbol{V}_{p}\left(s ight)$	=	<i>n</i> -dimensional velocity vector of the p^{th} particle at the s^{th} iteration		
v_{max}^i, v_{min}^i	=	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, <i>km/sec</i>		
$\boldsymbol{v}_{k_t^-}, \boldsymbol{v}_{k_t^+}$	=	velocity vectors at the instant just before and just after the k^{th}		
		impulse, <i>km/sec</i>		
v_0	=	initial velocity vector, km/sec		
$X_{p}(s)$	=	<i>n</i> -dimensional position vector of the p^{th} particle at the s^{th} iteration		
x_{max}^i, x_{min}^i	=	upper and lower bounds on the position of the i^{th} design variable		
χ	=	swarm constriction factor		
φ, λ	=	geocentric latitude and longitude, °		
$ heta_k$	=	angle defining position of spacecraft on the k^{th} exclusion ellipse, rad		
<i>V_{enter}</i>	=	true anomaly of the spacecraft as it enters the latitude band of the		
		exclusion zone		
μ	=	Earth's gravitational parameter, km^3/sec^2		
ΔV_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 g_{best} and the best position ever visited by that specific particle 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. 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 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 X_p and velocity V_p in *S* are *n*-dimensional vectors where the *i*th dimension of each vector corresponds to the *i*th design variable:

$$X_p = \begin{bmatrix} x_p^1, x_p^2, \dots, x_p^n \end{bmatrix}$$

$$V_p = \begin{bmatrix} v_p^1, v_p^2, \dots, v_p^n \end{bmatrix}$$
(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 $v_{min}^i = -v_{max}^i$:

$$\begin{aligned} x_{min}^{i} &\leq x_{p}^{i} &\leq x_{max}^{i} \\ v_{min}^{i} &\leq v_{p}^{i} &\leq v_{max}^{i} \end{aligned}$$
 (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 $(J_{p_{best}})$, then $J_{p_{best}}$ is set equal to $J_p(s)$ and the best position ever visited by the particle p_{best} is updated to the current particle position $X_p(s)$.

The velocity of each particle at the s^{th} iteration $V_p(s)$ is a function of the position and velocity of that particle at the previous iteration, as well as p_{best} . The velocity update for the global version of the PSO is also dependent on 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

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}$$
(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:

$$V_{p}(s) = \chi \left[V_{p}(s-1) + c_{1}z_{1} \left(\boldsymbol{p}_{best} - \boldsymbol{X}_{p}(s-1) \right) + c_{2}z_{2} \left(\boldsymbol{g}_{best} - \boldsymbol{X}_{p}(s-1) \right) \right]$$
(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 I_{best} is set equal to $X_p(s)$. The velocity update for the local PSO variant used in this research is shown in Equation 3.5:

$$V_{p}(s) = \chi \left[V_{p}(s-1) + c_{1}r_{1} \left(\boldsymbol{p}_{best} - \boldsymbol{X}_{p}(s-1) \right) + c_{2}r_{2} \left(\boldsymbol{l}_{best} - \boldsymbol{X}_{p}(s-1) \right) \right]$$
(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:

$$X_p(s) = X_p(s-1) + V_p(s)$$
 (3.6)

Similarly, any component of 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, \mathbf{r}_0 and \mathbf{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 r_1 , and the velocity vector at entry v_1 . The twobody and spherical Earth assumptions make it possible to analytically determine the true anomaly of the spacecraft v_{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 v_{enter} is given by

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

The true anomaly v_{enter} corresponds to the inertial position and velocity vectors denoted with \mathbf{r}_{enter} and \mathbf{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 v_{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 v_{enter} . A similar process is used to determine the true anomaly v_{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, v_1 , is equal to v_{enter} . The second case occurs when $\lambda_{enter} < \lambda_{min}$ and $\lambda_{min} < \lambda_{exit}$. This scenario implies $v_{enter} < v_1 < v_{exit}$ and requires interpolation to determine v_1 . The satellite's expected entry state into the exclusion zone can be found from v_1 .

The expected specific angular momentum vector of the orbit h_1 is defined by r_1 and v_1 :

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

Additionally, a unit vector perpendicular to v_1 and 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 \mathbf{r}_1 and oriented such that a_e is aligned with \mathbf{v}_1 . The satellite must arrive at some point on the exclusion ellipse rather than \mathbf{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 \mathbf{v}_1 in the direction of \mathbf{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):

$$R_{e} = \frac{a_{e}b_{e}}{\sqrt{b_{e}^{2}\cos^{2}\theta_{1} + a_{e}^{2}\sin^{2}\theta_{1}}}$$
(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):

$$\boldsymbol{r}_{1}^{*} = \boldsymbol{r}_{1} + R_{e} \cos \theta_{1} \frac{\boldsymbol{v}_{1}}{|\boldsymbol{v}_{1}|} + R_{e} \sin \theta_{1} \boldsymbol{g}_{1}$$
(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 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 r_{1_t} and 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 $\mathbf{r}_{1_t^-}$ and $\mathbf{v}_{1_t^-}$ to \mathbf{r}_1^* in T_1 s is ΔV_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 $(\mathbf{r}_0, \mathbf{v}_0, t_0)$ and determines the arrival state into the exclusion zone defined by t_1 , \mathbf{r}_1 , and \mathbf{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 , \mathbf{r}_1^* and \mathbf{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 \mathbf{r}_2 and \mathbf{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:

minimize $J = \Delta V_1$ m/s		
subject to:		
$r_0 = [6800 \ 0 \ 0] \text{ km}$	$v_0 = [0 5.41377 5.41377]$ km/sec	
$(\phi_{min},\phi_{max})=(-10^\circ,10^\circ)$	$(\lambda_{min}, \lambda_{max}) = (-50^\circ, -10^\circ)$	(3.13)
$a_e = 150 \text{ km}, b_e = 0.1 a_e$		
$1200 \text{ s} \le T_1 \le P$	$0 \le \theta_1 \le 2\pi$	
$R_{a_1} \leq 6850 \text{ km}$	$6750 \text{ km} \le R_{p_1}$	

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 $a_e = 150 \text{ km}$ and $b_e = 15 \text{ km}$

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.

		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%

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

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 $\mathbf{r}_0 = [7300\ 0\ 0]$ km and $\mathbf{v}_0 = [0\ 5.22507\ 5.22507]$ 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

			a_e/b_e (km)												
<i>r</i> 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			

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





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:

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

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

							a_e/b_e (km)				
<i>r</i> 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

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



(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 ($r_0 = 6800 \text{ km}$)

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.

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

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.

						a_e/b	_e km					
<i>r</i> 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
	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
	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	1.719	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

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



(a) T_1 , T_2 and T_3 as functions of exclusion ellipse (b) θ_1 , θ_2 , and θ_3 as functions of exclusion ellipse size size

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

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.

a_e/b_e , km	<i>T</i> 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 3.5: Optimal single pass RTM for various exclusion ellipse sizes

a_e/b_e , km	<i>T</i> ₁ , s	θ_1 , rad	<i>T</i> ₂ , s	θ_2 , rad	Δ_{V_1}	Δ_{V_2}	J, m/s	Time, sec	No. Optimal/Total, %
				$r_0 =$	= 6800 k	m			
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
				<i>r</i> ₀ =	= 7300 k	m			
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 3.6: Optimal double pass RTM for various exclusion ellipse sizes

a_e/b_e , km	<i>T</i> ₁ , s	θ_1 , rad	<i>T</i> ₂ , s	θ_2 , rad	<i>T</i> ₃ , s	θ_3 , rad	J, m/s	Time, s	No. Optimal/Total, %
				r	₀ = 6800 kı	n			
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	$_{0} = 7300 \text{ km}$	n			
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

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

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

 (\mathbf{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 delivering the same effects. As low-thrust propulsion technology improves, the low-thrust responsive theater maneuvers delivering the same effects.

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

- 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*
- \mathbf{r}_k = expected position vector of satellite at k^{th} time of entry into exclusion zone, km
- \mathbf{r}_{k}^{*} = actual position vector of spacecraft at k^{th} time of entry into exclusion zone, km
- \mathbf{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

$$\mathbf{r}_0$$
 = initial position vector, km

$$t_0$$
 = initial time, sec

- t_k = expected k^{th} time of entry into exclusion zone, sec
- v_0 = initial velocity vector, km/sec
- v_k = expected velocity vector of spacecraft at k^{th} time of entry into exclusion zone, km/sec

- v_k^* = actual velocity vector of spacecraft at k^{th} time of entry into exclusion zone, km/sec
- v_{k-} = velocity vector of spacecraft at the instant just before the k^{th} impulse, km/sec
- $v_{k_{t}^{+}}$ = 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, g_{best} , and the best position ever visited by that specific particle, 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 g_{best} is replaced by 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, \mathbf{r}_0 and \mathbf{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, r_1 and v_1 , respectively. These quantities define the specific angular momentum vector h_1 and the g_1 vector, which define the orientation of the exclusion ellipse centered at r_1 . The definition of 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 v_1 and semiminor axis b_e is aligned with 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 v_1 in the direction of g_1 .

$$\boldsymbol{r}_{1}^{*} = \boldsymbol{r}_{1} + R_{e} \cos \theta_{1} \frac{\boldsymbol{v}_{1}}{|\boldsymbol{v}_{1}|} + R_{e} \sin \theta_{1} \boldsymbol{g}_{1}$$
(4.2)

where

$$R_{e} = \frac{a_{e}b_{e}}{\sqrt{b_{e}^{2}\cos^{2}\theta_{1} + a_{e}^{2}\sin^{2}\theta_{1}}}$$
(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 \mathbf{r}_{1_t} and velocity \mathbf{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 .

$$\dot{r} = V_r$$

$$\dot{\psi} = \frac{V_{\psi}}{r}$$

$$\dot{V}_r = \frac{V_{\psi}^2}{r} - \frac{\mu}{r^2} + A_T \sin \eta$$

$$\dot{V}_{\psi} = -\frac{V_{\psi}V_r}{r} + A_T \cos \eta$$
(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, \mathbf{r}_{1_t} and \mathbf{v}_{1_t} , respectively. As a result, r_{1_t} and v_{1_t} are simply the magnitudes of \mathbf{r}_{1_t} and \mathbf{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 r_1 .

$$\psi_1^* = \psi_1 + \delta \cos^{-1} \left(\frac{(R_e)^2 - (r_1)^2 - (r_1^*)^2}{-2r_1r_1^*} \right)$$
(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 \mathbf{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 .

$$\delta = \begin{cases} -1 & \text{if } \frac{\pi}{2} - \gamma_1 < \theta_1 < \frac{3\pi}{2} - \gamma_1 \\ 1 & \text{otherwise} \end{cases}$$
(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 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 , r_1^* , and 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.

minimize
$$J = \Delta V_1 = \int_{t_0}^{t_1} A_T dt$$

subject to:
Exclusion zone: $(\phi_{min}, \phi_{max}), (\lambda_{min}, \lambda_{max})$ (4.7)
 $R_{a_1} \le R_{a_{max}}$ $R_{p_{min}} \le R_{p_1}$
 $A_{T_{min}} \le A_T \le A_{T_{max}}$ $0 \le \eta \le 2\pi$

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

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

$$\sin \eta = -\frac{\lambda_{V_r}}{\lambda_V}$$

$$\cos \eta = -\frac{\lambda_{V_{\psi}}}{\lambda_V}$$
(4.9)

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

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

$$\begin{array}{rcl}
0 & \leq \theta_{1} \leq 2\pi \\
\sqrt{\left(\frac{2\mu}{R_{p_{min}}+R_{a_{max}}}\right)\frac{R_{p_{min}}}{R_{a_{max}}}} & \leq \mathbf{v}_{1}^{*} \leq \sqrt{\left(\frac{2\mu}{R_{p_{min}}+R_{a_{max}}}\right)\frac{R_{a_{max}}}{R_{p_{min}}}} \\
-\frac{\pi}{2} & < \gamma_{1}^{*} < \frac{\pi}{2}
\end{array}$$
(4.11)

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. 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 $r_0 = 6800 \text{ km}$ and $a_e = 150 \text{ km}$. 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].

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		

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

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- (b) Propellant mass savings for a 500 kg satellite thrust RTMs $(r_0 = 6800 \text{ km})$



(c) Propellant mass savings for a 500 kg satellite

$$(r_0 = 7300 \text{ km})$$

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.

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

r_0		a_e/b_e (km)										
(<i>km</i>)	(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

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

Figure 4.5 shows the thrust acceleration and switching condition along with the optimal pointing direction for each maneuver for the case with $r_0 = 6800 \text{ km}$ and $a_e = 150 \text{ km}$. 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 ($I_{sp} = 250$ s) 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 func-

(b) 1st maneuver optimal thrust pointing







tion

tion

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- (b) Propellant mass savings for a 500 kg satellite thrust RTMs $(r_0 = 6800 \text{ km})$



(c) Propellant mass savings for a 500 kg satellite ($r_0 = 7300 \text{ km}$)

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.

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

$$(4.13)$$

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 $r_0 = 6800 \text{ km}$ and $a_e = 150 \text{ km}$. The figures are representative of those seen for the other cases.

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

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

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 to arrive on the exclusion ellipse at the desired arrival time for significantly less ΔV than required for maneuvers occurring in fewer orbital revolutions.

r_0			a_e/b_e (km)									
(<i>km</i>)	(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
	ΔV_1	1.234	1.480	1.729	1.973	2.223	2.472	2.718	2.962	3.211	3.520	3.697
7300	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

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

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 func-



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







(b) 1st maneuver optimal thrust pointing



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



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

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

tion

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- (b) Propellant mass savings for a 500 kg satellite thrust RTMs ($r_0 = 6800 \text{ km}$)



(c) Propellant mass savings in for a 500 kg satellite

 $(r_0 = 7300 \text{ km})$

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

4.8 Appendix: Coordinate Transformation Matrices

$$R1(\tau) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \tau & \sin \tau \\ 0 & -\sin \tau & \cos \tau \end{bmatrix}$$
(4.14)
$$R2(\tau) = \begin{bmatrix} \cos \tau & 0 & -\sin \tau \\ 0 & 1 & 0 \\ \sin \tau & 0 & \cos \tau \end{bmatrix}$$
(4.15)
$$R3(\tau) = \begin{bmatrix} \cos \tau & \sin \tau & 0 \\ -\sin \tau & \cos \tau & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
(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	Nomenclatu	ıre	
l_{GEO}		=	true longitude at epoch of the arrival location on the
			geostationary orbit, rad
\mathbf{r}_{IJK}^{c} ,	$\mathbf{r}_{RSW}^c, \mathbf{r}_{CYL}^c$	=	position vectors of the chaser in the inertial, local vertical, local
			horizontal, cylinder coordinate frames, km
r^{g}_{IJK}		=	inertial position vector of the ground
			site, <i>km</i>
r^m_{IJK}		=	inertial position vector of the <i>m</i> th target
R_\oplus		=	radius of the earth, km
t_f		=	maneuver completion time, sec
t_{enter}^k	, t_{exit}^k	=	entry, exit times of the <i>m</i> th target's <i>k</i> th pass over the ground site, <i>sec</i>
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
<i>t</i> ₃		=	coast time following cooperative inspection phase, sec
t_4		=	time of flight for maneuver to the final mission orbit, sec
$\boldsymbol{v}_{IJK}^{c},$	\boldsymbol{v}_{RSW}^c	=	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, <i>rad</i>

ϵ_{max}^{c}	=	maximum allowable elevation angle of the chaser with respect
		to the ground site, <i>rad</i>
ϵ^{g}_{min}	=	minimum elevation angle required by ground site for line-of-sight
		contact with the <i>m</i> th target, <i>rad</i>
ϵ^m	=	elevation angle of the target satellite, rad
ϕ, λ	=	geocentric latitude and longitude, rad
ω_\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
$oldsymbol{ ho}_{RSW}^{\hat{R}},oldsymbol{ ho}_{RSW}^{\hat{S}},oldsymbol{ ho}_{RSW}^{\hat{W}}$	=	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, <i>km</i>
$oldsymbol{ ho}_{SEZ}^{\hat{Z}}$	=	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 outerloop 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 bilevel 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 *m*th of *M* targets for the duration of the target's *k*th 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 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 *m*th 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 \mathbf{r}_{IJK}^g at any

time t is

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

Where, R_{\oplus} and ω_{\oplus} are the radius and the rotation rate of the Earth, respectively. The position vector of the target at time *t* is $\mathbf{r}_{IJK}^m(t)$ 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

$$\boldsymbol{\rho}_{IJK}(t) = \boldsymbol{r}_{IJK}^{m}(t) - \boldsymbol{r}_{IJK}^{g}(t)$$
(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].

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

Where *R*1, *R*2, and *R*3 are right handed rotation matrices about an angle τ and are defined in Equations 5.4, 5.5, and 5.6.

$$R1(\tau) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \tau & \sin \tau \\ 0 & -\sin \tau & \cos \tau \end{bmatrix}$$
(5.4)
$$R2(\tau) = \begin{bmatrix} \cos \tau & 0 & -\sin \tau \\ 0 & 1 & 0 \\ \sin \tau & 0 & \cos \tau \end{bmatrix}$$
(5.5)
$$R3(\tau) = \begin{bmatrix} \cos \tau & \sin \tau & 0 \\ -\sin \tau & \cos \tau & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
(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, $\rho_{sez}^{\hat{z}}$ is the Zenith component of ρ_{sez} and ϵ_{min}^{g} is the minimum elevation angle required for line of sight contact from the ground site.

$$\epsilon^{m} = \sin^{-1} \left(\frac{\rho_{sez}^{\hat{z}}(t)}{|\rho_{sez}(t)|} \right) > \epsilon_{min}^{g}$$
(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 (\hat{R}) axis aligned with r_{IJK}^m . A second (\hat{S}) 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 (\hat{Z}) axis points in the orbit normal direction. The coordinates of $\rho_{IJK}(t)$ 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, $(u^m(t))$.

$$\rho_{RSW}(t) = R3(u^{m}(t)) * R1(i^{m}) * R3(\Omega^{m}) * \rho_{IJK}(t)$$
(5.8)

The angle $\alpha(t)$, measured from the fundamental (orbital) plane in the RSW frame to $\rho_{RSW}(t)$ is found according to Equation 5.9, where $\hat{\rho}_{RSW}(t)$ is the unit vector corresponding to $\rho_{RSW}(t)$. Similarly, the angle $\beta(t)$, measured from the primary axis in the RSW frame to $\rho_{RSW}(t)$ 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.

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

$$\tan\beta(t) = \frac{\hat{\rho}_{RSW}^{S}(t)}{\hat{\rho}_{RSW}^{\hat{R}}(t)}$$
(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 $\rho_{RSW}(t)$. Any vector in the cylinder frame, $\mathbf{r}_{CYL}(t)$, can be converted to a vector in the RSW frame, $\mathbf{r}_{RSW}(t)$, according to Equation 5.11.

$$\boldsymbol{r}_{RSW}(t) = R3\left(-\beta(t)\right) * R2\left(\alpha(t)\right) * \boldsymbol{r}_{CYL}(t)$$
(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_{enter}^k and t_{exit}^k and inside the cylinder at all times in between. The cylinder is defined in the CYL frame such that one base is at x_{CYL}^{min} and the other is at x_{CYL}^{max} with a length equal to x_{CYL}^{max} minus x_{CYL}^{min} . The variables $x_{CYL}(t_{enter}^k)$ and $x_{CYL}(t_{exit}^k)$ 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.

$$\boldsymbol{r}_{CYL}^{c}\left(t_{enter}^{k}\right) = \begin{bmatrix} x_{CYL}\left(t_{enter}^{k}\right) \\ 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{r}_{CYL}^{c}\left(t_{exit}^{k}\right) = \begin{bmatrix} x_{CYL}\left(t_{exit}^{k}\right) \\ 0 \\ 0 \end{bmatrix} \qquad (5.12)$$

The vectors shown in Equation 5.12 are rotated into vectors in the RSW frame, $\mathbf{r}_{RSW}^c(t_{enter}^k)$ and $\mathbf{r}_{RSW}^c(t_{exit}^k)$, using Equation 5.11. [46] describes a method to find the entry and exit velocities in the RSW frame, $\mathbf{v}_{RSW}^c(t_{enter}^k)$ and $\mathbf{v}_{RSW}^c(t_{exit}^k)$, 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 $\mathbf{r}_{RSW}^c(t_{enter}^k)$ and $\mathbf{r}_{RSW}^c(t_{exit}^k)$, respectively. The time of flight is $t_{enter}^k - t_{exit}^k$ seconds. The relative position and velocity vectors at t_{enter}^k are converted to inertial coordinates using Equations 5.13 and 5.14. The inertial state of the chaser at t_{exit}^k 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.

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

$$\boldsymbol{v}_{IJK}^{c}\left(\boldsymbol{t}_{enter}^{k+}\right) = R3\left(-\Omega^{m}\right) * R1\left(-i^{m}\right) * R3\left(-u^{m}\left(\boldsymbol{t}_{enter}^{k}\right)\right) * \boldsymbol{v}_{RSW}^{c}\left(\boldsymbol{t}_{enter}^{k}\right)$$
(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 $u^c(t_1)$ where t_1 is the time of maneuver initiation. The position and velocity vectors, $\mathbf{r}_{IJK}^c(t_1^-)$ and $\mathbf{v}_{IJK}^c(t_1^-)$, 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_{enter}^k with the inertial position specified by $\mathbf{r}_{IJK}^c(t_{enter}^k)$. The time of flight to complete the maneuver is t_2 seconds, which implies maneuver initiation occurs at $t_1 = t_{enter}^k - t_2$ 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 $\mathbf{v}_{IJK}^c(t_1^+)$ and $\mathbf{v}_{IJK}^c(t_{enter}^{k-})$, 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.

$$\Delta V_1 = |\mathbf{v}_{IJK}^c(t_1^+) - \mathbf{v}_{IJK}^c(t_1^-)|$$
(5.15)

$$\Delta V_2 = |\mathbf{v}_{IJK}^c(t_{enter}^{k+}) - \mathbf{v}_{IJK}^c(t_{enter}^{k-})|$$
(5.16)

The path of the chaser is constrained for $t_1 \le t \le t_{enter}^k$ according to Equation 5.17, where $\epsilon^c(t)$ is the elevation angle of the chaser with respect to the ground site at any time and ϵ_{max}^c 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, $\mathbf{r}_{IJK}^c \left(t_{exit}^k + t_3^-\right)$ and $\mathbf{v}_{IJK}^c \left(t_{exit}^k + t_3^-\right)$. 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, $l_{GEO}(t_f)$, which corresponds to inertial position and velocity vectors, $\mathbf{r}_{IJK}^c \left(t_f^+\right)$ and $\mathbf{v}_{IJK}^c \left(t_f^+\right)$, respectively. The chaser travels from $\mathbf{r}_{IJK}^c \left(t_{exit}^k + t_3^-\right)$ to $\mathbf{r}_{IJK}^c \left(t_f\right)$ in t_4 seconds, which lends itself to a Lambert targeting solution. The initial and final velocities on the transfer orbit, $\mathbf{v}_{IJK}^c \left(t_{exit}^k + t_3^+\right)$ and $\mathbf{v}_{IJK}^c \left(t_f^-\right)$, 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.

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

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

The path of the chaser is constrained for $t_{exit}^k \le t \le t_f$ 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: $u_c(t_1), t_2, x_{CYL}(t_{enter}^k), x_{CYL}(t_{enter}^k), t_3, l_{GEO}(t_f)$, 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

minimize
$$J(\mathbf{x}) = \Delta V_1 + \Delta V_2 + \Delta V_3 + \Delta V_4$$
 km/s
where $\mathbf{x} = [m, k, u^c(t_1), t_2, x_{CYL}(t_{enter}^k), x_{CYL}(t_{exit}^k), t_3, l_{GEO}(t_f), t_4]$
subject to:
 $1 < m < M$
 $1 < k < K$
 $0 \le u^c(t_1) < 2\pi$
 $t_{exit}^k < t_{max}$
 $1 < t_2 < t_{enter}^k$
 104
 $x_{CYL}^{min} \le x_{CYL}(t_{enter}^k), x_{CYL}(t_{exit}^k) \le x_{CYL}^{max}$
 $0 \le t_3 < t_{enter}^{k+1} - t_{exit}^k$
(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 $\phi = 45^{\circ}$ and $\Lambda = 0^{\circ}$. Further, the cooperative inspection lasts for the duration of the target's horizon to horizon contact with the ground site. In other words, ϵ_{min}^g is set equal to zero. Finally, the chaser's elevation angle with respect to the ground site ϵ_{max}^c was defined to be equal to one degree. The cylinder bases x_{CYL}^{min} and x_{CYL}^{max} 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

	a (km)	i (°)	$u\left(t_0\right)(^o)$	# 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

Table 5.1: Initial conditions of the chaser and targets



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.

Rank	Satellite	Pass	$ar{J}$
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

Table 5.3: Best three target costs found by enumeration

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 targetpass combinations yielded no feasible solutions after 20 PSO runs, while no targetpass 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 *m*th target's *k*th 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.

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%		

Table 5.4: Outer-loop optimization routines

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 innerloop 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.

	m/k	$u^{c}(t_{1})$	t_2	x_{CYL}^{enter}	x_{CYL}^{exit}	<i>t</i> ₃	l_{GEO}	t_4	$ar{J}$
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

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

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 (b) Chaser path in the cylinder frame during inspecframe tion



(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

 \overline{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

Metric		PP	GP	PPi	GPi
	$ar{J}_{min}$	1.01747	1.01730	1.01726	1.01730
Cost	$ar{J}_{max}$	1.01881	1.01903	1.01838	1.01919
Cost	$ar{J}_{mean}$	1.01797	1.01799	1.01784	1.01798
	$\sigma_{ar{J}}$	0.00031	0.00043	0.00031	0.00038
	f_{min}	0.677	1.730	0.277	0.624
Millions of Cost Expetion Evolutions	f_{max}	2.329	3.351	0.979	1.308
withous of Cost Function Evaluations	<i>f</i> mean	1.512	2.744	0.607	0.942
	σ_{f}	0.459	0.358	0.150	0.142

Table 5.6: Bi-level algorithm performance comparison



(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 targetpass 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

	a (km)	<i>i</i> (°)	$\Omega\left(^{\circ} ight)$	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

Table 5.7: Target satellites' initial conditions

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 $\bar{J} = 2$.

Rank	m/k	$u^{c}\left(t_{1} ight)$	t_2	x_{CYL}^{enter}	x_{CYL}^{exit}	<i>t</i> ₃	l_{GEO}	t_4	$ar{J}$
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

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



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.





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 outerloop 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 deltavelocity 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 fifteentarget 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 v is the velocity vector of the spacecraft.

$$T = \frac{1}{2}m(\mathbf{v} \cdot \mathbf{v}) = \frac{1}{2}m(\dot{r}^2 + r^2\dot{\psi}^2)$$
(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

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

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

$$p_r = \frac{\partial \mathscr{L}}{\partial \dot{r}} = m\dot{r}$$

$$p_{\phi} = \frac{\partial \mathscr{L}}{\partial \dot{\psi}} = mr^2 \dot{\psi}$$
(A.4)

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

$$\dot{r} = \dot{q}_r = \frac{p_r}{m}$$

$$\dot{\psi} = \dot{q}_{\phi} = \frac{p_{\psi}}{mr^2}$$
(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.

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

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

$$\dot{p}_r = -\frac{\partial \mathscr{H}}{\partial r} = \frac{p_{\psi}^2}{mr^2} + \frac{\mu}{r^2}m$$

$$\dot{p}_{\psi} = -\frac{\partial \mathscr{H}}{\partial \psi} = 0$$
(A.7)

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

$$\ddot{r} = \frac{r^2 \dot{\psi}^2}{r} - \frac{\mu}{r^2}$$

$$\ddot{\psi} = -\frac{2i\dot{\psi}}{r^2}$$
(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.

$$V_r = \dot{r}$$

$$V_{\psi} = r\dot{\psi}$$
(A.9)

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

$$\dot{r} = V_r$$

$$\dot{\psi} = \frac{V_{\psi}}{r}$$

$$\dot{V}_r = \ddot{r}$$

$$\dot{V}_{\psi} = \dot{r}\dot{\psi} + r\ddot{\psi}$$
(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.

$$\dot{r} = V_r$$

$$\dot{\psi} = \frac{V_{\psi}}{r}$$

$$\dot{V}_r = \frac{V_{\psi}^2}{r} - \frac{\mu}{r^2}$$

$$\dot{V}_{\psi} = -\frac{V_r V_{\psi}}{r}$$
(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, r_{RSW} and v_{RSW} respectively, are given by Equation B.1.

$$r_{RSW} = x\hat{R} + y\hat{S} + z\hat{W}$$

$$v_{RSW} = \dot{x}\hat{R} + \dot{y}\hat{S} + \dot{z}\hat{W}$$
(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].

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

$$\begin{split} \tilde{x}(t) &= \frac{1}{2\pi} \dot{\tilde{x}}_0 \sin(2\pi \tilde{t}) - \left(3\tilde{x}_0 + \frac{1}{\pi}\right) \dot{\tilde{y}}_0 \cos(2\pi \tilde{t}) + \left(4\tilde{x}_0 + \frac{1}{\pi} \dot{\tilde{y}}_0\right) \\ \tilde{y}(t) &= \left(6\tilde{x}_0 + \frac{2}{\pi} \dot{\tilde{y}}_0\right) \sin(2\pi \tilde{t}) + \frac{1}{\pi} \dot{\tilde{x}}_0 \cos(2\pi \tilde{t}) - \left(12\pi \tilde{x}_0 + 3\dot{\tilde{y}}_0\right) \tilde{t} + \left(\tilde{y}_0 - \frac{1}{\pi} \dot{\tilde{y}}_0\right) \\ \tilde{z}(t) &= \tilde{z}_0 \cos(2\pi \tilde{t}) + \frac{1}{2\pi} \dot{\tilde{z}}_0 \sin(2\pi \tilde{t}) \\ \dot{\tilde{x}}(t) &= \dot{\tilde{x}}_0 \cos(2\pi \tilde{t}) + \left(6\pi \tilde{x}_0 + 2\dot{\tilde{y}}_0\right) \sin(2\pi \tilde{t}) \\ \dot{\tilde{y}}(t) &= \left(12\pi \tilde{x}_0 + 4\dot{\tilde{y}}_0\right) \cos(2\pi \tilde{t}) - 2\dot{\tilde{x}}_0 \sin(2\pi \tilde{t}) - \left(12\pi \tilde{x}_0 + 3\dot{\tilde{y}}_0\right) \\ \dot{\tilde{z}}(t) &= -2\pi \tilde{z}_0 \sin(2\pi \tilde{t}) + \dot{\tilde{z}}_0 \cos(2\pi \tilde{t}) \end{split}$$
(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 $c_1 + c_2 > 4$.

$$\begin{array}{rcl}
2 & < & c & \leq & 3.5 \\
20 & < & s & < & 200
\end{array}$$
(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.

S	с	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

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

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.

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

Iterations Convergence Percentage S с min max avg global local other 2.35 488.80 75** 70*** 298.15 2.11 2.17 182.85* 70*** 2.23 242.60 623.90*** 2.76 30* 3.00 563.45 35** 319.10 2.64 2.58 567.55 2.53 319.15 2.70 335.65 2.94 880.70* 651.10** 2.88 2.82 487.70 2.29 183.10** 2.05 311.70 80^{\star} 2.41 313.80 214.20*** 40*** 2.47

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

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.

$$2.05 \leq c \leq 2.5$$

$$30 \leq s \leq 120$$
(C.3)

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

S	с	min	max	avg	global	local	other
120	2.19	70	1000	398.65**	65	35	0
53	2.08	94	668	230.15	85*	15	0
69	2.11	80	773	219.85	65	35	0
86	2.13	79	553	223.30	70***	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***	45	55	0
109	2.30	73	629	215.70	65	35	0
75	2.28	102	1000	247.35	70***	30	0
30	2.36	45	839	161.05*	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**	20	0
103	2.22	60	682	211.50	70***	30	0
41	2.25	51	549	162.60**	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.

$$\begin{array}{rcl} 2.05 &\leq c &\leq 2.3 \\ 30 &\leq s &\leq 90 \end{array} \tag{C.4}$$

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

S	с	min	max	avg	global	local	other
56	2.08	77	761	217.85	70* * *	30	0
45	2.07	108	1000	280.15***	80*	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**	25	0
49	2.30	46	694	153.55	40*	60	0
41	2.21	55	220	108.40*	65	35	0
83	2.19	65	1000	307.35**	70***	30	0
60	2.18	66	908	212.60	60	40	0
30	2.20	46	728	178.85* * *	45**	55	0
75	2.28	51	694	259.75	70***	30	0
64	2.27	47	1000	248.55	50***	50	0
53	2.25	54	942	199.15	75**	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**	25	0
38	2.16	61	826	161.00**	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.

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

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

s	с	min	max	avg	global	local	other
80	2.13	74	880	269.35***	70***	30	0
43	2.08	96	966	298.55**	65	35	0
52	2.09	108	1000	327.60*	75**	25	0
61	2.10	77	546	177.65***	80*	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**	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*	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.

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

Appendix D: Code for Impulsive Responsive Theater Maneuvers

```
D.1 Single Pass RTMs
```

D.1.1 Single Pass RTM Data Script

```
1 t0 = 0;
2 \text{ GMST0} = 0;
3 latlim = [-10 10]*pi/180;
4 longlim = [-50 -10]*pi/180;
5
6 wgs84data
7 global MU
8 r0vec = [6800 7300;0 0;0 0];
9 v0vec = [0 0;5.41376581448788 sqrt(MU/7300)/sqrt(2);5.41376581448788
      sqrt(MU/7300)/sqrt(2)];
10 swarm = 30;
iter = 1000;
  aevec = [50 60 70 80 90 100 110 120 130 140 150];
12
<sup>13</sup> bevec = [5 6 7 8 9 10 11 12 13 14 15];
14 Rmaxvec = [6850 7350];
15 Rminvec = [6750 7250];
_{16} prec = [2;5;16];
17
  for k = 1:1
18
19
       r0 = r0vec(:,k);
20
       v0 = v0vec(:,k);
21
       Rmax = Rmaxvec(k);
22
       Rmin = Rminvec(k);
23
       [a, ecc, inc, RAAN, w, nu0] = RV2COE(r0, v0);
24
       period = 2*pi*sqrt(a^3/MU);
25
```

```
26
27
       state0=[r0 v0];
28
       fprintf(fid, '\n\n\r %s %3i\r\n', 'r0=', norm(r0));
29
30
       for aa = 11:11
31
32
           ae = aevec(aa);
33
           be = bevec(aa);
34
35
           fprintf(fid, '\n\n\r %s %3i\r\n', 'swarm=', swarm);
36
           fprintf(fid,'%s %3i\r\n','ae=',ae);
37
           fprintf(fid,'%s %3i\r\n','be=',be);
38
           fprintf(fid, '%s %3i\r\n', 'maxiter=',iter);
39
           fprintf(fid,'%2s %10s %8s %8s %8s %8s\r\n','run #','T1','theta1'
40
               ,'J','iterations','Run Time');
41
           itn = zeros(20,1);
42
           rt = zeros(20, 1);
43
           tot_time = 0;
44
45
           for h = 20:20
46
47
                clear JG Jpbest gbest manDV
48
49
                tstart = tic;
50
51
                [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,
52
                   lat_exit,long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,
                   latlim,longlim);
```

53

```
[JG, Jpbest, gbest, x, iter_needed, preburn_state1, initial_target
54
                    ] = PSO_RTM_analytical_prec(2,[1200 period;0 2*pi],prec,
                   iter,swarm,rf1,vf1,ae,be,Rmax,Rmin,latlim,longlim,tf1);
55
                tend = toc(tstart)
56
57
                DV1 = norm(preburn_state1(8:10)*1000);
58
                manDV = round(JG*1000*10^{5})/10^{5};
59
                itn(h) = iter_needed;
60
                rt(h) = tend;
61
62
                if h == 1
63
                    minDV = manDV;
64
                    mincount = 1;
65
                elseif manDV < minDV</pre>
66
                    minDV = manDV;
67
                    mincount = 1;
68
                elseif manDV == minDV
69
                    mincount = mincount + 1;
70
                end
71
72
                fprintf(fid,'%2i %10.2f %8.5f %10.5f %4i %10.4f\r\n',h,gbest
73
                    (1),gbest(2),manDV,itn(h),rt(h));
           end
74
75
           gpercent = mincount/h*100;
76
           tot_time = tot_time + sum(rt);
77
           mintime = \min(rt);
78
           maxtime = max(rt);
79
           meantime = mean(rt);
80
           miniter = min(itn);
81
82
           maxiter = max(itn);
```

```
meaniter = mean(itn);
83
          fprintf(fid, '%s %8.5f\r\n', 'min time=', mintime);
84
         fprintf(fid,'%s %8.5f\r\n','max time=',maxtime);
85
          fprintf(fid,'%s %8.5f\r\n','avg time=',meantime);
86
         fprintf(fid,'%s %8.5f\r\n','min iter=',miniter);
87
          fprintf(fid,'%s %8.5f\r\n','max iter=',maxiter);
88
         fprintf(fid, '%s %8.5f\r\n', 'avg iter=', meaniter);
89
         fprintf(fid, '%s %i\r\n', 'global conv=', gpercent);
90
      end
91
92
      fprintf(fid, '\n\n\r %s','
```

');

93 end

D.1.1.1 Constants and Parameters

```
1 function wgs84data
```

2 %

3 %% function wgs84data

4 %% This script provides global conversion factors and WGS 84 constants

5 %% that may be referenced by subsequent MatLab script files and functions.

6 %% Note these variables are case-specific and must be referenced as such

- 7 %%
- 8 %% The function must be called once in either the MatLab workspace or from a
- 9 %% main program script or function. Any function requiring all or some of the
- 10 %% variables defined must be listed in a global statement as follows,
- 11 **%%**

140

```
12 %% global Deg Rad MU RE OmegaEarth SidePerSol RadPerDay SecDay Flat
     EEsqrd ...
           EEarth J2 J3 J4 GMM GMS AU HalfPI TwoPI Zero_IE Small
13 %%
     Undefined
14 %%
15 %% in part or in its entirety. Order is not relevent. Case is.
16 %%
17 %
     Originally written by Capt Dave Vallado
18 %%
        Modified and Extended for Ada by Dr Ron Lisowski
19 %%
        Extended from DFASMath.adb by Thomas L. Yoder, LtCol, Spring 00
20 %%
21 %
     22 global Deg Rad MU RE OmegaEarth SidePerSol RadPerDay SecDay Flat EEsqrd
     . . .
        EEarth J2 J3 J4 GMM GMS AU HalfPI TwoPI Zero_IE Small Undefined
23
           g0
24
25 %% Degrees and Radians
       Deg=180.0/pi;
                                                   %% deg/rad
26
       Rad= pi/180.0;
                                                   %% rad/deg
27
28
 %% Earth Characteristics from WGS 84
29
       MU=398600.5;
                                                          %% km^3/
30
          sec<sup>2</sup>
       RE=6378.137;
                                                          %% km
31
       OmegaEarth=0.000072921151467; %% rad/sec
32
       SidePerSol=1.00273790935;
                                        %% Sidereal Days/Solar Day
33
       RadPerDay=6.30038809866574;
                                    %% rad/day
34
```

```
141
```

```
SecDay=86400.0;
                                                                %% sec/day
35
         Flat=1.0/298.257223563;
36
                                                       %%
         EEsqrd=(2.0-Flat)*Flat;
37
         EEarth=sqrt(EEsqrd);
38
         J2 = 0.00108263;
39
         J3 = -0.00000254;
40
         J4 = -0.00000161;
41
         g0 = 9.81;
42
43
      Moon & Sun Characteristics from WGS 84
44
  %%
         GMM= 4902.774191985;
                                                                %% km^3/sec^2
45
                                                                %% km^3/sec^2
         GMS= 1.32712438E11;
46
         AU= 149597870.0;
                                                                %% km
47
48
      HALFPI, PI2
                             PI/2, & 2PI in various names
  %%
49
         HalfPI= pi/2.0;
50
         TwoPI= 2.0*pi;
51
52
         Zero_{IE} = 0.015;
                                              %% Small number for incl & ecc
53
             purposes
                   = 1.0E-6;
                                              %% Small number used for
         Small
54
             tolerance purposes
         Undefined= 999999.1;
55
```

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

```
1 function [a,ecc,inc,RAAN,w,nu] = RV2COE(r,v)
2
3 %Author: Dan Showalter 18 Oct 2012
4
5 %Purpose: Compute classical orbital elements for a position and velocity
6 %vector. Based on algorithm in Bate/Mueller/White Fundamentals of
```

```
7 %Astrodynamics
8
9 %% Algorithm
10 global MU
11
12 khat = [0;0;1];
13
14 % calculate angular momentum vector
15 h = cross(r,v);
16
17 % calculate nodal vector
18 n = cross(khat, h);
19
20 %calculate eccentricity vector
21 evec = 1/MU*((norm(v)^2 - MU/norm(r))*r - dot(r,v)*v);
22
23 % eccentricity
24 ecc = norm(evec);
25
  % compute specific mechanical energy
26
27
  SME = norm(v)^2/2 - MU/norm(r);
28
29
30 % compute semimajor axis
a = -MU/(2*SME);
32
33 %compute inclination
_{34} inc = acos(h(3)/norm(h));
35
36 % compute RAAN
37 RAAN = acos(n(1)/norm(n));
38
```

```
143
```

```
39 if n(2) < 0
       RAAN = 2*pi - RAAN;
40
  end
41
42
  if ecc <= 0.00001
43
       ecc = 0;
44
       w = 0;
45
       nu = acos(dot(n,r)/(norm(n)*norm(r)));
46
47
48
           if imag(nu) ~= 0
49
               temp = dot(n,r)/(norm(n)*norm(r));
50
               if abs(temp) > 1
51
                   temp = sign(temp)*1;
52
                   nu = acos(temp);
53
               end
54
           end
55
56
           if r(3) < 0
57
               nu = 2*pi - nu;
58
           end
59
  else
60
       w = acos(dot(n, evec)/(norm(n)*norm(evec)));
61
62
       if evec(3) < 0
63
           w = 2*pi - w ;
64
       end
65
       nu = acos(dot(evec,r)/(norm(evec)*norm(r)));
66
       if imag(nu) ~= 0
67
           temp = dot(evec,r)/(norm(evec)*norm(r));
68
           if abs(temp) > 1
69
                temp = sign(temp)*1;
70
```

```
71 nu = acos(temp);
72 end
73 end
74 if dot(r,v) < 0
75 nu = 2*pi - nu;
76 end
77 end</pre>
```

D.1.1.3 Determine Spacecraft Entry into Exclusion Zone

```
1 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)
2 %UNTITLED2 This function takes a spacecraft's initial position/velocity
3 %vectors, initial time, initial greenwich mean time and latitude and
4 %longitude limits and produces the spacecraft's first entry and exit
5 %conditions into the exclusion zone
6
7 %INPUTS
      r0 = inertial initial position vector (km)
  %
8
      v0 = inertial initial velocity vector (km)
  %
9
  %
      GMST0 = initial greenwhich mean standard time
10
  %
      t0 = initial time (sec)
11
12
13 %OUTPUTS
  %
      R_enter = inertial entry position into exclusion zone (km)
14
      V_enter = inertial velocity vector into exclusion zone (km)
15
  %
      t_enter = entry time into exclusion zone
  %
16
      lat_enter = latitude of spacecraft when it enters exclusion zone (
17 %
      rad)
18 %
      long_enter = longitude of spacecraft when it enters exclusion zone (
      rad)
      R_exit = inertial exit position out of exclusion zone (km)
19 %
```

```
%
       V_exit = inertial velocity vector out of exclusion zone (km)
20
       t_exit = exit time into exclusion zone
21
  %
       lat_exit = latitude of spacecraft when it exits exclusion zone (rad)
  %
22
  %
       long_exit = longitude of spacecraft when it exits exclusion zone (
23
      rad)
  %%
24
25 wgs84data
  global MU
26
27
  longlim_temp = longlim;
28
  if longlim(2) < 0</pre>
29
       longlim_temp(2) = 2*pi + longlim(2);
30
  end
31
  if longlim(1) < 0
32
       longlim_temp(1) = 2*pi + longlim(1);
33
   end
34
35
  if longlim_temp(1) > longlim_temp(2)
36
       longlim_temp(1) = longlim_temp(1) - 2*pi;
37
       weird_flag = 1;
38
  else
39
       weird_flag = 0;
40
  end
41
42
   zone_long_diff = longlim_temp(2) - longlim_temp(1);
43
44
45
  [a, ecc, inc, RAAN, w, nu0] = RV2COE(r0, v0);
46
47
  period = 2*pi*sqrt(a^3/MU);
48
49 %% Find spacecraft entry and exit points into exclusion zone
50 %determine exclusion zone entry/exit times underneath orbit plane
```

```
146
```

```
51 [nu_enter_AN,nu_exit_AN,nu_enter_DN,nu_exit_DN] = exclusion_nu_intercept
      (latlim,inc,w);
52
53 %=======Ascending node opportunity
      ______
  %Determine inertial position vector to nu_enter_AN and nu_exit_AN
54
  [R_enter_AN, V_enter_AN] = COE2RV(a, ecc, inc, RAAN, w, nu_enter_AN);
55
  [R_exit_AN,V_exit_AN] = COE2RV(a,ecc,inc,RAAN,w,nu_exit_AN);
56
57
  %determine time of flight from nu0 to nu_enter_AN and nu_exit_AN
58
  [TOF_enter_AN] = TOF_from_nu(a,ecc,nu0,nu_enter_AN,0);
59
  [TOF_exit_AN] = TOF_from_nu(a,ecc,nu0,nu_exit_AN,0);
60
61
  if TOF_enter_AN < 20 && TOF_enter_AN > 0
62
      TOF_enter_AN = TOF_enter_AN + period;
63
      TOF_exit_AN = TOF_exit_AN + period;
64
  end
65
66
  if TOF_exit_AN < TOF_enter_AN</pre>
67
      if TOF_enter_AN > 0
68
          TOF_exit_AN = TOF_exit_AN + period;
69
70
      else
          TOF_enter_AN = TOF_enter_AN + 2*period;
71
          TOF_exit_AN = TOF_exit_AN + period;
72
      end
73
  end
74
75
76 %=======Descending node opportunity
      _____
77 %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);
78
79
  [R_exit_DN,V_exit_DN] = COE2RV(a,ecc,inc,RAAN,w,nu_exit_DN);
```

```
147
```

```
%determine time of flight from nu0 to nu_enter_DN and nu_exit_DN
81
   [TOF_enter_DN] = TOF_from_nu(a,ecc,nu0,nu_enter_DN,0);
82
   [TOF_exit_DN] = TOF_from_nu(a,ecc,nu0,nu_exit_DN,0);
83
84
   if TOF_enter_DN < 20 && TOF_enter_DN > 0
85
       TOF_enter_DN = TOF_enter_DN + period;
86
       TOF_exit_DN = TOF_exit_DN + period;
87
   end
88
89
  if TOF_exit_DN < TOF_enter_DN</pre>
90
       if TOF_enter_DN > 0
91
           TOF_exit_DN = TOF_exit_DN + period;
92
93
       else
           TOF_enter_DN = TOF_enter_DN + 2*period;
94
           TOF_exit_DN = TOF_exit_DN + period;
95
       end
96
   end
97
98
  flag = 0;
99
   count = 0;
100
101
  % determine is satellite is/is not in correct longitude range when it is
102
        in correct
103 % latitude range. If not, find the next time it will be in the correct
      longitude
104 % range
   while flag == 0;
105
       %Determine lattitude and longitude of spacecraft at nu_enter_AN and
106
           nu_exit_AN
```

80

```
[lat_enter_AN,long_enter_AN,GMST_enter_AN] = IJK_to_LATLONG(
107
           R_enter_AN(1), R_enter_AN(2), R_enter_AN(3), GMST0, t0+T0F_enter_AN)
           ;
       [lat_exit_AN,long_exit_AN,GMST_exit_AN] = IJK_to_LATLONG(R_exit_AN
108
           (1),R_exit_AN(2),R_exit_AN(3),GMST0,t0+TOF_exit_AN);
109
       if long_enter_AN < 0</pre>
110
            if weird_flag == 0
111
                long_enter_AN_temp = 2*pi + long_enter_AN;
112
            else
113
                long_enter_AN_temp = long_enter_AN;
114
            end
115
       else
116
117
            long_enter_AN_temp = long_enter_AN;
       end
118
119
       if long_exit_AN < 0</pre>
120
            if weird_flag == 0
121
                long_exit_AN_temp = 2*pi + long_exit_AN;
122
            else
123
                long_exit_AN_temp = long_exit_AN;
124
125
            end
       else
126
            long_exit_AN_temp = long_exit_AN;
127
       end
128
129
130
       %Determine lattitude and longitude of spacecraft at nu_enter_AN and
131
           nu_exit_AN
       [lat_enter_DN,long_enter_DN,GMST_enter_DN] = IJK_to_LATLONG(
132
           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
133
            (1),R_exit_DN(2),R_exit_DN(3),GMST0,t0+TOF_exit_DN);
        flag;
134
135
        if long_enter_DN < 0</pre>
136
            if weird_flag == 0
137
                 long_enter_DN_temp = 2*pi + long_enter_DN;
138
            else
139
                 long_enter_DN_temp = long_enter_DN;
140
141
            end
        else
142
             long_enter_DN_temp = long_enter_DN;
143
        end
144
145
        if long_exit_DN < 0</pre>
146
            if weird_flag == 0
147
                 long_exit_DN_temp = 2*pi + long_exit_DN;
148
            else
149
                 long_exit_DN_temp = long_exit_DN;
150
             end
151
        else
152
            long_exit_DN_temp = long_exit_DN;
153
        end
154
155
156
157
        if (longlim_temp(1) <= long_enter_AN_temp && long_enter_AN_temp <=</pre>
158
            longlim_temp(2)) || (longlim_temp(1) <= long_exit_AN_temp &&</pre>
            long_exit_AN_temp < longlim_temp(2))</pre>
            flag = 1;
159
            AN = 1;
160
161
```

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

184	<pre>nu_exit = nu_exit_DN;</pre>
185	<pre>t_exit = t0 + TOF_exit_DN;</pre>
186	<pre>elseif (longlim_temp(1) <= long_enter_DN_temp &&</pre>
	<pre>long_enter_DN_temp <= longlim_temp(2)) %Exact entry location</pre>
	known, but exact exit unknown
187	<pre>nu_enter = nu_enter_DN;</pre>
188	<pre>t_enter = t0 + TOF_enter_DN;</pre>
189	t_exit_guess = t0+ TOF_exit_DN;
190	<pre>[nu_exit,t_exit] = exclusion_exit_condition_dual2(a,ecc,inc,</pre>
	<pre>RAAN,w,nu0,longlim,t_exit_guess,t_enter,GMST0);</pre>
191	else %Exact entry unknown, but exact exit location known
192	<pre>nu_exit = nu_exit_DN;</pre>
193	<pre>t_exit = t0 + TOF_exit_DN;</pre>
194	<pre>t_enter_guess = t0 + TOF_enter_DN;</pre>
195	<pre>[nu_enter,t_enter] = exclusion_entry_condition_dual2(a,ecc,</pre>
	<pre>inc,RAAN,w,nu0,longlim,t_exit,t_enter_guess,GMST0);</pre>
196	end
197	elseif flag ~= 1
198	<pre>long_diff_AN_temp = long_exit_AN - long_enter_AN;</pre>
199	<pre>if long_diff_AN_temp < 0</pre>
200	<pre>long_diff_AN_temp = long_diff_AN_temp + 2*pi;</pre>
201	end
202	<pre>long_diff_DN_temp = long_exit_DN_temp - long_enter_AN_temp;</pre>
203	<pre>if long_exit_DN_temp < long_enter_DN_temp</pre>
204	<pre>long_diff_DN_temp = long_diff_DN_temp + 2*pi;</pre>
205	end
206	<pre>if long_diff_AN_temp > zone_long_diff && long_enter_AN < longlim</pre>
	<pre>(1) && long_exit_AN > longlim(2)</pre>
207	flag = 1;
208	<pre>nu_exit_guess = nu_exit_AN;</pre>
209	<pre>t_exit_guess = t0 + TOF_exit_AN;</pre>

211	[nu_enter,t_enter,phi_enter,lam_enter] =
	<pre>exclusion_entry_condition_dual2(a,ecc,inc,RAAN,w,nu0,</pre>
	<pre>longlim,t_exit_guess,t_enter_guess,GMST0);</pre>
212	[nu_exit,t_exit,phi_exit,lam_exit] =
	<pre>exclusion_exit_condition_dual2(a,ecc,inc,RAAN,w,nu0,</pre>
	<pre>longlim,t_exit_guess,t_enter,GMST0);</pre>
213	<pre>%not a valid entry if before t0 or if the longitude does not</pre>
	match
214	%the limits
215	<pre>if t_enter < 0 abs(lam_enter - longlim(1)) > 0.001 </pre>
	<pre>abs(lam_exit - longlim(2)) > 0.001</pre>
216	<pre>flag = 0;</pre>
217	end
218	end
219	<pre>if long_diff_DN_temp > zone_long_diff && long_enter_DN < longlim</pre>
	<pre>(1) && long_exit_DN > longlim(2)</pre>
220	flag = 1;
221	<pre>nu_exit_guess = nu_exit_DN;</pre>
222	<pre>t_exit_guess = t0 + TOF_exit_DN;</pre>
223	<pre>t_enter_guess = t0 + TOF_enter_DN;</pre>
224	<pre>[nu_enter,t_enter,phi_enter,lam_enter] =</pre>
	<pre>exclusion_entry_condition_dual2(a,ecc,inc,RAAN,w,nu0,</pre>
	<pre>longlim,t_exit_guess,t_enter_guess,GMST0);</pre>
225	[nu_exit,t_exit,phi_exit,lam_exit] =
	<pre>exclusion_exit_condition_dual2(a,ecc,inc,RAAN,w,nu0,</pre>
	<pre>longlim,t_exit_guess,t_enter,GMST0);</pre>
226	%not a valid entry if before t0 or if the longitude does not
	match
227	%the limits
228	<pre>if t_enter < 0 abs(lam_enter - longlim(1)) > 0.001 </pre>
	<pre>abs(lam_exit - longlim(2)) > 0.001</pre>
229	flag = 0;

```
end
230
231
             end
232
        end
233
        %Exact and exit unknown but spacecraft passes through the exclusion
234
            zone
235
236
237
        if flag ~= 0
238
            if t_enter < t0</pre>
239
                 flag = 0;
240
             end
241
        end
242
243
        if flag == 0
244
             TOF_enter_AN = TOF_enter_AN + period;
245
            TOF_exit_AN = TOF_exit_AN + period;
246
             TOF_enter_DN = TOF_enter_DN + period;
247
             TOF_exit_DN = TOF_exit_DN + period;
248
249
            count = count + 1;
250
251
            if count == 100
252
                 error
253
             end
254
        end
255
256
257
   end
258
   [R_enter,V_enter] = COE2RV(a,ecc,inc,RAAN,w,nu_enter);
259
```

```
260 [lat_enter,long_enter,GMST_enter] = IJK_to_LATLONG(R_enter(1),R_enter(2)
,R_enter(3),GMST0,t_enter);
261
262 [R_exit,V_exit] = COE2RV(a,ecc,inc,RAAN,w,nu_exit);
263 [lat_exit,long_exit,GMST_enter] = IJK_to_LATLONG(R_exit(1),R_exit(2),
R_exit(3),GMST0,t_exit);
264
265
266 end
```

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

```
1 function [nu_enter_AN, nu_exit_AN, nu_enter_DN, nu_exit_DN] =
      exclusion_nu_intercept(latlim,incl,omega)
2 %exclusion_zone_orbit_intercept determines
3 %
      1) the true anomalies of the orbit when it intersects the minimum
      and
      maximum latitudes of the exclusion zone for botht he ascending node
  %
4
       (AN) and descending node (DN) passes.
5
  %
6
7
  %INPUTS:
8
  %
      latlim = [phi_min phi_max]
9
           phi_min = the minimum latitude bound (rad)
10
  %
           phi_max = the maximum latitude bound (rad)
11
  %
  %
       incl = orbit inclination (rad)
12
      omega = orbit argument of perigee
13
  %
14
15
  %OUTPUTS
16
      nu_enter_AN = limit of true anomaly of spacecraft at entry into
  %
17
  %
      exclusion zone on AN pass
18
```

```
19 %
     nu_exit_AN = limit of true anomaly of spacecraft at exit exclusion
     zone
20 %
     on AN pass
 %
     nu_enter_DN = limit of true anomaly of spacecraft at entry into
21
22 %
     exclusion zone on DN pass
     nu_exit_DN = limit of true anomaly of spacecraft at exit exclusion
23 %
     zone
     on DN pass
24 %
25
26
27 %%
     ______
28 phi_min = latlim(1);
29 phi_max = latlim(2);
30
31 %%
32 if incl \tilde{} = pi/2
***********************
34 if incl < pi/2
     alpha = incl;
35
36 else
     alpha = pi - incl;
37
 end
38
39
     %======AN passes
40
        _____
     % 1) Exclusion zone 1st point beneath orbit plane (phi_min,
41
        lambda_max)
     delta_nu_enter_AN = asin(sin(norm(phi_min))/sin(alpha));
42
43
```

```
156
```
```
if phi_min > 0
44
45
           nu_enter_AN = delta_nu_enter_AN - omega;
      elseif phi_min < 0</pre>
46
           nu_enter_AN = 2*pi - omega - delta_nu_enter_AN;
47
      else
48
          nu_enter_AN = 2*pi - omega;
49
      end
50
51
      % 2) Exclusion zone last point out from under orbit plane (phi_max,
52
          lambda_min)
      delta_nu_exit_AN = asin(sin(norm(phi_max))/sin(alpha));
53
54
      if phi_max > 0
55
           nu_exit_AN = delta_nu_exit_AN - omega;
56
      elseif phi_max < 0</pre>
57
          nu_exit_AN = 2*pi - omega - delta_nu_exit_AN;
58
      else
59
          nu_exit_AN = 2*pi - omega;
60
      end
61
62
      %=====DN passes
63
          _____
      %Exclusion zone 1st point beneath orbit plane (phi_max, lambda_max)
64
      delta_nu_enter_DN = asin(sin(norm(phi_max))/sin(alpha));
65
66
      if phi_max > 0
67
           nu_enter_DN = pi - omega - delta_nu_enter_DN;
68
      elseif phi_max < 0</pre>
69
           nu_enter_DN = pi - omega + delta_nu_enter_DN;
70
      else
71
          nu_enter_DN = pi - omega;
72
73
      end
```

```
75
      %Exclusion zone last point out from under orbit plane (phi_min,
76
         lambda_min)
      delta_nu_exit_DN = asin(sin(norm(phi_min))/sin(alpha));
77
78
      if phi_min > 0
79
          nu_exit_DN = pi - omega - delta_nu_exit_DN;
80
      elseif phi_min < 0</pre>
81
          nu_exit_DN = pi - omega + delta_nu_exit_DN;
82
      else
83
          nu_exit_DN = pi - omega;
84
      end
85
  elseif incl == pi/2
86
  87
      ******
      %======AN PASS
88
         _____
      %Exclusion zone 1st point in lambda_max
89
      if phi_min > 0
90
          nu_enter_AN = norm(phi_min) - omega;
91
      elseif phi_min < 0</pre>
92
          nu_enter_AN = 2*pi - norm(phi_min) - omega;
93
      else
94
          nu_enter_AN = 2*pi - omega;
95
      end
96
97
      if phi_max > 0
98
          nu_exit_AN = norm(phi_max) - omega;
99
      elseif phi_max < 0</pre>
100
          nu_exit_AN = 2*pi - norm(phi_max) - omega;
101
102
      else
```

```
nu_exit_AN = 2*pi - omega;
103
104
       end
105
       %=====DN PASS
106
           _____
       if phi_max > 0
107
           nu_enter_DN = pi - norm(phi_max) - omega;
108
       elseif phi_max < 0</pre>
109
           nu_enter_DN = pi + norm(phi_max) - omega;
110
       else
111
           nu_enter_DN = pi - omega;
112
       end
113
114
       if phi_min > 0
115
           nu_exit_DN = pi - norm(phi_min) - omega;
116
       elseif phi_min < 0</pre>
117
           nu_exit_DN = pi + norm(phi_min) - omega;
118
       else
119
           nu_exit_DN = pi - omega;
120
       end
121
   end
122
123
   if incl > pi || incl < 0</pre>
124
       disp('Error in exclusion_zone_orbit_intercept: inclination not
125
           feasible')
       clear nu_enter_AN
126
  end
127
128
   if nu_enter_AN < 0</pre>
129
       nu_enter_AN = 2*pi + nu_enter_AN;
130
   elseif nu_enter_AN >= 2*pi
131
132
       nu_enter_AN = 2*pi - nu_enter_AN;
```

```
end
133
134
   if nu_exit_AN < 0</pre>
135
        nu_exit_AN = 2*pi + nu_exit_AN;
136
   elseif nu_exit_AN >= 2*pi
137
        nu_exit_AN = 2*pi - nu_exit_AN;
138
   end
139
140
   if nu_enter_DN < 0</pre>
141
        nu_enter_DN = 2*pi + nu_enter_DN;
142
   elseif nu_enter_DN >= 2*pi
143
        nu_enter_DN = 2*pi - nu_enter_DN;
144
   end
145
146
   if nu_exit_DN < 0</pre>
147
        nu_exit_DN = 2*pi + nu_exit_DN;
148
   elseif nu_exit_DN >= 2*pi
149
        nu_exit_DN = 2*pi - nu_exit_DN;
150
   end
151
```

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)
2 %This function computes the the entry states of the spacecraft
3 %into a rectangular exclusion zone (direct orbits only)
4
5 %INPUTS
      a = orbit semimajor axis (km)
 %
6
      ecc = orbit eccentricity
 %
7
      inc = orbit inclination (rad)
 %
8
      nu_exit = true anomaly of spacecraft upon exit from exclusion zone
9 %
```

```
%
                  (rad)
10
       lambda_exit = longitude of spacecraft upon exclusion zone exit (rad)
11
  %
       latlim = [phi_min phi_max]
  %
12
  %
           phi_min = the minimum latitude bound (rad)
13
           phi_max = the maximum latitude bound (rad)
  %
14
       longlim = [lambda_min lambda_max]
  %
15
  %
           lambda_min = the minimum longitude bound (rad)
16
           lambda_max = the maximum longitude bound (rad)
  %
17
18
  %OUTPUTS
19
20 %
       nu_enter_ex = true anomaly of spacecraft upon exclusion zone entry (
      rad)
      phi_enter = latitude of spacecraft upon entry into exclusion zone (
21 %
      rad)
22 %
       lambda_enter = longitude of spacecraft upon entry in exclusion zone
      (rad)
23 %%
24 wgs84data
25 global OmegaEarth
26
27 if inc < pi/2
       alpha = inc;
28
  elseif inc > pi/2
29
       alpha = pi - inc;
30
  else
31
       disp('ERROR:Inclination must be valid')
32
       clear alpha
33
34
  end
  longlim_temp = longlim(1);
35
  if longlim(1) < 0
36
       longlim_temp = longlim(1) + 2*pi;
37
38
  end
```

```
39
40
41
42
43
   [nu_guess] = nuf_from_TOF(nu0,t_enter,a,ecc);
44
45
46
  [R_guess,V_guess] = COE2RV(a,ecc,inc,RAAN,omega,nu_guess);
47
48
  [lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1), R_guess(2), R_guess(3)
49
      ,GMST0,t_enter);
50
  if longlim(1) < 0</pre>
51
       long_guess_temp = long_guess;
52
       if long_guess < 0</pre>
53
           long_guess_temp = 2*pi + long_guess;
54
       end
55
  %
         plot((long_guess)*180/pi,lat_guess*180/pi,'b0')
56
       del_lambda = longlim_temp - long_guess_temp;
57
  else
58
       del_lambda = longlim(1) - long_guess;
59
  end
60
61
  gamma = acos(sin(alpha)*cos(del_lambda));
62
  del_nu = acos(cot(gamma)*cot(alpha));
63
  nu_guess2 = nu_guess + del_nu;
64
  if nu_guess2 > 2*pi
65
       nu_guess2 = nu_guess2 - 2*pi;
66
  end
67
  delt = TOF_from_nu(a,ecc,nu_guess,nu_guess2,0);
68
69 t_guess = t_enter + delt;
```

```
70 [R_guess, V_guess] = COE2RV(a, ecc, inc, RAAN, omega, nu_guess2);
71 [lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1), R_guess(2), R_guess(3)
       ,GMST0,t_guess);
72 if longlim(1) < 0
       long_guess_temp = long_guess;
73
       if long_guess < 0</pre>
74
            long_guess_temp = 2*pi + long_guess;
75
            diff = longlim_temp - long_guess_temp;
76
       else
77
            diff = longlim(1) - long_guess_temp;
78
       end
79
80
   %
         plot((long_guess_temp)*180/pi,lat_guess*180/pi,'b0')
  %
81
82
   else
   %
         plot(long_guess*180/pi,lat_guess*180/pi,'kX')
83
       diff = longlim(1) - long_guess;
84
   end
85
86
87
  count = 0;
88
89
   while abs(diff) > 1e-6
90
       del_lambda = del_lambda + diff;
91
       gamma = acos(sin(alpha)*cos(del_lambda));
92
       del_nu = acos(cot(gamma)*cot(alpha));
93
       nu_guess2 = nu_guess + del_nu;
94
       if nu_guess2 > 2*pi
95
           nu_guess2 = nu_guess2 - 2*pi;
96
       end
97
       delt = TOF_from_nu(a,ecc,nu_guess,nu_guess2,0);
98
       t_guess = t_enter + delt;
99
100
       [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),
101
           R_guess(3),GMST0,t_guess);
        if longlim(1) < 0</pre>
102
            long_guess_temp = long_guess;
103
            if long_guess < 0</pre>
104
                 long_guess_temp = 2*pi + long_guess;
105
            end
106
            diff = longlim_temp - long_guess_temp;
107
              plot((long_guess_temp)*180/pi,lat_guess*180/pi,'b0')
   %
108
109
        else
            diff = longlim(1) - long_guess;
110
              plot(long_guess*180/pi,lat_guess*180/pi,'kX')
111
   %
        end
112
113
   end
114
115
116
   t_enter = t_guess;
117
118
   nu_enter = nu_guess2;
119
120
   R_enter = R_guess;
121
  V_enter = V_guess;
122
   lat_enter = lat_guess;
123
   long_enter = long_guess;
124
125
   % hold on
126
   % plot(long_guess(:)*180/pi,lat_guess(:)*180/pi,'b.')
127
128 % plot(long_enter*180/pi,lat_enter*180/pi,'gD',lambda_exit*180/pi,
       phi_exit*180/pi,'g0')
```



```
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)
2 %This function computes the the entry states of the spacecraft
3 %into a rectangular exclusion zone (direct orbits only)
5 %INPUTS
      a = orbit semimajor axis (km)
6 %
7 %
      ecc = orbit eccentricity
      inc = orbit inclination (rad)
8
  %
  %
      nu_exit = true anomaly of spacecraft upon exit from exclusion zone
9
                 (rad)
10
  %
      lambda_exit = longitude of spacecraft upon exclusion zone exit (rad)
  %
11
      latlim = [phi_min phi_max]
12
  %
           phi_min = the minimum latitude bound (rad)
  %
13
           phi_max = the maximum latitude bound (rad)
  %
14
       longlim = [lambda_min lambda_max]
  %
15
  %
           lambda_min = the minimum longitude bound (rad)
16
           lambda_max = the maximum longitude bound (rad)
17 %
18
19 %OUTPUTS
      nu_enter_ex = true anomaly of spacecraft upon exclusion zone entry (
20 %
      rad)
21 %
      phi_enter = latitude of spacecraft upon entry into exclusion zone (
      rad)
      lambda_enter = longitude of spacecraft upon entry in exclusion zone
22 %
      (rad)
23 %%
24
_{25} if inc < pi/2
      alpha = inc;
26
27 elseif inc > pi/2
```

```
alpha = pi - inc;
28
  else
29
       disp('ERROR:Inclination must be valid')
30
       clear alpha
31
  end
32
33
  longlim_temp = longlim(2);
34
  if longlim(2) < 0
35
       longlim_temp = longlim(2) + 2*pi;
36
37
  end
38
  lambda_max = longlim(2);
39
40
   [nu_guess] = nuf_from_TOF(nu0,t_exit,a,ecc);
41
42
43
  [R_guess,V_guess] = COE2RV(a,ecc,inc,RAAN,omega,nu_guess);
44
45
  [lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1), R_guess(2), R_guess(3)
46
       ,GMST0,t_exit);
47
  if longlim(2) < 0
48
       long_guess_temp = long_guess;
49
       if long_guess < 0</pre>
50
           long_guess_temp = 2*pi + long_guess;
51
       end
52
         plot((long_guess)*180/pi,lat_guess*180/pi,'b0')
  %
53
       del_lambda = long_guess_temp - longlim_temp;
54
55
  else
       del_lambda = long_guess -longlim(2);
56
  %
         plot(long_guess*180/pi,lat_guess*180/pi,'r0')
57
58
  end
```

```
59
  gamma = acos(sin(alpha)*cos(del_lambda));
60
  del_nu = acos(cot(gamma)*cot(alpha));
61
  nu_guess2 = nu_guess - del_nu;
62
  if nu_guess2 < 0</pre>
63
      nu_guess2 = nu_guess2 + 2*pi;
64
  end
65
  delt = TOF_from_nu(a,ecc,nu_guess2,nu_guess,0);
66
  t_guess = t_exit - delt;
67
  [R_guess,V_guess] = COE2RV(a,ecc,inc,RAAN,omega,nu_guess2);
68
  [lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1), R_guess(2), R_guess(3)
69
      ,GMST0,t_guess);
  if longlim(2) < 0</pre>
70
       long_guess_temp = long_guess;
71
       if long_guess < 0</pre>
72
           long_guess_temp = 2*pi + long_guess;
73
       end
74
       diff = long_guess_temp - longlim_temp;
75
  %
         plot((long_guess)*180/pi,lat_guess*180/pi,'b0')
76
  else
77
         plot(long_guess*180/pi,lat_guess*180/pi,'kX')
  %
78
       diff = long_guess -longlim(2);
79
  end
80
81
82
  count = 0;
83
84
  while abs(diff) > 1e-6
85
       del_lambda = del_lambda + diff;
86
       gamma = acos(sin(alpha)*cos(del_lambda));
87
       del_nu = acos(cot(gamma)*cot(alpha));
88
89
       nu_guess2 = nu_guess - del_nu;
```

```
if nu_guess2 < 0</pre>
90
91
            nu_guess2 = nu_guess2 + 2*pi;
        end
92
        delt = TOF_from_nu(a,ecc,nu_guess2,nu_guess,0);
93
        t_guess = t_exit - delt;
94
        [R_guess, V_guess] = COE2RV(a, ecc, inc, RAAN, omega, nu_guess2);
95
        [lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1), R_guess(2),
96
           R_guess(3),GMST0,t_guess);
        if longlim(2) < 0</pre>
97
            long_guess_temp = long_guess;
98
            if long_guess < 0</pre>
99
                 long_guess_temp = 2*pi + long_guess;
100
            end
101
            diff = long_guess_temp -longlim_temp;
102
              plot((long_guess)*180/pi,lat_guess*180/pi,'b0')
   %
103
        else
104
            diff = long_guess -longlim(2);
105
              plot(long_guess*180/pi,lat_guess*180/pi,'kX')
   %
106
        end
107
108
   end
109
110
111
   t_exit = t_guess;
112
113
   nu_exit = nu_guess2;
114
115
116 R_exit = R_guess;
117 V_exit = V_guess;
  lat_exit = lat_guess;
118
  long_exit = long_guess;
119
```

D.1.1.7 Convert Inertial State into Latitude and Longitude

```
1 function [lat,long,GMST] = IJK_to_LATLONG(x,y,z,GMST0,t)
2
3 global OmegaEarth
4
s r = sqrt(x^2 + y^2 + z^2);
6
7 \text{ alpha} = \text{atan2}(y, x);
  GMST = GMST0 + OmegaEarth*t;
9
10
  if GMST >= 2*pi
11
       GMST = GMST - 2*pi;
12
  end
13
14
  long = alpha - GMST;
15
16
  if long <= -pi</pre>
17
       long = 2*pi+long;
18
   elseif long >= pi
19
20
       long = -2*pi+long;
  end
21
22
23 lat = asin(z/r);
```

D.1.2 Single Pass RTM PSO Algorithm

```
1 function [JGmin, Jpbest, gbest, x, k, preburn_state1, initial_target] =
PS0_RTM_analytical_prec(n, limits, prec, iter, swarm, rf1, vf1, ae, be, Rmax,
Rmin, latlim, longlim, tf1)
2
3 %Author: Dan Showalter 18 Oct 2012
4
```

```
5 %Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem
6
7 %generic PSO variable
  %
      n: # of design variables
8
      limits: bounds on design variables (n x 2 vector) with first element
  %
9
           in row n being lower bound for element n and 2nd element in row
10 %
      n being
  %
           upper bound for element n
11
 %
      iter: number of iterations
12
      swarm: swarm size
13
  %
  %
      prec: defines the number of decimal places to keep for each design
14
           variable and the cost function evalution size: (n+1,1)
15
  %
16
17 %Problem specific PSO variables
  %
      n = 4
18
          n1 = TOF1 = TOF of first maneuver
  %
19
20 %
          n2 = theta1 = location on exclusion ellipse where spacecraft
      will
21 %
          arrive upon completion of maneuver 1
          n3 = TOF2 = TOF of 2nd maneuver
  %
22
          n4 = theta2 = location on exclusion ellipse where spacecraft
23 %
      will
24 %
          arrive upon completion of maneuver 2
25
26
 %Specific Problem Variables
27
      rf1: expected position vector when spacecraft enters exclusion zone
  %
28
      vf1: expected velocity vector when spacecraft enters exclusion zone
29
  %
  %
      ae: semimajor axis of exclusion ellipse
30
      be: semiminor axis of exclusion ellipse
  %
31
      Rmax: maximum allowable distance from Earth (constraint on maneuvers
32 %
      )
```

```
170
```

```
%
       Rmin: minimum alowable distance from Earth (constraint on maneuvers)
33
       latlim: vector defining latitude bounds on exclusion zone
34
  %
       longlim: vector defining longitude bounds on exclusion zone
  %
35
  %
       end time of maneuver sequence
36
37
38
  %%
39
40
  [N,M] = size(limits);
41
42
43 llim = limits(:,1);
44 ulim = limits(:,2);
45
  if N~=n
46
       fprintf('Error! limits size does not match number of variables')
47
       stop
48
  end
49
50
51 gbest = zeros(n,1);
52 x = zeros(n, swarm);
53 v = zeros(n,swarm);
54 pbest = zeros(n,swarm);
55 Jpbest = zeros(swarm,1);
56 d = (ulim - llim);
57 JG = zeros(iter,1);
58 J = zeros(swarm,1);
59
  count = 0;
60
  IND = 0;
61
62
63 CoreNum = 6;
64 if (matlabpool('size')) <=0
```

```
matlabpool('open','local',CoreNum);
65
66
   else
       disp('Parallel Computing Enabled')
67
   end
68
69
  %loop until maximum iteration have been met
70
   for k = 1:iter
71
72
       %create particles dictated by swarm size input
73
74
75
       % if this is the first iteration
76
       if k == 1
77
           for h = 1:swarm
78
                x(:,h) = random('unif',llim,ulim,[n,1]);
79
                v(:,h) = random('unif',-d,d,[n,1]);
80
           end
81
82
           %if this is after the first iteration, update velocity and
83
               position
           %of each particle in the swarm
84
       else
85
           parfor h = 1:swarm
86
87
                %set random weighting for each component
88
                c1 = 2.1;
89
                c2 = 2.1;
90
                phi = c1+c2;
91
                ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));
92
                               ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
                %
93
                cc = c1*random('unif',0,1);
94
                cs = c2*random('unif', 0, 1);
95
```

96 97 vdum = v(:,h);98 % 99 if h ~= IND % 100 vdum = v(:,h);% 101 else % 102 % vdum = 0;103 % end 104 %update velocity 105 vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x 106 (:,h))); 107 108 %check to make sure velocity doesn't exceed max velocity for 109 each %variable 110 for w = 1:n111 112 %if the variable velocity is less than the min, set it 113 to the min if vdum(w) < -d(w)114 vdum(w) = -d(w);115 %if the variable velocity is more than the max, set 116 it to the max elseif vdum(w) > d(w); 117 vdum(w) = d(w);118 end 119 end 120 121 v(:,h) = vdum;122 123

%update position 124 xdum = x(:,h) + v(:,h);125 126 for r = 1:n127 128 %if particle has passed lower limit 129 if xdum(r) < llim(r)</pre> 130 xdum(r) = llim(r);131 132 elseif xdum(r) > ulim(r) 133 xdum(r) = ulim(r);134 end 135 136 x(:,h) = xdum;137 138 end 139 140end 141 142 end 143 144 % round variables to get finite precision 145 for aa = 1:n146 x(aa,:) = round(x(aa,:)*10^(prec(aa)))/10^prec(aa); 147 v(aa,:) = round(v(aa,:)*10^(prec(aa)))/10^prec(aa); 148 end 149 150 151 ***** parfor m = 1:swarm 152 % ********************** Cost function evaluation here 153 ******

154 [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] = 155 Single_Burn_Maneuver(rf1,vf1,x(1,m),x(2,m),ae,be); 156 $[Ra1, Rp1] = Ra_Rp_from_RV(r01, v01+DV1vec');$ 157 158 if Ra1 > Rmax 159 J(m) = Inf;160 elseif Rp1 < Rmin</pre> 161 J(m) = Inf;162 else 163 J(m) = manDV1;164 end 165 166 end 167 168 169 ***** %% 170 %% 171 172 %round cost to nearest precision required 173 $J = round(J*10^{prec}(n+1))/10^{prec}(n+1);$ 174 175 if k == 1176 177 Jpbest(1:swarm) = J(1:swarm); 178 pbest(:,1:swarm) = x(:,1:swarm); 179 180 [Jgbest,IND] = min(Jpbest(:)); 181

```
175
```

gbest(:) = x(:,IND); else for h=1:swarm if J(h) < Jpbest(h)</pre> Jpbest(h) = J(h);pbest(:,h) = x(:,h); if Jpbest(h) < Jgbest</pre> 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</pre> count = count+1; end end

```
JG(k) = Jgbest;
214
        manDV = Jgbest;
215
        JGmin = Jgbest;
216
217
        if count == swarm
218
            break
219
        end
220
221
            figure(1)
222
            plot(x(1,:),x(2,:),'x',gbest(1),gbest(2),'r0')
223
            axis([llim(1) ulim(1) llim(2) ulim(2)])
224
225
   end
226
227
   TOF1 = gbest(1);
228
   theta1 = gbest(2);
229
230
231
   [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] = Single_Burn_Maneuver(rf1,
232
       vf1,gbest(1),gbest(2),ae,be);
233
234
   initial_target = [rtijk1;vtijk1;rmiss1];
235
   preburn_state1 = [r01;v01;tf1-TOF1;DV1vec'];
236
237
238
  % figure
239
  % plot(1:iter,JG)
240
241 % title('Cost vs. Iteration #')
242 % xlabel('# iterations')
243 % ylabel('cost')
244 % grid
```

245 % axis square

D.1.3 Single Burn Maneuver

```
1 function [r0,v0,rtijk,vtijk,manDV,DV1vec,rmiss] = Single_Burn_Maneuver(
      rf,vf,TOF,theta,ae,be)
2 %UNTITLED2 Summary of this function goes here
3 %
      Detailed explanation goes here
4 wgs84data;
5
6 global Small MU
7
8 %% determine orbit elements at spacecraft entrance into exclusion zone
9 [a,ecc,inc,RAAN,w,nu] = RV2COE(rf,vf);
10
11 %determine position vector of new arrival location
12 h = cross(rf, vf);
13
14 hunit = h/norm(h);
15
16 vunit = vf/norm(vf);
17
  gunit = cross(vunit, hunit);
18
19
  re = ae*be/sqrt((be*cos(theta))^2 + (ae*sin(theta))^2);
20
21
 rtijk = rf + re*cos(theta)*vunit + re*sin(theta)*gunit;
22
23
24 rmiss = norm(rtijk - rf);
25
26
27 %% determine orbital elements/position vector of departure location
28 [nu0] = nuf_from_TOF(nu,-TOF,a,ecc);
```

```
29
  [r0,v0] = COE2RV(a,ecc,inc,RAAN,w,nu0);
30
31
  %% solve lambert's problem both ways
32
  [V1s, V2s, extremal_distances_s, exitflag_s] = lambert2(r0',rtijk',TOF
33
      /(3600*24),0,MU);
34
  [V11, V21, extremal_distances_1, exitflag_1] = lambert2(r0',rtijk',-TOF
35
      /(3600*24),0,MU);
36
  DVS = norm(V1s - v0');
37
  DVL = norm(V11 - v0');
38
39
  if DVL < DVS
40
41
       manDV = DVL;
42
       DV1vec = V11 - v0';
43
       vtijk = V21';
44
45
  else
46
47
       manDV = DVS;
48
       DV1vec = V1s - v0';
49
       vtijk = V2s';
50
51
52 end
```

D.1.4 Lambert Targeting Algorithm

Code provided by [41].

D.1.5 Position and Velocity Vectors from Classical Orbital Elements

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

```
2
  3 %Author: Dan Showalter 18 Oct 2012
   4
   5 %Purpose: find inertial position and velocity vector gievn classical
  6 %orbital elements
   7
  8 %% Algorithm
   9
10 MU = 398600.5;
11
12 %find magnitude of position vector
13 p = a^{(1-ecc^2)};
14
15 r = p/(1+ecc*cos(nu));
16
17 \operatorname{Rpqw} = r^{*}[\cos(nu); \sin(nu); 0];
18 Vpqw = sqrt(MU/p)*[-sin(nu);(ecc+cos(nu));0];
19
20 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)+
21
                                                                                             cos(RAAN)*cos(w)*cos(inc),-cos(RAAN)*sin(inc);...
                                                                         sin(w)*sin(inc), cos(w)*sin(inc), cos(inc)];
22
23
24
25
26 Rijk = ROT*Rpqw;
27 Vijk = ROT*Vpqw;
28
29
30
31
```

32 **end**

D.1.6 Kepler's Equation

```
function [nuf] = nuf_from_TOF(nu0,TOF,a,e)
2 %This function computes the final true anomaly based on the initial trua
3 %anomaly and the time of flight
4
5 %INPUTS
         = semi-major axis (km)
6 % a
7 % nu0
         = initial true anomaly (rad)
8 % TOF
         = Time of flight (sec)
         = eccentricity (unitless)
9 % e
10
11 %OUTPUTS
12 % nuf = final true anomaly (rad)
13
14 %GLOBALS
15 % MU = Earth's gravitational parameter (km<sup>3</sup>/sec<sup>2</sup>)
16
17 %INTERNALS
         = mean motion (rad/sec)
18 % n
19 % EO
          = initial eccentric anomaly (rad)
          = initial mean anomaly (rad)
20 % MO
21 % Mf
         = final mean anomaly (rad)
22 % Ef = final eccentric anomaly (rad)
         = guess for final eccentric anomaly (rad)
23 % Eg
24
25 global MU
26 %% 1) compute orbital mean motion
n = sqrt(MU/a^3);
28
29
```

```
30 %% 2) convert initial true anomaly to initial mean anomaly
31
32 if nu0 == 0;
33
     MO = O;
34 elseif nu0 == pi
       M0 = pi;
35
  else
36
       E0 = acos((e+cos(nu0))/(1+e*cos(nu0)));
37
       if (nu0 > pi)
38
           E0 = 2*pi - E0;
39
       end
40
     M0 = E0 - e*sin(E0);
41
42 end
43
44 %% 3) compute final mean anomaly
45 Mf = M0 + n*TOF;
46 while Mf > 2*pi
    Mf = Mf - 2*pi;
47
  end
48
49
50 if Mf < 0
     Mf = 2*pi + Mf;
51
52 end
53
54 Eg = Mf;
55 Ef = Eg + (Mf - Eg + e*sin(Eg))/(1 - e*cos(Eg));
56
       while (abs(Ef-Eg) > 1e-12)
57
           Eg = Ef;
58
           Ef = Eg + (Mf - Eg + e*sin(Eg))/(1 - e*cos(Eg));
59
       end
60
61
```

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

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

```
1 function [Ra,Rp] = Ra_Rp_from_RV(rvec,vvec)
2
3 %rvec = position vector km
4 %vvec = velocity vector km/s
5
6 %Ra = radius of apogee
7 %Rp = radius of perigee
8
9 global MU
10
11 %magnitudes of r and v
12 R = norm(rvec);
13 V = norm(vvec);
14
15 %specific mechanical energy
_{16} E = V<sup>2</sup>/2 - MU/R;
17
18 %semimajor axis from specirfic mechanical energy
19 a = -MU/(2*E);
20
21 %specific angular momentum vector from rvec and vvec
22 h = cross(rvec,vvec);
23
24 %magnitude of specific angular momentum vector
25 H = norm(h);
26
```

```
27 %eccectricity
28 e = sqrt(1 + 2*E*H^2/MU^2);
29
30 Ra = a*(1+e);
31 Rp = a*(1-e);
32
33
34 end
```

D.2 Double Pass RTMs

D.2.1 Double Pass RTM Data Script

```
t = 0;
2 \text{ GMST0} = 0;
3 latlim = [-10 10]*pi/180;
4 longlim = [-50 -10]*pi/180;
5
6 wgs84data
7 global MU
8 r0vec = [6800 7300;0 0;0 0];
9 v0vec = [0 0;5.41376581448788 sqrt(MU/7300)/sqrt(2);5.41376581448788
      sqrt(MU/7300)/sqrt(2)];
10 r0 = 6800*[1 \ 0 \ 0];
11 v0 = sqrt(MU/6800)/sqrt(2)*[0 1 1];
12
13 swarm = 30;
14 iter = 1000;
15 aevec = [50 60 70 80 90 100 110 120 130 140 150];
16 bevec = [5 6 7 8 9 10 11 12 13 14 15];
17 Rmaxvec = [6850 7350];
18 Rminvec = [6750 7250];
19 prec = [2;5;2;5;16];
20
```

```
21 for k = 1:2
22
       r0 = r0vec(:,k);
23
       v0 = v0vec(:,k);
24
       Rmax = Rmaxvec(k);
25
       Rmin = Rminvec(k);
26
       [a, ecc, inc, RAAN, w, nu0] = RV2COE(r0, v0);
27
       period = 2*pi*sqrt(a^3/MU);
28
29
       state0=[r0 v0];
30
31
       fprintf(fid, '\n\n\r %s %3i\r\n', 'r0=', norm(r0));
32
33
       for aa = 1:11
34
35
           ae = aevec(aa);
36
           be = bevec(aa);
37
38
           fprintf(fid,'\n\n\r %s %3i\r\n','swarm=',swarm);
39
           fprintf(fid,'%s %3i\r\n','ae=',ae);
40
           fprintf(fid,'%s %3i\r\n','be=',be);
41
           fprintf(fid,'%s %3i\r\n','maxiter=',iter);
42
           fprintf(fid, '%2s %10s %8s %8s %8s %8s %8s %8s %8s %8s \r\n', 'run
43
               #','T1','theta1','T2','theta2','DV1','DV2','J','iterations',
               'Run Time');
44
           itn = zeros(20,1);
45
           rt = zeros(20,1);
46
           tot_time = 0;
47
48
           for h = 1:20
49
50
```

```
clear JG Jpbest gbest manDV
51
52
                tstart = tic;
53
                [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,
54
                   lat_exit,long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,
                   latlim,longlim);
55
                [JG, Jpbest, gbest, x, iter_needed, preburn_state1,
56
                   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);
57
                tend = toc(tstart)
58
59
                DV1 = norm(preburn_state1(8:10)*1000);
60
                DV2 = norm(preburn_state2(8:10)*1000);
61
               manDV = round(JG*1000*10^{5})/10^{5};
62
                itn(h) = iter_needed;
63
               rt(h) = tend;
64
65
               if h == 1
66
                    minDV = manDV;
67
                    mincount = 1;
68
                elseif manDV < minDV</pre>
69
                    minDV = manDV;
70
                    mincount = 1;
71
                elseif manDV == minDV
72
                    mincount = mincount + 1;
73
                end
74
75
```

```
186
```

```
fprintf(fid,'%2i %10.2f %8.5f %10.2f %8.5f %10.5f %10.5f
76
                   %10.5f %4i %10.4f\r\n',h,gbest(1),gbest(2),gbest(3),
                    gbest(4),DV1,DV2,manDV,itn(h),rt(h));
            end
77
78
            gpercent = mincount/h*100;
79
80
81
            tot_time = tot_time + sum(rt);
82
           mintime = \min(rt);
83
           maxtime = max(rt);
84
            meantime = mean(rt);
85
86
           miniter = min(itn);
87
           maxiter = max(itn);
88
            meaniter = mean(itn);
89
90
91
            fprintf(fid,'%s %8.5f\r\n','min time=',mintime);
92
            fprintf(fid, '%s %8.5f\r\n', 'max time=', maxtime);
93
            fprintf(fid, '%s %8.5f\r\n', 'avg time=', meantime);
94
            fprintf(fid, '%s %8.5f\r\n', 'min iter=',miniter);
95
            fprintf(fid,'%s %8.5f\r\n','max iter=',maxiter);
96
            fprintf(fid, '%s %8.5f\r\n', 'avg iter=', meaniter);
97
            fprintf(fid,'%s %i\r\n','global conv=',gpercent);
98
       end
99
       fprintf(fid,'\n\n\r %s','
100
           ');
```

101 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)
2
3
  [N,M] = size(limits);
4
5
6 llim = limits(:,1);
7 ulim = limits(:,2);
8
9 if N^{\sim}=n
       fprintf('Error! limits size does not match number of variables')
10
       stop
11
12
   end
13
  gbest = zeros(n,1);
14
15 x = zeros(n, swarm);
16 v = zeros(n, swarm);
17 pbest = zeros(n,swarm);
  Jpbest = zeros(swarm,1);
18
19 d = (ulim - 1lim);
_{20} JG = zeros(iter,1);
J = zeros(swarm, 1);
22
  count = 0;
23
  IND = 0;
24
25
  CoreNum = 12;
26
  if (matlabpool('size'))<=0</pre>
27
       matlabpool('open','local',CoreNum);
28
  else
29
30
       disp('Parallel Computing Enabled')
```

```
31 end
32
  %loop until maximum iteration have been met
33
  for k = 1:iter
34
35
       %create particles dictated by swarm size input
36
       parfor h = 1:swarm
37
38
           % if this is the first iteration
39
           if k == 1
40
                x(:,h) = random('unif',llim,ulim,[n,1]);
41
                v(:,h) = random('unif',-d,d,[n,1]);
42
43
44
               %if this is after the first iteration, update velocity and
45
                   position
                %of each particle in the swarm
46
           else
47
48
                %set random weighting for each component
49
                c1 = 2.1;
50
                c2 = 2.1;
51
                phi = c1+c2;
52
                ci = 2/abs(2-phi - sqrt(phi<sup>2</sup> - 4*phi));
53
                %
                               ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
54
                cc = c1*random('unif',0,1);
55
                cs = c2*random('unif', 0, 1);
56
57
58
                vdum = v(:,h);
59
                %
60
                               if h ~= IND
61
                %
```

% vdum = v(:,h);62 else 63 % vdum = 0;% 64 % end 65 %update velocity 66 vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x 67 (:,h))); 68 69 %check to make sure velocity doesn't exceed max velocity for 70 each %variable 71 for w = 1:n72 73 %if the variable velocity is less than the min, set it 74 to the min if vdum(w) < -d(w)75 vdum(w) = -d(w);76 %if the variable velocity is more than the max, set 77 it to the max elseif vdum(w) > d(w); 78 vdum(w) = d(w);79 end 80 end 81 82 v(:,h) = vdum;83 84 %update position 85 xdum = x(:,h) + v(:,h);86 87 for r = 1:n88 89

%if particle has passed lower limit 90 if xdum(r) < llim(r)</pre> 91 xdum(r) = llim(r);92 93 elseif xdum(r) > ulim(r) 94 xdum(r) = ulim(r);95 end 96 97 x(:,h) = xdum;98 99 end 100101 end 102 103 end 104 105 % round variables to get finite precision 106 for aa = 1:n107 x(aa,:) = round(x(aa,:)*10^(prec(aa)))/10^prec(aa); 108 $v(aa,:) = round(v(aa,:)*10^(prec(aa)))/10^prec(aa);$ 109 end 110 111 112 *********** parfor m = 1:swarm 113 % *********************** Cost function evaluation here 114 OmegaEarth=0.000072921151467; 115 [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] = 116 Single_Burn_Maneuver(rf1,vf1,x(1,m),x(2,m),ae,be); 117 118 $[Ra1, Rp1] = Ra_Rp_from_RV(r01, v01+DV1vec');$

if Ra1 > Rmax 120 J(m) = Inf;121 elseif Rp1 < Rmin</pre> 122 J(m) = Inf;123 else 124 125 [rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2, 126 lat_exit2,long_exit2] = zone_entry_exit2(rtijk1,vtijk1 ,0+OmegaEarth*tf1,0,latlim,longlim); 127 [r02,v02,rtijk2,vtijk2,manDV2,DV2vec,rmiss2] = 128 Single_Burn_Maneuver(rf2,vf2,x(3,m),x(4,m),ae,be); 129 $[Ra2,Rp2] = Ra_Rp_from_RV(r02,v02+DV2vec');$ 130 131 if Ra2 > Rmax 132 J(m) = Inf;133 elseif Rp2 < Rmin</pre> 134 J(m) = Inf;135 else 136 137 J(m) = manDV1 + manDV2;138 end 139 end 140 141 end 142 143 144 *****
```
%%
145
          146
       %%
147
       %round cost to nearest precision required
148
       J = round(J*10^{prec}(n+1))/10^{prec}(n+1);
149
150
       if k == 1
151
152
           Jpbest(1:swarm) = J(1:swarm);
153
           pbest(:,1:swarm) = x(:,1:swarm);
154
155
           [Jgbest,IND] = min(Jpbest(:));
156
157
           gbest(:) = x(:,IND);
158
159
160
       else
161
           for h=1:swarm
162
               if J(h) < Jpbest(h)</pre>
163
                   Jpbest(h) = J(h);
164
                   pbest(:,h) = x(:,h);
165
                   if Jpbest(h) < Jgbest</pre>
166
167
                       Jgbest = Jpbest(h);
168
                       gbest(:) = x(:,h);
169
                       IND = h;
170
171
                   end
172
               end
173
174
           end
```

```
end
175
176
        count = 0;
177
178
        for y = 1:swarm
179
180
            diff = Jgbest - Jpbest(y);
181
182
            if abs(diff)<1e-10</pre>
183
                 count = count+1;
184
             end
185
186
        end
187
188
        JG(k) = Jgbest;
189
        manDV = Jgbest;
190
        JGmin = Jgbest;
191
192
        if count == swarm
193
            break
194
        end
195
   end
196
   OmegaEarth=0.000072921151467;
197
   TOF1 = gbest(1);
198
   theta1 = gbest(2);
199
   TOF2 = gbest(3);
200
   theta2 = gbest(4);
201
202
203
   [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] = Single_Burn_Maneuver(rf1,
204
       vf1,gbest(1),gbest(2),ae,be);
```

```
[rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2,lat_exit2,
206
       long_exit2] = zone_entry_exit2(rtijk1,vtijk1,0+OmegaEarth*tf1,0,
      latlim,longlim);
207
   [r02,v02,rtijk2,vtijk2,manDV2,DV2vec,rmiss2] = Single_Burn_Maneuver(rf2,
208
      vf2,gbest(3),gbest(4),ae,be);
209
210
   initial_target1 = [rtijk1;vtijk1;rmiss1];
211
   preburn_state1 = [r01;v01;tf1-TOF1;DV1vec'];
212
   initial_target2 = [rtijk2;vtijk2;rmiss2];
213
   preburn_state2 = [r02;v02;tf2-TOF2;DV2vec'];
214
215
216
217 % figure
  % plot(1:iter,JG)
218
219 % title('Cost vs. Iteration #')
  % xlabel('# iterations')
220
221 % ylabel('cost')
222 % grid
223 % axis square
   D.3 Triple Pass RTMs
```

```
D.3.1 Triple Pass RTM Data Script
```

```
1 t0 = 0;
2 GMST0 = 0;
3 latlim = [-10 10]*pi/180;
4 longlim = [-50 -10]*pi/180;
5
6 wgs84data
7 global MU
8 r0vec = [6800 7300;0 0;0 0];
```

```
9 v0vec = [0 0;5.41376581448788 sqrt(MU/7300)/sqrt(2);5.41376581448788
      sqrt(MU/7300)/sqrt(2)];
10
11
12 r0 = 6800*[1 0 0];
13 v0 = sqrt(MU/6800)/sqrt(2)*[0 1 1];
14
15 swarm = 60;
16 iter = 1000;
17 aevec = [50 60 70 80 90 100 110 120 130 140 150];
<sup>18</sup> bevec = [5 6 7 8 9 10 11 12 13 14 15];
19 Rmaxvec = [6850 7350];
20 Rminvec = [6750 7250];
21 prec = [2;5;2;5;2;5;16];
22
_{23} mincase(1) = 4;
  mincase(2) = 1;
24
25
  for k = 2:2
26
       r0 = r0vec(:,k);
27
       v0 = v0vec(:,k);
28
       Rmax = Rmaxvec(k);
29
       Rmin = Rminvec(k);
30
       [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0,v0);
31
       period = 2*pi*sqrt(a^3/MU);
32
33
       state0=[r0 v0];
34
35
       fprintf(fid, '\n\n\r %s %3i\r\n', 'r0=', norm(r0));
36
37
       for aa = 8:8
38
39
```

```
196
```

```
ae = aevec(aa);
40
          be = bevec(aa);
41
42
          fprintf(fid, '\n\n\r %s %3i\r\n', 'swarm=', swarm);
43
          fprintf(fid, '%s %3i\r\n', 'ae=',ae);
44
          fprintf(fid,'%s %3i\r\n','be=',be);
45
          fprintf(fid,'%s %3i\r\n','maxiter=',iter);
46
          47
              s\r\n','run #','T1','theta1','T2','theta2','T3','theta3','
              DV1', 'DV2', 'DV3', 'J', 'iterations', 'Run Time');
48
          itn = zeros(20,1);
49
          rt = zeros(20,1);
50
          tot_time = 0;
51
52
          for h = 41:60
53
54
              clear JG Jpbest gbest manDV
55
56
              tstart = tic;
57
58
              [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,
59
                  lat_exit,long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,
                  latlim,longlim);
60
              [JG, Jpbest, gbest, x, iter_needed, preburn_state1,
61
                  initial_target1, preburn_state2, initial_target2,
                  preburn_state3, initial_target3] = ...
                  PS0_3pass_RTM_analytical_prec(6,[1200 period;0 2*pi;1200
62
                       period;0 2*pi;1200 period;0 2*pi],prec,iter,swarm,
                      rf1,vf1,ae,be,Rmax,Rmin,latlim,longlim,tf1);
```

```
tend = toc(tstart)
64
65
                DV1 = norm(preburn_state1(8:10)*1000);
66
                DV2 = norm(preburn_state2(8:10)*1000);
67
                DV3 = norm(preburn_state3(8:10)*1000);
68
                manDV = round(JG*1000*10^{5})/10^{5};
69
                itn(h) = iter_needed;
70
                rt(h) = tend;
71
72
                if h == 1 || h == 21 || h == 41
73
                    minDV = manDV;
74
                    mincount = 1;
75
                elseif manDV < minDV</pre>
76
                    minDV = manDV;
77
                    mincount = 1;
78
                elseif manDV == minDV
79
                    mincount = mincount + 1;
80
                end
81
82
                fprintf(fid, '%2i %10.2f %8.5f %10.2f %8.5f %10.2f %8.5f
83
                   %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
84
85
           gpercent = mincount/h*100;
86
           tot_time = tot_time + sum(rt);
87
           mintime = min(rt);
88
           maxtime = max(rt);
89
           meantime = mean(rt);
90
           miniter = min(itn);
91
92
           maxiter = max(itn);
```

```
198
```

```
meaniter = mean(itn);
93
94
95
            fprintf(fid,'%s %8.5f\r\n','min time=',mintime);
96
            fprintf(fid, '%s %8.5f\r\n', 'max time=', maxtime);
97
            fprintf(fid,'%s %8.5f\r\n','avg time=',meantime);
98
            fprintf(fid, '%s %8.5f\r\n', 'min iter=',miniter);
99
            fprintf(fid,'%s %8.5f\r\n','max iter=',maxiter);
100
            fprintf(fid,'%s %8.5f\r\n','avg iter=',meaniter);
101
            fprintf(fid,'%s %i\r\n','global conv=',gpercent);
102
       end
103
       fprintf(fid, '\n\n\r %s','
104
```

');

105 **end**

D.3.2 Triple Pass RTM PSO Algorithm

```
1 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)
2
3 %Author: Dan Showalter 18 Oct 2012
4
5 %Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem
6
7 %generic PSO variable
  %
      n: # of design variables
8
      limits: bounds on design variables (n x 2 vector) with first element
  %
9
          in row n being lower bound for element n and 2nd element in row
 %
10
      n being
          upper bound for element n
11 %
```

```
iter: number of iterations
12 %
13
  %
       swarm: swarm size
      prec: defines the number of decimal places to keep for each design
  %
14
  %
           variable and the cost function evalution size: (n+1,1)
15
16
  %Problem specific PSO variables
17
  %
      n = 4
18
          n1 = TOF1 = TOF of first maneuver
  %
19
20 %
          n2 = theta1 = location on exclusion ellipse where spacecraft
      will
  %
           arrive upon completion of maneuver 1
21
          n3 = TOF2 = TOF of 2nd maneuver
22
  %
          n4 = theta2 = location on exclusion ellipse where spacecraft
23 %
      will
24 %
           arrive upon completion of maneuver 2
25
26
  %Specific Problem Variables
27
       rf1: expected position vector when spacecraft enters exclusion zone
  %
28
       vf1: expected velocity vector when spacecraft enters exclusion zone
  %
29
       ae: semimajor axis of exclusion ellipse
  %
30
       be: semiminor axis of exclusion ellipse
31
  %
      Rmax: maximum allowable distance from Earth (constraint on maneuvers
  %
32
      )
      Rmin: minimum alowable distance from Earth (constraint on maneuvers)
  %
33
       latlim: vector defining latitude bounds on exclusion zone
  %
34
       longlim: vector defining longitude bounds on exclusion zone
  %
35
       end time of maneuver sequence
36
  %
37
38
39 %%
40
```

```
41 [N,M] = size(limits);
42
43 llim = limits(:,1);
44 ulim = limits(:,2);
45
  if N~=n
46
        fprintf('Error! limits size does not match number of variables')
47
        stop
48
   end
49
50
51 gbest = zeros(n,1);
52 \mathbf{x} = \mathbf{z}\mathbf{e}\mathbf{r}\mathbf{o}\mathbf{s}(\mathbf{n}, \mathbf{s}\mathbf{w}\mathbf{a}\mathbf{r}\mathbf{m});
53 v = zeros(n,swarm);
54 pbest = zeros(n,swarm);
55 Jpbest = zeros(swarm,1);
56 d = (ulim - llim);
57 JG = zeros(iter,1);
58 J = zeros(swarm,1);
59
  count = 0;
60
  IND = 0;
61
62
  CoreNum = 12;
63
  if (matlabpool('size'))<=0</pre>
64
        matlabpool('open','local',CoreNum);
65
  else
66
        disp('Parallel Computing Enabled')
67
   end
68
69
70 %loop until maximum iteration have been met
71 for k = 1:iter
72
```

```
%create particles dictated by swarm size input
73
74
75
       % if this is the first iteration
76
       if k == 1
77
            for h = 1:swarm
78
                x(:,h) = random('unif',llim,ulim,[n,1]);
79
                v(:,h) = random('unif',-d,d,[n,1]);
80
            end
81
82
            %if this is after the first iteration, update velocity and
83
                position
            %of each particle in the swarm
84
       else
85
            parfor h = 1:swarm
86
87
                %set random weighting for each component
88
                c1 = 2.1;
89
                c2 = 2.1;
90
                phi = c1+c2;
91
                ci = 2/abs(2-phi - sqrt(phi<sup>2</sup> - 4*phi));
92
                %
                                ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
93
                cc = c1*random('unif', 0, 1);
94
                cs = c2*random('unif', 0, 1);
95
96
97
                vdum = v(:,h);
98
99
                %
                %
                                if h ~= IND
100
                                     vdum = v(:,h);
                %
101
                                else
                %
102
103
                %
                                     vdum = 0;
```

```
202
```

% end 104 %update velocity 105 vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x 106 (:,h))); 107 108 %check to make sure velocity doesn't exceed max velocity for 109 each %variable 110 for w = 1:n111 112 %if the variable velocity is less than the min, set it 113 to the min if vdum(w) < -d(w)114 vdum(w) = -d(w);115 %if the variable velocity is more than the max, set 116 it to the max elseif vdum(w) > d(w); 117 vdum(w) = d(w);118 end 119 end 120 121 v(:,h) = vdum;122 123 %update position 124 xdum = x(:,h) + v(:,h);125 126 for r = 1:n127 128 %if particle has passed lower limit 129 if xdum(r) < llim(r)</pre> 130 xdum(r) = llim(r);131

elseif xdum(r) > ulim(r) 133 xdum(r) = ulim(r); 134 end 135 136 x(:,h) = xdum;137 138 end 139 140141 end 142 end 143 144 % round variables to get finite precision 145 for aa = 1:n146 x(aa,:) = round(x(aa,:)*10^(prec(aa)))/10^prec(aa); 147 v(aa,:) = round(v(aa,:)*10^(prec(aa)))/10^prec(aa); 148 end 149 150 151 ****** parfor m = 1:swarm 152 % ********************** Cost function evaluation here 153 *********************** OmegaEarth=0.000072921151467; 154 [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,~] = Single_Burn_Maneuver(155 rf1,vf1,x(1,m),x(2,m),ae,be); 156 [Ra1,Rp1] = Ra_Rp_from_RV(r01,v01+DV1vec'); 157 158 if Ra1 > Rmax 159 J(m) = Inf;160

```
elseif Rp1 < Rmin</pre>
161
                J(m) = Inf;
162
            else
163
164
                [rf2,vf2,tf2,~,~,~,~,~,~] = zone_entry_exit2(rtijk1,vtijk1
165
                    ,0+OmegaEarth*tf1,0,latlim,longlim);
166
                [r02,v02,rtijk2,vtijk2,manDV2,DV2vec,~] =
167
                    Single_Burn_Maneuver(rf2,vf2,x(3,m),x(4,m),ae,be);
168
                [Ra2, Rp2] = Ra_Rp_from_RV(r02, v02+DV2vec');
169
170
                if Ra2 > Rmax
171
                     J(m) = Inf;
172
                elseif Rp2 < Rmin</pre>
173
                     J(m) = Inf;
174
                else
175
176
                     [rf3,vf3,tf3,~,~,~,~,~,~] = zone_entry_exit2(rtijk2,
177
                         vtijk2,0+OmegaEarth*(tf1+tf2),0,latlim,longlim);
178
                     [r03,v03,rtijk3,vtijk3,manDV3,DV3vec,~] =
179
                         Single_Burn_Maneuver(rf3,vf3,x(5,m),x(6,m),ae,be);
180
                     [Ra3,Rp3] = Ra_Rp_from_RV(r03,v03+DV3vec');
181
182
                       figure
  %
183
                       % plot(long1(:)*180/pi,lat1(:)*180/pi,'r.')
184
   %
  %
185
   %
186
                       longmin_plot(1:2) = longlim(1);
  %
187
188
  %
                       longmax_plot(1:2) = longlim(2);
```

```
%
                       latmax_plot(1:2) = latlim(1);
189
190
   %
                       latmin_plot(1:2) = latlim(2);
   %
191
   %
                       plot(longmin_plot(:)*180/pi,latlim(:)*180/pi,'c-')
192
                       xlabel('longitude (deg)')
   %
193
                       ylabel('latitude (deg)')
   %
194
                       axis([-180 180 -90 90])
   %
195
   %
                       grid
196
   %
                       hold on
197
                       plot(longmax_plot(:)*180/pi,latlim(:)*180/pi,'c-')
198
   %
   %
                       if longlim(2) > longlim(1)
199
                           plot(longlim(:)*180/pi,latmax_plot(:)*180/pi,'c-')
200
   %
                           plot(longlim(:)*180/pi,latmin_plot(:)*180/pi,'c-')
   %
201
                       else
202
   %
                           longlim_plot = [longlim(1) pi -pi longlim(2)];
   %
203
                           plot(longlim_plot(1:2)*180/pi,latmax_plot(:)*180/
   %
204
       pi,'c-')
  %
                           plot(longlim_plot(3:4)*180/pi,latmax_plot(:)*180/
205
       pi,'c-')
                           plot(longlim_plot(1:2)*180/pi,latmin_plot(:)*180/
   %
206
       pi,'c-')
                           plot(longlim_plot(3:4)*180/pi,latmin_plot(:)*180/
207
  %
       pi,'c-')
                       end
   %
208
                       plot(long_enter2*180/pi,lat_enter2*180/pi,'r0',
209
   %
       long_exit2*180/pi,lat_exit2*180/pi,'b0')
                       plot(long_enter3*180/pi,lat_enter3*180/pi,'r0',
  %
210
       long_exit3*180/pi,lat_exit3*180/pi,'b0')
   %
211
                       close all
   %
212
213
214
                    if Ra3 > Rmax
```

215	J(m) = Inf;
216	<pre>elseif Rp3 < Rmin</pre>
217	J(m) = Inf;
218	else
219	
220	J(m) = manDV1 + manDV2 + manDV3;
221	end
222	end
223	end
224	
225	end
226	
227	%% *****************************Constraint Equations

228	%%

229	%%
230	
231	%round cost to nearest precision required
232	<pre>J = round(J*10^prec(n+1))/10^prec(n+1);</pre>
233	
234	if k == 1
235	
236	<pre>Jpbest(1:swarm) = J(1:swarm);</pre>
237	<pre>pbest(:,1:swarm) = x(:,1:swarm);</pre>
238	
239	[Jgbest,IND] = min(Jpbest(:));
240	
241	<pre>gbest(:) = x(:,IND);</pre>

else

```
for h=1:swarm
245
                  if J(h) < Jpbest(h)</pre>
246
                       Jpbest(h) = J(h);
247
                       pbest(:,h) = x(:,h);
248
                       if Jpbest(h) < Jgbest</pre>
249
250
                             Jgbest = Jpbest(h);
251
                             gbest(:) = x(:,h);
252
                             IND = h;
253
254
                       end
255
                  end
256
             end
257
        end
258
259
        count = 0;
260
261
        for y = 1:swarm
262
263
             diff = Jgbest - Jpbest(y);
264
265
             if abs(diff)<1e-10</pre>
266
                  count = count+1;
267
             end
268
269
        end
270
271
        JG(k) = Jgbest;
272
        manDV = Jgbest;
273
        JGmin = Jgbest;
274
275
```

```
if count == swarm
276
277
           break
       end
278
   end
279
   OmegaEarth=0.000072921151467;
280
   TOF1 = gbest(1);
281
   TOF2 = gbest(3);
282
   TOF3 = gbest(5);
283
284
   [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] = Single_Burn_Maneuver(rf1,
285
      vf1,gbest(1),gbest(2),ae,be);
286
   [rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2,lat_exit2,
287
       long_exit2] = zone_entry_exit2(rtijk1,vtijk1,0+OmegaEarth*tf1,0,
      latlim,longlim);
288
   [r02,v02,rtijk2,vtijk2,manDV2,DV2vec,rmiss2] = Single_Burn_Maneuver(rf2,
289
      vf2,gbest(3),gbest(4),ae,be);
290
   [rf3,vf3,tf3,lat_enter3,long_enter3,R_exit3,V_exit3,t_exit3,lat_exit3,
291
       long_exit3] = zone_entry_exit2(rtijk2,vtijk2,0+OmegaEarth*(tf1+tf2)
       ,0,latlim,longlim);
292
   [r03,v03,rtijk3,vtijk3,manDV3,DV3vec,rmiss3] = Single_Burn_Maneuver(rf3,
293
      vf3,gbest(5),gbest(6),ae,be);
294
295
   initial_target1 = [rtijk1;vtijk1;rmiss1];
296
   preburn_state1 = [r01;v01;tf1-TOF1;DV1vec'];
297
   initial_target2 = [rtijk2;vtijk2;rmiss2];
298
   preburn_state2 = [r02;v02;tf2-TOF2;DV2vec'];
299
300
   initial_target3 = [rtijk3;vtijk3;rmiss3];
```

```
209
```

```
301 preburn_state3 = [r03;v03;tf3-TOF3;DV3vec'];
302
303
304 % figure
305 % plot(1:iter,JG)
306 % title('Cost vs. Iteration #')
307 % xlabel('# iterations')
308 % ylabel('cost')
309 % grid
310 % axis square
```

D.3.3 Triple Pass Multiple Revolution RTM PSO Algorithm

```
1 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)
2
 %Author: Dan Showalter 18 Oct 2012
3
  %Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem
5
6
7 %generic PSO variable
8 %
      n: # of design variables
      limits: bounds on design variables (n x 2 vector) with first element
9
  %
           in row n being lower bound for element n and 2nd element in row
10 %
      n being
           upper bound for element n
  %
11
      iter: number of iterations
12
  %
      swarm: swarm size
13
  %
      prec: defines the number of decimal places to keep for each design
  %
14
  %
          variable and the cost function evalution size: (n+1,1)
15
16
17 %Problem specific PSO variables
```

```
210
```

```
18 %
      n = 4
           n1 = TOF1 = TOF of first maneuver
19
  %
           n2 = theta1 = location on exclusion ellipse where spacecraft
  %
20
      will
           arrive upon completion of maneuver 1
  %
21
           n3 = TOF2 = TOF of 2nd maneuver
  %
22
23 %
           n4 = theta2 = location on exclusion ellipse where spacecraft
      will
  %
           arrive upon completion of maneuver 2
24
25
26
  %Specific Problem Variables
27
       rf1: expected position vector when spacecraft enters exclusion zone
  %
28
       vf1: expected velocity vector when spacecraft enters exclusion zone
29
  %
  %
       ae: semimajor axis of exclusion ellipse
30
       be: semiminor axis of exclusion ellipse
  %
31
       Rmax: maximum allowable distance from Earth (constraint on maneuvers
  %
32
      )
  %
       Rmin: minimum alowable distance from Earth (constraint on maneuvers)
33
       latlim: vector defining latitude bounds on exclusion zone
  %
34
       longlim: vector defining longitude bounds on exclusion zone
  %
35
       end time of maneuver sequence
36
  %
37
38
  %%
39
40
  [N,M] = size(limits);
41
42
43 llim = limits(:,1);
44 ulim = limits(:,2);
45
46 if N^{\sim}=n
```

47 **fprintf('**Error! limits size does not match number of variables'**)**

48

stop

```
end
49
50
51 lbest = zeros(n,swarm);
52 x = zeros(n, swarm);
53 v = zeros(n, swarm);
54 pbest = zeros(n,swarm);
55 Jpbest = zeros(swarm,1);
56 d = (ulim - llim);
57 JG = zeros(iter,1);
J = zeros(swarm, 1);
59 Jsubs = zeros(3,swarm);
60 Jsubp = zeros(3, swarm);
  Jsubout = zeros(1,3);
61
62
  count = 0;
63
  IND = 0;
64
65
  CoreNum = 12;
66
  if (matlabpool('size'))<=0</pre>
67
       matlabpool('open','local',CoreNum);
68
  else
69
       disp('Parallel Computing Enabled')
70
  end
71
72
  %loop until maximum iteration have been met
73
  for k = 1:iter
74
75
       %create particles dictated by swarm size input
76
77
78
```

```
% if this is the first iteration
79
        if k == 1
80
            for h = 1:swarm
81
                x(:,h) = random('unif',llim,ulim,[n,1]);
82
                v(:,h) = random('unif',-d,d,[n,1]);
83
            end
84
85
            %if this is after the first iteration, update velocity and
86
                position
            %of each particle in the swarm
87
        else
88
            parfor h = 1:swarm
89
90
91
                %set random weighting for each component
92
                c1 = 2.09;
93
                c2 = 2.09;
94
                phi = c1+c2;
95
                ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));
96
                                ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
                %
97
                cc = c1*random('unif',0,1);
98
                 cs = c2*random('unif', 0, 1);
99
100
101
                vdum = v(:,h);
102
                %
103
                                if h ~= IND
                %
104
                                     vdum = v(:,h);
105
                %
                %
                                else
106
                                     vdum = 0;
                %
107
                                end
                %
108
                %update velocity
109
```

110	vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(lbest(:,h)
	- x(:,h)));
111	
112	
113	%check to make sure velocity doesn't exceed max velocity for
	each
114	%variable
115	for $w = 1:n$
116	
117	%if the variable velocity is less than the min, set it
	to the min
118	if vdum(w) < -d(w)
119	vdum(w) = -d(w);
120	%if the variable velocity is more than the max, set
	it to the max
121	<pre>elseif vdum(w) > d(w);</pre>
122	vdum(w) = d(w);
123	end
124	end
125	
126	v(:,h) = vdum;
127	
128	%update position
129	xdum = x(:,h) + v(:,h);
130	
131	for $r = 1:n$
132	
133	%if particle has passed lower limit
134	<pre>if xdum(r) < llim(r)</pre>
135	<pre>xdum(r) = llim(r);</pre>
136	
137	<pre>elseif xdum(r) > ulim(r)</pre>

```
xdum(r) = ulim(r);
138
                   end
139
140
                   x(:,h) = xdum;
141
142
               end
143
144
           end
145
146
147
       end
148
      % round variables to get finite precision
149
       for aa = 1:n
150
           x(aa,:) = round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
151
           v(aa,:) = round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
152
       end
153
154
       155
          ***************
       parfor m = 1:swarm
156
           % ********************** Cost function evaluation here
157
              ******
           OmegaEarth=0.000072921151467;
158
           [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,~] = Single_Burn_Maneuver(
159
              rf1,vf1,x(1,m),x(2,m),ae,be);
160
           [Ra1, Rp1] = Ra_Rp_from_RV(r01, v01+DV1vec');
161
162
           if Ra1 > Rmax
163
               J(m) = Inf;
164
               Jsubs(:,m) = [Inf;Inf;Inf];
165
           elseif Rp1 < Rmin</pre>
166
```

J(m) = Inf;167 Jsubs(:,m) = [Inf;Inf;Inf]; 168 else 169 170 [rf2,vf2,tf2,lat_enter2,long_enter2,~,~,t2_exit,lat_exit2, 171 long_exit2] = zone_entry_exit2(rtijk1,vtijk1,0+ OmegaEarth*tf1,0,latlim,longlim); 172 [r02,v02,rtijk2,vtijk2,manDV2,DV2vec,~] = 173 Single_Burn_Maneuver(rf2,vf2,x(3,m),x(4,m),ae,be); 174 $[Ra2, Rp2] = Ra_Rp_from_RV(r02, v02+DV2vec');$ 175 176 if Ra2 > Rmax 177 J(m) = Inf;178 Jsubs(:,m) = [manDV1; Inf; Inf]; 179 elseif Rp2 < Rmin</pre> 180 J(m) = Inf;181 Jsubs(:,m) = [manDV1; Inf; Inf]; 182 else 183 184 [rf3,vf3,tf3,lat_enter3,long_enter3,~,~,~,lat_exit3, 185 long_exit3] = zone_entry_exit2(rtijk2,vtijk2,0+ OmegaEarth*(tf1+tf2),0,latlim,longlim); 186 if x(5,m) > (tf2+tf3-(t2_exit)) 187 J(m) = Inf;188 Jsubs(:,m) = [manDV1;manDV2;Inf]; 189 else 190 191

[r03,v03,rtijk3,vtijk3,manDV3,DV3vec,~] = 192 Single_Burn_Maneuver_nrev(rf3,vf3,x(5,m),x(6,m), ae,be); 193 [Ra3,Rp3] = Ra_Rp_from_RV(r03,v03+DV3vec'); 194 195 if Ra3 > Rmax 196 J(m) = Inf;197 Jsubs(:,m) = [manDV1;manDV2;Inf]; 198 elseif Rp3 < Rmin</pre> 199 J(m) = Inf;200 Jsubs(:,m) = [manDV1;manDV2;Inf]; 201 else 202 203 J(m) = manDV1 + manDV2 + manDV3;204 Jsubs(:,m) = [manDV1;manDV2;manDV3]; 205 end 206 % J(m) = manDV1 + manDV2 +207 manDV3; end 208 end 209 end 210 211 end 212 213 214 ***** %% 215 %% 216 217

```
%round cost to nearest precision required
218
        J = round(J*10^{prec}(n+1))/10^{prec}(n+1);
219
220
221
        if k == 1
222
            Jpbest = J;
223
            pbest = x;
224
            Jsubp = Jsubs;
225
            parfor aa = 1:swarm
226
                 Jtemp = J;
227
                 nup = aa+nhood/2;
228
                 ndown = aa-nhood/2;
229
230
                 indl = (ndown:1:nup);
231
                 inddown = find(indl < 1);</pre>
232
                 indl(inddown) = swarm+indl(inddown);
233
                 indup = find(indl > swarm);
234
                 indl(indup) = indl(indup)-swarm;
235
236
                 [Jlbest(aa), indmin] = min(Jtemp(indl));
237
                 lbest(:,aa) = x(:,indl(indmin));
238
239
            end
240
        else
241
            parfor aa = 1:swarm
242
                 Jtemp = J;
243
                 nup = aa+nhood/2;
244
                 ndown = aa-nhood/2;
245
246
                 indl = (ndown:1:nup);
247
                 inddown = find(indl < 1);</pre>
248
249
                 indl(inddown) = swarm+indl(inddown);
```

```
218
```

```
indup = find(indl > swarm);
250
                 indl(indup) = indl(indup)-swarm;
251
252
                 [Jmintemp, indmin] = min(Jtemp(indl));
253
                 if Jmintemp < Jlbest(aa)</pre>
254
                      Jlbest(aa) = Jmintemp;
255
                      lbest(:,aa) = x(:,indl(indmin));
256
                 end
257
258
                 if Jtemp(aa) < Jpbest(aa)</pre>
259
                      Jpbest(aa) = Jtemp(aa);
260
                      pbest(:,aa) = x(:,aa);
261
                      Jsubp(:,aa) = Jsubs(:,aa);
262
                 end
263
             end
264
265
        end
266
267
        [Jgbest, indgbest] = min(Jpbest);
268
        gbest = pbest(:,indgbest);
269
        Jsubout = Jsubp(:,indgbest);
270
271
        diff = zeros(swarm,1);
272
        parfor y = 1:swarm
273
             diff(y) = Jgbest - Jpbest(y);
274
        end
275
276
        indcount = find(abs(diff)<10^(-prec(n+1)));</pre>
277
278
        JG(k) = Jgbest;
279
        manDV = Jgbest;
280
        JGmin = Jgbest;
281
```

```
282
        if k > 1
283
            if JG(k) == JG(k-1)
284
                 count = count + 1;
285
            else
286
                   MinCost = Jgbest*1000
   %
287
                   k
   %
288
                 count = 0;
289
            end
290
                        if length(indcount) > previndcount
291
            %
                           length(indcount)
            %
292
                        end
293
            %
294
                        if length(indcount) > 100
295
            %
            %
                           keyboard
296
            %
                        end
297
298
        end
299
300
        if length(indcount) > 0.75*swarm
301
            ex_flag = 0;
302
            break
303
        end
304
305
               figure(1)
   %
306
              plot(x(1,:),x(2,:),'x',lbest(1,:),lbest(2,:),'k0',pbest(1,:),
307
   %
       pbest(2,:),'m.',gbest(1),gbest(2),'r0')
               axis([llim(1) ulim(1) llim(2) ulim(2)])
   %
308
   %
309
              figure(2)
   %
310
              plot(x(3,:),x(4,:),'x',lbest(3,:),lbest(4,:),'k0',pbest(3,:),
   %
311
       pbest(4,:),'m.',gbest(3),gbest(4),'r0')
```

```
%
              axis([llim(3) ulim(3) llim(4) ulim(4)])
312
313
   %
              figure(3)
   %
314
              plot(x(5,:),x(6,:),'x',lbest(5,:),lbest(6,:),'k0',pbest(5,:),
   %
315
       pbest(6,:),'m.',gbest(5),gbest(6),'r0')
              axis([llim(5) ulim(5) llim(6) ulim(6)])
   %
316
317
318
        if count > 1000
319
            ex_flag = 1;
320
            break
321
322
        end
   end
323
324
   if k == iter
325
        ex_flag = 2;
326
   end
327
328
   % figure
329
   % plot(1:iter,JG)
330
   % title('Cost vs. Iteration #')
331
  % xlabel('# iterations')
332
  % ylabel('cost')
333
334 % grid
335 % axis square
```

D.3.4 Single Burn Maneuver with Multiple Revolutions

```
1 function [r0,v0,rtijk,vtijk,manDV,DV1vec,rmiss] =
    Single_Burn_Maneuver_nrev(rf,vf,TOF,theta,ae,be)
2 %UNTITLED2 Summary of this function goes here
3 % Detailed explanation goes here
4 wgs84data;
```

```
5
6 global MU
7
8 %% determine orbit elements at spacecraft entrance into exclusion zone
9 [a,ecc,inc,RAAN,w,nu] = RV2COE(rf,vf);
10
11 %determine position vector of new arrival location
h = cross(rf, vf);
13
14 hunit = h/norm(h);
15
  vunit = vf/norm(vf);
16
17
  gunit = cross(vunit, hunit);
18
19
  re = ae*be/sqrt((be*cos(theta))^2 + (ae*sin(theta))^2);
20
21
  rtijk = rf + re*cos(theta)*vunit + re*sin(theta)*gunit;
22
23
  rmiss = norm(rtijk - rf);
24
25
26
27 %% determine orbital elements/position vector of departure location
  [nu0] = nuf_from_TOF(nu,-TOF,a,ecc);
28
29
  [r0,v0] = COE2RV(a,ecc,inc,RAAN,w,nu0);
30
31
32 P0 = 2*pi*sqrt(a^3/MU);
33
34 rat = TOF/P0;
35 m = floor(rat);
36
```

```
37 %% solve lambert's problem both ways
38 [V1s, V2s, extremal_distances_s, exitflag_s] = lambert2(r0',rtijk',TOF
      /(3600*24),m,MU);
39
40 [V11, V21, extremal_distances_1, exitflag_1] = lambert2(r0',rtijk',-TOF
      /(3600*24),m,MU);
41
42 if isnan(V1s(1)) == 1
       [V1s, V2s, extremal_distances_s, exitflag_s] = lambert2(r0',rtijk',
43
          TOF/(3600*24),0,MU);
44 elseif isnan(V1l(1)) == 1
       [V11, V21, extremal_distances_1, exitflag_1] = lambert2(r0',rtijk',-
45
          TOF/(3600*24),0,MU);
46
  end
47
48 DVS = norm(V1s - v0');
49 DVL = norm(V11 - v0');
50
51 if DVL < DVS
52
      manDV = DVL;
53
      DV1vec = V11 - v0';
54
      vtijk = V21';
55
56
57 else
58
      manDV = DVS;
59
      DV1vec = V1s - v0';
60
      vtijk = V2s';
61
62
63 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

```
1 t0 = 0;
2 \text{ GMST0} = 0;
3 latlim = [-10 10]*pi/180;
4 longlim = [-50 -10]*pi/180;
5
6 wgs84data
7 global MU MU2
8 r0vec = [7300;0;0];
  v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];
9
10
  [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
11
  period = 2*pi*sqrt(a^3/MU);
12
13
  aevec = [150 \ 140 \ 130 \ 120 \ 110 \ 100 \ 90 \ 80 \ 70 \ 60 \ 50];
14
15 bevec = [15 14 13 12 11 10 9 8 7 6 5];
  Rmaxvec = norm(r0vec)+50;
16
  Rminvec = norm(r0vec) - 50;
17
18
19 DU = norm(r@vec);
20 TU = period/(2*pi);
MU2 = MU*TU^2/DU^3;
22
23 \text{ m0} = 1000;
24 r0 = r0vec;
25 v0 = v0vec;
```

```
26 Rmax = Rmaxvec;
27
  Rmin = Rminvec;
28
  %Energy of most elliptical orbit
29
  ab = (Rmax + Rmin)/2; %semi-major axis of orbit
30
  Eb = -MU/(2*ab); %energy of orbit
31
32 Vmax = sqrt(2*(MU/Rmin + Eb));
  Vmin = sqrt(2*(MU/Rmax + Eb));
33
34
35 state0=[r0 v0];
_{36} Tmax = 2e-3;
37 dir = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\Single
     Pass\Data\';
38
 fid = fopen([dir 'PSOSinglePassData_Final_5312014.txt'],'a');
39
40
41 swarm = 40;
42 iter = 1000;
43 prec = [3;6;3;6];
44
_{45} for bb = 2:11
46
      if bb == 1
47
                   fprintf(fid,'%s %i\r\n','r0 (km) =',r0vec(1));
48
                   fprintf(fid,'%s %i\r\n','swarm =',swarm);
49
                   fprintf(fid,'%s\t %s\t %s\t %s\t %s\t %s\t %s\t \r\n','
50
                      TOF','Phi','Vf','fpa','DV','iter','time');
                   fprintf(fid,'%s\r\n','
51
                      ······');
          endval = 20;
52
      else endval = 20;
53
54
      end
```

```
55
56
       ae = aevec(bb);
       be = bevec(bb);
57
58
59
       for aa = 1:endval
60
61
           tstart = tic;
62
63
           [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
64
               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
65
               ;-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;
66
           tend = toc(tstart);
67
68
           fprintf(fid,'%i \t\t %10.5f\t %4.3f\t %7.6f\t %4.3f\t %7.6f\t %i
69
               \t %4.1f\r\n',ae,tf1,gbest(1),gbest(2),gbest(3),Cost1,k,tend
               );
70
       end
71
72
73 end
```

E.1.1.2 Single Pass LTRTM PSO Algorithm

```
5 %Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem
6
7 %generic PSO variable
  %
      n: # of design variables
8
      limits: bounds on design variables (n x 2 vector) with first element
  %
9
      in row n being lower bound for element n and 2nd element in row n
  %
10
      being
      upper bound for element n
  %
11
12 %
      iter: number of iterations
13 %
      swarm: swarm size
14
15 %Problem specific PSO variables
  %
      n = 4
16
          n1 = TOF1 = TOF of first maneuver
17
  %
18 %
          n2 = theta1 = location on exclusion ellipse where spacecraft
      will
           arrive upon completion of maneuver 1
  %
19
  %
          n3 = TOF2 = TOF of 2nd maneuver
20
          n4 = theta2 = location on exclusion ellipse where spacecraft
21 %
      will
           arrive upon completion of maneuver 2
22 %
23
24
  %Specific Problem Variables
25
      rf1: expected position vector when spacecraft enters exclusion zone
  %
26
      vf1: expected velocity vector when spacecraft enters exclusion zone
  %
27
      ae: semimajor axis of exclusion ellipse
  %
28
      be: semiminor axis of exclusion ellipse
29
  %
30 %
      Rmax: maximum allowable distance from Earth (constraint on maneuvers
      )
      Rmin: minimum alowable distance from Earth (constraint on maneuvers)
  %
31
32 %
      latlim: vector defining latitude bounds on exclusion zone
```

```
227
```

```
33 %
        longlim: vector defining longitude bounds on exclusion zone
        end time of maneuver sequence
34
  %
35
36
37 %%
38
  [N,M] = size(limits);
39
40
41 llim = limits(:,1);
42 ulim = limits(:,2);
43
44 if N^{\sim}=n
        fprintf('Error! limits size does not match number of variables')
45
46
        stop
   end
47
48
49 gbest = zeros(n,1);
50 \mathbf{x} = \mathbf{z}\mathbf{e}\mathbf{r}\mathbf{o}\mathbf{s}(\mathbf{n}, \mathbf{s}\mathbf{w}\mathbf{a}\mathbf{r}\mathbf{m});
51 v = zeros(n, swarm);
52 pbest = zeros(n,swarm);
53 Jpbest = zeros(swarm,1);
54 d = (ulim - llim);
55 JG = zeros(iter,1);
56 J = zeros(iter,swarm);
57
58 \text{ count} = 0;
  IND = 0;
59
60
61 CoreNum = 12;
62 if (matlabpool('size')) <=0</pre>
        matlabpool('open','local',CoreNum);
63
64 else
```
```
disp('Parallel Computing Enabled')
65
66
  end
67
  %loop until maximum iteration have been met
68
  for k = 1:iter
69
70
      %create particles dictated by swarm size input
71
       parfor h = 1:swarm
72
73
           % if this is the first iteration
74
           if k == 1
75
               x(:,h) = random('unif',llim,ulim,[n,1]);
76
               v(:,h) = random('unif',-d,d,[n,1]);
77
78
               %if this is after the first iteration, update velocity and
79
                   position
               %of each particle in the swarm
80
           else
81
               %set random weighting for each component
82
               ci = 2/abs(2-2*2.09 - sqrt(4.18^2 - 4*4.18));
83
                              ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
               %
84
               cc = 2.09*random('unif',0,1);
85
               cs = 2.09*random('unif',0,1);
86
87
88
               vdum = v(:,h);
89
               %update velocity
90
               vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x
91
                   (:,h)));
92
93
```

%check to make sure velocity doesn't exceed max velocity for 94 each %variable 95 for w = 1:n96 97 %if the variable velocity is less than the min, set it 98 to the min if vdum(w) < -d(w)99 vdum(w) = -d(w);100%if the variable velocity is more than the max, set 101 it to the max elseif vdum(w) > d(w); 102 vdum(w) = d(w);103 104 end end 105 106 v(:,h) = vdum;107 108 %update position 109 xdum = x(:,h) + v(:,h);110 111 for r = 1:n112 113 %if particle has passed lower limit 114 if xdum(r) < llim(r)</pre> 115 xdum(r) = llim(r);116 117 elseif xdum(r) > ulim(r) 118 xdum(r) = ulim(r);119 end 120 121 x(:,h) = xdum;122

123 124 end 125 end 126 127 end 128 129 ************* parfor m = 1:swarm 130 % *********************** Cost function evaluation here 131 ****** $MU2 = MU*TU^2/DU^3;$ 132 133 phi = x(1,m);134 $Vt_mag = x(2,m);$ 135 $fpa_t = x(3,m);$ 136 137 [DV, ~, ~, ~, ~, ~, ~, ~, ~, ~, rt_ijk, vt_ijk, ~, ~] = Single_LT_Maneuver(138 rfvec,vfvec,tf,phi,ae,be,Vt_mag,fpa_t,DU,TU,MU2); 139 % $maxT = maxT*DU/TU^2;$ 140 141 [a,ecc,inc,RAAN,w,nu] = RV2COE(rt_ijk,vt_ijk); 142 143 $Ra = a^{*}(1+ecc);$ 144 $Rp = a^{*}(1 - ecc);$ 145 146 if maxT > Tmax/m0 147 % % J(m) = Inf;148 % else 149 if Ra > Rmax || Rp < Rmin</pre> 150 J(m) = Inf;151

152	else
153	
154	J(m) = DV;
155	end
156	
157	end
158	
159	%% ******************************Constraint Equations

160	%%

```
%%
161
162
        %round cost to nearest precision required
163
        J = round(J*10^{prec}(n+1))/10^{prec}(n+1);
164
165
        if k == 1
166
            count = 0;
167
            Jpbest(1:swarm) = J(1:swarm);
168
            pbest(:,1:swarm) = x(:,1:swarm);
169
170
            [Jgbest,IND] = min(Jpbest(:));
171
            gbest(:) = x(:,IND);
172
173
        else
174
175
            Jtemp = J;
176
            parfor h=1:swarm
177
                 if Jtemp(h) < Jpbest(h)</pre>
178
                      Jpbest(h) = J(h);
179
                      pbest(:,h) = x(:,h);
180
```

```
end
181
             end
182
183
             [Jgbest, indgbest] = min(Jpbest);
184
             gbest = pbest(:,indgbest);
185
186
        end
187
188
189
190
        diff = zeros(swarm,1);
191
        parfor y = 1:swarm
192
193
             diff(y) = Jgbest - Jpbest(y);
194
        end
195
196
        indcount = find(abs(diff)<10^(-prec(n+1)));</pre>
197
198
199
200
201
        JG(k) = Jgbest;
202
        JGmin = Jgbest;
203
204
205
        if length(indcount) == swarm
206
             break
207
        end
208
209
        if k > 1
210
             if JG(k) == JG(k-1)
211
                  count = count + 1;
212
```

```
213 else

214 count = 0;

215 end

216

217 end

218

219

220 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)
2 %Single_LT_Maneuver computes a feasible low thrust maneuver to intercept
       rf
3 %at a specified time
4 wgs84data;
5
6 %INPUTS
7 % rf = inertial position vector at expected arrival location (DU)
% vf = inertial velocity vector at expected arrival location (DU/TU)
9 % TOF = time of flight (TU)
10 % phi = angle of exclusion ellipse (rad)
11 % ae = exclusion ellipse semi-major axis (DU)
12 % be = exclusion ellipse semi-minor axis (DU)
13 % Vt_mag = velocity magnitude at new arrival location (DU/TU)
14 % fpa_t = flight path angle at new arrival location (rad)
15
16 %OUTPUTS
17 %LT_DV = total delta V required for shape-based maneuver (DU/TU)
18 %maxT = maximum thrust acceleration allowed (DY/TU^2)
```

```
234
```

```
19 %r = vector of radius values (DU) in perifocal frame
20 %T_a = thrust acceleration profile (DU/TU^2)
21 %thetaf_int = vector of theta values (rad)
22 %theta_dot = vector of time rate of change of thetaf_int (rad/TU)
23 %theta_ddot = vector of time rate of change of theta_dot (rad/TU^2)
24 %rdot = vector of rate time rate of change of r (DU/TU)
25 %Tvec = vector of time values (TU)
26 %TOF_calc = calculated time of flight (TU) - should match TOF
27 %rt_ijk = inertial position vector ,vt_ijk
28
29 %
_{30} MU = 398600.5;
31 %% determine inertial position vectors of maneuver initiaion and
      completion
32 [a,ecc,inc,RAAN,w,nu] = RV2COE(rf,vf);
33
  period = 2*pi*sqrt(a^3/MU);
34
35
36 %determine position vector of new arrival location
h = cross(rf, vf);
38
 hunit = h/norm(h);
39
40
  vunit = vf/norm(vf);
41
42
  gunit = cross(vunit, hunit);
43
44
45 re = ae*be/sqrt((be*cos(phi))^2 + (ae*sin(phi))^2);
46
47 %inertial position vector of new arrival position
48 rt_ijk = rf + re*cos(phi)*vunit + re*sin(phi)*gunit;
49
```

```
50 %inertial velocity vector at arrival
51 %maneuver is coplanar so expected angular momentum is in same direction
      as
52 %actual angular momentum at arrival
53
54 %unit vector used to help determine actual velocity vector
  funit = cross(hunit,rt_ijk)/norm(rt_ijk);
55
56
57 vt_ijk = Vt_mag*sin(fpa_t)*rt_ijk/norm(rt_ijk) + Vt_mag*cos(fpa_t)*funit
      ;
58
59
  rmiss = norm(rt_ijk - rf);
60
61
62 % determine orbital elements/position vector of departure location
  [nu0] = nuf_from_TOF(nu,-TOF,a,ecc);
63
64
  [r0_ijk,v0_ijk] = COE2RV(a,ecc,inc,RAAN,w,nu0);
65
66
67
 % convert inertial coordinates to perifocal frame
68
  [r0_pqw,v0_pqw] = IJK_to_PQW(r0_ijk,v0_ijk,inc,RAAN,w);
69
70 [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);
71
72
73 %determine total transfer angle
74 cos_psi = (re^2 - norm(rf_pqw)^2 - norm(rt_pqw)^2)/(-2*norm(rf_pqw)*norm
      (rt_pqw));
75 psi = acos(cos_psi);
76
77 %expected flight path angle
78 [fpa_1] = fpa_calc(ecc,nu);
```

```
79
  if phi > pi/2-fpa_1 && phi < 3*pi/2-fpa_1</pre>
80
       psi = -psi;
81
82
  end
83
84 %total transfer angle
85 revs = TOF/period;
86
87 nrevs = floor(revs);
88
89 if nu > nu0
     ang1 = nu-nu0;
90
91 else
       ang1 = 2*pi + nu-nu0;
92
   end
93
94
   ang = ang1+2*pi*nrevs;
95
96
   thetaf = ang + psi;
97
98
  %flight path angle of satellite at maneuver initiation
99
   [gamma0] = fpa_calc(ecc,nu0);
100
101
  %% scale vectors
102
103 r_0_pqw = r_0_pqw/DU;
  v0_pqw = v0_pqw/DU*TU;
104
105 rt_pqw = rt_pqw/DU;
106 vt_pqw = vt_pqw/DU*TU;
107 rf_pqw = rf_pqw/DU;
  vf_pqw = vf_pqw/DU*TU;
108
  TOF = TOF/TU;
109
110
```

237

[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

```
1 function [fpa] = fpa_calc(e,nu)
2 %Generates flight path angle as a function of orbit eccentricity and
      flight
3 %path angle
4
5 %INPUTS
6 % e = orbit eccentricity (unitless)
7 % nu = orbit true anomaly (rad)
8
9 %OUTPUT
10 % fpa = flight path angle (rad)
11
12 %sin of flight path angle
  sin_fpa = (e^{sin}(nu))/sqrt(1+2*e^{cos}(nu)+e^{2});
13
14
15 %cos of flight path angle
  \cos_{fpa} = (1 + e^{\cos(nu)})/sqrt(1 + 2^{e^{\cos(nu)}} + e^{2});
16
17
  fpa = atan2(sin_fpa,cos_fpa);
18
19
20
21 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)
```

```
2 %UNTITLED2 Summary of this function goes here
      Detailed explanation goes here
3 %
4
5 %INPUTS
_{6} %r1vec = position vector (3x1) of initial orbit at theta0 (DU)
7 %v1vec = velocity vector (3x1) of initial orbit at theta0 (DU/TU)
% % rfvec = position vector (3x1) of final orbit at thetaf (DU)
9 %vfvec = velocity vector (3x1) of initial orbit at theta0 (DU/TU)
10 %gamma1 = flight path angle of initial orbit at theta1 (rad)
11 %gamma2 = flight path angle of final orbit at thetaf (rad)
12 %
     _____
13
14 hlvec = cross(rlvec,vlvec); %specific angular momentum of body 1
15 h1 = norm(h1vec); %magnitude of specific angular momentum
16 r1 = norm(r1vec); %magnitude of position vector
17
18 hfvec = cross(rfvec,vfvec); %specific angular momentum of body2 at
     thetaf
19 hf = norm(hfvec); %magnitude of specific angular momentum
20 rf = norm(rfvec); %magnitude of position vector
21 vf = norm(vfvec);
22
a = 1/r1; %parameter a
24 b = -tan(gamma1)/r1; %parameter b
25
 thetadot1 = h1/(r1^2); %rate of change of theta1
26
thetadotf = hf/(rf^2);
28
29 c = 1/(2*r1)*(MU/(r1^3*thetadot1^2)-1); %parameter c
30
```

```
239
```

```
31 flag = 0;
32 guess = 0;
33 n = 0;
34 step = .1;
  total = 20;
35
  while flag == 0
36
       options = optimset('Display', 'off');
37
       [d,FVAL,ex_flag] = fzero(@(x) TF_PARAM_d_RTM_f0(x,rf,TOF,thetaf,
38
           gammaf,a,b,c,thetadotf,MU,n,step),guess,options);
       if d == guess && FVAL == 0
39
           if n < total && n >= 0
40
                guess = guess + step;
41
               n = n + 1;
42
           elseif n == total
43
                guess = -step;
44
               n = -1;
45
           else
46
47
                guess = guess - step;
                n = n - 1;
48
                if n == -step*total;
49
                    flag = 1;
50
                end
51
           end
52
       else
53
           flag = 1;
54
       end
55
56
  end
57
58
  if ex_flag ~=1
59
  %
         disp('Fzero did not converge to a solution')
60
       LT_DV = Inf;
61
```

```
maxT = Inf;
62
63
       r = 0;
        gamma = 0;
64
        T_a = 0;
65
        thetaf_int = 0;
66
        theta_dot = 0;
67
        theta_ddot = 0;
68
       rdot = 0;
69
        Tvec = 0;
70
        TOF_calc = 0;
71
72 %
          gammaf
  %
          vf
73
74
  else
75
76
77
        mat1 = [30*thetaf<sup>2</sup> -10*thetaf<sup>3</sup> thetaf<sup>4</sup>;...
78
            -48*thetaf 18*thetaf<sup>2</sup> -2*thetaf<sup>3</sup>;...
79
            20 -8*thetaf thetaf^2];
80
81
       mat2 = [1/rf-(a+b*thetaf+c*thetaf^2+d*thetaf^3);...
82
            -tan(gammaf)/rf-(b+2*c*thetaf+3*d*thetaf^2);...
83
            MU/(rf^4*thetadotf^2) - (1/rf+2*c+6*d*thetaf)];
84
85
        soln_vec = 1/(2*thetaf^6)*mat1*mat2;
86
87
        e = soln_vec(1); %parameter d
88
        f = soln_vec(2); %parameter e
89
        g = soln_vec(3); %parameter f
90
91
        thetaf_int = linspace(0,thetaf,100); %set up
92
93
```

```
theta = thetaf_int; %theta values
94
95
       r = 1./(a + b*thetaf_int + c*thetaf_int.^2 + d*thetaf_int.^3 + e*
96
           thetaf_int.^4 + f*thetaf_int.^5 + g*thetaf_int.^6); %r values
          based on parametric representation as a function of theta
97
       tan_gamma = -r.*(b + 2*c.*thetaf_int + 3*d.*thetaf_int.^2 + 4*e.*
98
           thetaf_int.^3 + 5*f.*thetaf_int.^4 + 6*g*thetaf_int.^5); %
           tangent of flight path angle (thrust assumed along fpa)
99
       gamma = atan(tan_gamma); %actual flight path angle
100
101
       denom = (1./r + 2*c + 6*d.*theta + 12*e.*theta.^2 + 20*f.*theta.^3 +
102
            30*g.*theta.^4); %denominator of terms used to compute angular
          velocity (theta_dot) acceleration (theta_ddot) and thrust
           acceleration (T_a)
103
       term1 = 4.*tan_gamma./denom; %term used for angular acceleration (
104
           theta_ddot)
       term2 = (6*d + 24*e.*theta + 60*f.*theta.^2 + 120*g.*theta.^3 -
105
           tan_gamma./r)./denom.^2; %term used for angular acceleration (
           theta_ddot)
106
       theta_ddot = -MU./(2.*r.^4).*(term1 + term2); %angular acceleration
107
       theta_dot = sqrt(MU./(r.^4).*(1./denom)); %angular velocity
108
       T_a = -MU./(2.*(r.^3).*cos(gamma)).*term2; %thrust acceleration
109
110
       rdot = -r.^2.*(b + 2*c.*theta + 3*d.*theta.^2 + 4*e.*theta.^3 + 5*f
111
           .*theta.^4 + 6*g.*theta.^5).*theta_dot;
112
       maxT = max(abs(T_a));
113
114
```

```
242
```

```
time_func = sqrt((r.^4/MU.*denom)); %function values used for
115
           quadrature integration of time of flight
116
       dT = zeros(length(theta),1);
117
       Tvec = zeros(length(theta),1);
118
119
       for aa = 2:length(theta)
120
            fa = time_func(aa-1);
121
            fb = time_func(aa);
122
123
            dT(aa) = (theta(aa) - theta(aa-1))*(fa + fb)/2;
124
            Tvec(aa) = Tvec(aa-1) + dT(aa);
125
       end
126
127
       TOF_calc = sum(dT);
128
129
       for bb = 2:length(theta)
130
            % Delta V
131
            fa_DV = abs(T_a(bb-1))/theta_dot(bb-1);
132
            fb_DV = abs(T_a(bb))/theta_dot(bb);
133
            DV_vec(bb) = (theta(bb) - theta(bb-1))*(fa_DV + fb_DV)/2;
134
135
       end
136
       LT_DV = sum(DV_vec);
137
138
139
140 end
```

E.1.1.6 Root Finding Equation

```
3
4 mat1 = [30*thetaf^2 -10*thetaf^3 thetaf^4;...
           -48*thetaf 18*thetaf^2 -2*thetaf^3;...
5
           20 -8*thetaf thetaf^2];
6
7
  mat2 = [1/rf-(a+b*thetaf+c*thetaf^2+d*thetaf^3);...
8
          -tan(gammaf)/rf-(b+2*c*thetaf+3*d*thetaf^2);...
0
           MU2/(rf^4*thetadotf^2) - (1/rf+2*c+6*d*thetaf)];
10
11
  soln_vec = 1/(2*thetaf^6)*mat1*mat2;
12
13
14 e = soln_vec(1); %parameter d
15 f = soln_vec(2); %parameter e
16 g = soln_vec(3); %parameter f
17
  thetaf_int = linspace(0,thetaf,100); %set up
18
19
  theta = thetaf_int; %theta values
20
21
22 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
23
24 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)
25
_{26} ind = find(denom < 0);
27
28 time_func = sqrt(((r.^4)/MU2).*denom); %function values used for
      quadrature integration of time of flight
```

```
244
```

```
29
  for aa = 2:length(theta)
30
       fa = time_func(aa-1);
31
       fb = time_func(aa);
32
33
       dT(aa) = (theta(aa) - theta(aa-1))*(fa + fb)/2;
34
  end
35
36
  TOF_calc = sum(dT);
37
38
  func = TOF_calc - TOF;
39
40
       if norm(x - n*step) < 1e-6 && isreal(TOF_calc) == 0</pre>
41
            func = TOF - sqrt(real(TOF_calc)^2 + imag(TOF_calc)^2);
42
       end
43
```

```
44 end
```

E.1.2 Direct Collocation Algorithms

E.1.2.1 Single Pass LTRTM Driver

```
_{1} for zz = 2:2
2
      if zz == 1
3
           load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
4
              Single Pass\Data\data6800_LT_1RTMsort.mat')
          PSO_data = data6800_LT_1RTMsort;
5
          rmag = 6800;
6
      elseif zz == 2
7
           load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
8
              Single Pass\Data\data7300_LT_1RTMsort.mat')
          PSO_data = data7300_LT_1RTMsort;
9
          rmag = 7300;
10
      end
11
```

```
for cc = 1:22
13
           fid2 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust
14
                RTM\Single Pass\Data\PS02GP0PSSinglePassData.txt','a');
           clear guess setup limits output
15
16
           clc
17
           tstart = tic;
18
19
           fid = fopen('PSO_to_GPOPS.txt','a');
20
           t0 = 0;
21
           GMST0 = 0;
22
           latlim = [-10 10]*pi/180;
23
           longlim = [-50 -10]*pi/180;
24
25
           wgs84data
26
           global MU
27
           r0vec = [rmag;0;0];
28
           v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];
29
30
           [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
31
           period = 2*pi*sqrt(a^3/MU);
32
33
           swarm = 30;
34
           iter = 1000;
35
           Rmaxvec = norm(r0vec)+50;
36
           Rminvec = norm(r0vec)-50;
37
           prec = [2;5;16];
38
39
           r0 = r0vec;
40
           v0 = v0vec;
41
42
           Rmax = Rmaxvec;
```

12

```
Rmin = Rminvec;
43
44
           ae = PSO_data(cc,1);
45
           be = ae/10;
46
47
           TOF = PSO_data(cc,2);
48
           phi = PSO_data(cc,3);
49
           Vt_mag = PSO_data(cc,4);
50
           fpa_t = PSO_data(cc,5);
51
52
           tstart = tic;
53
54
           [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
55
               long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim)
               ;
56
           DU = norm(rf1);
57
           TU = period/(2*pi);
58
59
           ae1 = ae/DU;
60
           be1 = be/DU;
61
           r01 = norm(r0)/DU;
62
           MU2 = MU*TU^2/DU^3;
63
64
           tOmin = 0; % minimum initial time
65
           tOmax = 0; % maximum initial time
66
           tfmin = period/TU; % minimum final time
67
           tfmax = period/TU;
68
           n0 = sqrt(MU2/(norm(r0)/DU)^3);
69
70
           [LT_DV,maxT,r,gamma,T_a,thetaf_int,theta_dot,theta_ddot,rdot,
71
               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);
72
            delt = (tf1 - TOF)/TU;
73
            time_mod = Tvec + delt;
74
75
            [rf_pqw,vf_pqw] = IJK_to_PQW(rf1,vf1,inc,RAAN,w);
76
            rf_pqw = rf_pqw/DU;
77
           vf_pqw = vf_pqw/DU*TU;
78
79
80
            vunit = vf_pqw/norm(vf_pqw);
81
            hfp = cross(rf_pqw,vf_pqw);
82
            hunit = hfp/norm(hfp);
83
84
            gunit = cross(vunit, hunit);
85
86
            ang = (0:0.001:2*pi);
87
            re = (ae1*be1)./sqrt((be1*cos(ang)).^2 + (ae1*sin(ang)).^2);
88
89
90
            theta_rf = atan2(rf_pqw(2),rf_pqw(1));
91
            if theta_rf < 0</pre>
92
                theta_rf = 2*pi + theta_rf;
93
            end
94
            [rtest] = IJK_to_PQW(r0,v0,inc,RAAN,w);
95
            theta0 = atan2(rtest(2),rtest(1));
96
97
            theta_mod = thetaf_int + atan2(r0_pqw(2),r0_pqw(1));
98
99
            coast_length = 1;
100
101
```

102	<pre>time_guess = zeros(coast_length+length(time_mod),1);</pre>
103	<pre>time_guess(1:coast_length) = 0;</pre>
104	<pre>time_guess(coast_length+1:end) = time_mod;</pre>
105	<pre>theta_guess = zeros(coast_length+length(theta_mod),1);</pre>
106	<pre>theta_guess(coast_length+1:end) = theta_mod;</pre>
107	<pre>theta_guess(1:coast_length) = theta0;</pre>
108	<pre>r_guess = zeros(coast_length+length(theta_mod),1);</pre>
109	<pre>r_guess(1:coast_length) = norm(r0)/DU;</pre>
110	<pre>r_guess(coast_length+1:end) = r;</pre>
111	<pre>vr_guess = zeros(coast_length+length(theta_mod),1);</pre>
112	<pre>vr_guess(1:coast_length) = 0;</pre>
113	<pre>vr_guess(coast_length+1:end) = rdot;</pre>
114	<pre>vtheta_guess = zeros(coast_length+length(theta_mod),1);</pre>
115	<pre>vtheta_guess(1:coast_length) = sqrt(MU2/(norm(r0)/DU));</pre>
116	<pre>vtheta_guess(coast_length+1:end) = r.*theta_dot;</pre>
117	<pre>T_guess = zeros(coast_length+length(time_mod),1);</pre>
118	<pre>T_guess(1:coast_length) = 0;</pre>
119	T_guess(coast_length+1:end) = T_a;
120	<pre>B_guess = zeros(coast_length+length(time_mod),1);</pre>
121	<pre>B_guess(1:coast_length) = 0;</pre>
122	<pre>B_guess(coast_length+1:end) = gamma;</pre>
123	
124	<pre>ind = find(T_guess ~= 0);</pre>
125	
126	%inertial position vector of new arrival position
127	<pre>for aa = 1:length(ang)</pre>
128	r_ell(:,aa) = rf_pqw + re(aa)*cos(ang(aa))*vunit + re(aa)*
	<pre>sin(ang(aa))*gunit;</pre>
129	end
130	
131	<pre>for dd = 1:length(r_guess)</pre>

132	rg_pqw = DU*[r_guess(dd)*cos(theta_guess(dd));r_guess(dd)*
	<pre>sin(theta_guess(dd));0];</pre>
133	<pre>[rgi(dd,:)] = PQW_to_IJK(rg_pqw,[],inc,RAAN,w);</pre>
134	end
135	
136	<pre>for ee = 1:length(ang)</pre>
137	<pre>rell_pqw = [r_ell(1,ee)*DU;r_ell(2,ee)*DU;0];</pre>
138	<pre>rnom_pqw = norm(r0)*[cos(ang(ee));sin(ang(ee));0];</pre>
139	<pre>[rell_ijk(:,ee)] = PQW_to_IJK(rell_pqw,[],inc,RAAN,w);</pre>
140	<pre>[rnom_ijk(ee,:)] = PQW_to_IJK(rnom_pqw,[],inc,RAAN,w);</pre>
141	end
142	%% GPOPS RUN
143	% variables from PSo phase
144	r1 = 1;
145	<pre>rf = norm(rt_pqw);</pre>
146	rmax = r1 + be/DU;
147	<pre>rmin = r1 - be/DU;</pre>
148	<pre>thetaf_min = theta_rf - atan(ae/norm(r0));</pre>
149	<pre>thetaf_max = theta_rf + atan(ae/norm(r0));</pre>
150	
151	%colocation points and fraction
152	colnum = 4;
153	colp = 20;
154	
155	% Control and time boundaries
156	if phi > pi
157	umin = 0; % minimum control angle
158	umax = 2*pi; % maximum control angle
159	else
160	umin = -pi; % minimum control angle
161	umax = pi; % maximum control angle
162	end

163	Tmax = 2*0.0001160;
164	Tmin = Tmax/1000;
165	
166	% GPOPS Setup
167	% Phase 1 Information
168	<pre>iphase = 1;</pre>
169	<pre>limits(iphase).intervals = 1;</pre>
170	<pre>limits(iphase).nodesperint = 100;</pre>
171	<pre>bounds.phase(iphase).initialtime.lower = t0min;</pre>
172	<pre>bounds.phase(iphase).initialtime.upper = t0max;</pre>
173	<pre>bounds.phase(iphase).finaltime.lower = tf1/TU;</pre>
174	<pre>bounds.phase(iphase).finaltime.upper = tf1/TU;</pre>
175	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
176	<pre>bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU</pre>
	<pre>0 sqrt(MU2/r1)];</pre>
177	bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU
	<pre>0 sqrt(MU2/r1)];</pre>
178	<pre>bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.2</pre>
	0];
179	<pre>bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.2</pre>
	1.1];
180	<pre>bounds.phase(iphase).state.lower = [r1-0.1 theta_rf-n0*tf1/TU</pre>
	-0.2 0];
181	<pre>bounds.phase(iphase).state.upper = [r1+0.1 thetaf_max 0.2 1.1];</pre>
182	<pre>bounds.phase(iphase).control.lower = [Tmin umin];</pre>
183	<pre>bounds.phase(iphase).control.upper = [Tmax umax];</pre>
184	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
185	<pre>bounds.parameter.lower = 0;</pre>
186	<pre>bounds.parameter.upper = 2*pi;</pre>
187	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
188	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
189	<pre>bounds.phase(iphase).integral.lower = 0;</pre>

190	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
191	<pre>bounds.eventgroup(iphase).lower = [0 0 Rmin/DU Rmin/DU]; % None</pre>
192	<pre>bounds.eventgroup(iphase).upper = [0 0 Rmax/DU Rmax/DU]; % None</pre>
193	% GUESS SOLUTION
194	<pre>guess.phase(iphase).time = time_guess;</pre>
195	<pre>guess.phase(iphase).state(:,1) = r_guess;</pre>
196	<pre>guess.phase(iphase).state(:,2) = theta_guess;</pre>
197	<pre>guess.phase(iphase).state(:,3) = vr_guess;</pre>
198	<pre>guess.phase(iphase).state(:,4) = vtheta_guess;</pre>
199	% Control guess :
200	<pre>guess.phase(iphase).control(:,1) = T_guess;</pre>
201	<pre>guess.phase(iphase).control(:,2) = B_guess;</pre>
202	guess.parameter = phi;
203	<pre>guess.phase(iphase).integral = LT_DV;</pre>
204	
205	%auxiliary data
206	auxdata.MU = MU2;
207	auxdata.ae = ae1;
208	auxdata.be = be1;
209	<pre>auxdata.rf_pqw = rf_pqw;</pre>
210	<pre>auxdata.vunit = vunit;</pre>
211	<pre>auxdata.gunit = gunit;</pre>
212	
213	% NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
	required
214	<pre>setup.name = ['TIME_FIXED_INTERCEPT'];</pre>
215	
216	<pre>setup.functions.continuous = @LT_RTM_Continuous;</pre>
217	<pre>setup.functions.endpoint = @LT_RTM_Endpoint;</pre>
218	<pre>setup.nlp.solver = 'ipopt';</pre>
219	<pre>setup.mesh.maxiteration = 10;</pre>
220	<pre>setup.mesh.tolerance = 1e-12;</pre>

221	<pre>setup.mesh.colpointsmin = 40;</pre>
222	<pre>setup.mesh.colpointsmax = 200;</pre>
223	<pre>setup.mesh.phase(iphase).colpoints = colnum*ones(1,colp);</pre>
224	<pre>setup.mesh.phase(iphase).fraction = (1/colp)*ones(1,colp);</pre>
225	<pre>setup.bounds = bounds;</pre>
226	<pre>setup.guess = guess;</pre>
227	setup.auxdata = auxdata;
228	<pre>setup.mesh.method = 'RPMintegration';</pre>
229	<pre>setup.derivatives.supplier = 'sparseFD';</pre>
230	<pre>setup.derivativelevel ='second';</pre>
231	<pre>setup.dependencies = 'sparseNaN';</pre>
232	<pre>setup.scales = 'none';</pre>
233	
234	<pre>output = gpops2(setup);</pre>
235	<pre>solution = output.result.solution;</pre>
236	
237	<pre>r_GPOPS = solution.phase.state(:,1);</pre>
238	<pre>theta_GPOPS = solution.phase.state(:,2);</pre>
239	<pre>Vr_GPOPS = solution.phase.state(:,3);</pre>
240	<pre>Vt_GPOPS = solution.phase.state(:,4);</pre>
241	<pre>lambda_r = solution.phase.costate(:,1);</pre>
242	<pre>lambda_theta = solution.phase.costate(:,2);</pre>
243	<pre>lambda_Vr = solution.phase.costate(:,3);</pre>
244	<pre>lambda_Vt = solution.phase.costate(:,4);</pre>
245	<pre>tvec = solution.phase.time;</pre>
246	
247	<pre>thetadot_GPOPS = Vt_GPOPS./r_GPOPS;</pre>
248	<pre>T_GPOPS = solution.phase.control(:,1);</pre>
249	<pre>Beta_GPOPS = solution.phase.control(:,2);</pre>
250	<pre>phi_GPOPS = solution.parameter;</pre>
251	<pre>re_GPOPS = ae1*be1/sqrt((be1*cos(phi_GPOPS))^2 + (ae1*sin(</pre>
	<pre>phi_GPOPS))^2);</pre>

252	
253	<pre>Cost = solution.phase.integral*DU/TU*1000</pre>
254	
255	<pre>rt = rf_pqw*DU + re_GPOPS*DU*cos(phi_GPOPS)*vunit + re_GPOPS*DU*</pre>
	<pre>sin(phi_GPOPS)*gunit;</pre>
256	
257	
258	%%
259	clear setup guess bounds
260	
261	colnum = 4;
262	colp = 40;
263	% GPOPS Setup
264	% Phase 1 Information
265	<pre>iphase = 1;</pre>
266	<pre>limits(iphase).intervals = 1;</pre>
267	<pre>limits(iphase).nodesperint = 100;</pre>
268	<pre>bounds.phase(iphase).initialtime.lower = t0min;</pre>
269	<pre>bounds.phase(iphase).initialtime.upper = t0max;</pre>
270	<pre>bounds.phase(iphase).finaltime.lower = tf1/TU;</pre>
271	<pre>bounds.phase(iphase).finaltime.upper = tf1/TU;</pre>
272	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
273	<pre>bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU</pre>
	<pre>0 sqrt(MU2/r1)];</pre>
274	<pre>bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU</pre>
	<pre>0 sqrt(MU2/r1)];</pre>
275	<pre>bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.2</pre>
	0];
276	<pre>bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.2</pre>
	1.1];
277	<pre>bounds.phase(iphase).state.lower = [r1-0.1 theta_rf-n0*tf1/TU</pre>
	-0.2 0];

```
bounds.phase(iphase).state.upper = [r1+0.1 thetaf_max 0.2 1.1];
278
           bounds.phase(iphase).control.lower = [0 umin];
279
           bounds.phase(iphase).control.upper = [Tmax umax];
280
           % LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
281
           bounds.parameter.lower = 0;
282
           bounds.parameter.upper = 2*pi;
283
           bounds.phase(iphase).path.lower = []; % None
284
           bounds.phase(iphase).path.upper = []; % None
285
           bounds.phase(iphase).integral.lower = 0;
286
           bounds.phase(iphase).integral.upper = 1;
287
           bounds.eventgroup(iphase).lower = [0 0 Rmin/DU Rmin/DU]; % None
288
           bounds.eventgroup(iphase).upper = [0 0 Rmax/DU Rmax/DU]; % None
289
           % bounds.eventgroup(iphase).lower = [0 0]; % None
290
           % bounds.eventgroup(iphase).upper = [0 0]; % None
291
           % GUESS SOLUTION
292
           guess.phase(iphase).time = tvec;
293
            guess.phase(iphase).state(:,1) = r_GPOPS;
294
           guess.phase(iphase).state(:,2) = theta_GPOPS;
295
            guess.phase(iphase).state(:,3) = Vr_GPOPS;
296
            guess.phase(iphase).state(:,4) = Vt_GPOPS;
297
           % Control guess :
298
            guess.phase(iphase).control(:,1) = T_GPOPS;
299
            guess.phase(iphase).control(:,2) = Beta_GPOPS;
300
            guess.parameter = phi_GPOPS;
301
            guess.phase(iphase).integral = LT_DV;
302
303
           %auxiliary data
304
            auxdata.MU = MU2;
305
            auxdata.ae = ae1;
306
            auxdata.be = be1;
307
            auxdata.rf_pqw = rf_pqw;
308
309
            auxdata.vunit = vunit;
```

310	auxdata.gunit = gunit;
311	
312	% NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
	required
313	r0string = num2str(norm(r0vec));
314	<pre>aestr = num2str(ae);</pre>
315	<pre>itstr = num2str(PS0_data(cc,end));</pre>
316	<pre>tempstr = [aestr itstr];</pre>
317	<pre>aestring = num2str(tempstr);</pre>
318	<pre>setup.name = ['SinglePass' r0string aestring];</pre>
319	
320	<pre>setup.functions.continuous = @LT_RTM_Continuous;</pre>
321	<pre>setup.functions.endpoint = @LT_RTM_Endpoint;</pre>
322	<pre>setup.nlp.solver = 'ipopt';</pre>
323	<pre>setup.mesh.maxiteration = 50;</pre>
324	<pre>setup.mesh.tolerance = 1e-12;</pre>
325	<pre>setup.mesh.colpointsmin = 40;</pre>
326	<pre>setup.mesh.colpointsmax = 200;</pre>
327	<pre>setup.mesh.phase(iphase).colpoints = colnum*ones(1,colp);</pre>
328	<pre>setup.mesh.phase(iphase).fraction = (1/colp)*ones(1,colp);</pre>
329	<pre>setup.bounds = bounds;</pre>
330	<pre>setup.guess = guess;</pre>
331	setup.auxdata = auxdata;
332	<pre>setup.mesh.method = 'RPMintegration';</pre>
333	<pre>setup.derivatives.supplier = 'sparseFD';</pre>
334	<pre>setup.derivativelevel ='second';</pre>
335	<pre>setup.dependencies = 'sparseNaN';</pre>
336	<pre>setup.scales = 'none';</pre>
337	
338	<pre>output = gpops2(setup);</pre>
339	<pre>solution2 = output.result.solution;</pre>
340	

341	r_GPOPS2 = solution2.phase.state(:,1);
342	<pre>theta_GPOPS2 = solution2.phase.state(:,2);</pre>
343	<pre>Vr_GPOPS2 = solution2.phase.state(:,3);</pre>
344	<pre>Vt_GPOPS2 = solution2.phase.state(:,4);</pre>
345	<pre>lambda_r2 = solution2.phase.costate(:,1);</pre>
346	<pre>lambda_theta2 = solution2.phase.costate(:,2);</pre>
347	<pre>lambda_Vr2 = solution2.phase.costate(:,3);</pre>
348	<pre>lambda_Vt2 = solution2.phase.costate(:,4);</pre>
349	<pre>tvec2 = solution2.phase.time;</pre>
350	
351	<pre>thetadot_GPOPS2 = Vt_GPOPS2./r_GPOPS2;</pre>
352	<pre>T_GPOPS2 = solution2.phase.control(:,1);</pre>
353	<pre>Beta_GPOPS2 = solution2.phase.control(:,2);</pre>
354	<pre>phi_GPOPS2 = solution2.parameter;</pre>
355	<pre>re_GPOPS2 = ae1*be1/sqrt((be1*cos(phi_GPOPS2))^2 + (ae1*sin(</pre>
	<pre>phi_GPOPS2))^2);</pre>
356	
357	<pre>Cost2 = solution2.phase.integral*DU/TU*1000</pre>
358	
359	
360	<pre>fprintf(fid,'%i\t %10.5f\t %4.3f\t %4.3f\t %4.3f\t %6.5f\t %6.5f</pre>
	<pre>\t %4.3f\r\n',ae,TOF,phi,Vt_mag,fpa_t,PSO_data(cc,6)*DU/TU,</pre>
	Cost2*DU/TU*1000,phi_GPOPS2);
361	
362	
363	<pre>optans.ics = struct('r0',r0vec,'v0',v0vec,'t0',t0,'ae',ae,'be',</pre>
	<pre>be,'Rmax',Rmax,'Rmin',Rmin,'rf1',rf1,'vf1',vf1,'tf1',tf1,'</pre>
	<pre>latlim',latlim,'longlim',longlim,'GMST0',GMST0,</pre>
364	<pre>'inc',inc,'RAAN',RAAN,'w',w,'ang',ang);</pre>
365	<pre>optans.scale = struct('TU',TU,'DU',DU,'MU',MU2);</pre>

366	<pre>optans.entry = struct('lat_enter', lat_enter, 'long_enter',</pre>
	<pre>long_enter,'lat_exit',lat_exit,'long_exit',long_exit,'r_ell'</pre>
	<pre>,r_ell,'rtijk',rt_ijk,'vtijk',vt_ijk);</pre>
367	<pre>optans.phase = struct('state', solution2.phase(1).state, 'costate'</pre>
	<pre>,solution2.phase(1).costate,'control',solution2.phase(1).</pre>
	<pre>control,'time',tvec2);</pre>
368	optans.parameter = solution2.parameter;
369	
370	<pre>dir = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\</pre>
	Single Pass\Images\';
371	<pre>dir2 = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\</pre>
	Single Pass\Data\';
372	
373	<pre>tend = toc(tstart);</pre>
374	
375	<pre>exflag = output.result.nlpinfo;</pre>
376	
377	
378	
379	if cc == 1
380	<pre>fprintf(fid2,'%s %i %s %i %s\r\n','colpoint = ',colnum,'(1,',</pre>
	colp,')');
381	end
382	
383	<pre>if exflag == 0</pre>
384	<pre>fprintf(fid2,'%i\t %i \t %4.3f\t %4.3f\t %6.5f\t %6.5f\t %6.5f\t %6.5f</pre>
	%5.2f\t %i\t\r\n',
385	<pre>norm(r0),ae,phi,phi_GPOPS2,PS0_data(cc,6),Cost,Cost2,tend,</pre>
	exflag);
386	
387	<pre>[optfin] = LT_SINGLE_PASS_PLOTS(optans,r0string,aestring,dir);</pre>
388	

```
dataname = ['SinglePass' r0string aestring];
389
390
             save(strcat(dir2,[dataname]),'output');
391
             end
392
393
            close all
394
            clear optfin
395
396
        end
397
398
  end
```

E.1.2.2 Single Pass LTRTM Equations of Motion and Cost Function

```
1 function phaseout = LT_RTM_Continuous(input)
2
3 s = input.phase.state;
4 u = input.phase.control;
5
6 %% Equations of Motion
7 %
```

```
8 r = s(:,1);
9 vr = s(:,3);
10 vtheta = s(:,4);
11
12 T = u(:,1);
13 B = u(:,2);
14
15 MU2 = input.auxdata.MU;
16
17 r_dot = vr;
18 theta_dot = vtheta./r;
```

```
19 vr_dot = (vtheta.^2)./r - MU2./(r.^2) + T.*sin(B);
20 vtheta_dot = -vtheta.*vr./r + T.*cos(B);
21
22 % Form matrix output
23 daeout = [r_dot theta_dot vr_dot vtheta_dot];
24
25 phaseout.dynamics = daeout;
26 %
```

```
27 %% Cost Function
```

```
28 phaseout.integrand = T;
```

E.1.2.3 Single Pass LTRTM Constraints

```
1 function output = LT_RTM_Endpoint(input)
2
3
4 %% Cost Function Evaluation
5 %
6 J = input.phase(1).integral;
7 output.objective = J;
```

```
8 %
```

```
9 %% Event Constraints
10
```

- u t0 = input.phase(1).initialtime;
- 12 tf = input.phase(1).finaltime;
- 13 x0 = input.phase(1).initialstate;
- 14 xf = input.phase(1).finalstate;

```
15
16 rf = xf(1);
17 thetaf = xf(2);
18 Vrf = xf(3);
19 Vtf = xf(4);
20
21 p = input.parameter;
22 phi = p(1);
23
24 ae1 = input.auxdata.ae;
25 be1 = input.auxdata.be;
26 MU2 = input.auxdata.MU;
27 rf_pqw = input.auxdata.rf_pqw;
28 vunit = input.auxdata.vunit;
  gunit = input.auxdata.gunit;
29
30
  term1 = (be1*cos(phi))^2 + (ae1*sin(phi))^2;
31
32
  re = ae1*be1/sqrt(term1);
33
34
35 rt = rf_pqw + re*cos(phi)*vunit + re*sin(phi)*gunit;
36
37 %final position constraints
  event1 = rf*cos(thetaf) - rt(1);
38
  event2 = rf*sin(thetaf) - rt(2);
39
40
41 % output.eventgroup(1).event = [event1 event2];
42
43 %apogee and perigee constraints
44 Vf_mag = sqrt(Vrf<sup>2</sup> + Vtf<sup>2</sup>);
45 fpa = atan(Vrf/Vtf);
46
```

```
47 vt = Vf_mag*[-sin(thetaf-fpa);cos(thetaf-fpa);0];
48
49 [a,ecc,~,~,~,~] = RV2COE_MU(rt,vt,MU2);
50 Ra = a*(1+ecc);
51 Rp = a*(1-ecc);
52
53 event3 = Ra;
54 event4 = Rp;
55
56 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

```
1 wgs84data
2 global MU
3 OmegaEarth = 0.000072921151467;
4
5 \text{ for } bb = 10:10
6
       t0 = 0;
7
       GMST0 = 0;
8
       latlim = [-10 10]*pi/180;
9
       longlim = [-50 -10]*pi/180;
10
11
       r0vec = [7300;0;0];
12
       v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];
13
14
       [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
15
       period = 2*pi*sqrt(a^3/MU);
16
17
       aevec = [150 140 130 120 110 100 90 80 70 60 50];
18
```

```
bevec = [15 14 13 12 11 10 9 8 7 6 5];
19
       Rmaxvec = norm(r0vec) + 50;
20
       Rminvec = norm(r0vec) - 50;
21
22
       DU = norm(r@vec);
23
       TU = period/(2*pi);
24
       MU2 = MU*TU^2/DU^3;
25
26
       m0 = 1000;
27
       r0 = r0vec;
28
       v0 = v0vec;
29
       Rmax = Rmaxvec;
30
       Rmin = Rminvec;
31
32
       %Energy of most elliptical orbit
33
       ab = (Rmax + Rmin)/2; %semi-major axis of orbit
34
       Eb = -MU/(2*ab); %energy of orbit
35
       Vmax = sqrt(2*(MU/Rmin + Eb));
36
       Vmin = sqrt(2*(MU/Rmax + Eb));
37
38
       fid = fopen([dir 'PSODoublePassDataFinal_06012014.txt'],'a');
39
       state0=[r0 v0];
40
41
       Tmax = 2e-3;
42
       swarm = 40;
43
       iter = 1000;
44
       prec = [5;5;5;9];
45
46
       if bb == 1
47
48
           fprintf(fid,'%s %i\r\n','r0 (km) =',r0vec(1));
49
           fprintf(fid,'%s %i\r\n','swarm =',swarm);
50
```

```
51
          52
             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','
53
             ');
      end
54
55
      if bb == 10
56
          endval = 1;
57
      else
58
          endval = 20;
59
      end
60
61
      ae = aevec(bb);
62
      be = bevec(bb);
63
64
      [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
65
          long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim);
66
      for aa = 1:endval
67
68
          tstart = tic;
69
70
          [JGmin, Jpbest, gbest, x, k] = LT_RTM_PSO_TFIXED(3, [0 2*pi; Vmin Vmax
71
             ;-pi/2+0.000001 pi/2-0.000001],iter,swarm,prec,rf1,vf1,tf1,
             ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0);
72
          Cost1 = JGmin*DU/TU*1000
73
74
```

```
264
```
```
tend = toc(tstart)
75
76
           tstart2 = tic;
77
78
           [~,~,~,~,~,~,~,~,~,~,~,~,rt_ijk,vt_ijk] = Single_LT_Maneuver(rf1,
79
              vf1,tf1,gbest(1),ae,be,gbest(2),gbest(3),DU,TU,MU2);
80
           [rf2,vf2,tf2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0+OmegaEarth*
81
              tf1,0,latlim,longlim);
82
           [JGmin2, Jpbest2, gbest2, x2, k2] = LT_RTM_PSO_TFIXED(3, [0 2*pi; Vmin
83
                Vmax;-pi/2+0.000001 pi/2-0.000001],iter,swarm,prec,rf2,vf2,
              tf2,ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0);
84
           Cost2 = JGmin2*DU/TU*1000
85
86
           CostTOT = Cost1 + Cost2
87
88
           tend2 = toc(tstart2)
89
90
91
92
           fprintf(fid,'%i\t %i \t %10.5f\t %6.5f\t %7.6f\t %6.5f\t %10.5f\
93
              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),
94
                   gbest2(2),gbest2(3),Cost1,Cost2,CostT0T,k,k2,k+k2,tend,
                   tend2,tend+tend2);
           tend + tend2
95
96
```

```
97 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
98 end
99
100 end
```

E.2.2 Direct Collocation Algorithms

E.2.2.1 Double Pass LTRTM Driver

```
1 \text{ for } zz = 1:1
      clear guess setup limits output
2
      close all
3
      clc
4
5
      if zz == 1
6
             load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
7 %
      Double Pass\Journal Data\data6800_LT_2RTMsort.mat')
             [ind0] = find(data6800_LT_2RTMsort(:,1) ~= 0);
8 %
             PSO_data = data6800_LT_2RTMsort(ind0,:);
  %
9
           load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
10
              Double Pass\Journal Data\data6800_LT_2RTM_2ndTier.mat')
           [ind0] = find(data6800_LT_2RTM_2ndTier(:,1) ~= 0);
11
           PSO_data = data6800_LT_2RTM_2ndTier(ind0,:);
12
           cmax = length(PS0_data);
13
           cmin = 1;
14
           rmag = 6800;
15
       elseif zz == 2
16
17 %
             load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
      Double Pass\Journal Data\data7300_LT_2RTMsort.mat')
18 %
             [ind0] = find(data7300_LT_2RTMsort(:,1) ~= 0);
19 %
             PSO_data = data7300_LT_2RTMsort(ind0,:);
```

```
load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
20
               Double Pass\Journal Data\data7300_LT_2RTM_2ndTier.mat')
           [ind0] = find(data7300_LT_2RTM_2ndTier(:,1) ~= 0);
21
           PSO_data = data7300_LT_2RTM_2ndTier(ind0,:);
22
           cmax = length(PS0_data);
23
           cmin = 1;
24
           rmag = 7300;
25
       end
26
27
28
       tstart = tic;
29
       for cc = cmin:cmax
30
           fid = fopen('PSO_to_GPOPS_2RTM.txt','a');
31
           clear guess setup limits output
32
           close all
33
           clc
34
35
           t0 = 0;
36
           GMST0 = 0;
37
           latlim = [-10 10]*pi/180;
38
           longlim = [-50 -10]*pi/180;
39
40
           wgs84data
41
           global MU2 MU
42
           OmegaEarth = 0.000072921151467;
43
           r0vec = [rmag;0;0];
44
           v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];
45
46
           [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
47
           period = 2*pi*sqrt(a^3/MU);
48
49
50
           swarm = 30;
```

```
iter = 1000;
51
52
           Rmaxvec = rmag + 50;
           Rminvec = rmag - 50;
53
           prec = [2;5;16];
54
55
           r0 = r0vec;
56
           v0 = v0vec;
57
           Rmax = Rmaxvec;
58
           Rmin = Rminvec;
59
60
           ae = PSO_data(cc,2);
61
           be = ae/10;
62
63
           TOF = PSO_data(cc,3);
64
           phi = PSO_data(cc,4);
65
           Vt_mag = PSO_data(cc,5);
66
           fpa_t = PSO_data(cc,6);
67
68
           tstart = tic;
69
70
           [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
71
               long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim)
               ;
72
           DU = norm(rf1);
73
           TU = period/(2*pi);
74
75
           ae1 = ae/DU;
76
           be1 = be/DU;
77
           r01 = norm(r0)/DU;
78
           MU2 = MU*TU^{2}/DU^{3};
79
           tOmin = 0; % minimum initial time
80
```

81	tOmax = 0; % maximum initial time
82	tfmin = tf1; % minimum final time
83	tfmax = tf1;
84	$n0 = sqrt(MU2/(norm(r0)/DU)^3);$
85	
86	%% First Maneuver
87	[LT_DV,maxT,r,gamma,T_a,thetaf_int,theta_dot,theta_ddot,rdot,
	<pre>Tvec,TOF_calc,rt_ijk,vt_ijk,rt_pqw,vt_pqw,r0_pqw,v0_pqw] =</pre>
	<pre>Single_LT_Maneuver(rf1,vf1,TOF,phi,ae,be,Vt_mag,fpa_t,DU,TU,</pre>
	MU2);
88	
89	<pre>delt = (tf1 - TOF)/TU;</pre>
90	<pre>time_mod = Tvec + delt;</pre>
91	
92	<pre>[rf_pqw,vf_pqw] = IJK_to_PQW(rf1,vf1,inc,RAAN,w);</pre>
93	<pre>rf_pqw = rf_pqw/DU;</pre>
94	vf_pqw = vf_pqw/DU*TU;
95	
96	
97	<pre>vunit = vf_pqw/norm(vf_pqw);</pre>
98	hfp = cross(rf_pqw,vf_pqw);
99	<pre>hunit = hfp/norm(hfp);</pre>
100	
101	<pre>gunit = cross(vunit,hunit);</pre>
102	
103	ang = (0:0.001:2*pi);
104	<pre>re = (ae1*be1)./sqrt((be1*cos(ang)).^2 + (ae1*sin(ang)).^2);</pre>
105	
106	
107	<pre>theta_rf = atan2(rf_pqw(2),rf_pqw(1));</pre>
108	<pre>if theta_rf < 0</pre>
109	<pre>theta_rf = 2*pi + theta_rf;</pre>

110	end
111	<pre>[rtest] = IJK_to_PQW(r0,v0,inc,RAAN,w);</pre>
112	<pre>theta0 = atan2(rtest(2),rtest(1));</pre>
113	
114	<pre>theta_mod = thetaf_int + atan2(r0_pqw(2),r0_pqw(1));</pre>
115	
116	<pre>coast_length = 1;</pre>
117	
118	<pre>time_guess = zeros(coast_length+length(time_mod),1);</pre>
119	<pre>time_guess(1:coast_length) = 0;</pre>
120	<pre>time_guess(coast_length+1:end) = time_mod;</pre>
121	<pre>theta_guess = zeros(coast_length+length(theta_mod),1);</pre>
122	<pre>theta_guess(coast_length+1:end) = theta_mod;</pre>
123	<pre>theta_guess(1:coast_length) = theta0;</pre>
124	<pre>r_guess = zeros(coast_length+length(theta_mod),1);</pre>
125	<pre>r_guess(1:coast_length) = norm(r0)/DU;</pre>
126	<pre>r_guess(coast_length+1:end) = r;</pre>
127	<pre>vr_guess = zeros(coast_length+length(theta_mod),1);</pre>
128	<pre>vr_guess(1:coast_length) = 0;</pre>
129	<pre>vr_guess(coast_length+1:end) = rdot;</pre>
130	<pre>vtheta_guess = zeros(coast_length+length(theta_mod),1);</pre>
131	<pre>vtheta_guess(1:coast_length) = sqrt(MU2/(norm(r0)/DU));</pre>
132	<pre>vtheta_guess(coast_length+1:end) = r.*theta_dot;</pre>
133	<pre>T_guess = zeros(coast_length+length(time_mod),1);</pre>
134	<pre>T_guess(1:coast_length) = 0;</pre>
135	<pre>T_guess(coast_length+1:end) = T_a;</pre>
136	<pre>B_guess = zeros(coast_length+length(time_mod),1);</pre>
137	<pre>B_guess(1:coast_length) = 0;</pre>
138	<pre>B_guess(coast_length+1:end) = gamma;</pre>
139	
140	<pre>ind = find(T_guess ~= 0);</pre>
141	

%inertial position vector of new arrival position 142 for aa = 1:length(ang) 143 r_ell(:,aa) = rf_pqw + re(aa)*cos(ang(aa))*vunit + re(aa)* 144 sin(ang(aa))*gunit; end 145 146 for dd = 1:length(r_guess) 147 rg_pqw = DU*[r_guess(dd)*cos(theta_guess(dd));r_guess(dd)* 148 sin(theta_guess(dd));0]; 149 [rgi(dd,:)] = PQW_to_IJK(rg_pqw,[],inc,RAAN,w); end 150 151 for ee = 1:length(ang) 152 rell_pqw = [r_ell(1,ee)*DU;r_ell(2,ee)*DU;0]; 153 rnom_pqw = norm(r0)*[cos(ang(ee));sin(ang(ee));0]; 154 [rell_ijk(:,ee)] = PQW_to_IJK(rell_pqw,[],inc,RAAN,w); 155 [rnom_ijk(ee,:)] = PQW_to_IJK(rnom_pqw,[],inc,RAAN,w); 156 end 157 158 159 %% determine limits on subsequent passes into exclusion zone 160 % assume upper limit based on circular orbit with phi = pi/2161 % assume lower limit based on circular orbit with phi = 2pi/2 162 $phi_low = 3*pi/2;$ 163 phi_upp = pi/2; 164 165 [rf_upp,vf_upp] = Single_LT_Limits(rf1,vf1,phi_upp,ae,be,DU,TU); 166 167 [rf2_upp,vf2_upp,tf2_upp] = zone_entry_exit2(rf_upp,vf_upp,GMST0 168 +OmegaEarth*tf1,0,latlim,longlim); 169 170 ang_upp = sqrt(MU/norm(rf_upp)^3)*tf2_upp;

thetaf2_max = theta_guess(end) + ang_upp; 172 $tf2_max = (tf1 + tf2_upp)/TU;$ 173 174 175 [rf_low,vf_low] = Single_LT_Limits(rf1,vf1,phi_low,ae,be,DU,TU); 176 177 [rf2_low,vf2_low,tf2_low] = zone_entry_exit2(rf_low,vf_low,GMST0 178 +OmegaEarth*tf1,0,latlim,longlim); 179 ang_low = sqrt(MU/norm(rf_low)^3)*tf2_low; 180181 thetaf2_min = theta_guess(end) + ang_low; 182 $tf2_min = (tf1 + tf2_low)/TU;$ 183 184 185 186 %% Second Maneuver 187 [rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2, 188 lat_exit2,long_exit2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0 +OmegaEarth*tf1,0,latlim,longlim); 189 190 TOF2 = PS0_data(cc,7); 191 phi2 = PSO_data(cc,8); 192 Vt_mag2 = PSO_data(cc,9); 193 fpa_t2 = PSO_data(cc,10); 194 195 [LT_DV2,maxT2,r2,gamma2,T_a2,thetaf_int2,theta_dot2,theta_ddot2, 196 rdot2,Tvec2,TOF_calc2,rt_ijk2,vt_ijk2] = Single_LT_Maneuver(rf2,vf2,TOF2,phi2,ae,be,Vt_mag2,fpa_t2,DU,TU,MU2);

197

```
theta02 = theta_guess(end);
198
199
            delt2 = (tf2 - TOF2)/TU;
200
            time_mod2 = Tvec2 + delt2 + time_guess(end);
201
202
            [rf_pqw2,vf_pqw2] = IJK_to_PQW(rf2,vf2,inc,RAAN,w);
203
            rf_pqw2 = rf_pqw2/DU;
204
            vf_pqw2 = vf_pqw2/DU*TU;
205
206
207
            vunit2 = vf_pqw2/norm(vf_pqw2);
            hfp2 = cross(rf_pqw2,vf_pqw2);
208
            hunit2 = hfp2/norm(hfp2);
209
210
            gunit2 = cross(vunit2, hunit2);
211
212
213
            %inertial position vector of new arrival position
214
            for bb = 1:length(ang)
215
                r_ell2(:,bb) = rf_pqw2 + re(bb)*cos(ang(bb))*vunit2 + re(bb)
216
                    *sin(ang(bb))*gunit2;
            end
217
218
            [rt_pqw2,vt_pqw2] = IJK_to_PQW(rt_ijk2,vt_ijk2,inc,RAAN,w);
219
            ang_mod2 = atan2(rt_pqw2(2),rt_pqw2(1));
220
            if ang_mod2 < 0</pre>
221
                ang_mod2 = ang_mod2 + 2*pi;
222
            end
223
            ang_mod1 = atan2(rt_pqw(2),rt_pqw(1));
224
225
            diff = ang_mod2 - ang_mod1;
226
            diff2 = thetaf_int2(end) - thetaf_int2(1);
227
            theta_diff = diff - diff2;
228
```

229	
230	<pre>theta_mod2 = thetaf_int2 + theta_diff + theta_guess(end);</pre>
231	
232	<pre>coast_length = 1;</pre>
233	
234	<pre>time_guess2 = zeros(coast_length+length(time_mod2),1);</pre>
235	<pre>time_guess2(1:coast_length) = time_guess(end);</pre>
236	<pre>time_guess2(coast_length+1:end) = time_mod2;</pre>
237	<pre>theta_guess2 = zeros(coast_length+length(theta_mod2),1);</pre>
238	<pre>theta_guess2(coast_length+1:end) = theta_mod2;</pre>
239	<pre>theta_guess2(1:coast_length) = theta02;</pre>
240	<pre>r_guess2 = zeros(coast_length+length(theta_mod2),1);</pre>
241	<pre>r_guess2(1:coast_length) = r_guess(end);</pre>
242	<pre>r_guess2(coast_length+1: end) = r2;</pre>
243	<pre>vr_guess2 = zeros(coast_length+length(theta_mod2),1);</pre>
244	<pre>vr_guess2(1:coast_length) = vr_guess(end);</pre>
245	<pre>vr_guess2(coast_length+1:end) = rdot2;</pre>
246	<pre>vtheta_guess2 = zeros(coast_length+length(theta_mod2),1);</pre>
247	<pre>vtheta_guess2(1:coast_length) = vtheta_guess(end);</pre>
248	<pre>vtheta_guess2(coast_length+1:end) = r2.*theta_dot2;</pre>
249	<pre>T_guess2 = zeros(coast_length+length(time_mod2),1);</pre>
250	<pre>T_guess2(1:coast_length) = 0;</pre>
251	T_guess2(coast_length+1: end) = T_a2;
252	<pre>B_guess2 = zeros(coast_length+length(time_mod2),1);</pre>
253	<pre>B_guess2(1:coast_length) = B_guess(end);</pre>
254	<pre>B_guess2(coast_length+1: end) = gamma2;</pre>
255	
256	<pre>ind2 = find(T_guess2 ~= 0);</pre>
257	
258	nom_orb2_time = [(0:1:tf2) tf2];
259	
260	<pre>[at,et,it,Ot,ot,nut]= RV2COE(rt_ijk,vt_ijk);</pre>

for ee = 1:length(nom_orb2_time) 262 [nutf] = nuf_from_TOF(nut,nom_orb2_time(ee),at,et); 263 [Rdum(:,ee),Vdum] = COE2RV(at,et,it,Ot,ot,nutf); 264 [nom_orb2_R] = IJK_to_PQW(Rdum(:,ee),Vdum,inc,RAAN,w); 265 266 ROrb2_PQW(ee,:) = nom_orb2_R; 267 268 end 269 if nutf < nut</pre> 270 nutf = nutf + 2*pi; 271 end 272 273 %Angle of expected 2nd pass entry location into exclusion zone 274 thetaf2 = (nutf - nut) + 0;275 276 277 %% GPOPS RUN (1st Run Through assigns a non-zero minimum thrust 278 to help GPOPS-II converge) % variables from PSo phase 279 r1 = 1;280 rf = norm(rt_pqw); 281 rmax = r1 + be/DU;282 rmin = r1 - be/DU;283 thetaf_min = theta_rf - atan(ae/norm(r0)); 284 thetaf_max = theta_rf + atan(ae/norm(r0)); 285 286 % Control and time boundaries 287 umin = -pi; % minimum control angle 288 umax = pi; % maximum control angle 289 Tmax = 2*0.0001160;290 291 Tmin = Tmax/1000;

292	
293	min1 = -0.5;
294	min2 = -0.5;
295	$\max 1 = 2 * pi + 0.5;$
296	$\max 2 = 2 * pi + 0.5;$
297	
298	%colocation points and fraction
299	colnum = 4;
300	colp = 40;
301	
302	% GPOPS Setup
303	% Phase 1 Information
304	<pre>iphase = 1;</pre>
305	<pre>limits(iphase).intervals = 1;</pre>
306	<pre>limits(iphase).nodesperint = 100;</pre>
307	<pre>bounds.phase(iphase).initialtime.lower = t0min;</pre>
308	<pre>bounds.phase(iphase).initialtime.upper = t0max;</pre>
309	<pre>bounds.phase(iphase).finaltime.lower = tf1/TU;</pre>
310	<pre>bounds.phase(iphase).finaltime.upper = tf1/TU;</pre>
311	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
312	<pre>bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU</pre>
	<pre>0 sqrt(MU2/r1)];</pre>
313	<pre>bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU</pre>
	<pre>0 sqrt(MU2/r1)];</pre>
314	<pre>bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.2</pre>
	0];
315	<pre>bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.2</pre>
	1.2];
316	<pre>bounds.phase(iphase).state.lower = [r1-0.1 theta_rf-n0*tf1/TU</pre>
	-0.2 0];
317	<pre>bounds.phase(iphase).state.upper = [r1+0.1 thetaf_max 0.2 1.2];</pre>
318	<pre>bounds.phase(iphase).control.lower = [Tmin umin1];</pre>

319	<pre>bounds.phase(iphase).control.upper = [Tmax umax1];</pre>
320	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
321	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
322	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
323	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
324	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
325	<pre>bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 Rmin/DU Rmin/</pre>
	DU]; % None
326	<pre>bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 Rmax/DU Rmax/</pre>
	DU]; % None
327	% GUESS SOLUTION
328	<pre>guess.phase(iphase).time = time_guess;</pre>
329	<pre>guess.phase(iphase).state(:,1) = r_guess;</pre>
330	<pre>guess.phase(iphase).state(:,2) = theta_guess;</pre>
331	<pre>guess.phase(iphase).state(:,3) = vr_guess;</pre>
332	<pre>guess.phase(iphase).state(:,4) = vtheta_guess;</pre>
333	% Control guess :
334	<pre>guess.phase(iphase).control(:,1) = T_guess;</pre>
335	<pre>guess.phase(iphase).control(:,2) = B_guess;</pre>
336	<pre>guess.phase(iphase).integral = LT_DV;</pre>
337	
338	% Phase 2 Information (second Maneuver
339	<pre>iphase = 2;</pre>
340	limits(iphase).intervals = 1;
341	<pre>limits(iphase).nodesperint = 100;</pre>
342	<pre>bounds.phase(iphase).initialtime.lower = tf1/TU;</pre>
343	<pre>bounds.phase(iphase).initialtime.upper = tf1/TU;</pre>
344	<pre>bounds.phase(iphase).finaltime.lower = tf2_min-1;</pre>
345	<pre>bounds.phase(iphase).finaltime.upper = tf2_max+1;</pre>
346	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
347	<pre>bounds.phase(iphase).initialstate.lower = [rmin thetaf_min -0.2</pre>
	0];

348	<pre>bounds.phase(iphase).initialstate.upper = [rmax thetaf_max 0.2</pre>
	1.2];
349	bounds.phase(iphase).finalstate.lower = [rmin thetaf2_min-1 -0.2
	0];
350	<pre>bounds.phase(iphase).finalstate.upper = [rmax thetaf2_max+1 0.2</pre>
	1.2];
351	<pre>bounds.phase(iphase).state.lower = [r1-0.1 thetaf_min -0.2 0];</pre>
352	<pre>bounds.phase(iphase).state.upper = [r1+0.1 thetaf2_max+1 0.2</pre>
	1.2];
353	<pre>bounds.phase(iphase).control.lower = [Tmin umin2];</pre>
354	<pre>bounds.phase(iphase).control.upper = [Tmax umax2];</pre>
355	<pre>bounds.parameter.lower = [0 0];</pre>
356	<pre>bounds.parameter.upper = [2*pi 2*pi];</pre>
357	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
358	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
359	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
360	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
361	<pre>bounds.eventgroup(iphase).lower = [0 0 0 Rmin/DU Rmin/DU]; %</pre>
	None
362	<pre>bounds.eventgroup(iphase).upper = [0 0 0 Rmax/DU Rmax/DU]; %</pre>
	None
363	% GUESS SOLUTION
364	<pre>guess.phase(iphase).time = time_guess2;</pre>
365	<pre>guess.phase(iphase).state(:,1) = r_guess2;</pre>
366	<pre>guess.phase(iphase).state(:,2) = theta_guess2;</pre>
367	<pre>guess.phase(iphase).state(:,3) = vr_guess2;</pre>
368	<pre>guess.phase(iphase).state(:,4) = vtheta_guess2;</pre>
369	% Control guess :
370	<pre>guess.phase(iphase).control(:,1) = T_guess2;</pre>
371	<pre>guess.phase(iphase).control(:,2) = B_guess2;</pre>
372	<pre>guess.parameter = [phi phi2];</pre>
373	% guess.parameter = [phi];

guess.phase(iphase).integral = LT_DV2; 374 375 %auxiliary data 376 auxdata.MU = MU2; 377 auxdata.ae = ae1;378 auxdata.be = be1; 379 auxdata.rf_pqw = rf_pqw; 380 auxdata.vunit = vunit; 381 auxdata.gunit = gunit; 382 383 auxdata.inc = inc; auxdata.RAAN = RAAN; 384 auxdata.w = w;385 auxdata.latlim = latlim; 386 auxdata.longlim = longlim; 387 auxdata.GMST0 = GMST0; 388 auxdata.OmegaEarth = OmegaEarth; 389 auxdata.DU = DU; 390 auxdata.TU = TU;391 392 % NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae" 393 required r0string = num2str(norm(r0vec)); 394 aestr = num2str(ae); 395 itstr = num2str(PS0_data(cc,end)); 396 tempstr = [aestr itstr]; 397 aestring = num2str(tempstr); 398 setup.name = ['DoublePass' r0string aestring]; 399 400 setup.functions.continuous = @LT_2RTM_Continuous; 401 setup.functions.endpoint = @LT_2RTM_Endpoint; 402 setup.nlp.solver = 'ipopt'; 403 404 setup.mesh.maxiteration = 10;

```
setup.mesh.tolerance = 1e-10;
405
            setup.mesh.colpointsmin = 40;
406
            setup.mesh.colpointsmax = 400;
407
            for ival = 1:2
408
                setup.mesh.phase(ival).colpoints = colnum*ones(1,colp);
409
                setup.mesh.phase(ival).fraction = (1/colp)*ones(1,colp);
410
            end
411
            setup.bounds = bounds;
412
            setup.guess = guess;
413
            setup.auxdata = auxdata;
414
            setup.mesh.method = 'RPMintegration';
415
            setup.derivatives.supplier = 'sparseFD';
416
            setup.derivativelevel ='second';
417
            setup.dependencies = 'sparseNaN';
418
            setup.scales = 'none';
419
420
            output = gpops2(setup);
421
            solution = output.result.solution;
422
           %%
423
           %States and costates from phase 1 (first maneuver)
424
            r_GPOPS_P1 = solution.phase(1).state(:,1);
425
            theta_GPOPS_P1 = solution.phase(1).state(:,2);
426
            Vr_GPOPS_P1 = solution.phase(1).state(:,3);
427
            Vt_GPOPS_P1 = solution.phase(1).state(:,4);
428
            lambda_r_P1 = solution.phase(1).costate(:,1);
429
            lambda_theta_P1 = solution.phase(1).costate(:,2);
430
            lambda_Vr_P1 = solution.phase(1).costate(:,3);
431
            lambda_Vt_P1 = solution.phase(1).costate(:,4);
432
            tvec_P1 = solution.phase(1).time;
433
434
            thetadot_GPOPS_P1 = Vt_GPOPS_P1./r_GPOPS_P1;
435
            T_GPOPS_P1 = solution.phase(1).control(:,1);
436
```

437	<pre>Beta_GPOPS_P1 = solution.phase(1).control(:,2);</pre>
438	<pre>phi_GPOPS_P1 = solution.parameter(1);</pre>
439	<pre>re_GPOPS_P1 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P1))^2 + (ae1*sin(</pre>
	<pre>phi_GPOPS_P1))^2);</pre>
440	
441	%

- -

_ _

442	
443	%States and Costates from pahse 2 (second maneuver)
444	<pre>r_GPOPS_P2 = solution.phase(2).state(:,1);</pre>
445	<pre>theta_GPOPS_P2 = solution.phase(2).state(:,2);</pre>
446	<pre>Vr_GPOPS_P2 = solution.phase(2).state(:,3);</pre>
447	<pre>Vt_GPOPS_P2 = solution.phase(2).state(:,4);</pre>
448	<pre>lambda_r_P2 = solution.phase(2).costate(:,1);</pre>
449	<pre>lambda_theta_P2 = solution.phase(2).costate(:,2);</pre>
450	<pre>lambda_Vr_P2 = solution.phase(2).costate(:,3);</pre>
451	<pre>lambda_Vt_P2 = solution.phase(2).costate(:,4);</pre>
452	<pre>tvec_P2 = solution.phase(2).time;</pre>
453	
454	<pre>thetadot_GPOPS_P2 = Vt_GPOPS_P2./r_GPOPS_P2;</pre>
455	<pre>T_GPOPS_P2 = solution.phase(2).control(:,1);</pre>
456	<pre>Beta_GPOPS_P2 = solution.phase(2).control(:,2);</pre>
457	<pre>phi_GPOPS_P2 = solution.parameter(2);</pre>
458	<pre>re_GPOPS_P2 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P2))^2 + (ae1*sin(</pre>
	<pre>phi_GPOPS_P2))^2);</pre>
459	%
460	<pre>Cost = (solution.phase(1).integral + solution.phase(2).integral)</pre>

*DU/TU*1000;

```
%% GPOPS Run two (Minimum thrust is set to zero in run 2 to
462
               generate true optimal solution
           clear guess setup bound limits
463
464
           %colocation points and fraction
465
           colnum = 4;
466
           colp = 40;
467
468
           % Phase 1 Information
469
470
           iphase = 1;
           limits(iphase).intervals = 1;
471
           limits(iphase).nodesperint = 100;
472
           bounds.phase(iphase).initialtime.lower = t0min;
473
           bounds.phase(iphase).initialtime.upper = t0max;
474
           bounds.phase(iphase).finaltime.lower = tf1/TU;
475
           bounds.phase(iphase).finaltime.upper = tf1/TU;
476
           % LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
477
           bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU
478
                0 sqrt(MU2/r1)];
           bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU
479
                0 sqrt(MU2/r1)];
           bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.2
480
               0];
           bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.2
481
               1.1];
           bounds.phase(iphase).state.lower = [r1-0.1 theta_rf-n0*tf1/TU
482
               -0.2 0];
           bounds.phase(iphase).state.upper = [r1+0.1 thetaf_max 0.2 1.1];
483
           bounds.phase(iphase).control.lower = [0 umin1];
484
           bounds.phase(iphase).control.upper = [Tmax umax1];
485
           % LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
486
487
           bounds.phase(iphase).path.lower = []; % None
```

```
282
```

488	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
489	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
490	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
491	<pre>bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 Rmin/DU Rmin/</pre>
	DU]; % None
492	<pre>bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 Rmax/DU Rmax/</pre>
	DU]; % None
493	% GUESS SOLUTION
494	<pre>guess.phase(iphase).time = tvec_P1;</pre>
495	<pre>guess.phase(iphase).state(:,1) = r_GPOPS_P1;</pre>
496	<pre>guess.phase(iphase).state(:,2) = theta_GPOPS_P1;</pre>
497	<pre>guess.phase(iphase).state(:,3) = Vr_GPOPS_P1;</pre>
498	<pre>guess.phase(iphase).state(:,4) = Vt_GPOPS_P1;</pre>
499	% Control guess :
500	<pre>guess.phase(iphase).control(:,1) = T_GPOPS_P1;</pre>
501	<pre>guess.phase(iphase).control(:,2) = Beta_GPOPS_P1;</pre>
502	<pre>guess.phase(iphase).integral = LT_DV;</pre>
503	
504	% Phase 2 Information (second Maneuver
505	iphase = 2;
506	<pre>limits(iphase).intervals = 1;</pre>
507	<pre>limits(iphase).nodesperint = 100;</pre>
508	<pre>bounds.phase(iphase).initialtime.lower = tf1/TU;</pre>
509	<pre>bounds.phase(iphase).initialtime.upper = tf1/TU;</pre>
510	<pre>bounds.phase(iphase).finaltime.lower = tf2_min-1;</pre>
511	<pre>bounds.phase(iphase).finaltime.upper = tf2_max+1;</pre>
512	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
513	<pre>bounds.phase(iphase).initialstate.lower = [rmin thetaf_min -0.2</pre>
	0];
514	<pre>bounds.phase(iphase).initialstate.upper = [rmax thetaf_max 0.2</pre>
	1.1];

515	<pre>bounds.phase(iphase).finalstate.lower = [rmin thetaf2_min-1 -0.2</pre>
	0];
516	<pre>bounds.phase(iphase).finalstate.upper = [rmax thetaf2_max+1 0.2</pre>
	1.1];
517	<pre>bounds.phase(iphase).state.lower = [r1-0.1 thetaf_min -0.2 0];</pre>
518	<pre>bounds.phase(iphase).state.upper = [r1+0.1 thetaf2_max+1 0.2</pre>
	1.1];
519	<pre>bounds.phase(iphase).control.lower = [0 umin2];</pre>
520	<pre>bounds.phase(iphase).control.upper = [Tmax umax2];</pre>
521	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
522	<pre>bounds.parameter.lower = [0 0];</pre>
523	<pre>bounds.parameter.upper = [2*pi 2*pi];</pre>
524	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
525	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
526	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
527	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
528	<pre>bounds.eventgroup(iphase).lower = [0 0 0 Rmin/DU Rmin/DU]; %</pre>
	None
529	<pre>bounds.eventgroup(iphase).upper = [0 0 0 Rmax/DU Rmax/DU]; %</pre>
	None
530	% GUESS SOLUTION
531	<pre>guess.phase(iphase).time = tvec_P2;</pre>
532	<pre>guess.phase(iphase).state(:,1) = r_GPOPS_P2;</pre>
533	<pre>guess.phase(iphase).state(:,2) = theta_GPOPS_P2;</pre>
534	<pre>guess.phase(iphase).state(:,3) = Vr_GPOPS_P2;</pre>
535	<pre>guess.phase(iphase).state(:,4) = Vt_GPOPS_P2;</pre>
536	% Control guess :
537	<pre>guess.phase(iphase).control(:,1) = T_GPOPS_P2;</pre>
538	<pre>guess.phase(iphase).control(:,2) = Beta_GPOPS_P2;</pre>
539	<pre>guess.parameter = [phi_GPOPS_P1 phi_GPOPS_P2];</pre>
540	% guess.parameter = [phi];
541	guess.phase(iphase).integral = Cost;

542	
543	%auxiliary data
544	auxdata.MU = MU2;
545	auxdata.ae = ae1;
546	auxdata.be = be1;
547	<pre>auxdata.rf_pqw = rf_pqw;</pre>
548	auxdata.vunit = vunit;
549	auxdata.gunit = gunit;
550	auxdata.inc = inc;
551	auxdata.RAAN = RAAN;
552	auxdata.w = w;
553	auxdata.latlim = latlim;
554	auxdata.longlim = longlim;
555	auxdata.GMST0 = GMST0;
556	<pre>auxdata.OmegaEarth = OmegaEarth;</pre>
557	auxdata.DU = DU;
558	auxdata.TU = TU;
559	
560	% NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
	required
561	r0string = num2str(norm(r0vec));
562	<pre>aestr = num2str(ae);</pre>
563	<pre>itstr = num2str(PS0_data(cc,end));</pre>
564	<pre>tempstr = [aestr itstr];</pre>
565	<pre>aestring = num2str(tempstr);</pre>
566	<pre>setup.name = ['DoublePass' r0string aestring];</pre>
567	
568	<pre>setup.functions.continuous = @LT_2RTM_Continuous;</pre>
569	<pre>setup.functions.endpoint = @LT_2RTM_Endpoint;</pre>
570	<pre>setup.nlp.solver = 'ipopt';</pre>
571	<pre>setup.mesh.maxiteration = 10;</pre>
572	<pre>setup.mesh.tolerance = 1e-10;</pre>

```
setup.mesh.colpointsmin = 40;
573
           setup.mesh.colpointsmax = 400;
574
            for ival = 1:2
575
                setup.mesh.phase(ival).colpoints = colnum*ones(1,colp);
576
                setup.mesh.phase(ival).fraction = (1/colp)*ones(1,colp);
577
           end
578
           setup.bounds = bounds;
579
           setup.guess = guess;
580
            setup.auxdata = auxdata;
581
582
           setup.mesh.method = 'RPMintegration';
            setup.derivatives.supplier = 'sparseFD';
583
            setup.derivativelevel ='second';
584
            setup.dependencies = 'sparseNaN';
585
            setup.scales = 'none';
586
587
           output = gpops2(setup);
588
           solution2 = output.result.solution;
589
           %%
590
           %States and costates from phase 1 (first maneuver)
591
           r_GPOPS_P12 = solution2.phase(1).state(:,1);
592
           theta_GPOPS_P12 = solution2.phase(1).state(:,2);
593
           Vr_GPOPS_P12 = solution2.phase(1).state(:,3);
594
           Vt_GPOPS_P12 = solution2.phase(1).state(:,4);
595
            lambda_r_P12 = solution2.phase(1).costate(:,1);
596
            lambda_theta_P12 = solution2.phase(1).costate(:,2);
597
            lambda_Vr_P12 = solution2.phase(1).costate(:,3);
598
            lambda_Vt_P12 = solution2.phase(1).costate(:,4);
599
            tvec_P12 = solution2.phase(1).time;
600
601
            thetadot_GPOPS_P12 = Vt_GPOPS_P12./r_GPOPS_P12;
602
           T_GPOPS_P12 = solution2.phase(1).control(:,1);
603
604
            Beta_GPOPS_P12 = solution2.phase(1).control(:,2);
```

```
ind1_large = find(Beta_GPOPS_P12 > 2*pi);
605
            ind1_small = find(Beta_GPOPS_P12 < 0);</pre>
606
607
            while isempty(ind1_large) == 0
608
                Beta_GPOPS_P12(ind1_large) = Beta_GPOPS_P12(ind1_large) - 2*
609
                    pi;
                ind1_large = find(Beta_GPOPS_P12 > 2*pi);
610
            end
611
612
            while isempty(ind1_small) == 0
613
                Beta_GPOPS_P12(ind1_small) = Beta_GPOPS_P12(ind1_small) + 2*
614
                    pi;
                ind1_small = find(Beta_GPOPS_P12 < 0);</pre>
615
            end
616
617
            phi_GPOPS_P12 = solution2.parameter(1);
618
            re_GPOPS_P12 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P12))^2 + (ae1*
619
               sin(phi_GPOPS_P12))^2);
620
           %States and Costates from pahse 2 (second maneuver)
621
            r_GPOPS_P22 = solution2.phase(2).state(:,1);
622
            theta_GPOPS_P22 = solution2.phase(2).state(:,2);
623
            Vr_GPOPS_P22 = solution2.phase(2).state(:,3);
624
            Vt_GPOPS_P22 = solution2.phase(2).state(:,4);
625
            lambda_r_P22 = solution2.phase(2).costate(:,1);
626
            lambda_theta_P22 = solution2.phase(2).costate(:,2);
627
            lambda_Vr_P22 = solution2.phase(2).costate(:,3);
628
            lambda_Vt_P22 = solution2.phase(2).costate(:,4);
629
            tvec_P22 = solution2.phase(2).time;
630
631
            thetadot_GPOPS_P22 = Vt_GPOPS_P22./r_GPOPS_P22;
632
            T_GPOPS_P22 = solution2.phase(2).control(:,1);
633
```

```
287
```

```
Beta_GPOPS_P22 = solution2.phase(2).control(:,2);
634
635
            ind2_large = find(Beta_GPOPS_P22 > 2*pi);
636
            ind2_small = find(Beta_GPOPS_P22 < 0);</pre>
637
638
            while isempty(ind2_large) == 0
639
                Beta_GPOPS_P22(ind2_large) = Beta_GPOPS_P22(ind2_large) - 2*
640
                    pi;
                ind2_large = find(Beta_GPOPS_P22 > 2*pi);
641
642
            end
643
            while isempty(ind2_small) == 0
644
                Beta_GPOPS_P22(ind2_small) = Beta_GPOPS_P22(ind2_small) + 2*
645
                    pi;
                ind2_small = find(Beta_GPOPS_P22 < 0);</pre>
646
            end
647
            phi_GPOPS_P22 = solution2.parameter(2);
648
            re_GPOPS_P22 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P22))^2 + (ae1*
649
                sin(phi_GPOPS_P22))^2);
650
            Cost2 = (solution2.phase(1).integral + solution2.phase(2).
651
                integral)*DU/TU*1000;
652
653
            %%
654
            %
655
            % Determine entry condition for second maneuver
656
            rt = [r_GPOPS_P12(end)*cos(theta_GPOPS_P12(end));r_GPOPS_P12(end
657
               )*sin(theta_GPOPS_P12(end));0];
```

659	%apogee and perigee constraints
660	<pre>Vf_mag = sqrt(Vr_GPOPS_P12(end)^2 + Vt_GPOPS_P12(end)^2);</pre>
661	<pre>fpa = atan(Vr_GPOPS_P12(end)/Vt_GPOPS_P12(end));</pre>
662	
663	%perifocal velocity
664	<pre>vt = Vf_mag*[-sin(theta_GPOPS_P12(end)-fpa); cos(theta_GPOPS_P12(</pre>
	end)-fpa);0];
665	
666	<pre>[rt_ijk_P12,vt_ijk_P12] = PQW_to_IJK(rt,vt,inc,RAAN,w);</pre>
667	<pre>rt_ijk_P12 = rt_ijk_P12*DU;</pre>
668	<pre>vt_ijk_P12 = vt_ijk_P12*DU/TU;</pre>
669	
670	<pre>[r2,v2,t2] = zone_entry_exit2(rt_ijk_P12,vt_ijk_P12,GMST0+</pre>
	<pre>OmegaEarth*tvec_P12(end)*TU,0,latlim,longlim);</pre>
671	
672	<pre>[rf_pqw2,vf_pqw2] = IJK_to_PQW(r2,v2,inc,RAAN,w);</pre>
673	
674	<pre>rf_pqw2 = rf_pqw2/DU;</pre>
675	vf_pqw2 = vf_pqw2/DU*TU;
676	
677	<pre>vunit2 = vf_pqw2/norm(vf_pqw2);</pre>
678	hfp2 = cross(rf_pqw2,vf_pqw2);
679	<pre>hunit2 = hfp2/norm(hfp2);</pre>
680	
681	<pre>gunit2 = cross(vunit2, hunit2);</pre>
682	%
683	
684	
685	% for plotting purposes in PQW frame

686	%
687	<pre>term12 = (be1*cos(phi_GPOPS_P22))^2 + (ae1*sin(phi_GPOPS_P22))</pre>
	^2;
688	<pre>re2 = ae1*be1/sqrt(term12);</pre>
689	
690	rt2 = rf_pqw2 + re2*cos(phi_GPOPS_P22)*vunit2 + re2*sin(
	<pre>phi_GPOPS_P22)*gunit2;</pre>
691	
692	<pre>for aa = 1:length(ang)</pre>
693	r_ell2(:,aa) = rf_pqw2 + re(aa)*cos(ang(aa))*vunit2 + re(aa)
	<pre>*sin(ang(aa))*gunit2;</pre>
694	end
695	
696	
697	%First maneuver inertial position and velocity
698	<pre>for dd = 1:length(r_GPOPS_P12)</pre>
699	%perifocal position vector
700	<pre>rg_pqw = DU*[r_GPOPS_P12(dd)*cos(theta_GPOPS_P12(dd));</pre>
	<pre>r_GPOPS_P12(dd)*sin(theta_GPOPS_P12(dd));0];</pre>
701	%velocity magnitude
702	<pre>Vf_mag = sqrt(Vr_GPOPS_P12(dd)^2 + Vt_GPOPS_P12(dd)^2);</pre>
703	<pre>fpa = atan(Vr_GPOPS_P12(dd)/Vt_GPOPS_P12(dd));</pre>
704	
705	%perifocal velocity
706	<pre>vg_pqw = DU/TU*Vf_mag*[-sin(theta_GPOPS_P12(dd)-fpa);cos(</pre>
	<pre>theta_GPOPS_P12(dd)-fpa);0];</pre>
707	
708	<pre>[rgi(dd,:),vgi(dd,:)] = PQW_to_IJK(rg_pqw,vg_pqw,inc,RAAN,w)</pre>
	;
709	end

710	
711	% actual arrival in exclusion zone location at tf1
712	<pre>[lat_act_enter,long_act_enter] = IJK_to_LATLONG(rgi(end,1),rgi(</pre>
	<pre>end,2),rgi(end,3),GMST0+OmegaEarth*tf1,0);</pre>
713	<pre>figure(1)</pre>
714	<pre>plot(long_act_enter*180/pi,lat_act_enter*180/pi,'b0')</pre>
715	
716	%expected arrival condition in exclusion zone at tf2
717	<pre>[rf2exp,vf2exp,tf2exp,lat_enter2exp,long_enter2exp] =</pre>
	<pre>zone_entry_exit2(rgi(end,:),vgi(end,:),GMST0+OmegaEarth*(tf1</pre>
),t0,latlim,longlim);
718	
719	<pre>plot(long_enter2exp*180/pi,lat_enter2exp*180/pi,'r0')</pre>
720	
721	%Second maneuver inertial position and velocity2
722	<pre>for dd = 1:length(r_GPOPS_P22)</pre>
723	%perifocal position vector
724	<pre>rg_pqw2 = DU*[r_GPOPS_P22(dd)*cos(theta_GPOPS_P22(dd));</pre>
	r_GPOPS_P22(dd)*sin(theta_GPOPS_P22(dd));0];
725	
726	%velocity magnitude and flight path angle
727	<pre>Vf_mag = sqrt(Vr_GPOPS_P22(dd)^2 + Vt_GPOPS_P22(dd)^2);</pre>
728	<pre>fpa = atan(Vr_GPOPS_P22(dd)/Vt_GPOPS_P22(dd));</pre>
729	
730	%perifocal velocity
731	<pre>vg_pqw2 = DU/TU*Vf_mag*[-sin(theta_GPOPS_P22(dd)-fpa);cos(</pre>
	<pre>theta_GPOPS_P22(dd)-fpa);0];</pre>
732	
733	<pre>[rgi2(dd,:),vgi2(dd,:)] = PQW_to_IJK(rg_pqw2,vg_pqw2,inc,</pre>
	RAAN,w);
734	
735	end

737	% actual arrival in exclusion zone location at tf2
738	<pre>[lat_act_enter2,long_act_enter2] = IJK_to_LATLONG(rgi2(end,1),</pre>
	<pre>rgi2(end,2),rgi2(end,3),GMST0+OmegaEarth*(tf1+tf2exp),0);</pre>
739	<pre>figure(1)</pre>
740	<pre>plot(long_act_enter2*180/pi,lat_act_enter2*180/pi,'b0')</pre>
741	%
742	<pre>% [a2,e2,i2,02,o2,nu2]= RV2COE(rgi2(end,:),vgi2(end,:));</pre>
743	
744	
745	
746	%save optimal path in structure
747	<pre>optans2.ics = struct('r0',r0vec,'v0',v0vec,'t0',t0,'ae',ae,'be',</pre>
	<pre>be,'Rmax',Rmax,'Rmin',Rmin,'latlim',latlim,'longlim',longlim</pre>
	,'GMST0',GMST0,
748	<pre>'inc',inc,'RAAN',RAAN,'w',w,'ang',ang);</pre>
749	<pre>optans2.scale = struct('TU',TU,'DU',DU,'MU',MU2);</pre>
750	<pre>optans2.entry(1) = struct('lat_enter',lat_enter,'long_enter',</pre>
	<pre>long_enter,'r_ell',r_ell,'rtijk',rgi(end,:),'vtijk',vgi(end</pre>
	<pre>,:),'rt_pqw',rg_pqw,'rf_pqw',rf_pqw,</pre>
751	<pre>'lat_act_enter',lat_act_enter,'long_act_enter',</pre>
	<pre>long_act_enter,'rf1',rf1,'vf1',vf1,'tf1',tf1);</pre>
752	<pre>optans2.entry(2) = struct('lat_enter',lat_enter2exp,'long_enter'</pre>
	<pre>,long_enter2exp,'r_ell',r_ell2,'rtijk',rgi2(end,:),'vtijk',</pre>
	<pre>vgi2(end,:),'rt_pqw',rg_pqw2,'rf_pqw',rf_pqw2,</pre>
753	<pre>'lat_act_enter',lat_act_enter2,'long_act_enter',</pre>
	<pre>long_act_enter2,'rf1',rf2exp,'vf1',vf2exp,'tf1',tf2exp);</pre>
754	<pre>optans2.phase(1) = struct('state',solution2.phase(1).state,'</pre>
	<pre>costate',solution2.phase(1).costate,'control',solution2.</pre>
	<pre>phase(1).control,'time',tvec_P12,'rgi',rgi);</pre>

755	<pre>optans2.phase(2) = struct('state',solution2.phase(2).state,'</pre>
	<pre>costate',solution2.phase(2).costate,'control',solution2.</pre>
	<pre>phase(2).control,'time',tvec_P22,'rgi',rgi2);</pre>
756	optans2.parameter = solution2.parameter;
757	
758	<pre>r0string = num2str(norm(r0vec));</pre>
759	<pre>aestr = num2str(ae);</pre>
760	<pre>itstr = num2str(PS0_data(cc,end));</pre>
761	<pre>tempstr = [aestr itstr];</pre>
762	<pre>aestring = num2str(tempstr);</pre>
763	
764	<pre>dir = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\</pre>
	Double Pass\Images\';
765	<pre>dir2 = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\</pre>
	Double Pass\Data\';
766	
767	<pre>tend = toc(tstart);</pre>
768	
769	<pre>exflag = output.result.nlpinfo;</pre>
770	
771	<pre>fid2 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust</pre>
	<pre>RTM\Double Pass\Data\PS02GP0PSDoublePassDataB.txt','a');</pre>
772	
773	<pre>fprintf(fid2,'%i\t %i\t %4.3f\t %4.3f\t %4.3f\t %4.3f\t %4.3f\t%6.5f\t</pre>
	%6.5f\t %6.5f\t %6.2f\t %i\r\n',
774	<pre>norm(r0), ae, phi, phi_GPOPS_P12, phi2, phi_GPOPS_P22, PS0_data(cc</pre>
	,13),Cost,Cost2,tend,exflag);
775	
776	<pre>fid3 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust</pre>
	<pre>RTM\Double Pass\Data\DoublePassCostB.txt','a');</pre>
777	<pre>fprintf(fid3,'%i\t %i \t %4.3f\t %4.3f\t %4.3f\t %i\t\r\n',</pre>

```
norm(r0),ae,solution2.phase(1).integral*DU/TU*1000,solution2
778
                     .phase(2).integral*DU/TU*1000,Cost2,exflag);
779
780
781
            if exflag == 0
782
            %plot optimal results
783
            [optout] = LT_DOUBLE_PASS_PLOTS(optans2,r0string,aestring,dir);
784
785
            dataname = ['DoublePass' r0string aestring];
786
787
            %save data
788
            save(strcat(dir2,[dataname]),'output');
789
            end
790
791
            clear optans
792
            close all
793
        end
794
795 end
```



```
function phaseout = LT_2RTM_Continuous(input)
%
% Phase 1
%
s s1 = input.phase(1).state;
u1 = input.phase(1).control;
%
% Equations of Motion
9 %
```

```
10 r1 = s1(:,1);
vr1 = s1(:,3);
12 vtheta1 = s1(:,4);
13
14 T1 = u1(:, 1);
15 B1 = u1(:,2);
16
17 MU2 = input.auxdata.MU;
18
19 r_dot1 = vr1;
20 theta_dot1 = vtheta1./r1;
21 vr_dot1 = (vtheta1.^2)./r1 - MU2./(r1.^2) + T1.*sin(B1);
22 vtheta_dot1 = -vtheta1.*vr1./r1 + T1.*cos(B1);
23
24 % Form matrix output
25 daeout1 = [r_dot1 theta_dot1 vr_dot1 vtheta_dot1];
26
27 phaseout(1).dynamics = daeout1;
28 %
     _____
```

```
29 % Cost Function
30 phaseout(1).integrand = T1;
31
32 %% Phase 2
33
34 s2 = input.phase(2).state;
35 u2 = input.phase(2).control;
36
37 % Equations of Motion
```

38 %

```
r2 = s2(:,1);
40 vr2 = s2(:,3);
41 vtheta2 = s2(:,4);
42
43 T2 = u2(:,1);
44 B2 = u2(:,2);
45
_{46} r_dot2 = vr2;
47 theta_dot2 = vtheta2./r2;
48 vr_dot2 = (vtheta2.^2)./r2 - MU2./(r2.^2) + T2.*sin(B2);
49 vtheta_dot2 = -vtheta2.*vr2./r2 + T2.*cos(B2);
50
51 % Form matrix output
52 daeout2 = [r_dot2 theta_dot2 vr_dot2 vtheta_dot2];
53
54 phaseout(2).dynamics = daeout2;
55 %
          _____
```

```
56 % Cost Function
57 phaseout(2).integrand = T2;
```

E.2.2.3 Double Pass LTRTM Constraints

```
1 function output = LT_2RTM_Endpoint(input)
2
3
4 %% Cost Function Evaluation
```

```
_____
       _____
6 J = input.phase(1).integral + input.phase(2).integral;
7 output.objective = J;
8 %
              _____
9 %% Event Constraints
10
11 %% Phase 1 (First Maneuver)
12 %phase 2 variables
13 tf1 = input.phase(1).finaltime;
14 xf1 = input.phase(1).finalstate;
15 p = input.parameter;
16 phi = p(1);
17
18 %phase 2 variables
19 t02 = input.phase(2).initialtime;
20 tf2 = input.phase(2).finaltime;
x02 = input.phase(2).initialstate;
xf2 = input.phase(2).finalstate;
_{23} phi2 = p(2);
24
25 \text{ rf} = xf1(1);
_{26} thetaf = xf1(2);
27 Vrf = xf1(3);
_{28} Vtf = xf1(4);
29
30 ae1 = input.auxdata.ae; %semimajor axis of exclusion ellipse
31 be1 = input.auxdata.be; % semiminor axis of exclusion ellipse
32 MU2 = input.auxdata.MU; %gravitational parameter scaled by DU and TU
```

5 **%**

```
33 rf_pqw = input.auxdata.rf_pqw; % perifocal position vector of initial
      corssing into exclusion zone
34 vunit = input.auxdata.vunit; %perifocal unit velocity vector of initial
      crossing into exclusion zone
35 gunit = input.auxdata.gunit; %perifocal unit vetcor of initial crossing
      into exclusion zone
36
37 term1 = (be1*cos(phi))^2 + (ae1*sin(phi))^2;
38
39 re = ae1*be1/sqrt(term1);
40
41 rt = rf_pqw + re*cos(phi)*vunit + re*sin(phi)*gunit;
42
43 %final position constraints
44 event1 = rf^*cos(thetaf) - rt(1);
45 event2 = rf*sin(thetaf) - rt(2);
46
47 %velocity magnitude and flight path angle
48 Vf_mag = sqrt(Vrf<sup>2</sup> + Vtf<sup>2</sup>);
49 fpa = atan(Vrf/Vtf);
50
51 %perifocal velocity
s2 vt = Vf_mag*[-sin(thetaf-fpa); cos(thetaf-fpa); 0];
53
54 [a,ecc,~,~,~,~] = RV2COE_MU(rt,vt,MU2);
55 Ra = a*(1+ecc);
56 Rp = a^{(1-ecc)};
57
58 event3 = Ra;
59 event4 = Rp;
60
61 % Linkage Constraints
```

```
62 event1_link_state = x02 - xf1;
63 event1_link_time = t02 - tf1;
64
65 output.eventgroup(1).event = [event1_link_state event1_link_time event1
      event2 event3 event4];
66
67 %% Phase 2 (Second Maneuver)
68
69 % constant variables
70 inc = input.auxdata.inc; %inclination of initial orbit (used to convert
      everything into perifocal frame of initial orbit)
71 RAAN = input.auxdata.RAAN; %RAAN of initial orbit (used to convert
      everything into perifocal frame of initial orbit)
72 w = input.auxdata.w; %argument of perigee of initial orbit (used to
      convert everything into perifocal frame of initial orbit)
73 latlim = input.auxdata.latlim;
74 longlim = input.auxdata.longlim;
75 GMSTO = input.auxdata.GMSTO;
76 OmegaEarth = input.auxdata.OmegaEarth;
77 DU = input.auxdata.DU;
78 TU = input.auxdata.TU;
79
80
rf2 = xf2(1);
x_{2} thetaf2 = xf2(2);
83 Vrf2 = xf2(3);
84 Vtf2 = xf2(4);
85
86 %position and velocity of initial intercept in perifocal frame of
      initial
87 %orbit
```

```
ss if isnan(rf) == 1 || isnan(thetaf) == 1 || isnan(Vf_mag) == 1 || isnan(
      tf1) == 1 || isnan(phi) == 1
          event21 = NaN;
89
          event22 = NaN;
90
          event25 = NaN;
91
          event23 = NaN;
92
          event24 = NaN;
93
94 else
95 [rt_ijk,vt_ijk] = PQW_to_IJK(rt,vt,inc,RAAN,w);
96 rt_ijk = rt_ijk*DU;
97 vt_ijk = vt_ijk*DU/TU;
98
  [r2,v2,t2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0+OmegaEarth*tf1*TU,0,
99
      latlim,longlim);
100
   [rf_pqw2,vf_pqw2] = IJK_to_PQW(r2,v2,inc,RAAN,w);
101
102
  rf_pqw2 = rf_pqw2/DU;
103
   vf_pqw2 = vf_pqw2/DU*TU;
104
105
  vunit2 = vf_pqw2/norm(vf_pqw2);
106
   hfp2 = cross(rf_pqw2,vf_pqw2);
107
   hunit2 = hfp2/norm(hfp2);
108
109
   gunit2 = cross(vunit2, hunit2);
110
111
   term12 = (be1*cos(phi2))^2 + (ae1*sin(phi2))^2;
112
   re2 = ae1*be1/sqrt(term12);
113
114
  rt2 = rf_pqw2 + re2*cos(phi2)*vunit2 + re2*sin(phi2)*gunit2;
115
116
117 %final position constraints
```
```
event21 = rf2*cos(thetaf2) - rt2(1);
118
   event22 = rf2*sin(thetaf2) - rt2(2);
119
120
   %apogee and perigee constraints
121
   Vf_mag2 = sqrt(Vrf2^2 + Vtf2^2);
122
   fpa2 = atan(Vrf2/Vtf2);
123
124
  %perifocal velocity
125
   vt2 = Vf_mag2*[-sin(thetaf2-fpa2); cos(thetaf2-fpa2); 0];
126
127
   [a2,ecc2,~,~,~,~] = RV2COE_MU(rt2,vt2,MU2);
128
  Ra2 = a2*(1+ecc2);
129
  Rp2 = a2*(1-ecc2);
130
131
  event23 = Ra2;
132
   event24 = Rp2;
133
134
   event25 = tf2 - (tf1 + t2/TU);
135
   end
136
137
  output.eventgroup(2).event = [event21 event22 event25 event23 event24];
138
```

E.3 Triple Pass LTRTMs

E.3.1 Particle Swarm Algorithms

E.3.1.1 Triple Pass LTRTM PSO Driver

```
1 wgs84data
2 global MU
3 OmegaEarth = 0.000072921151467;
4
5 for bb = 10:10
6
7 t0 = 0;
```

```
GMST0 = 0;
8
       latlim = [-10 10]*pi/180;
9
       longlim = [-50 -10]*pi/180;
10
11
       r0vec = [7300;0;0];
12
       v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];
13
14
       [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
15
       period = 2*pi*sqrt(a^3/MU);
16
17
       aevec = [150 140 130 120 110 100 90 80 70 60 50];
18
       bevec = [15 14 13 12 11 10 9 8 7 6 5];
19
       Rmaxvec = norm(r0vec) + 50;
20
       Rminvec = norm(r0vec) - 50;
21
22
       DU = norm(r0vec);
23
       TU = period/(2*pi);
24
       MU2 = MU*TU^2/DU^3;
25
26
       m0 = 1000;
27
       r0 = r0vec;
28
       v0 = v0vec;
29
       Rmax = Rmaxvec;
30
       Rmin = Rminvec;
31
32
       %Energy of most elliptical orbit
33
       ab = (Rmax + Rmin)/2; %semi-major axis of orbit
34
       Eb = -MU/(2*ab); %energy of orbit
35
       Vmax = sqrt(2*(MU/Rmin + Eb));
36
       Vmin = sqrt(2*(MU/Rmax + Eb));
37
38
39
       fid = fopen([dir 'PSODoublePassDataFinal_06012014.txt'],'a');
```

```
state0=[r0 v0];
40
41
      Tmax = 2e-3;
42
      swarm = 40;
43
      iter = 1000;
44
      prec = [5;5;5;9];
45
46
      if bb == 1
47
48
          fprintf(fid,'%s %i\r\n','r0 (km) =',r0vec(1));
49
          fprintf(fid,'%s %i\r\n','swarm =',swarm);
50
51
          52
             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','
53
             ');
      end
54
55
      if bb == 10
56
          endval = 1;
57
      else
58
          endval = 20;
59
      end
60
61
      ae = aevec(bb);
62
      be = bevec(bb);
63
64
      [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
65
         long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim);
```

```
303
```

```
66
67
      for aa = 1:endval
68
          tstart = tic;
69
70
          [JGmin, Jpbest, gbest, x, k] = LT_RTM_PSO_TFIXED(3, [0 2*pi; Vmin Vmax
71
              ;-pi/2+0.000001 pi/2-0.000001],iter,swarm,prec,rf1,vf1,tf1,
             ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0);
72
          Cost1 = JGmin*DU/TU*1000
73
74
          tend = toc(tstart)
75
76
77
          tstart2 = tic;
78
          79
             vf1,tf1,gbest(1),ae,be,gbest(2),gbest(3),DU,TU,MU2);
80
          [rf2,vf2,tf2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0+OmegaEarth*
81
             tf1,0,latlim,longlim);
82
          [JGmin2, Jpbest2, gbest2, x2, k2] = LT_RTM_PSO_TFIXED(3, [0 2*pi; Vmin
83
              Vmax;-pi/2+0.000001 pi/2-0.000001],iter,swarm,prec,rf2,vf2,
             tf2,ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0);
84
          Cost2 = JGmin2*DU/TU*1000
85
86
          CostTOT = Cost1 + Cost2
87
88
          tend2 = toc(tstart2)
89
90
91
```

```
304
```

```
fprintf(fid,'%i\t %i \t %10.5f\t %6.5f\t %7.6f\t %6.5f\t %10.5f\
93
               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),
94
                   gbest2(2),gbest2(3),Cost1,Cost2,CostTOT,k,k2,k+k2,tend,
                   tend2,tend+tend2);
           tend + tend2
95
96
           clear tstart JGmin Jpbest gbest x k Cost1 tend tstart2 rt_ijk
97
               vt_ijk rf2 vf2 tf2 JGmin2 Jpbest2 gbest2 x2 k2 Cost2
               CostTOT tend2
       end
98
99
100 end
```

E.3.2 Direct Collocation Algorithms

92

E.3.2.1 Triple Pass LTRTM Driver

```
1 \text{ for } zz = 2:2
       clear guess setup limits output
2
       close all
3
       clc
4
5
       if zz == 1
6
           load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
7
               Triple Pass\Journal Data\data6800_LT_3RTMsort.mat')
           [ind0] = find(data6800_LT_3RTMsort(:,1) ~= 0);
8
           PSO_data = data6800_LT_3RTMsort(ind0,:);
9
           cmax = 67;
10
           cmin = 67;
11
           rmag = 6800;
12
       elseif zz == 2
13
```

```
load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
14
               Triple Pass\Journal Data\data7300_LT_3RTMsort.mat')
           [ind0] = find(data7300_LT_3RTMsort(:,1) ~= 0);
15
           PSO_data = data7300_LT_3RTMsort(ind0,:);
16
           cmax = 27;
17
           cmin = 22;
18
           rmag = 7300;
19
       end
20
21
       for cc = cmin:cmax
22
           fid = fopen('PSO_to_GPOPS_3RTM4.txt','a');
23
           clear guess setup limits output
24
           close all
25
           clc
26
           t0 = 0;
27
           GMST0 = 0;
28
           latlim = [-10 10]*pi/180;
29
           longlim = [-50 -10]*pi/180;
30
31
           wgs84data
32
           global MU2 MU
33
           OmegaEarth = 0.000072921151467;
34
           r0vec = [rmag;0;0];
35
           v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];
36
37
           [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
38
           period = 2*pi*sqrt(a^3/MU);
39
40
           swarm = 30;
41
           iter = 1000;
42
           Rmaxvec = rmag + 50;
43
44
           Rminvec = rmag - 50;
```

```
45
           r0 = r0vec;
46
           v0 = v0vec;
47
           Rmax = Rmaxvec;
48
           Rmin = Rminvec;
49
50
           ae = PSO_data(cc,2);
51
           be = ae/10;
52
53
           TOF = PSO_data(cc,3);
54
           phi = PSO_data(cc,4);
55
           Vt_mag = PSO_data(cc,5);
56
           fpa_t = PSO_data(cc,6);
57
58
           tstart = tic;
59
60
           [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
61
               long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim)
               ;
62
           lat_exp_enter = lat_enter;
63
           long_exp_enter = long_enter;
64
65
           DU = norm(rf1);
66
           TU = period/(2*pi);
67
68
           ae1 = ae/DU;
69
           be1 = be/DU;
70
           r01 = norm(r0)/DU;
71
           MU2 = MU*TU^2/DU^3;
72
           tOmin = 0; % minimum initial time
73
           tOmax = 0; % maximum initial time
74
```

```
307
```

75	tfmin = tf1; % minimum final time
76	tfmax = tf1;
77	$n0 = sqrt(MU2/(norm(r0)/DU)^3);$
78	
79	%% First Maneuver
80	[LT_DV,maxT,r,gamma,T_a,thetaf_int,theta_dot,theta_ddot,rdot,
	<pre>Tvec,TOF_calc,rt_ijk,vt_ijk,rt_pqw,vt_pqw,r0_pqw,v0_pqw] =</pre>
	<pre>Single_LT_Maneuver(rf1,vf1,tf1,phi,ae,be,Vt_mag,fpa_t,DU,TU,</pre>
	MU2);
81	
82	<pre>time_mod = Tvec;</pre>
83	
84	<pre>[rf_pqw,vf_pqw] = IJK_to_PQW(rf1,vf1,inc,RAAN,w);</pre>
85	<pre>rf_pqw = rf_pqw/DU;</pre>
86	$vf_pqw = vf_pqw/DU*TU;$
87	
88	
89	<pre>vunit = vf_pqw/norm(vf_pqw);</pre>
90	hfp = cross(rf_pqw,vf_pqw);
91	<pre>hunit = hfp/norm(hfp);</pre>
92	
93	<pre>gunit = cross(vunit, hunit);</pre>
94	
95	ang = (0:0.001:2*pi);
96	<pre>re = (ae1*be1)./sqrt((be1*cos(ang)).^2 + (ae1*sin(ang)).^2);</pre>
97	
98	
99	<pre>theta_rf = atan2(rf_pqw(2),rf_pqw(1));</pre>
100	<pre>if theta_rf < 0</pre>
101	<pre>theta_rf = 2*pi + theta_rf;</pre>
102	end
103	<pre>[rtest] = IJK_to_PQW(r0,v0,inc,RAAN,w);</pre>

```
theta0 = atan2(rtest(2),rtest(1));
104
105
            theta_mod = thetaf_int + atan2(r0_pqw(2),r0_pqw(1));
106
107
            time_guess = time_mod;
108
            theta_guess = theta_mod;
109
            r_guess = r;
110
            vr_guess = rdot;
111
            vtheta_guess = r.*theta_dot;
112
113
            T_guess = T_a;
            B_guess = gamma;
114
115
            ind = find(T_guess ~= 0);
116
117
           %inertial position vector of new arrival position
118
            for aa = 1:length(ang)
119
                r_ell(:,aa) = rf_pqw + re(aa)*cos(ang(aa))*vunit + re(aa)*
120
                    sin(ang(aa))*gunit;
            end
121
122
            rgi = zeros(length(r_guess),3);
123
            for dd = 1:length(r_guess)
124
                rg_pqw = DU*[r_guess(dd)*cos(theta_guess(dd));r_guess(dd)*
125
                    sin(theta_guess(dd));0];
                [rgi(dd,:)] = PQW_to_IJK(rg_pqw,[],inc,RAAN,w);
126
            end
127
128
            for ee = 1:length(ang)
129
                rell_pqw = [r_ell(1,ee)*DU;r_ell(2,ee)*DU;0];
130
                rnom_pqw = norm(r0)*[cos(ang(ee));sin(ang(ee));0];
131
                [rell_ijk(:,ee)] = PQW_to_IJK(rell_pqw,[],inc,RAAN,w);
132
133
                [rnom_ijk(ee,:)] = PQW_to_IJK(rnom_pqw,[],inc,RAAN,w);
```

```
end
134
135
136
           %% determine limits on subsequent passes into exclusion zone
137
           % assume upper limit based on circular orbit with phi = pi/2
138
           % assume lower limit based on circular orbit with phi = 2pi/2
139
           phi_low = 3*pi/2;
140
           phi_upp = pi/2;
141
142
            [rf_upp,vf_upp] = Single_LT_Limits(rf1,vf1,phi_upp,ae,be,DU,TU);
143
144
            [rf2_upp,vf2_upp,tf2_upp] = zone_entry_exit2(rf_upp,vf_upp,GMST0
145
               +OmegaEarth*tf1,0,latlim,longlim);
146
            ang_upp = sqrt(MU/norm(rf_upp)^3)*tf2_upp;
147
148
            thetaf2_max = theta_guess(end) + ang_upp;
149
            tf2_max = (tf1 + tf2_upp)/TU;
150
151
152
            [rf_low,vf_low] = Single_LT_Limits(rf1,vf1,phi_low,ae,be,DU,TU);
153
154
            [rf2_low,vf2_low,tf2_low] = zone_entry_exit2(rf_low,vf_low,GMST0
155
               +OmegaEarth*tf1,0,latlim,longlim);
156
            ang_low = sqrt(MU/norm(rf_low)^3)*tf2_low;
157
158
            thetaf2_min = theta_guess(end) + ang_low;
159
            tf2_min = (tf1 + tf2_low)/TU;
160
161
162
163
```

```
164
           %% Second Maneuver
165
            [rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2,
166
               lat_exit2,long_exit2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0
               +OmegaEarth*tf1,0,latlim,longlim);
167
168
            TOF2 = PSO_data(cc,7);
169
           phi2 = PSO_data(cc,8);
170
171
            Vt_mag2 = PSO_data(cc,9);
            fpa_t2 = PSO_data(cc,10);
172
173
            [LT_DV2,maxT2,r2,gamma2,T_a2,thetaf_int2,theta_dot2,theta_ddot2,
174
               rdot2,Tvec2,TOF_calc2,rt_ijk2,vt_ijk2] = Single_LT_Maneuver(
               rf2,vf2,tf2,phi2,ae,be,Vt_mag2,fpa_t2,DU,TU,MU2);
175
            theta02 = theta_guess(end);
176
177
            delt2 = (tf2 - TOF2)/TU;
178
            time_mod2 = Tvec2 + time_guess(end) + delt2;
179
180
            [rf_pqw2,vf_pqw2] = IJK_to_PQW(rf2,vf2,inc,RAAN,w);
181
           rf_pqw2 = rf_pqw2/DU;
182
           vf_pqw2 = vf_pqw2/DU*TU;
183
184
           vunit2 = vf_pqw2/norm(vf_pqw2);
185
           hfp2 = cross(rf_pqw2,vf_pqw2);
186
           hunit2 = hfp2/norm(hfp2);
187
188
            gunit2 = cross(vunit2, hunit2);
189
190
            [rt_pqw2,vt_pqw2] = IJK_to_PQW(rt_ijk2,vt_ijk2,inc,RAAN,w);
191
```

```
ang_mod2 = atan2(rt_pqw2(2),rt_pqw2(1));
192
            while ang_mod2 < theta_guess(end)</pre>
193
                ang_mod2 = ang_mod2 + 2*pi;
194
            end
195
            ang_mod1 = atan2(rt_pqw(2),rt_pqw(1));
196
197
            diff = ang_mod2 - ang_mod1;
198
            diff2 = thetaf_int2(end) - thetaf_int2(1);
199
            theta_diff = diff - diff2;
200
201
202
           %inertial position vector of new arrival position
203
            for bb = 1:length(ang)
204
                r_ell2(:,bb) = rf_pqw2 + re(bb)*cos(ang(bb))*vunit2 + re(bb)
205
                    *sin(ang(bb))*gunit2;
            end
206
207
            theta_mod2 = thetaf_int2 + theta_diff + theta_guess(end);
208
209
            coast_length = 1;
210
211
            time_guess2 = zeros(coast_length+length(time_mod2),1);
212
            time_guess2(1:coast_length) = time_guess(end);
213
            time_guess2(coast_length+1:end) = time_mod2;
214
            theta_guess2 = zeros(coast_length+length(theta_mod2),1);
215
            theta_guess2(coast_length+1:end) = theta_mod2;
216
            theta_guess2(1:coast_length) = theta02;
217
            r_guess2 = zeros(coast_length+length(theta_mod2),1);
218
            r_guess2(1:coast_length) = r_guess(end);
219
            r_guess2(coast_length+1:end) = r2;
220
            vr_guess2 = zeros(coast_length+length(theta_mod2),1);
221
222
            vr_guess2(1:coast_length) = vr_guess(end);
```

```
vr_guess2(coast_length+1:end) = rdot2;
223
            vtheta_guess2 = zeros(coast_length+length(theta_mod2),1);
224
            vtheta_guess2(1:coast_length) = vtheta_guess(end);
225
            vtheta_guess2(coast_length+1:end) = r2.*theta_dot2;
226
            T_guess2 = zeros(coast_length+length(time_mod2),1);
227
            T_guess2(1:coast_length) = 0;
228
            T_guess2(coast_length+1:end) = T_a2;
229
            B_guess2 = zeros(coast_length+length(time_mod2),1);
230
            B_guess2(1:coast_length) = B_guess(end);
231
            B_guess2(coast_length+1:end) = gamma2;
232
233
            ind2 = find(T_guess2 = 0);
234
235
            nom_orb2_time = [(0:1:tf2) tf2];
236
237
            [at,et,it,Ot,ot,nut] = RV2COE(rt_ijk,vt_ijk);
238
239
            for ee = 1:length(nom_orb2_time)
240
                [nutf] = nuf_from_TOF(nut,nom_orb2_time(ee),at,et);
241
                [Rdum(:,ee),Vdum] = COE2RV(at,et,it,Ot,ot,nutf);
242
                [nom_orb2_R] = IJK_to_PQW(Rdum(:,ee),Vdum,inc,RAAN,w);
243
244
                ROrb2_PQW(ee,:) = nom_orb2_R;
245
246
            end
247
            while nutf < nut</pre>
248
                nutf = nutf + 2*pi;
249
            end
250
251
            %Angle of expected 2nd pass entry location into exclusion zone
252
            thetaf2 = (nutf - nut) + 0;
253
254
```

```
%% Coasting phase
255
           %modify angle to match scenario angle
256
           %1) determine coes of post maneuver 2 orbit at t = time_guess2(
257
               end)
            [at2,et2,it2,Ot2,ot2,nut2]= RV2COE(rt_ijk2,vt_ijk2);
258
259
            tcoast3 = [(time_guess2(end)+.001:.1:time_guess2(end)+(t_exit2-
260
               tf2)/TU)';time_guess2(end)+(t_exit2-tf2)/TU];
            coast_length = length(tcoast3);
261
            time_guess3 = tcoast3;
262
            r_guess3 = zeros(coast_length,1);
263
            theta_guess3 = zeros(coast_length,1);
264
            vr_guess3 = zeros(coast_length,1);
265
            vtheta_guess3 = zeros(coast_length,1);
266
            T_guess3 = zeros(coast_length,1);
267
            Beta_guess3 = zeros(coast_length,1);
268
269
            for yy = 1:coast_length
270
                if yy == 1
271
                    tprev = time_guess2(end);
272
                    angprev = theta_guess2(end);
273
274
                    nu_prev = nut2;
                end
275
                %2) determine length of time step in seconds
276
                tstep = (tcoast3(yy)-tprev)*TU;
277
                %3) current time becomes previous time
278
                tprev = tcoast3(yy);
279
                %4) determine angle traveled during tstep
280
                angnew = nuf_from_TOF(nu_prev,tstep,at2,et2);
281
282
                if angnew < nu_prev</pre>
283
284
                    angtemp = angnew+2*pi;
```

```
else
285
                     angtemp = angnew;
286
                end
287
                ang_diff = angtemp - nu_prev;
288
                theta_guess3(yy) = angprev+ang_diff;
289
                angprev = theta_guess3(yy);
290
                nu_prev = angnew;
291
                %5) determine position and velocity in IJK
292
                [r3ijk,v3ijk]=COE2RV(at2,et2,it2,Ot2,ot2,angnew);
293
                %6) convert position and velocity to perifocal frame of
294
                    initial
                %orbit
295
                [r3pqw,v3pqw] = IJK_to_PQW(r3ijk,v3ijk,inc,RAAN,w);
296
297
                r_guess3(yy) = norm(r3pqw)/DU;
298
                % 7) Vr and Vtheta
299
300
                vr_guess3(yy) = (MU/at2*(1-et2^2))*et2*sin(angnew)/DU*TU;
301
                vtheta_guess3(yy) = sqrt(MU/at2*(1-et2^2))*(1+et2*cos(angnew
302
                    ))/DU*TU;
                T_guess(yy) = 0;
303
                Beta_guess(yy) = 0;
304
305
                if yy == coast_length
306
                     tend3 = tcoast3(yy)*TU;
307
                     rend3 = r3ijk;
308
                     vend3 = v3ijk;
309
                     theta3end = theta_guess3(yy);
310
                end
311
312
            end
313
314
```

315	%% Third Maneuver
316	<pre>[rf4,vf4,tf4,lat_enter3,long_enter3,R_exit3,V_exit3,t_exit3,</pre>
	<pre>lat_exit3,long_exit3] = zone_entry_exit2(rend3,vend3,GMST0+</pre>
	<pre>OmegaEarth*(tf1+t_exit2),0,latlim,longlim);</pre>
317	<pre>if tf4 < period+500</pre>
318	<pre>TOF3 = PS0_data(cc,11);</pre>
319	<pre>phi3 = PSO_data(cc,12);</pre>
320	<pre>Vt_mag3 = PSO_data(cc,13);</pre>
321	<pre>fpa_t3 = PSO_data(cc,14);</pre>
322	
323	[LT_DV3,maxT3,r3,gamma3,T_a3,thetaf_int3,theta_dot3,
	<pre>theta_ddot3,rdot3,Tvec3,TOF_calc3,rt_ijk3,vt_ijk3] =</pre>
	Single_LT_Maneuver(rf4,vf4,tf4,phi3,ae,be,Vt_mag3,fpa_t3
	,DU,TU,MU2);
324	
325	<pre>theta04 = theta_guess3(end);</pre>
326	
327	<pre>delt4 = (tf4 - TOF3)/TU;</pre>
328	<pre>time_mod4 = Tvec3 + time_guess3(end) + delt4;</pre>
329	
330	<pre>[rf_pqw4,vf_pqw4] = IJK_to_PQW(rf4,vf4,inc,RAAN,w);</pre>
331	$rf_pqw4 = rf_pqw4/DU;$
332	$vf_pqw4 = vf_pqw4/DU*TU;$
333	
334	<pre>vunit4 = vf_pqw4/norm(vf_pqw4);</pre>
335	hfp4 = cross(rf_pqw4,vf_pqw4);
336	<pre>hunit4 = hfp4/norm(hfp4);</pre>
337	
338	<pre>gunit4 = cross(vunit4, hunit4);</pre>
339	
340	<pre>[rt_pqw4,vt_pqw4] = IJK_to_PQW(rt_ijk3,vt_ijk3,inc,RAAN,w);</pre>
341	<pre>ang_mod4 = atan2(rt_pqw4(2),rt_pqw4(1));</pre>

```
while ang_mod4 < 0</pre>
342
                     ang_mod4 = ang_mod4 + 2*pi;
343
                end
344
345
346
                diff31 = ang_mod4 - ang_mod2;
347
                diff32 = thetaf_int3(end) - thetaf_int3(1);
348
                theta_diff3 = diff31 - diff32;
349
350
                %inertial position vector of new arrival position
351
                for zz = 1:length(ang)
352
                     r_ell4(:,zz) = rf_pqw4 + re(zz)*cos(ang(zz))*vunit4 + re
353
                        (zz)*sin(ang(zz))*gunit4;
                end
354
355
                theta_mod4 = thetaf_int3 + theta_diff3 + theta_guess2(end);
356
                while theta_mod4(1) < theta_guess3(end)</pre>
357
                     theta_mod4 = theta_mod4 + 2*pi;
358
                end
359
360
                if theta_mod4(2) - theta04 > 0.1
361
                     theta_mod4 = theta_mod4 - 2*pi;
362
                end
363
364
                coast_length = 1;
365
366
                time_guess4 = zeros(coast_length+length(time_mod4),1);
367
                time_guess4(1:coast_length) = time_guess3(end);
368
                time_guess4(coast_length+1:end) = time_mod4;
369
                theta_guess4 = zeros(coast_length+length(theta_mod4),1);
370
                theta_guess4(coast_length+1:end) = theta_mod4;
371
372
                theta_guess4(1:coast_length) = theta04;
```

```
317
```

373	r_guess4 = zeros(coast_length+length(theta_mod4),1);
374	<pre>r_guess4(1:coast_length) = r_guess3(end);</pre>
375	<pre>r_guess4(coast_length+1:end) = r3;</pre>
376	<pre>vr_guess4 = zeros(coast_length+length(theta_mod4),1);</pre>
377	<pre>vr_guess4(1:coast_length) = vr_guess3(end);</pre>
378	<pre>vr_guess4(coast_length+1:end) = rdot3;</pre>
379	<pre>vtheta_guess4 = zeros(coast_length+length(theta_mod4),1);</pre>
380	<pre>vtheta_guess4(1:coast_length) = vtheta_guess3(end);</pre>
381	<pre>vtheta_guess4(coast_length+1:end) = r3.*theta_dot3;</pre>
382	<pre>T_guess4 = zeros(coast_length+length(time_mod4),1);</pre>
383	<pre>T_guess4(1:coast_length) = 0;</pre>
384	T_guess4(coast_length+1:end) = T_a3;
385	<pre>B_guess4 = zeros(coast_length+length(time_mod4),1);</pre>
386	<pre>B_guess4(1:coast_length) = 0;</pre>
387	<pre>B_guess4(coast_length+1:end) = gamma3;</pre>
388	
389	nom_orb3_time = [(0:1:tf4) tf4];
390	
391	<pre>[at2,et2,it2,0t2,ot2,nut2]= RV2COE(rt_ijk2,vt_ijk2);</pre>
392	
393	<pre>for yy = 1:length(nom_orb3_time)</pre>
394	<pre>[nutf2] = nuf_from_TOF(nut2,nom_orb3_time(yy),at2,et2);</pre>
395	<pre>[Rdum2(:,yy),Vdum2] = COE2RV(at2,et2,it2,Ot2,ot2,nutf2);</pre>
396	<pre>[nom_orb3_R] = IJK_to_PQW(Rdum2(:,yy),Vdum2,inc,RAAN,w);</pre>
397	
398	<pre>ROrb3_PQW(ee,:) = nom_orb3_R;</pre>
399	
400	end
401	else
402	TOF3 = period;
403	<pre>phi3 = PSO_data(cc,12);</pre>
404	<pre>Vt_mag3 = PS0_data(cc,13);</pre>

405	<pre>fpa_t3 = PS0_data(cc,14);</pre>
406	
407	[LT_DV3,maxT3,r3,gamma3,T_a3,thetaf_int3,theta_dot3,
	<pre>theta_ddot3,rdot3,Tvec3,TOF_calc3,rt_ijk3,vt_ijk3] =</pre>
	<pre>Single_LT_Maneuver(rf4,vf4,TOF3,phi3,ae,be,Vt_mag3,</pre>
	<pre>fpa_t3,DU,TU,MU2);</pre>
408	
409	
410	
411	<pre>tcoast4 = (time_guess3(end)+.001:.1:time_guess3(end)+(tf4-</pre>
	<pre>period)/TU);</pre>
412	<pre>time_mod4 = time_guess3(end)+(tf4-period)/TU + Tvec3;</pre>
413	<pre>coast_length4 = length(tcoast4);</pre>
414	
415	%modify time to match scenario time
416	<pre>time_guess4 = zeros(coast_length4+length(time_mod4),1);</pre>
417	<pre>time_guess4(1:coast_length4) = tcoast4;</pre>
418	<pre>time_guess4(coast_length4+1:end) = time_mod4;</pre>
419	
420	<pre>r_guess4 = zeros(length(time_guess4),1);</pre>
421	<pre>theta_guess4 = zeros(length(time_guess4),1);</pre>
422	<pre>vr_guess4 = zeros(length(time_guess4),1);</pre>
423	<pre>vtheta_guess4 = zeros(length(time_guess4),1);</pre>
424	<pre>T_guess4 = zeros(length(time_guess4),1);</pre>
425	<pre>Beta_guess4 = zeros(length(time_guess4),1);</pre>
426	
427	%modify angle to match scenario angle
428	%1) determine coes of post maneuver 2 orbit at t =
	time_guess2(end)
429	<pre>[at2,et2,it2,Ot2,ot2,nut2]= RV2COE(rend3,vend3);</pre>
430	
431	<pre>for yy = 1:coast_length4</pre>

432	if $yy == 1$
433	<pre>tprev = time_guess3(end);</pre>
434	<pre>angprev = theta_guess3(end);</pre>
435	<pre>nu_prev = nut2;</pre>
436	end
437	%2) determine length of time step in seconds
438	<pre>tstep = (tcoast4(yy)-tprev)*TU;</pre>
439	%3) current time becomes previous time
440	<pre>tprev = tcoast4(yy);</pre>
441	%4) determine angle traveled during tstep
442	<pre>angnew = nuf_from_TOF(nu_prev,tstep,at2,et2);</pre>
443	
444	<pre>if angnew < nu_prev</pre>
445	<pre>angtemp = angnew+2*pi;</pre>
446	else
447	<pre>angtemp = angnew;</pre>
448	end
449	<pre>ang_diff = angtemp - nu_prev;</pre>
450	<pre>theta_guess4(yy) = angprev+ang_diff;</pre>
451	<pre>angprev = theta_guess4(yy);</pre>
452	<pre>nu_prev = angnew;</pre>
453	%5) determine position and velocity in IJK
454	<pre>[r3ijk,v3ijk]=COE2RV(at2,et2,it2,Ot2,ot2,angnew);</pre>
455	%6) convert position and velocity to perifocal frame of
	initial
456	%orbit
457	<pre>[r3pqw,v3pqw] = IJK_to_PQW(r3ijk,v3ijk,inc,RAAN,w);</pre>
458	
459	r_guess4(yy) = norm(r3pqw)/DU;
460	% 7) Vr and Vtheta
461	

vr_guess4(yy) = (MU/at2*(1-et2^2))*et2*sin(angnew)/DU*TU 462 ; vtheta_guess4(yy) = sqrt(MU/at2*(1-et2^2))*(1+et2*cos(463 angnew))/DU*TU; 464 465 end 466 467 [rf_pqw4,vf_pqw4] = IJK_to_PQW(rf4,vf4,inc,RAAN,w); 468 469 rf_pqw4 = rf_pqw4/DU; vf_pqw4 = vf_pqw4/DU*TU; 470 471 vunit4 = vf_pqw4/norm(vf_pqw4); 472 hfp4 = cross(rf_pqw4,vf_pqw4); 473 hunit4 = hfp4/norm(hfp4); 474 475 gunit4 = cross(vunit4, hunit4); 476 [rt_pqw4,vt_pqw4] = IJK_to_PQW(rt_ijk3,vt_ijk3,inc,RAAN,w); 477 478 479 ang_mod4 = atan2(rt_pqw4(2),rt_pqw4(1)); 480 while ang_mod4 < theta_guess4(yy)</pre> 481 ang_mod4 = ang_mod4 + 2*pi; 482 end 483 484 diff31 = ang_mod4 - theta_guess4(yy); 485 diff32 = thetaf_int3(end) - thetaf_int3(1); 486 theta_diff3 = diff31 - diff32; 487 488 489 490 491 %inertial position vector of new arrival position

for zz = 1:length(ang) 492 $r_ell4(:,zz) = rf_pqw4 + re(zz)*cos(ang(zz))*vunit4 + re$ 493 (zz)*sin(ang(zz))*gunit4; end 494 495 theta_mod4 = theta_guess4(yy) + thetaf_int3 + theta_diff3; 496 497 while theta_mod4(1) < theta_guess4(yy)</pre> 498 theta_mod4 = theta_mod4 + 2*pi; 499 500 end 501 502 time_guess4(coast_length4+1:end) = time_mod4; 503 theta_guess4(coast_length4+1:end) = theta_mod4; 504 r_guess4(coast_length4+1:end) = r3; 505 vr_guess4(coast_length4+1:end) = rdot3; 506 vtheta_guess4(coast_length4+1:end) = r3.*theta_dot3; 507 $T_guess4(1:coast_length4) = 0;$ 508 T_guess4(coast_length4+1:end) = T_a3; 509 Beta_guess4(1:coast_length4) = 0; 510 Beta_guess4(coast_length4+1:end) = gamma3; 511 512 end 513 514 515 516 %% GPOPS RUN (1st Run Through assigns a non-zero minimum thrust 517 to help GPOPS-II converge) % variables from PSo phase 518 r1 = 1;519 rf = norm(rt_pqw); 520 521 rmax = r1 + be/DU;

522	<pre>rmin = r1 - be/DU;</pre>
523	<pre>thetaf_min = theta_rf - atan(ae/norm(r0));</pre>
524	<pre>thetaf_max = theta_rf + atan(ae/norm(r0));</pre>
525	
526	% Control and time boundaries
527	umin = -0.5; % minimum control angle
528	<pre>umax = 2*pi+0.5; % maximum control angle</pre>
529	Tmax = 2*0.0001160;
530	Tmin = Tmax/1000;
531	%

% GPOPS Setup 532 % Phase 1 Information 533 iphase = 1;534 bounds.phase(iphase).initialtime.lower = t0min; 535 bounds.phase(iphase).initialtime.upper = t0max; 536 bounds.phase(iphase).finaltime.lower = tf1/TU; 537 bounds.phase(iphase).finaltime.upper = tf1/TU; 538 % LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY 539 bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU 540 0 sqrt(MU2/r1)]; bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU 541 0 sqrt(MU2/r1)]; bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.1 542 0]; bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.1 543 1.1]; bounds.phase(iphase).state.lower = [rmin theta_rf-n0*tf1/TU -0.1 544 0]; bounds.phase(iphase).state.upper = [rmax thetaf_max 0.1 1.1]; 545 546 bounds.phase(iphase).control.lower = [Tmin umin];

547	<pre>bounds.phase(iphase).control.upper = [Tmax umax];</pre>
548	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
549	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
550	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
551	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
552	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
553	<pre>bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 Rmin/DU Rmin/</pre>
	DU]; % None
554	<pre>bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 Rmax/DU Rmax/</pre>
	DU]; % None
555	% GUESS SOLUTION
556	<pre>guess.phase(iphase).time = time_guess;</pre>
557	<pre>guess.phase(iphase).state(:,1) = r_guess;</pre>
558	<pre>guess.phase(iphase).state(:,2) = theta_guess;</pre>
559	<pre>guess.phase(iphase).state(:,3) = vr_guess;</pre>
560	<pre>guess.phase(iphase).state(:,4) = vtheta_guess;</pre>
561	% Control guess :
562	<pre>guess.phase(iphase).control(:,1) = T_guess;</pre>
563	<pre>guess.phase(iphase).control(:,2) = B_guess;</pre>
564	<pre>guess.phase(iphase).integral = LT_DV;</pre>
565	%
566	% Phase 2 Information (second Maneuver
567	iphase = 2;

_ _ _ _ _ _

568	<pre>bounds.phase(iphase).initialtime.lower = tf1/TU;</pre>
569	<pre>bounds.phase(iphase).initialtime.upper = tf1/TU;</pre>
570	<pre>bounds.phase(iphase).finaltime.lower = tf2_min-1;</pre>
571	<pre>bounds.phase(iphase).finaltime.upper = tf2_max+1;</pre>
572	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
573	<pre>bounds.phase(iphase).initialstate.lower = [rmin thetaf_min -0.1</pre>
	0];

574	<pre>bounds.phase(iphase).initialstate.upper = [rmax thetaf_max 0.1</pre>
	1.1];
575	<pre>bounds.phase(iphase).finalstate.lower = [rmin thetaf2_min-1 -0.1</pre>
	0];
576	<pre>bounds.phase(iphase).finalstate.upper = [rmax thetaf2_max+1 0.1</pre>
	1.1];
577	<pre>bounds.phase(iphase).state.lower = [rmin thetaf_min -0.1 0];</pre>
578	<pre>bounds.phase(iphase).state.upper = [rmax thetaf2_max+1 0.1 1.1];</pre>
579	<pre>bounds.phase(iphase).control.lower = [Tmin umin];</pre>
580	<pre>bounds.phase(iphase).control.upper = [Tmax umax];</pre>
581	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
582	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
583	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
584	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
585	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
586	<pre>bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 0 Rmin/DU Rmin</pre>
	/DU]; % None
587	<pre>bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 0 Rmax/DU Rmax</pre>
	/DU]; % None
588	% GUESS SOLUTION
589	<pre>guess.phase(iphase).time = time_guess2;</pre>
590	<pre>guess.phase(iphase).state(:,1) = r_guess2;</pre>
591	<pre>guess.phase(iphase).state(:,2) = theta_guess2;</pre>
592	<pre>guess.phase(iphase).state(:,3) = vr_guess2;</pre>
593	<pre>guess.phase(iphase).state(:,4) = vtheta_guess2;</pre>
594	% Control guess :
595	<pre>guess.phase(iphase).control(:,1) = T_guess2;</pre>
596	<pre>guess.phase(iphase).control(:,2) = B_guess2;</pre>
597	<pre>guess.phase(iphase).integral = LT_DV2;</pre>
598	%

599	% Phase 3 (Coast)
600	iphase = 3;
601	<pre>bounds.phase(iphase).initialtime.lower = tf2_min-1;</pre>
602	<pre>bounds.phase(iphase).initialtime.upper = tf2_max+1;</pre>
603	<pre>bounds.phase(iphase).finaltime.lower = tcoast3(end)-1;</pre>
604	<pre>bounds.phase(iphase).finaltime.upper = tcoast3(end)+1;</pre>
605	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
606	<pre>bounds.phase(iphase).initialstate.lower = [rmin thetaf2_min-1</pre>
	-0.1 0];
607	<pre>bounds.phase(iphase).initialstate.upper = [rmax thetaf2_max+1</pre>
	0.1 1.1];
608	<pre>bounds.phase(iphase).finalstate.lower = [rmin theta3end-1 -0.1</pre>
	0];
609	<pre>bounds.phase(iphase).finalstate.upper = [rmax theta3end+1 0.1</pre>
	1.1];
610	<pre>bounds.phase(iphase).state.lower = [rmin thetaf2_min-1 -0.1 0];</pre>
611	<pre>bounds.phase(iphase).state.upper = [rmax theta3end+1 0.1 1.1];</pre>
612	<pre>bounds.phase(iphase).control.lower = [0 0];</pre>
613	<pre>bounds.phase(iphase).control.upper = [0 0];</pre>
614	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
615	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
616	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
617	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
618	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
619	<pre>bounds.eventgroup(iphase).lower = [zeros(1,5) 0]; % None</pre>
620	<pre>bounds.eventgroup(iphase).upper = [zeros(1,5) 0]; % None</pre>
621	% GUESS SOLUTION
622	<pre>guess.phase(iphase).time = time_guess3;</pre>
623	<pre>guess.phase(iphase).state(:,1) = r_guess3;</pre>
624	<pre>guess.phase(iphase).state(:,2) = theta_guess3;</pre>
625	<pre>guess.phase(iphase).state(:,3) = vr_guess3;</pre>
626	guess.phase(iphase).state(:,4) = vtheta_guess3;

627	% Control guess :
628	<pre>guess.phase(iphase).control(:,1) = T_guess3;</pre>
629	<pre>guess.phase(iphase).control(:,2) = Beta_guess3;</pre>
630	<pre>guess.phase(iphase).integral = 0;</pre>
631	
632	% Phase 4 Information (third Maneuver)
633	iphase = 4;
634	<pre>bounds.phase(iphase).initialtime.lower = tcoast3(end)-1;</pre>
635	<pre>bounds.phase(iphase).initialtime.upper = tcoast3(end)+1;</pre>
636	<pre>bounds.phase(iphase).finaltime.lower = (tf1+tf2+tf4)/TU-1;</pre>
637	<pre>bounds.phase(iphase).finaltime.upper = (tf1+tf2+tf4)/TU+1;</pre>
638	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
639	<pre>bounds.phase(iphase).initialstate.lower = [rmin theta3end-1 -0.1</pre>
	0];
640	<pre>bounds.phase(iphase).initialstate.upper = [rmax theta3end+1 0.1</pre>
	1.1];
641	<pre>bounds.phase(iphase).finalstate.lower = [rmin theta_guess4(end)</pre>
	-1 - 0.1 0];
642	<pre>bounds.phase(iphase).finalstate.upper = [rmax theta_guess4(end)</pre>
	+1 0.1 1.1];
643	<pre>bounds.phase(iphase).state.lower = [rmin theta3end-1 -0.1 0];</pre>
644	<pre>bounds.phase(iphase).state.upper = [rmax theta_guess4(end)+1 0.1</pre>
	1.1];
645	<pre>bounds.phase(iphase).control.lower = [Tmin umin];</pre>
646	<pre>bounds.phase(iphase).control.upper = [Tmax umax];</pre>
647	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
648	<pre>bounds.parameter.lower = [0 0 0];</pre>
649	<pre>bounds.parameter.upper = [2*pi 2*pi 2*pi];</pre>
650	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
651	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
652	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
653	<pre>bounds.phase(iphase).integral.upper = 1;</pre>

654	<pre>bounds.eventgroup(iphase).lower = [0 0 0 Rmin/DU Rmin/DU]; %</pre>
	None
655	<pre>bounds.eventgroup(iphase).upper = [0 0 0 Rmax/DU Rmax/DU]; %</pre>
	None
656	% GUESS SOLUTION
657	guess.phase(iphase).time = time_guess4;
658	<pre>guess.phase(iphase).state(:,1) = r_guess4;</pre>
659	guess.phase(iphase).state(:,2) = theta_guess4;
660	<pre>guess.phase(iphase).state(:,3) = vr_guess4;</pre>
661	guess.phase(iphase).state(:,4) = vtheta_guess4;
662	% Control guess :
663	<pre>guess.phase(iphase).control(:,1) = T_guess4;</pre>
664	<pre>guess.phase(iphase).control(:,2) = Beta_guess4;</pre>
665	guess.parameter = [phi phi2 phi3];
666	<pre>guess.phase(iphase).integral = LT_DV3;</pre>
667	%

668	%auxiliary data
669	auxdata.MU = MU2;
670	auxdata.ae = ae1;
671	auxdata.be = be1;
672	<pre>auxdata.rf_pqw = rf_pqw;</pre>
673	auxdata.vunit = vunit;
674	auxdata.gunit = gunit;
675	auxdata.inc = inc;
676	auxdata.RAAN = RAAN;
677	auxdata.w = w;
678	auxdata.latlim = latlim;
679	auxdata.longlim = longlim;
680	auxdata.GMST0 = GMST0;
681	<pre>auxdata.OmegaEarth = OmegaEarth;</pre>

682	auxdata.DU = DU;
683	auxdata.TU = TU;
684	
685	% NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
	required
686	<pre>setup.name = 'TIME_FIXED_INTERCEPT';</pre>
687	
688	
689	colp = 4;
690	colnum = 20;
691	colp2 = 4;
692	colnum2 = 80;
693	
694	<pre>setup.functions.continuous = @LT_3RTM_Continuous_4Phase;</pre>
695	<pre>setup.functions.endpoint = @LT_3RTM_Endpoint_4Phase;</pre>
696	<pre>setup.nlp.solver = 'ipopt';</pre>
697	<pre>setup.mesh.maxiteration = 10;</pre>
698	<pre>setup.mesh.tolerance = 1e-10;</pre>
699	<pre>setup.mesh.colpointsmin = 40;</pre>
700	<pre>setup.mesh.colpointsmax = 1000;</pre>
701	for $i = 1:4$
702	if i < 4
703	<pre>setup.mesh.phase(i).colpoints = colp*ones(1,colnum);</pre>
704	<pre>setup.mesh.phase(i).fraction = 1/colnum*ones(1,colnum);</pre>
705	elseif i == 3
706	<pre>setup.mesh.phase(i).colpoints = 1*ones(1,5);</pre>
707	<pre>setup.mesh.phase(i).fraction = 1/5*ones(1,5);</pre>
708	else
709	<pre>setup.mesh.phase(i).colpoints = colp2*ones(1,colnum2);</pre>
710	<pre>setup.mesh.phase(i).fraction = 1/colnum2*ones(1,colnum2)</pre>
	;
711	end

712	end
713	<pre>setup.bounds = bounds;</pre>
714	<pre>setup.guess = guess;</pre>
715	setup.auxdata = auxdata;
716	<pre>setup.mesh.method = 'RPMintegration';</pre>
717	<pre>setup.derivatives.supplier = 'sparseFD';</pre>
718	<pre>setup.derivativelevel ='second';</pre>
719	<pre>setup.dependencies = 'sparseNaN';</pre>
720	<pre>setup.scales = 'none';</pre>
721	
722	<pre>output = gpops2(setup);</pre>
723	<pre>solution = output.result.solution;</pre>
724	%%
725	%States and costates from phase 1 (first maneuver)
726	<pre>r_GPOPS_P1 = solution.phase(1).state(:,1);</pre>
727	<pre>theta_GPOPS_P1 = solution.phase(1).state(:,2);</pre>
728	<pre>Vr_GPOPS_P1 = solution.phase(1).state(:,3);</pre>
729	<pre>Vt_GPOPS_P1 = solution.phase(1).state(:,4);</pre>
730	<pre>lambda_r_P1 = solution.phase(1).costate(:,1);</pre>
731	<pre>lambda_theta_P1 = solution.phase(1).costate(:,2);</pre>
732	<pre>lambda_Vr_P1 = solution.phase(1).costate(:,3);</pre>
733	<pre>lambda_Vt_P1 = solution.phase(1).costate(:,4);</pre>
734	<pre>tvec_P1 = solution.phase(1).time;</pre>
735	
736	<pre>thetadot_GPOPS_P1 = Vt_GPOPS_P1./r_GPOPS_P1;</pre>
737	<pre>T_GPOPS_P1 = solution.phase(1).control(:,1);</pre>
738	<pre>Beta_GPOPS_P1 = solution.phase(1).control(:,2);</pre>
739	<pre>phi_GPOPS_P1 = solution.parameter(1);</pre>
740	<pre>re_GPOPS_P1 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P1))^2 + (ae1*sin(</pre>
	<pre>phi_GPOPS_P1))^2);</pre>

742	switch_func1 = lambda_Vr_P1.*sin(Beta_GPOPS_P1) + lambda_Vt_P1.*
	<pre>cos(Beta_GPOPS_P1) + 1;</pre>
743	
744	%

745	
746	%States and Costates from phase 2 (second maneuver)
747	<pre>r_GPOPS_P2 = solution.phase(2).state(:,1);</pre>
748	<pre>theta_GPOPS_P2 = solution.phase(2).state(:,2);</pre>
749	<pre>Vr_GPOPS_P2 = solution.phase(2).state(:,3);</pre>
750	<pre>Vt_GPOPS_P2 = solution.phase(2).state(:,4);</pre>
751	<pre>lambda_r_P2 = solution.phase(2).costate(:,1);</pre>
752	<pre>lambda_theta_P2 = solution.phase(2).costate(:,2);</pre>
753	<pre>lambda_Vr_P2 = solution.phase(2).costate(:,3);</pre>
754	<pre>lambda_Vt_P2 = solution.phase(2).costate(:,4);</pre>
755	<pre>tvec_P2 = solution.phase(2).time;</pre>
756	
757	<pre>thetadot_GPOPS_P2 = Vt_GPOPS_P2./r_GPOPS_P2;</pre>
758	<pre>T_GPOPS_P2 = solution.phase(2).control(:,1);</pre>
759	<pre>Beta_GPOPS_P2 = solution.phase(2).control(:,2);</pre>
760	<pre>phi_GPOPS_P2 = solution.parameter(2);</pre>
761	<pre>re_GPOPS_P2 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P2))^2 + (ae1*sin(</pre>
	<pre>phi_GPOPS_P2))^2);</pre>
762	
763	<pre>switch_func2 = lambda_Vr_P2.*sin(Beta_GPOPS_P2) + lambda_Vt_P2.*</pre>
	<pre>cos(Beta_GPOPS_P2) + 1;</pre>
764	%
765	

%States and Costates from phase 3 (third maneuver)

767	<pre>r_GPOPS_P3 = solution.phase(3).state(:,1);</pre>
768	<pre>theta_GPOPS_P3 = solution.phase(3).state(:,2);</pre>
769	<pre>Vr_GPOPS_P3 = solution.phase(3).state(:,3);</pre>
770	<pre>Vt_GPOPS_P3 = solution.phase(3).state(:,4);</pre>
771	<pre>lambda_r_P3 = solution.phase(3).costate(:,1);</pre>
772	<pre>lambda_theta_P3 = solution.phase(3).costate(:,2);</pre>
773	<pre>lambda_Vr_P3 = solution.phase(3).costate(:,3);</pre>
774	<pre>lambda_Vt_P3 = solution.phase(3).costate(:,4);</pre>
775	<pre>tvec_P3 = solution.phase(3).time;</pre>
776	
777	<pre>thetadot_GPOPS_P3 = Vt_GPOPS_P3./r_GPOPS_P3;</pre>
778	<pre>T_GPOPS_P3 = solution.phase(3).control(:,1);</pre>
779	<pre>Beta_GPOPS_P3 = solution.phase(3).control(:,2);</pre>
780	<pre>phi_GPOPS_P3 = solution.parameter(3);</pre>
781	<pre>re_GPOPS_P3 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P3))^2 + (ae1*sin(</pre>
	<pre>phi_GPOPS_P3))^2);</pre>
782	
783	<pre>switch_func3 = lambda_Vr_P3.*sin(Beta_GPOPS_P3) + lambda_Vt_P3.*</pre>
	<pre>cos(Beta_GPOPS_P3) + 1;</pre>
784	
785	%States and Costates from phase 3 (third maneuver)
786	<pre>r_GPOPS_P4 = solution.phase(4).state(:,1);</pre>
787	<pre>theta_GPOPS_P4 = solution.phase(4).state(:,2);</pre>
788	<pre>Vr_GPOPS_P4 = solution.phase(4).state(:,3);</pre>
789	<pre>Vt_GPOPS_P4 = solution.phase(4).state(:,4);</pre>
790	<pre>lambda_r_P4 = solution.phase(4).costate(:,1);</pre>
791	<pre>lambda_theta_P4 = solution.phase(4).costate(:,2);</pre>
792	<pre>lambda_Vr_P4 = solution.phase(4).costate(:,3);</pre>
793	<pre>lambda_Vt_P4 = solution.phase(4).costate(:,4);</pre>
794	The second se
	tvec_P4 = solution.pnase(4).time;
795	<pre>tvec_P4 = solution.pnase(4).time;</pre>

```
T_GPOPS_P4 = solution.phase(4).control(:,1);
797
           Beta_GPOPS_P4 = solution.phase(4).control(:,2);
798
           re_GPOPS_P4 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P3))^2 + (ae1*sin(
799
               phi_GPOPS_P3))^2);
800
            switch_func4 = lambda_Vr_P4.*sin(Beta_GPOPS_P4) + lambda_Vt_P4.*
801
               cos(Beta_GPOPS_P4) + 1;
802
           %
803
           Cost = (solution.phase(1).integral + solution.phase(2).integral
804
               + solution.phase(3).integral + solution.phase(4).integral)*
               DU/TU*1000
           %% GPOPS Run two (Minimum thrust is set to zero in run 2 to
805
               generate true optimal solution
           clear guess setup bound limits
806
807
           % variables from PSo phase
808
           r1 = 1;
809
           rmax = r1 + be/DU;
810
           rmin = r1 - be/DU;
811
           thetaf_min = theta_rf - atan(ae/norm(r0));
812
           thetaf_max = theta_rf + atan(ae/norm(r0));
813
814
           % Control and time boundaries
815
           umin = -0.5; % minimum control angle
816
           umax = 2*pi+0.5; % maximum control angle
817
           Tmax = 2*0.0001160;
818
                  Tmin = 0;
           %
819
820
           % Phase 1 Information
821
```

822	iphase = 1;
823	<pre>bounds.phase(iphase).initialtime.lower = t0min;</pre>
824	<pre>bounds.phase(iphase).initialtime.upper = t0max;</pre>
825	<pre>bounds.phase(iphase).finaltime.lower = tf1/TU;</pre>
826	<pre>bounds.phase(iphase).finaltime.upper = tf1/TU;</pre>
827	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
828	<pre>bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU</pre>
	<pre>0 sqrt(MU2/r1)];</pre>
829	<pre>bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU</pre>
	<pre>0 sqrt(MU2/r1)];</pre>
830	<pre>bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.1</pre>
	0];
831	<pre>bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.1</pre>
	1.1];
832	<pre>bounds.phase(iphase).state.lower = [rmin theta_rf-n0*tf1/TU -0.1</pre>
	0];
833	<pre>bounds.phase(iphase).state.upper = [rmax thetaf_max 0.1 1.1];</pre>
834	<pre>bounds.phase(iphase).control.lower = [Tmin umin];</pre>
835	<pre>bounds.phase(iphase).control.upper = [Tmax umax];</pre>
836	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
837	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
838	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
839	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
840	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
841	<pre>bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 Rmin/DU Rmin/</pre>
	DU]; % None
842	<pre>bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 Rmax/DU Rmax/</pre>
	DU]; % None
843	% GUESS SOLUTION
844	<pre>guess.phase(iphase).time = tvec_P1;</pre>
845	<pre>guess.phase(iphase).state(:,1) = r_GPOPS_P1;</pre>
846	guess.phase(iphase).state(:,2) = theta_GPOPS_P1;

847	<pre>guess.phase(iphase).state(:,3) = Vr_GPOPS_P1;</pre>
848	<pre>guess.phase(iphase).state(:,4) = Vt_GPOPS_P1;</pre>
849	% Control guess :
850	<pre>guess.phase(iphase).control(:,1) = T_GPOPS_P1;</pre>
851	<pre>guess.phase(iphase).control(:,2) = Beta_GPOPS_P1;</pre>
852	<pre>guess.phase(iphase).integral = solution.phase(1).integral;</pre>
853	%

854	% Phase 2 Information (second Maneuver
855	iphase = 2;
856	<pre>bounds.phase(iphase).initialtime.lower = tf1/TU;</pre>
857	<pre>bounds.phase(iphase).initialtime.upper = tf1/TU;</pre>
858	<pre>bounds.phase(iphase).finaltime.lower = tf2_min-1;</pre>
859	<pre>bounds.phase(iphase).finaltime.upper = tf2_max+1;</pre>
860	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
861	<pre>bounds.phase(iphase).initialstate.lower = [rmin thetaf_min -0.1</pre>
	0];
862	<pre>bounds.phase(iphase).initialstate.upper = [rmax thetaf_max 0.1</pre>
	1.1];
863	<pre>bounds.phase(iphase).finalstate.lower = [rmin thetaf2_min-1 -0.1</pre>
	0];
864	<pre>bounds.phase(iphase).finalstate.upper = [rmax thetaf2_max+1 0.1</pre>
	1.1];
865	<pre>bounds.phase(iphase).state.lower = [rmin thetaf_min -0.1 0];</pre>
866	<pre>bounds.phase(iphase).state.upper = [rmax thetaf2_max+1 0.1 1.1];</pre>
867	<pre>bounds.phase(iphase).control.lower = [Tmin umin];</pre>
868	<pre>bounds.phase(iphase).control.upper = [Tmax umax];</pre>
869	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
870	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
871	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
872	<pre>bounds.phase(iphase).integral.lower = 0;</pre>

873	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
874	<pre>bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 0 Rmin/DU Rmin</pre>
	/DU]; % None
875	<pre>bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 0 Rmax/DU Rmax</pre>
	/DU]; % None
876	% GUESS SOLUTION
877	<pre>guess.phase(iphase).time = tvec_P2;</pre>
878	<pre>guess.phase(iphase).state(:,1) = r_GPOPS_P2;</pre>
879	<pre>guess.phase(iphase).state(:,2) = theta_GPOPS_P2;</pre>
880	<pre>guess.phase(iphase).state(:,3) = Vr_GPOPS_P2;</pre>
881	<pre>guess.phase(iphase).state(:,4) = Vt_GPOPS_P2;</pre>
882	% Control guess :
883	<pre>guess.phase(iphase).control(:,1) = T_GPOPS_P2;</pre>
884	<pre>guess.phase(iphase).control(:,2) = Beta_GPOPS_P2;</pre>
885	<pre>guess.phase(iphase).integral = solution.phase(2).integral;</pre>
886	%

887	% Phase 3 (Coast)
888	iphase = 3;
889	<pre>bounds.phase(iphase).initialtime.lower = tf2_min-1;</pre>
890	<pre>bounds.phase(iphase).initialtime.upper = tf2_max+1;</pre>
891	<pre>bounds.phase(iphase).finaltime.lower = tcoast3(end)-1;</pre>
892	<pre>bounds.phase(iphase).finaltime.upper = tcoast3(end)+1;</pre>
893	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
894	<pre>bounds.phase(iphase).initialstate.lower = [rmin thetaf2_min-1</pre>
	-0.1 0];
895	<pre>bounds.phase(iphase).initialstate.upper = [rmax thetaf2_max+1</pre>
	0.1 1.1];
896	<pre>bounds.phase(iphase).finalstate.lower = [rmin theta3end-1 -0.1</pre>
	0];
897	<pre>bounds.phase(iphase).finalstate.upper = [rmax theta3end+1 0.1</pre>
-----	--
	1.1];
898	<pre>bounds.phase(iphase).state.lower = [rmin thetaf2_min-1 -0.1 0];</pre>
899	<pre>bounds.phase(iphase).state.upper = [rmax theta3end+1 0.1 1.1];</pre>
900	<pre>bounds.phase(iphase).control.lower = [0 0];</pre>
901	<pre>bounds.phase(iphase).control.upper = [0 0];</pre>
902	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
903	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
904	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
905	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
906	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
907	<pre>bounds.eventgroup(iphase).lower = [zeros(1,5) 0]; % None</pre>
908	<pre>bounds.eventgroup(iphase).upper = [zeros(1,5) 0]; % None</pre>
909	% GUESS SOLUTION
910	<pre>guess.phase(iphase).time = tvec_P3;</pre>
911	<pre>guess.phase(iphase).state(:,1) = r_GPOPS_P3;</pre>
912	<pre>guess.phase(iphase).state(:,2) = theta_GPOPS_P3;</pre>
913	<pre>guess.phase(iphase).state(:,3) = Vr_GPOPS_P3;</pre>
914	<pre>guess.phase(iphase).state(:,4) = Vt_GPOPS_P3;</pre>
915	% Control guess :
916	<pre>guess.phase(iphase).control(:,1) = T_GPOPS_P3;</pre>
917	<pre>guess.phase(iphase).control(:,2) = Beta_GPOPS_P3;</pre>
918	<pre>guess.phase(iphase).integral = solution.phase(3).integral;</pre>
919	
920	% Phase 4 Information (third Maneuver)
921	<pre>iphase = 4;</pre>
922	<pre>bounds.phase(iphase).initialtime.lower = tcoast3(end)-1;</pre>
923	<pre>bounds.phase(iphase).initialtime.upper = tcoast3(end)+1;</pre>
924	<pre>bounds.phase(iphase).finaltime.lower = (tf1+tf2+tf4)/TU-1;</pre>
925	<pre>bounds.phase(iphase).finaltime.upper = (tf1+tf2+tf4)/TU+1;</pre>
926	% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY

927	<pre>bounds.phase(iphase).initialstate.lower = [rmin theta3end-1 -0.1</pre>
	0];
928	<pre>bounds.phase(iphase).initialstate.upper = [rmax theta3end+1 0.1</pre>
	1.1];
929	<pre>bounds.phase(iphase).finalstate.lower = [rmin theta_guess4(end)</pre>
	-1 - 0.1 0];
930	bounds.phase(iphase).finalstate.upper = [rmax theta_guess4(end)
	+1 0.1 1.1];
931	<pre>bounds.phase(iphase).state.lower = [rmin theta3end-1 -0.1 0];</pre>
932	<pre>bounds.phase(iphase).state.upper = [rmax theta_guess4(end)+1 0.1</pre>
	1.1];
933	<pre>bounds.phase(iphase).control.lower = [Tmin umin];</pre>
934	<pre>bounds.phase(iphase).control.upper = [Tmax umax];</pre>
935	% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
936	<pre>bounds.parameter.lower = [0 0 0];</pre>
937	<pre>bounds.parameter.upper = [2*pi 2*pi 2*pi];</pre>
938	<pre>bounds.phase(iphase).path.lower = []; % None</pre>
939	<pre>bounds.phase(iphase).path.upper = []; % None</pre>
940	<pre>bounds.phase(iphase).integral.lower = 0;</pre>
941	<pre>bounds.phase(iphase).integral.upper = 1;</pre>
942	<pre>bounds.eventgroup(iphase).lower = [0 0 0 Rmin/DU Rmin/DU]; %</pre>
	None
943	<pre>bounds.eventgroup(iphase).upper = [0 0 0 Rmax/DU Rmax/DU]; %</pre>
	None
944	% GUESS SOLUTION
945	<pre>guess.phase(iphase).time = tvec_P4;</pre>
946	<pre>guess.phase(iphase).state(:,1) = r_GPOPS_P4;</pre>
947	<pre>guess.phase(iphase).state(:,2) = theta_GPOPS_P4;</pre>
948	<pre>guess.phase(iphase).state(:,3) = Vr_GPOPS_P4;</pre>
949	<pre>guess.phase(iphase).state(:,4) = Vt_GPOPS_P4;</pre>
950	% Control guess :
951	<pre>guess.phase(iphase).control(:,1) = T_GPOPS_P4;</pre>

952	<pre>guess.phase(iphase).control(:,2) = Beta_GPOPS_P4;</pre>
953	guess.parameter = [phi_GPOPS_P1 phi_GPOPS_P2 phi_GPOPS_P3];
954	<pre>guess.phase(iphase).integral = solution.phase(4).integral;</pre>
955	
956	%auxiliary data
957	auxdata.MU = MU2;
958	auxdata.ae = ae1;
959	<pre>auxdata.be = be1;</pre>
960	<pre>auxdata.rf_pqw = rf_pqw;</pre>
961	auxdata.vunit = vunit;
962	auxdata.gunit = gunit;
963	<pre>auxdata.inc = inc;</pre>
964	auxdata.RAAN = RAAN;
965	auxdata.w = w;
966	auxdata.latlim = latlim;
967	auxdata.longlim = longlim;
968	auxdata.GMST0 = GMST0;
969	<pre>auxdata.OmegaEarth = OmegaEarth;</pre>
970	auxdata.DU = DU;
971	auxdata.TU = TU;
972	
973	% NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
	required
974	<pre>setup.name = 'TIME_FIXED_INTERCEPT';</pre>
975	
976	
977	<pre>colp = 4;</pre>
978	colnum = 20;
979	colp2 = 4;
980	colnum2 = 80;
981	
982	<pre>setup.functions.continuous = @LT_3RTM_Continuous_4Phase;</pre>

983	<pre>setup.functions.endpoint = @LT_3RTM_Endpoint_4Phase;</pre>
984	<pre>setup.nlp.solver = 'ipopt';</pre>
985	<pre>setup.mesh.maxiteration = 10;</pre>
986	<pre>setup.mesh.tolerance = 1e-10;</pre>
987	<pre>setup.mesh.colpointsmin = 40;</pre>
988	<pre>setup.mesh.colpointsmax = 1000;</pre>
989	for i = 1:4
990	if i < 4
991	<pre>setup.mesh.phase(i).colpoints = colp*ones(1,colnum);</pre>
992	<pre>setup.mesh.phase(i).fraction = 1/colnum*ones(1,colnum);</pre>
993	elseif i == 3
994	<pre>setup.mesh.phase(i).colpoints = 1*ones(1,5);</pre>
995	<pre>setup.mesh.phase(i).fraction = 1/5*ones(1,5);</pre>
996	else
997	<pre>setup.mesh.phase(i).colpoints = colp2*ones(1,colnum2);</pre>
998	<pre>setup.mesh.phase(i).fraction = 1/colnum2*ones(1,colnum2)</pre>
	;
999	end
1000	end
1001	<pre>setup.bounds = bounds;</pre>
1002	
	<pre>setup.guess = guess;</pre>
1003	setup.guess = guess; setup.auxdata = auxdata;
1003 1004	<pre>setup.guess = guess; setup.auxdata = auxdata; setup.mesh.method = 'RPMintegration';</pre>
1003 1004 1005	<pre>setup.guess = guess; setup.auxdata = auxdata; setup.mesh.method = 'RPMintegration'; setup.derivatives.supplier = 'sparseFD';</pre>
1003 1004 1005 1006	<pre>setup.guess = guess; setup.auxdata = auxdata; setup.mesh.method = 'RPMintegration'; setup.derivatives.supplier = 'sparseFD'; setup.derivativelevel ='second';</pre>
1003 1004 1005 1006 1007	<pre>setup.guess = guess; setup.auxdata = auxdata; setup.mesh.method = 'RPMintegration'; setup.derivatives.supplier = 'sparseFD'; setup.derivativelevel ='second'; setup.dependencies = 'sparseNaN';</pre>
1003 1004 1005 1006 1007 1008	<pre>setup.guess = guess; setup.auxdata = auxdata; setup.mesh.method = 'RPMintegration'; setup.derivatives.supplier = 'sparseFD'; setup.derivativelevel ='second'; setup.dependencies = 'sparseNaN'; setup.scales = 'none';</pre>
1003 1004 1005 1006 1007 1008 1009	<pre>setup.guess = guess; setup.auxdata = auxdata; setup.mesh.method = 'RPMintegration'; setup.derivatives.supplier = 'sparseFD'; setup.derivativelevel ='second'; setup.dependencies = 'sparseNaN'; setup.scales = 'none';</pre>
1003 1004 1005 1006 1007 1008 1009 1010	<pre>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);</pre>
1003 1004 1005 1006 1007 1008 1009 1010	<pre>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;</pre>
1003 1004 1005 1006 1007 1008 1009 1010 1011 1012	<pre>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;</pre>

1014	%States and costates from phase 1 (first maneuver)
1015	<pre>r_GPOPS_P12 = solution2.phase(1).state(:,1);</pre>
1016	<pre>theta_GPOPS_P12 = solution2.phase(1).state(:,2);</pre>
1017	<pre>Vr_GPOPS_P12 = solution2.phase(1).state(:,3);</pre>
1018	<pre>Vt_GPOPS_P12 = solution2.phase(1).state(:,4);</pre>
1019	<pre>lambda_r_P12 = solution2.phase(1).costate(:,1);</pre>
1020	lambda_theta_P12 = solution2.phase(1).costate(:,2);
1021	lambda_Vr_P12 = solution2.phase(1).costate(:,3);
1022	lambda_Vt_P12 = solution2.phase(1).costate(:,4);
1023	<pre>tvec_P12 = solution2.phase(1).time;</pre>
1024	
1025	<pre>thetadot_GPOPS_P12 = Vt_GPOPS_P12./r_GPOPS_P12;</pre>
1026	<pre>T_GPOPS_P12 = solution2.phase(1).control(:,1);</pre>
1027	<pre>Beta_GPOPS_P12 = solution2.phase(1).control(:,2);</pre>
1028	<pre>phi_GPOPS_P12 = solution2.parameter(1);</pre>
1029	<pre>re_GPOPS_P12 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P12))^2 + (ae1*</pre>
	<pre>sin(phi_GPOPS_P12))^2);</pre>
1030	
1031	%States and Costates from pahse 2 (second maneuver)
1032	<pre>r_GPOPS_P22 = solution2.phase(2).state(:,1);</pre>
1033	<pre>theta_GPOPS_P22 = solution2.phase(2).state(:,2);</pre>
1034	<pre>Vr_GPOPS_P22 = solution2.phase(2).state(:,3);</pre>
1035	<pre>Vt_GPOPS_P22 = solution2.phase(2).state(:,4);</pre>
1036	
1037	lambda_r_P22 = solution2.pnase(2).costate(:,1);
	<pre>lambda_r_P22 = solution2.phase(2).costate(:,1); lambda_theta_P22 = solution2.phase(2).costate(:,2);</pre>
1038	<pre>lambda_r_P22 = solution2.phase(2).costate(:,1); lambda_theta_P22 = solution2.phase(2).costate(:,2); lambda_Vr_P22 = solution2.phase(2).costate(:,3);</pre>
1038 1039	<pre>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);</pre>
1038 1039 1040	<pre>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;</pre>
1038 1039 1040 1041	<pre>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;</pre>
1038 1039 1040 1041 1042	<pre>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;</pre>
1038 1039 1040 1041 1042 1043	<pre>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);</pre>

1045	<pre>phi_GPOPS_P22 = solution2.parameter(2);</pre>
1046	<pre>re_GPOPS_P22 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P22))^2 + (ae1*</pre>
	<pre>sin(phi_GPOPS_P22))^2);</pre>
1047	
1048	%States and Costates from pahse 2 (second maneuver)
1049	<pre>r_GPOPS_P32 = solution2.phase(3).state(:,1);</pre>
1050	<pre>theta_GPOPS_P32 = solution2.phase(3).state(:,2);</pre>
1051	<pre>Vr_GPOPS_P32 = solution2.phase(3).state(:,3);</pre>
1052	<pre>Vt_GPOPS_P32 = solution2.phase(3).state(:,4);</pre>
1053	<pre>lambda_r_P32 = solution2.phase(3).costate(:,1);</pre>
1054	<pre>lambda_theta_P32 = solution2.phase(3).costate(:,2);</pre>
1055	<pre>lambda_Vr_P32 = solution2.phase(3).costate(:,3);</pre>
1056	<pre>lambda_Vt_P32 = solution2.phase(3).costate(:,4);</pre>
1057	<pre>tvec_P32 = solution2.phase(3).time;</pre>
1058	
1059	<pre>thetadot_GPOPS_P32 = Vt_GPOPS_P32./r_GPOPS_P32;</pre>
1060	<pre>T_GPOPS_P32 = solution2.phase(3).control(:,1);</pre>
1061	<pre>Beta_GPOPS_P32 = solution2.phase(3).control(:,2);</pre>
1062	<pre>phi_GPOPS_P32 = solution2.parameter(3);</pre>
1063	<pre>re_GPOPS_P32 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P32))^2 + (ae1*</pre>
	<pre>sin(phi_GPOPS_P32))^2);</pre>
1064	
1065	%States and Costates from pahse 2 (second maneuver)
1066	<pre>r_GPOPS_P42 = solution2.phase(4).state(:,1);</pre>
1067	<pre>theta_GPOPS_P42 = solution2.phase(4).state(:,2);</pre>
1068	<pre>Vr_GPOPS_P42 = solution2.phase(4).state(:,3);</pre>
1069	<pre>Vt_GPOPS_P42 = solution2.phase(4).state(:,4);</pre>
1070	<pre>lambda_r_P42 = solution2.phase(4).costate(:,1);</pre>
1071	lambda_theta_P42 = solution2.phase(4).costate(:,2);
1072	<pre>lambda_Vr_P42 = solution2.phase(4).costate(:,3);</pre>
1073	<pre>lambda_Vt_P42 = solution2.phase(4).costate(:,4);</pre>
1074	<pre>tvec_P42 = solution2.phase(4).time;</pre>

1075	
1076	<pre>thetadot_GPOPS_P42 = Vt_GPOPS_P42./r_GPOPS_P42;</pre>
1077	<pre>T_GPOPS_P42 = solution2.phase(4).control(:,1);</pre>
1078	<pre>Beta_GPOPS_P42 = solution2.phase(4).control(:,2);</pre>
1079	<pre>re_GPOPS_P42 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P32))^2 + (ae1*</pre>
	<pre>sin(phi_GPOPS_P32))^2);</pre>
1080	
1081	<pre>Cost2 = (solution2.phase(1).integral + solution2.phase(2).</pre>
	<pre>integral + solution2.phase(3).integral + solution2.phase(4).</pre>
	integral)*DU/TU*1000
1082	
1083	
1084	%%
1085	clear rgi rgi2
1086	
1087	ang = (0:0.001:2*pi);
1088	<pre>re = (ae1*be1)./sqrt((be1*cos(ang)).^2 + (ae1*sin(ang)).^2);</pre>
1089	
1090	%
1091	% Determine entry condition for second maneuver
1092	<pre>rt = [r_GPOPS_P12(end)*cos(theta_GPOPS_P12(end));r_GPOPS_P12(end</pre>
)*sin(theta_GPOPS_P12(end));0];
1093	
1094	%apogee and perigee constraints
1095	<pre>Vf_mag = sqrt(Vr_GPOPS_P12(end)^2 + Vt_GPOPS_P12(end)^2);</pre>
1096	<pre>fpa = atan(Vr_GPOPS_P12(end)/Vt_GPOPS_P12(end));</pre>
1097	
1098	%perifocal velocity
1099	<pre>vt = Vf_mag*[-sin(theta_GPOPS_P12(end)-fpa); cos(theta_GPOPS_P12(</pre>
	end)-fpa);0];

1100	
1101	<pre>[rt_ijk_P12,vt_ijk_P12] = PQW_to_IJK(rt,vt,inc,RAAN,w);</pre>
1102	<pre>rt_ijk_P12 = rt_ijk_P12*DU;</pre>
1103	<pre>vt_ijk_P12 = vt_ijk_P12*DU/TU;</pre>
1104	
1105	<pre>[lat_act_enter,long_act_enter] = IJK_to_LATLONG(rt_ijk_P12(1),</pre>
	rt_ijk_P12(2),rt_ijk_P12(3),GMST0,tvec_P12(end)*TU);
1106	
1107	<pre>[r2,v2,t2] = zone_entry_exit2(rt_ijk_P12,vt_ijk_P12,GMST0+</pre>
	<pre>OmegaEarth*tvec_P12(end)*TU,0,latlim,longlim);</pre>
1108	
1109	rf2exp = r2;
1110	vf2exp = v2;
1111	tf2exp = t2;
1112	
1113	<pre>[lat_enter2exp,long_enter2exp] = IJK_to_LATLONG(r2(1),r2(2),r2</pre>
	(3),GMST0,tvec_P22(end)*TU);
1114	
1115	<pre>[rf_pqw2,vf_pqw2] = IJK_to_PQW(r2,v2,inc,RAAN,w);</pre>
1116	
1117	<pre>rf_pqw2 = rf_pqw2/DU;</pre>
1118	vf_pqw2 = vf_pqw2/DU*TU;
1119	
1120	<pre>vunit2 = vf_pqw2/norm(vf_pqw2);</pre>
1121	hfp2 = cross(rf_pqw2,vf_pqw2);
1122	<pre>hunit2 = hfp2/norm(hfp2);</pre>
1123	
1124	<pre>gunit2 = cross(vunit2, hunit2);</pre>
1125	
1126	%

1127	
1128	% for plotting purposes in PQW frame
1129	%
1130	<pre>term12 = (be1*cos(phi_GPOPS_P22))^2 + (ae1*sin(phi_GPOPS_P22))</pre>
	^2;
1131	<pre>re2 = ae1*be1/sqrt(term12);</pre>
1132	
1133	<pre>rt2 = rf_pqw2 + re2*cos(phi_GPOPS_P22)*vunit2 + re2*sin(</pre>
	<pre>phi_GPOPS_P22)*gunit2;</pre>
1134	
1135	%apogee and perigee constraints
1136	<pre>Vf_mag2 = sqrt(Vr_GPOPS_P22(end)^2 + Vt_GPOPS_P22(end)^2);</pre>
1137	<pre>fpa2 = atan(Vr_GPOPS_P22(end)/Vt_GPOPS_P22(end));</pre>
1138	
1139	%perifocal velocity
1140	<pre>vt2 = Vf_mag2*[-sin(theta_GPOPS_P22(end)-fpa2);cos(</pre>
	<pre>theta_GPOPS_P22(end)-fpa2);0];</pre>
1141	
1142	<pre>[rt_ijk_P22,vt_ijk_P22] = PQW_to_IJK(rt2,vt2,inc,RAAN,w);</pre>
1143	<pre>rt_ijk_P22 = rt_ijk_P22*DU;</pre>
1144	<pre>vt_ijk_P22 = vt_ijk_P22*DU/TU;</pre>
1145	rt2 = rt2*DU;
1146	<pre>[lat_act_enter2,long_act_enter2] = IJK_to_LATLONG(rt_ijk_P22(1),</pre>
	rt_ijk_P22(2),rt_ijk_P22(3),GMST0,tvec_P22(end)*TU);
1147	
1148	<pre>for aa = 1:length(ang)</pre>
1149	<pre>r_ell2(:,aa) = rf_pqw2 + re(aa)*cos(ang(aa))*vunit2 + re(aa)</pre>
	<pre>*sin(ang(aa))*gunit2;</pre>
1150	end
1151	

1152 1153 [r4,v4,t4] = zone_entry_exit2(rt_ijk_P22,vt_ijk_P22,GMST0+ OmegaEarth*tvec_P22(end)*TU,0,latlim,longlim); 1154 rf4exp = r4;1155 vf4exp = v4;1156 tf4exp = t4;1157 1158 [lat_enter4exp,long_enter4exp] = IJK_to_LATLONG(r4(1),r4(2),r4 1159 (3),GMST0,tvec_P42(end)*TU); 1160 [rf_pqw4,vf_pqw4] = IJK_to_PQW(r4,v4,inc,RAAN,w); 1161 1162 rf_pqw4 = rf_pqw4/DU; 1163 $vf_pqw4 = vf_pqw4/DU*TU;$ 1164 1165 vunit4 = vf_pqw4/norm(vf_pqw4); 1166 hfp4 = cross(rf_pqw4,vf_pqw4); 1167 hunit4 = hfp2/norm(hfp4); 1168 1169 gunit4 = cross(vunit4, hunit4); 1170 1171 for aa = 1:length(ang) 1172 $r_ell4(:,aa) = rf_pqw4 + re(aa)*cos(ang(aa))*vunit4 + re(aa)$ 1173 *sin(ang(aa))*gunit4; end 1174 1175 rt4 = [r_GPOPS_P42(end)*cos(theta_GPOPS_P42(end));r_GPOPS_P42(1176 end)*sin(theta_GPOPS_P42(end));0]; 1177 %apogee and perigee constraints 1178 1179 Vf_mag4 = sqrt(Vr_GPOPS_P42(end)^2 + Vt_GPOPS_P42(end)^2);

1180	<pre>fpa4 = atan(Vr_GPOPS_P42(end)/Vt_GPOPS_P42(end));</pre>
1181	
1182	%perifocal velocity
1183	<pre>vt4 = Vf_mag4*[-sin(theta_GPOPS_P42(end)-fpa4);cos(</pre>
	<pre>theta_GPOPS_P42(end)-fpa4);0];</pre>
1184	
1185	<pre>[rt_ijk_P42,vt_ijk_P42] = PQW_to_IJK(rt4,vt4,inc,RAAN,w);</pre>
1186	<pre>rt_ijk_P42 = rt_ijk_P42*DU;</pre>
1187	<pre>vt_ijk_P42 = vt_ijk_P42*DU/TU;</pre>
1188	rt4 = rt4*DU;
1189	
1190	<pre>[lat_act_enter4,long_act_enter4] = IJK_to_LATLONG(rt_ijk_P42(1),</pre>
	rt_ijk_P42(2),rt_ijk_P42(3),GMST0,tvec_P42(end)*TU);
1191	
1192	<pre>rgi = zeros(length(r_GPOPS_P12),3);</pre>
1193	<pre>vgi = zeros(length(r_GPOPS_P12),3);</pre>
1194	%First maneuver inertial position and velocity
1195	<pre>for dd = 1:length(r_GPOPS_P12)</pre>
1196	%perifocal position vector
1197	<pre>rg_pqw = DU*[r_GPOPS_P12(dd)*cos(theta_GPOPS_P12(dd));</pre>
	<pre>r_GPOPS_P12(dd)*sin(theta_GPOPS_P12(dd));0];</pre>
1198	%velocity magnitude
1199	<pre>Vf_mag = sqrt(Vr_GPOPS_P12(dd)^2 + Vt_GPOPS_P12(dd)^2);</pre>
1200	<pre>fpa = atan(Vr_GPOPS_P12(dd)/Vt_GPOPS_P12(dd));</pre>
1201	
1202	%perifocal velocity
1203	<pre>vg_pqw = DU/TU*Vf_mag*[-sin(theta_GPOPS_P12(dd)-fpa);cos(</pre>
	<pre>theta_GPOPS_P12(dd)-fpa);0];</pre>
1204	
1205	<pre>[rgi(dd,:),vgi(dd,:)] = PQW_to_IJK(rg_pqw,vg_pqw,inc,RAAN,w)</pre>
	;
1206	end

1207	
1208	<pre>rgi2 = zeros(length(r_GPOPS_P22),3);</pre>
1209	<pre>vgi2 = zeros(length(r_GPOPS_P22),3);</pre>
1210	%First maneuver inertial position and velocity
1211	<pre>for dd = 1:length(r_GPOPS_P22)</pre>
1212	%perifocal position vector
1213	<pre>rg_pqw2 = DU*[r_GPOPS_P22(dd)*cos(theta_GPOPS_P22(dd));</pre>
	<pre>r_GPOPS_P22(dd)*sin(theta_GPOPS_P22(dd));0];</pre>
1214	%velocity magnitude
1215	<pre>Vf_mag = sqrt(Vr_GPOPS_P22(dd)^2 + Vt_GPOPS_P22(dd)^2);</pre>
1216	<pre>fpa = atan(Vr_GPOPS_P22(dd)/Vt_GPOPS_P22(dd));</pre>
1217	
1218	%perifocal velocity
1219	<pre>vg_pqw2 = DU/TU*Vf_mag*[-sin(theta_GPOPS_P22(dd)-fpa);cos(</pre>
	<pre>theta_GPOPS_P22(dd)-fpa);0];</pre>
1220	
1221	<pre>[rgi2(dd,:),vgi2(dd,:)] = PQW_to_IJK(rg_pqw2,vg_pqw2,inc,</pre>
	RAAN,w);
1222	end
1223	
1224	<pre>rgi3 = zeros(length(r_GPOPS_P32),3);</pre>
1225	<pre>vgi3 = zeros(length(r_GPOPS_P32),3);</pre>
1226	%First maneuver inertial position and velocity
1227	<pre>for dd = 1:length(r_GPOPS_P32)</pre>
1228	%perifocal position vector
1229	<pre>rg_pqw3 = DU*[r_GPOPS_P32(dd)*cos(theta_GPOPS_P32(dd));</pre>
	<pre>r_GPOPS_P32(dd)*sin(theta_GPOPS_P32(dd));0];</pre>
1230	%velocity magnitude
1231	<pre>Vf_mag = sqrt(Vr_GPOPS_P32(dd)^2 + Vt_GPOPS_P32(dd)^2);</pre>
1232	<pre>fpa = atan(Vr_GPOPS_P32(dd)/Vt_GPOPS_P32(dd));</pre>
1233	
1234	%perifocal velocity

1235	<pre>vg_pqw3 = DU/TU*Vf_mag*[-sin(theta_GPOPS_P32(dd)-fpa);cos(</pre>
	<pre>theta_GPOPS_P32(dd)-fpa);0];</pre>
1236	
1237	<pre>[rgi3(dd,:),vgi3(dd,:)] = PQW_to_IJK(rg_pqw3,vg_pqw3,inc,</pre>
	RAAN,w);
1238	end
1239	
1240	<pre>rgi4 = zeros(length(r_GPOPS_P42),3);</pre>
1241	<pre>vgi4 = zeros(length(r_GPOPS_P42),3);</pre>
1242	%First maneuver inertial position and velocity
1243	<pre>for dd = 1:length(r_GPOPS_P42)</pre>
1244	%perifocal position vector
1245	<pre>rg_pqw4 = DU*[r_GPOPS_P42(dd)*cos(theta_GPOPS_P42(dd));</pre>
	<pre>r_GPOPS_P42(dd)*sin(theta_GPOPS_P42(dd));0];</pre>
1246	%velocity magnitude
1247	<pre>Vf_mag = sqrt(Vr_GPOPS_P42(dd)^2 + Vt_GPOPS_P42(dd)^2);</pre>
1248	<pre>fpa = atan(Vr_GPOPS_P42(dd)/Vt_GPOPS_P42(dd));</pre>
1249	
1250	%perifocal velocity
1251	<pre>vg_pqw4 = DU/TU*Vf_mag*[-sin(theta_GPOPS_P42(dd)-fpa);cos(</pre>
	<pre>theta_GPOPS_P42(dd)-fpa);0];</pre>
1252	
1253	<pre>[rgi4(dd,:),vgi4(dd,:)] = PQW_to_IJK(rg_pqw4,vg_pqw4,inc,</pre>
	RAAN,w);
1254	end
1255	
1256	%%
1257	
1258	%save optimal path in structure
1259	<pre>optans2.ics = struct('r0',r0vec,'v0',v0vec,'t0',t0,'ae',ae,'be',</pre>
	<pre>be,'Rmax',Rmax,'Rmin',Rmin,'latlim',latlim,'longlim',longlim</pre>
	, 'GMST0', GMST0,

1260	<pre>'inc',inc,'RAAN',RAAN,'w',w,'ang',ang);</pre>
1261	<pre>optans2.scale = struct('TU',TU,'DU',DU,'MU',MU2);</pre>
1262	<pre>optans2.entry(1) = struct('lat_enter',lat_enter,'long_enter',</pre>
	<pre>long_enter,'r_ell',r_ell,'rtijk',rgi(end,:),'vtijk',vgi(end</pre>
	,:),'rt_pqw',rg_pqw,'rf_pqw',rf_pqw,
1263	<pre>'lat_act_enter',lat_act_enter,'long_act_enter',</pre>
	<pre>long_act_enter,'rf1',rf1,'vf1',vf1,'tf1',tf1);</pre>
1264	<pre>optans2.entry(2) = struct('lat_enter',lat_enter2exp,'long_enter'</pre>
	<pre>,long_enter2exp,'r_ell',r_ell2,'rtijk',rgi2(end,:),'vtijk',</pre>
	<pre>vgi2(end,:),'rt_pqw',rt2,'rf_pqw',rf_pqw2,</pre>
1265	<pre>'lat_act_enter',lat_act_enter2,'long_act_enter',</pre>
	<pre>long_act_enter2,'rf1',rf2exp,'vf1',vf2exp,'tf1',tf2exp);</pre>
1266	<pre>optans2.entry(4) = struct('lat_enter',lat_enter4exp,'long_enter'</pre>
	<pre>,long_enter4exp,'r_ell',r_ell4,'rtijk',rgi4(end,:),'vtijk',</pre>
	<pre>vgi4(end,:),'rt_pqw',rt4,'rf_pqw',rf_pqw4,</pre>
1267	<pre>'lat_act_enter',lat_act_enter4,'long_act_enter',</pre>
	<pre>long_act_enter4,'rf1',rf4exp,'vf1',vf4exp,'tf1',tf4exp);</pre>
1268	<pre>optans2.phase(1) = struct('state', solution2.phase(1).state,'</pre>
	<pre>costate',solution2.phase(1).costate,'control',solution2.</pre>
	<pre>phase(1).control,'time',tvec_P12,'rgi',rgi);</pre>
1269	<pre>optans2.phase(2) = struct('state', solution2.phase(2).state,'</pre>
	<pre>costate',solution2.phase(2).costate,'control',solution2.</pre>
	<pre>phase(2).control,'time',tvec_P22,'rgi',rgi2);</pre>
1270	<pre>optans2.phase(3) = struct('state',solution2.phase(3).state,'</pre>
	<pre>costate',solution2.phase(3).costate,'control',solution2.</pre>
	<pre>phase(3).control,'time',tvec_P32,'rgi',rgi3);</pre>
1271	<pre>optans2.phase(4) = struct('state', solution2.phase(4).state,'</pre>
	<pre>costate',solution2.phase(4).costate,'control',solution2.</pre>
	<pre>phase(4).control,'time',tvec_P42,'rgi',rgi4);</pre>
1272	optans2.parameter = solution2.parameter;
1273	
1274	r0string = num2str(norm(r0vec));

1275	<pre>aestr = num2str(ae);</pre>
1276	<pre>itstr = num2str(PS0_data(cc,end));</pre>
1277	<pre>tempstr = [aestr itstr];</pre>
1278	<pre>aestring = num2str(tempstr);</pre>
1279	
1280	<pre>dir = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\</pre>
	Triple Pass\Images\';
1281	<pre>dir2 = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\</pre>
	Triple Pass\Data\';
1282	
1283	<pre>tend = toc(tstart);</pre>
1284	
1285	<pre>exflag = output.result.nlpinfo;</pre>
1286	
1287	<pre>fid2 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust</pre>
	<pre>RTM\Triple Pass\Data\PS02GP0PSTriplePassData120.txt','a');</pre>
1288	
1289	<pre>fprintf(fid2,'%i\t %i\t %4.3f\t %4.3f\t %4.3f\t %6.5f\t %6.5f\t</pre>
	%6.5f\t %6.2f\t %i\r\n',
1290	<pre>norm(r0),ae,phi_GPOPS_P12,phi_GPOPS_P22,phi_GPOPS_P32,</pre>
	<pre>PS0_data(cc,18),Cost,Cost2,tend,exflag);</pre>
1291	
1292	<pre>fid3 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust</pre>
	<pre>RTM\Triple Pass\Data\TriplePassCost120.txt','a');</pre>
1293	<pre>fprintf(fid3,'%i\t %i \t %4.3f\t %4.3f\t %4.3f\t %4.3f\t %i\t\r</pre>
	\n',
1294	<pre>norm(r0),ae,solution2.phase(1).integral*DU/TU*1000,solution2</pre>
	.phase(2).integral*DU/TU*1000,solution2.phase(4).
	<pre>integral*DU/TU*1000,Cost2,exflag);</pre>
1295	
1296	<pre>if exflag == 0</pre>
1297	%plot optimal results

```
[optout] = LT_TRIPLE_PASS_PLOTS(optans2,r0string,aestring,
1298
                      dir);
1299
                  dataname = ['TriplePass' r0string aestring];
1300
1301
                  %save data
1302
                  save(strcat(dir2,[dataname]),'output');
1303
             end
1304
1305
             clear optans
1306
             close all
1307
1308
        end
1309
1310 end
```

E.3.2.2 Triple Pass LTRTM Equations of Motion and Cost Function

```
14 T1 = u1(:,1);
```

```
15 B1 = u1(:,2);

16

17 MU2 = input.auxdata.MU;

18

19 r_dot1 = vr1;

20 theta_dot1 = vtheta1./r1;

21 vr_dot1 = (vtheta1.^2)./r1 - MU2./(r1.^2) + T1.*sin(B1);

22 vtheta_dot1 = -vtheta1.*vr1./r1 + T1.*cos(B1);

23

24 % Form matrix output

25 daeout1 = [r_dot1 theta_dot1 vr_dot1 vtheta_dot1];

26

27 phaseout(1).dynamics = daeout1;

28 %
```

```
29 % Cost Function
30 phaseout(1).integrand = T1;
31
32 %% Phase 2
33
34 s2 = input.phase(2).state;
35 u2 = input.phase(2).control;
36
37 % Equations of Motion
38 %
```

```
39 r2 = s2(:,1);
40 vr2 = s2(:,3);
41 vtheta2 = s2(:,4);
42
```

```
43 T2 = u2(:,1);
44 B2 = u2(:,2);
45
46 r_dot2 = vr2;
47 theta_dot2 = vtheta2./r2;
48 vr_dot2 = (vtheta2.^2)./r2 - MU2./(r2.^2) + T2.*sin(B2);
49 vtheta_dot2 = -vtheta2.*vr2./r2 + T2.*cos(B2);
50
51 % Form matrix output
52 daeout2 = [r_dot2 theta_dot2 vr_dot2 vtheta_dot2];
53
54 phaseout(2).dynamics = daeout2;
55 %
```

```
56 % Cost Function
57 phaseout(2).integrand = T2;
58
59 %% Phase 3
60
61 s3 = input.phase(3).state;
62 u3 = input.phase(3).control;
63
64 % Equations of Motion
65 %
```

```
66 r3 = s3(:,1);
67 vr3 = s3(:,3);
68 vtheta3 = s3(:,4);
69
70 T3 = u3(:,1);
```

```
71 B3 = u3(:,2);
72
73 r_dot3 = vr3;
74 theta_dot3 = vtheta3./r3;
75 vr_dot3 = (vtheta3.^2)./r3 - MU2./(r3.^2) + T3.*sin(B3);
76 vtheta_dot3 = -vtheta3.*vr3./r3 + + T3.*cos(B3);
77
78 % Form matrix output
79 daeout3 = [r_dot3 theta_dot3 vr_dot3 vtheta_dot3];
80
81 phaseout(3).dynamics = daeout3;
82 %
```

```
% Cost Function
phaseout(3).integrand = zeros(length(r3),1);
%
% Phase 4
%
% Phase 4
%
u4 = input.phase(4).state;
u4 = input.phase(4).control;
%
u5 Equations of Motion
2 %
```

```
93 r4 = s4(:,1);
94 vr4 = s4(:,3);
95 vtheta4 = s4(:,4);
96
97 T4 = u4(:,1);
98 B4 = u4(:,2);
```

```
99
100 r_dot4 = vr4;
  theta_dot4 = vtheta4./r4;
101
  vr_dot4 = (vtheta4.^2)./r4 - MU2./(r4.^2) + T4.*sin(B4);
102
  vtheta_dot4 = -vtheta4.*vr4./r4 + T4.*\cos(B4);
103
104
  % Form matrix output
105
  daeout4 = [r_dot4 theta_dot4 vr_dot4 vtheta_dot4];
106
107
  phaseout(4).dynamics = daeout4;
108
109 %
       _____
```

```
110 % Cost Function
```

phaseout(4).integrand = T4;

E.3.2.3 Triple Pass LTRTM Constraints

```
11 %% Phase 1 (First Maneuver)
12 %phase 2 variables
13 tf1 = input.phase(1).finaltime;
14 xf1 = input.phase(1).finalstate;
15 p = input.parameter;
  phi = p(1);
16
17
  %phase 2 variables
18
  t02 = input.phase(2).initialtime;
19
  tf2 = input.phase(2).finaltime;
20
x02 = input.phase(2).initialstate;
22 xf2 = input.phase(2).finalstate;
_{23} phi2 = p(2);
24
25 %phase 3 variables
  t03 = input.phase(3).initialtime;
26
27 tf3 = input.phase(3).finaltime;
  x03 = input.phase(3).initialstate;
28
  xf3 = input.phase(3).finalstate;
29
30
31
  %phase 3 variables
32
  t04 = input.phase(4).initialtime;
33
  tf4 = input.phase(4).finaltime;
34
 x04 = input.phase(4).initialstate;
35
 xf4 = input.phase(4).finalstate;
36
  phi3 = p(3);
37
38
39 rf = xf1(1);
40 thetaf = xf1(2);
41 Vrf = xf1(3);
42 Vtf = xf1(4);
```

```
44 ae1 = input.auxdata.ae; %semimajor axis of exclusion ellipse
45 be1 = input.auxdata.be; % semiminor axis of exclusion ellipse
46 MU2 = input.auxdata.MU; %gravitational parameter scaled by DU and TU
47 rf_pqw = input.auxdata.rf_pqw; % perifocal position vector of initial
      corssing into exclusion zone
48 vunit = input.auxdata.vunit; %perifocal unit velocity vector of initial
      crossing into exclusion zone
49 gunit = input.auxdata.gunit; %perifocal unit vetcor of initial crossing
      into exclusion zone
50
  term1 = (be1*cos(phi))^2 + (ae1*sin(phi))^2;
51
52
53 re = ae1*be1/sqrt(term1);
54
  rt = rf_pqw + re*cos(phi)*vunit + re*sin(phi)*gunit;
55
56
57 %final position constraints
  event1 = rf*cos(thetaf) - rt(1);
58
  event2 = rf*sin(thetaf) - rt(2);
59
60
61 %velocity magnitude and flight path angle
62 Vf_mag = sqrt(Vrf<sup>2</sup> + Vtf<sup>2</sup>);
  fpa = atan(Vrf/Vtf);
63
64
65 %perifocal velocity
66 vt = Vf_mag*[-sin(thetaf-fpa); cos(thetaf-fpa); 0];
67
68 [a,ecc,~,~,~,~] = RV2COE_MU(rt,vt,MU2);
69 Ra = a^{*}(1+ecc);
70 Rp = a*(1-ecc);
71
```

```
358
```

```
72 event3 = Ra;
73 event4 = Rp;
74
75
76
77 % Linkage Constraints
78 event1_link_state = x02 - xf1;
 event1_link_time = t02 - tf1;
79
80
81 output.eventgroup(1).event = [event1_link_state event1_link_time event1
      event2 event3 event4];
82
83 %% Phase 2 (Second Maneuver)
84
85 % constant variables
% inc = input.auxdata.inc; %inclination of initial orbit (used to convert
      everything into perifocal frame of initial orbit)
87 RAAN = input.auxdata.RAAN; %RAAN of initial orbit (used to convert
      everything into perifocal frame of initial orbit)
88 w = input.auxdata.w; %argument of perigee of initial orbit (used to
      convert everything into perifocal frame of initial orbit)
89 latlim = input.auxdata.latlim;
90 longlim = input.auxdata.longlim;
91 GMST0 = input.auxdata.GMST0;
92 OmegaEarth = input.auxdata.OmegaEarth;
93 DU = input.auxdata.DU;
94 TU = input.auxdata.TU;
95
96
97 rf2 = xf2(1);
98 thetaf2 = xf2(2);
99 Vrf2 = xf2(3);
```

```
359
```

```
Vtf2 = xf2(4);
100
101
  %position and velocity of initial intercept in perifocal frame of
102
       initial
  %orbit
103
  if isnan(rf) == 1 \mid | isnan(thetaf) == 1 \mid | isnan(Vf_mag) == 1 \mid | isnan(
104
       tf1) == 1 || isnan(phi) == 1 ...
       || isnan(rf2) == 1 || isnan(thetaf2) == 1 || isnan(tf2) == 1 ||
105
           isnan(phi2) == 1
           event21 = NaN;
106
           event22 = NaN;
107
           event25 = NaN;
108
           event23 = NaN;
109
           event24 = NaN;
110
          Vf_mag2 = NaN;
111
  else
112
113 [rt_ijk,vt_ijk] = PQW_to_IJK(rt,vt,inc,RAAN,w);
  rt_ijk = rt_ijk*DU;
114
   vt_ijk = vt_ijk*DU/TU;
115
116
   [r2,v2,t2,~,~,~,~,t2_exit] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0+
117
       OmegaEarth*tf1*TU,0,latlim,longlim);
118
   [rf_pqw2,vf_pqw2] = IJK_to_PQW(r2,v2,inc,RAAN,w);
119
120
   rf_pqw2 = rf_pqw2/DU;
121
   vf_pqw2 = vf_pqw2/DU*TU;
122
123
  vunit2 = vf_pqw2/norm(vf_pqw2);
124
125 hfp2 = cross(rf_pqw2,vf_pqw2);
  hunit2 = hfp2/norm(hfp2);
126
127
```

```
360
```

```
gunit2 = cross(vunit2, hunit2);
128
129
   term12 = (be1*cos(phi2))^2 + (ae1*sin(phi2))^2;
130
   re2 = ae1*be1/sqrt(term12);
131
132
   rt2 = rf_pqw2 + re2*cos(phi2)*vunit2 + re2*sin(phi2)*gunit2;
133
134
   %final position constraints
135
   event21 = rf2*cos(thetaf2) - rt2(1);
136
   event22 = rf2*sin(thetaf2) - rt2(2);
137
138
   %apogee and perigee constraints
139
  Vf_mag2 = sqrt(Vrf2^2 + Vtf2^2);
140
   fpa2 = atan(Vrf2/Vtf2);
141
142
   %perifocal velocity
143
   vt2 = Vf_mag2*[-sin(thetaf2-fpa2);cos(thetaf2-fpa2);0];
144
145
   [a2,ecc2,~,~,~,~] = RV2COE_MU(rt2,vt2,MU2);
146
  Ra2 = a2*(1+ecc2);
147
  Rp2 = a2*(1-ecc2);
148
149
   event23 = Ra2;
150
   event24 = Rp2;
151
152
   event25 = tf2 - (tf1 + t2/TU);
153
   end
154
155
156 % Linkage Constraints
  event2_link_state = x03 - xf2;
157
   event2_link_time = t03 - tf2;
158
159
```

```
160 output.eventgroup(2).event = [event2_link_state event2_link_time event21
        event22 event25 event23 event24];
161
   %% Phase 3 (Coast Phase)
162
163
  rf3 = xf3(1);
164
   thetaf3 = xf3(2);
165
  Vrf3 = xf3(3);
166
   Vtf3 = xf3(4);
167
168
169
170 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
171
                 || isnan(tf2) == 1 || isnan(phi2) == 1 ...
            || isnan(t2_exit)
172
       event31 = NaN;
173
   else
174
       event31 = tf3 - (tf1+t2_exit/TU);
175
176
   end
177
   event3_link_state = x04 - xf3;
178
   event3_link_time = t04 - tf3;
179
180
181
   output.eventgroup(3).event = [event3_link_state event3_link_time event31
182
       ];
183
   %% Phase 4 (Third Maneuver)
184
185
186
187 \text{ rf4} = xf4(1);
```

```
188 thetaf4 = xf4(2);
   Vrf4 = xf4(3);
189
   Vtf4 = xf4(4);
190
191
192 %position and velocity of initial intercept in perifocal frame of
       initial
  %orbit
193
  if isnan(rf) == 1 || isnan(thetaf) == 1 || isnan(Vf_mag) == 1 || isnan(
194
       tf1) == 1 || isnan(phi) == 1 ...
            || isnan(rf2) == 1 || isnan(thetaf2) == 1 || isnan(Vf_mag2) == 1
195
                || isnan(tf2) == 1 || isnan(phi2) == 1 ...
            || isnan(tf3) == 1 || isnan(Vrf3) == 1 || isnan(Vtf3) == 1 ||
196
               isnan(thetaf3) == 1 || isnan(rf3) == 1
       event41 = NaN;
197
       event42 = NaN;
198
       event45 = NaN;
199
       event43 = NaN;
200
       event44 = NaN;
201
   else
202
203
       %apogee and perigee constraints at terminus of third phase
204
       Vf_mag3 = sqrt(Vrf3^2 + Vtf3^2);
205
       fpa3 = atan(Vrf3/Vtf3);
206
207
       %perifocal velocity at terminus of thrid phase
208
       vt3 = Vf_mag3*[-sin(thetaf3-fpa3);cos(thetaf3-fpa3);0];
209
210
       rt3 = [rf3*cos(thetaf3);rf3*sin(thetaf3);0];
211
212
       [rt_ijk3,vt_ijk3] = PQW_to_IJK(rt3,vt3,inc,RAAN,w);
213
       rt_ijk3 = rt_ijk3*DU;
214
       vt_ijk3 = vt_ijk3*DU/TU;
215
```

```
217
       [r4,v4,t4] = zone_entry_exit2(rt_ijk3,vt_ijk3,GMST0+OmegaEarth*(tf3)
           *TU,0,latlim,longlim);
218
       [rf_pqw4,vf_pqw4] = IJK_to_PQW(r4,v4,inc,RAAN,w);
219
220
       rf_pqw4 = rf_pqw4/DU;
221
       vf_pqw4 = vf_pqw4/DU*TU;
222
223
224
       vunit4 = vf_pqw4/norm(vf_pqw4);
       hfp4 = cross(rf_pqw4,vf_pqw4);
225
       hunit4 = hfp4/norm(hfp4);
226
227
228
       gunit4 = cross(vunit4, hunit4);
229
       term14 = (be1*cos(phi3))^2 + (ae1*sin(phi3))^2;
230
       re4 = ae1*be1/sqrt(term14);
231
232
       rt4 = rf_pqw4 + re4*cos(phi3)*vunit4 + re4*sin(phi3)*gunit4;
233
234
       %final position constraints
235
       event41 = rf4*cos(thetaf4) - rt4(1);
236
       event42 = rf4*sin(thetaf4) - rt4(2);
237
238
       %apogee and perigee constraints
239
       Vf_mag4 = sqrt(Vrf4^2 + Vtf4^2);
240
       fpa4 = atan(Vrf4/Vtf4);
241
242
       %perifocal velocity
243
       vt4 = Vf_mag4*[-sin(thetaf4-fpa4);cos(thetaf4-fpa4);0];
244
245
       [a4,ecc4,~,~,~,~] = RV2COE_MU(rt4,vt4,MU2);
246
```

```
Ra4 = a4*(1+ecc4);
247
       Rp4 = a4*(1-ecc4);
248
249
250
       event43 = Ra4;
       event44 = Rp4;
251
252
       event45 = tf4 - (tf3 + t4/TU);
253
   end
254
255
256
257 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

```
1 %%
2
3 %maximum number of entries into exclusion zone before maneuver is
4 %required
5 close all
6 % clear all
7 clc
8
9 el_val_pass = 0;
10 el_val_shadow = 1;
11 GMST0 = 0;
12 lat_site = pi/4;
13 long_site = 0;
14 t0 = 0;
15 tf_max = 36*3600;
16 tstep = 1;
17 r_cyl = 1;
18 \text{ xmin} = 1;
19 xmax = 3;
20 %design variables
21 %entry and exit locations on relative lobe
22 % psi0 = 0;
23 % psif = pi;
24 % x_cyl_in = 5;
25 % x_cyl_out = xmin;
26 %
```

```
27 % coast0 = 0;
_{28} % coastf = 0;
29
30
31 wgs84data
32 global MU OmegaEarth RE
33
34 %% Determine Chief Satellite Entry/Exit over Exclusion Zone
35
36 %Initial COEs of chief satellite
37 a_chief_vec = [26581.76 7378 6878];
38 e_chief = 0;
39 i_chief = 55*pi/180;
40 0_chief = 0;
41 o_chief = 0;
42 % nu_chief_vec = 0;
43 nu_chief = 0;
44
45 chief_params = [e_chief;i_chief;0_chief;o_chief;nu_chief];
46 %Initial COEs of deputy satellite
a_{dep} = 6578;
_{48} e_{dep} = 0;
49 i_dep = 55*pi/180;
50 0_dep = 0;
51 o_dep = 0;
52 nu_dep0 = 0;
53
54 dep_params = [a_dep;e_dep;i_dep;0_dep;o_dep];
55
56 a_GEO = 42164.14;
57 e_GE0 = 0;
58 i_GE0 = 0;
```

```
59 O_GEO = 0;
60 O_{GEO} = 0;
61
  GEO_params = [a_GEO;e_GEO;i_GEO;O_GEO;o_GEO];
62
63
64
  for aa = 1:length(a_chief_vec)
65
       [C_times_c,C_ind_c,Rijk_c,Vijk_c,~,~,rho_vec_cw_c,Tvec_c,~,~] =
66
          contact_times(a_chief_vec(aa),i_chief,e_chief,0_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);</pre>
67
       max_coastf = C_times_c(max(val_ind)+1,1) - C_times_c(max(val_ind),2);
68
       C_times_c = C_times_c(val_ind,:);
69
       [max_ind,~] = size(C_times_c);
70
       ThreePassEnumData(aa).times = C_times_c;
71
       ThreePassEnumData(aa).ind = C_ind_c;
72
       ThreePassEnumData(aa).Rc = Rijk_c;
73
       ThreePassEnumData(aa).Vc = Vijk_c;
74
       ThreePassEnumData(aa).rho_c = rho_vec_cw_c;
75
       ThreePassEnumData(aa).Tc = Tvec_c;
76
       ThreePassEnumData(aa).max_ind = max_ind;
77
       ThreePassEnumData(aa).max_coastf = max_coastf;
78
       clear C_times_c C_ind_c Rijk_c Vijk_c rho_vec_cw_c Tvec_c max_ind
79
          max_coastf
  end
80
81
82 save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
      Article_Data\Rev2\ThreePassEnumData.mat', 'ThreePassEnumData')
83
_{84} for bb = 3:3
85
```

```
C_times_c = ThreePassEnumData(bb).times;
86
       C_ind_c = ThreePassEnumData(bb).ind;
87
       Rijk_c = ThreePassEnumData(bb).Rc;
88
       Vijk_c = ThreePassEnumData(bb).Vc;
89
       rho_vec_cw_c = ThreePassEnumData(bb).rho_c;
90
       Tvec_c = ThreePassEnumData(bb).Tc;
91
       max_ind = ThreePassEnumData(bb).max_ind;
92
93
94
       a_chief = a_chief_vec(bb);
95
       if C_times_c(1,1) == 0
96
            min_start = 2;
97
       else
98
99
            min_start = 1;
       end
100
101
       for cc = 14:max_ind
102
            if cc == 14
103
                startval = 19;
104
            else
105
                startval = 1;
106
107
            end
108
            for dd = startval:20
109
110
                tstart = tic;
111
                %Period of Chief satellite's orbit
112
                Pc = 2*pi*sqrt(a_chief^3/MU);
113
114
115
                t_enter = C_times_c(cc,1);
116
117
                t_exit = C_times_c(cc,2);
```

118	t_zone = t_exit - t_enter;
119	
120	%determine indices of minimum duration contact
121	<pre>C_ind_contact = C_ind_c(cc,:);</pre>
122	
123	%find unit vector pointing towards the deputy that puts
	chief between
124	%ground site and deputy
125	<pre>rho_unit_cw = rho_vec_cw_c(:,C_ind_contact(1):C_ind_contact</pre>
	(2));
126	
127	%determine alpha and beta angles during contact times
128	[alphavec,betavec] = alphabeta(rho_unit_cw);
129	
130	%Vector of times for propogation
131	<pre>T_prop = Tvec_c(C_ind_contact(1):C_ind_contact(2)) - Tvec_c(</pre>
	C_ind_contact(1))*ones(length(Tvec_c(C_ind_contact(1):
	<pre>C_ind_contact(2)),1);</pre>
132	
133	%Determine position/velocity vectors of chief satellite upon
	intial/final
134	%contact
135	<pre>chief_pos0 = Rijk_c(:,C_ind_c(cc,1));</pre>
136	<pre>chief_vel0 = Vijk_c(:,C_ind_c(cc,1));</pre>
137	<pre>chief_posf = Rijk_c(:,C_ind_c(cc,2));</pre>
138	<pre>chief_velf = Vijk_c(:,C_ind_c(cc,2));</pre>
139	
140	<pre>if cc < max_ind</pre>
141	<pre>max_coastf = C_times_c(cc+1,1) - C_times_c(cc,2);</pre>
142	else
143	<pre>max_coastf = ThreePassEnumData(bb).max_coastf;</pre>
144	end

145				
146				%time variables have precision to 1 second. Others have
147				%precision to 0.001 units (km,rad)
148				prec2 = [2;0;3;3;0;2;0;6];
149				
150				
151				[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,
152				chief_velf,dep_params,GEO_params,alphavec,betavec,t_zone
				,Pc,t_enter,t_exit,r_cyl,T_prop,GMST0,lat_site,
				<pre>long_site,tstep,el_val_shadow);</pre>
153				
154				<pre>tend = toc(tstart);</pre>
155				
156				<pre>infvec = find(JG == Inf);</pre>
157				<pre>inftot = length(infvec);</pre>
158				
159				<pre>fid = fopen('C:\Users\Dan Showalter\Documents\MATLAB\PS0\</pre>
				Relative Motion\Article_Data\Rev2\
				ThreePassEnumerationSat3.txt','a');
160				<pre>fprintf(fid,'%2i \t\t %2i \t\t %2i \t\t %3.2f \t\t %6i \t\t</pre>
				%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
				<pre>(2),gbest(3),gbest(4),gbest(5),gbest(6),gbest(7),Jmin,k,</pre>
				<pre>inftot,tend);</pre>
161				
162			end	
163		end		
164	end			

F.1.1.1 Determine Target Contact Times with Ground Site

```
1 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)
2
3 %INPUTS
4 % a = satellite semimajor axis (km)
5 % inc = satellite inclination (rad)
6 % ecc = satellite eccentricity
7 % Omega = satellite RAAN (rad)
8 % omega = satellite argument of perigee (rad)
9 % nu0 = initial true anomaly (rad)
10 % lambda0 = initial GMST of ground site
11 % tmax = maximum scenario time (sec)
12 % tstep = time step (sec)
13
14 %OUTPUTS
15 % C_times = times satellite is in contact with the ground site
16 % C_ind = indices of satellite contat times
17 % rijk = position vectors of satellite at discretized times
18 % vijk = velocity vectors of satellite at discretized times
19 % rgs = position vectors of the ground site at discretized times
20 % rho_sez = vector from ground site to satellite in SEZ coordinates
21 % rho_RIC = vector from ground site to satellite in RIC coordinates
22 %
     _____
_{23} tvec = (0:tstep:tmax)';
24
25 % determine true anomaly of spacecraft at each time step
26 [nu_vec] = nuf_from_TOF_vec(nu0,tvec,a,ecc);
27
```

```
372
```
```
28 %determine inertial position and velocity vectors at each tiem step
29
  [rijk,vijk,r] = COE2RV_vec(a,ecc,inc,Omega,omega,nu_vec);
30
r = r';
32
 wgs84data
33
34
  global OmegaEarth RE
35
36
  long_site = lambda0 + GMST0 + OmegaEarth*tvec;
37
38
  Rsite = zeros(3,length(tvec));
39
40 Rsite(1,:) = RE*cos(lat_site).*cos(long_site);
41 Rsite(2,:) = RE*cos(lat_site).*sin(long_site);
42 Rsite(3,:) = RE*sin(lat_site);
43
  rho_ijk = rijk - Rsite;
44
45
 temp = zeros(3,length(tvec));
46
47 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
48
      (2,:);
  temp(3,:) = rho_ijk(3,:);
49
50
  rho_sez = zeros(3,length(tvec));
51
52
s3 rho_sez(1,:) = cos(pi/2 - lat_site)*temp(1,:) - sin(pi/2 - lat_site)*
      temp(3,:);
_{54} rho_sez(2,:) = temp(2,:);
s5 rho_sez(3,:) = sin(pi/2 - lat_site)*temp(1,:) + cos(pi/2 - lat_site)*
      temp(3,:);
```

```
56
57
58
59
60
61
62
63 \text{ ind}_1 = 1;
  while ind_l == 1
64
65
      ind7 = find(long_site > 2*pi);
66
      if isempty(ind7)
67
          ind_1 = 0;
68
      else
69
      long_site(ind7) = long_site(ind7) - 2*pi;
70
      end
71
72
73
  end
74
75 uvec = omega + nu_vec;
76
  zone_ind = zeros(length(tvec),1);
77
  num_in = 0;
78
79
80 %inertial coordinates of ground site
s1 rgs = zeros(3,length(tvec));
s2 rgs(1,:) = RE*cos(lat_site)*cos(long_site');
83 rgs(2,:) = RE*cos(lat_site)*sin(long_site');
84 rgs(3,:) = RE*sin(lat_site);
85
86 %vector from ground site to satellite
87 rho_ijk = rijk - rgs;
```

```
89 %transform into sez coordinates
90 temp = zeros(3,length(tvec));
91 temp(1,:) = cos(long_site').*rho_ijk(1,:) + sin(long_site').*rho_ijk
       (2,:);
92 temp(2,:) = -sin(long_site').*rho_ijk(1,:) + cos(long_site').*rho_ijk
       (2,:);
  temp(3,:) = rho_ijk(3,:);
93
94
  rho_sez = zeros(3,length(tvec));
95
96
97 rho_sez(1,:) = cos(pi/2 - lat_site)*temp(1,:) - sin(pi/2 - lat_site)*
       temp(3,:);
  rho_sez(2,:) = temp(2,:);
98
  rho_sez(3,:) = sin(pi/2 - lat_site)*temp(1,:) + cos(pi/2 - lat_site)*
99
       temp(3,:);
100
   rho_mag = sqrt(rho_sez(1,:).^2 + rho_sez(2,:).^2 + rho_sez(3,:).^2);
101
102
   el_vec = asind(rho_sez(3,:)./rho_mag);
103
104
105
   for aa = 1:length(tvec)
106
       if el_vec(aa) > el_val
107
           zone_ind(aa) = 1;
108
           if aa == 1
109
                num_in = num_in + 1;
110
                C_{times(num_in,1)} = tvec(aa);
111
                C_ind(num_in, 1) = aa;
112
           else
113
                if zone_ind(aa - 1) == 0
114
115
                    num_in = num_in + 1;
```

88

```
C_times(num_in,1) = tvec(aa);
116
                     C_ind(num_in, 1) = aa;
117
                end
118
            end
119
       else
120
            zone_ind(aa) = 0;
121
            if aa ~= 1
122
                if zone_ind(aa - 1) == 1
123
                     C_times(num_in,2) = tvec(aa-1);
124
                     C_ind(num_in, 2) = aa - 1;
125
                end
126
127
            end
       end
128
       if aa == length(tvec) && zone_ind(aa -1) == 1
129
            C_times(num_in,2) = tvec(aa);
130
            C_ind(num_in, 2) = aa;
131
       end
132
   end
133
134
   %convert ijk to RIC
135
   temp2 = zeros(3,length(tvec));
136
   temp2(1,:) = cos(Omega)*rho_ijk(1,:) + sin(Omega)*rho_ijk(2,:);
137
   temp2(2,:) = cos(Omega)*rho_ijk(2,:) - sin(Omega)*rho_ijk(1,:);
138
   temp2(3,:) = rho_ijk(3,:);
139
140
   temp3 = zeros(3,length(tvec));
141
   temp3(1,:) = temp2(1,:);
142
   temp3(2,:) = cos(inc)*temp2(2,:) + sin(inc)*temp2(3,:);
143
   temp3(3,:) = cos(inc)*temp2(3,:) - sin(inc)*temp2(2,:);
144
145
   rho_RIC = zeros(3,length(tvec));
146
147
```

```
rho_RIC(1,:) = cos(uvec').*temp3(1,:) + sin(uvec').*temp3(2,:);
148
   rho_RIC(2,:) = cos(uvec').*temp3(2,:) - sin(uvec').*temp3(1,:);
149
   rho_RIC(3,:) = temp3(3,:);
150
151
   norm_vec = sqrt(rho_RIC(1,:).^2 + rho_RIC(2,:).^2 + rho_RIC(3,:).^2);
152
153
   rho_RIC(1,:) = rho_RIC(1,:)./norm_vec;
154
   rho_RIC(2,:) = rho_RIC(2,:)./norm_vec;
155
   rho_RIC(3,:) = rho_RIC(3,:)./norm_vec;
156
157
   if num_in == 0
158
       C_{times} = 0;
159
       C_ind = 0;
160
161
   end
```

F.1.1.2 Determine Final True Anomaly Given Time of Flight

```
1 function [nuf] = nuf_from_TOF_vec(nu0,TOF_vec,a,e)
2
3 wgs84data
  global MU
4
5
6 nuf = zeros(length(TOF_vec),1);
7 Eg = zeros(length(TOF_vec),1);
8
9
  %% 1) compute orbital mean motion
10
n = sqrt(MU/abs(a)^3);
12
13
  %% 2) convert initial true anomaly to initial mean anomaly
14
 if e < 1
15
      if nu0 == 0;
16
```

```
MO = O;
17
18
       elseif nu0 == pi
19
20
            M0 = pi;
21
22
       else
23
24
            E0 = acos((e+cos(nu0))/(1+e*cos(nu0)));
25
26
27
            if (nu0 > pi)
28
29
                E0 = 2*pi - E0;
30
31
            end
32
33
            M0 = E0 - e^*sin(E0);
34
35
       end
36
37
       M0 = M0*ones(length(TOF_vec),1);
38
39
40
       %% 3) compute final mean anomaly
41
       Mold = M0 + n*TOF_vec;
42
       N = Mold/(2*pi);
43
       Mf = Mold - floor(N)*2*pi;
44
45
46
       Mflag = 1;
47
48
```

```
while Mflag == 1
49
            ind_Mf = find(Mf > 2*pi);
50
51
           if isempty(ind_Mf) == 0
52
53
                Mf(ind_Mf) = Mf(ind_Mf) - 2*pi;
54
            else
55
                Mflag = 0;
56
            end
57
58
       end
59
60
61
       ind_Eg1 = find(Mf > pi);
62
       ind_Eg2 = find(Mf <= pi);</pre>
63
64
       Eg(ind_Eg1) = Mf(ind_Eg1) - e;
65
       Eg(ind_Eg2) = Mf(ind_Eg2) + e;
66
67
       Ef = Eg + (Mf - Eg + e*sin(Eg))./(1 - e*cos(Eg));
68
69
       Eflag = 1;
70
71
       while Eflag == 1
72
            diff = abs(Ef - Eg);
73
74
            ind_Ef = find(diff > 1e-8);
75
76
           if isempty(ind_Ef) == 0
77
                Eg = Ef;
78
                Ef = Eg + (Mf - Eg + e*sin(Eg))./(1 - e*cos(Eg));
79
80
            else
```

```
Eflag = 0;
81
82
            end
        end
83
84
       nuf = acos((cos(Ef)-e)./(1-e*cos(Ef)));
85
86
        ind_quad = find(Ef > pi);
87
88
       nuf(ind_quad) = 2*pi - nuf(ind_quad);
89
90
91
92
  elseif e > 1
93
   %% Hyperbolic orbits
94
        sinh_H0 = sin(nu0)*sqrt(e^2 - 1)/(1+e*cos(nu0));
95
96
       H = zeros(length(TOF_vec),1);
97
       M = zeros(length(TOF_vec),1);
98
       M0 = e*sinh_H0 - asinh(sinh_H0);
99
100
       Mold = M0 + n*TOF_vec;
101
        N = Mold/(2*pi);
102
       M = Mold - floor(N) * 2*pi;
103
104
        if e < 1.6
105
            H = M + e;
106
            ind1 = find(-pi < M & M < 0);
107
            ind4 = find(M > pi);
108
            H(ind1) = M(ind1) - e;
109
            H(ind4) = M(ind4) - e;
110
        else
111
            H = M/(e - 1);
112
```

```
if e < 3.6
113
                 ind2 = find(abs(M) > pi);
114
                 if isempty(ind2) == 0
115
                     H(ind2) = M(ind2) - sign(M(ind2))*e;
116
                 end
117
            end
118
        end
119
120
        Hflag = 1;
121
        Hg = H;
122
123
        while Hflag == 1
124
           Hnew = Hg + (M - e*sinh(Hg) + Hg)./(e*cosh(Hg) - 1);
125
126
           diff = abs(Hnew - Hg);
127
128
           ind_H = find(diff > 1e-8);
129
           if isempty(ind_H) == 0
130
                Hg = Hnew;
131
           else
132
                Hflag = 0;
133
           end
134
        end
135
136
        sin_nu = (-sinh(Hnew)*sqrt(e^2 - 1))./(1 - e*cosh(Hnew));
137
138
        cos_nu = (cosh(Hnew) - e)./(1-e*cosh(Hnew));
139
140
       nuf = atan2(sin_nu,cos_nu);
141
142
143
144 end
```

```
145
   if isreal(nuf) == 0
146
       а
147
       e
148
       nu0
149
       save nuf
150
151 end
152 % if e == 1
153 %
        keyboard
154 % end
155 % zer_val = find(nuf == 0);
156 % if zer_val == length(nuf)
157 %
        keyboard
158 % end
```

F.1.1.3 Determine State Given COEs

```
1 function [R_ijk,V_ijk,r] = COE2RV_vec(a,ecc,inc,RAAN,w,nu_vec)
2
3 %Author: Dan Showalter 18 Oct 2012
4
5 %Purpose: find inertial position and velocity vector gievn classical
6 %orbital elements
7
8 %% Algorithm
9 MU = 398600.5;
10
ii dim = length(nu_vec);
12
13 p = a*(1-ecc^2);
14
15 r = p./(1 + ecc * cos(nu_vec));
16
```

```
17
18 R_pqw = zeros(3, dim);
19 V_pqw = zeros(3, dim);
20 R_{ijk} = zeros(3, dim);
V_i = zeros(3, dim);
22
23
24
25 R_pqw(1,:) = (r.*cos(nu_vec))';
R_pqw(2,:) = (r.*sin(nu_vec))';
27 V_pqw(1,:) = sqrt(MU/p)*(-sin(nu_vec)');
  V_pqw(2,:) = sqrt(MU/p)*(ecc+cos(nu_vec))';
28
29
30
  %first rotation about vertical axis by -w
31 R_temp1(1,:) = cos(-w)*R_pqw(1,:) + sin(-w)*R_pqw(2,:);
32 R_temp1(2,:) = cos(-w)*R_pqw(2,:) - sin(-w)*R_pqw(1,:);
R_{temp1(3,:)} = R_pqw(3,:);
34
  V_temp1(1,:) = cos(-w)*V_pqw(1,:) + sin(-w)*V_pqw(2,:);
35
  V_{temp1(2,:)} = \cos(-w) * V_{pqw}(2,:) - \sin(-w) * V_{pqw}(1,:);
36
V_{temp1(3,:)} = V_pqw(3,:);
38
  %2nd rotation about primary axis by -inc
39
  R_{temp2(1,:)} = R_{temp1(1,:)};
40
41 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,:);
42
43
 V_{temp2(1,:)} = V_{temp1(1,:)};
44
45 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,:);
46
47
48 %3rd rotation about vertical axis by -RAAN
```

```
49 R_ijk(1,:) = cos(-RAAN)*R_temp2(1,:) + sin(-RAAN)*R_temp2(2,:);
50 R_ijk(2,:) = cos(-RAAN)*R_temp2(2,:) - sin(-RAAN)*R_temp2(1,:);
51 R_ijk(3,:) = R_temp2(3,:);
52 
53 V_ijk(1,:) = cos(-RAAN)*V_temp2(1,:) + sin(-RAAN)*V_temp2(2,:);
54 V_ijk(2,:) = cos(-RAAN)*V_temp2(2,:) - sin(-RAAN)*V_temp2(1,:);
55 V_ijk(3,:) = V_temp2(3,:);
```

F.1.1.4 Inner Loop PSO Algorithm

2

```
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)

```
3
4
5
6 %Author: Dan Showalter 18 Oct 2012
7
8 %Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem
9
10 %generic PSO variable
11
  %
      n: # of design variables
      limits: bounds on design variables (n x 2 vector) with first element
12 %
          in row n being lower bound for element n and 2nd element in row
13 %
      n being
          upper bound for element n
14 %
```

```
15 %
       iter: number of iterations
       swarm: swarm size
16
  %
       prec: defines the number of decimal places to keep for each design
  %
17
  %
           variable and the cost function evalution size: (n+1,1)
18
19
  %Problem specific PSO variables
20
21
22
23
  %Specific Problem Variables
24
25
26
27
  %%
28
29
  [N,~] = size(limits);
30
31
32 llim = limits(:,1);
33 ulim = limits(:,2);
34
35 if N^{\sim}=n
       fprintf('Error! limits size does not match number of variables')
36
       stop
37
   end
38
39
40
41
42 gbest = zeros(n,1);
43 x = zeros(n, swarm);
44 v = zeros(n, swarm);
45 pbest = zeros(n,swarm);
46 Jpbest = zeros(swarm,1);
```

```
47 \, d = (ulim - llim);
48 JG = zeros(iter,1);
49 J = zeros(swarm, 1);
50
51 llim2 = ones(n,swarm);
52 ulim2 = ones(n,swarm);
53
54 for aa = 1:n
       llim2(aa,:) = llim(aa)*llim2(aa,:);
55
       ulim2(aa,:) = ulim(aa)*ulim2(aa,:);
56
  end
57
58
59 \ d2 = ulim2 - llim2;
60
61 CoreNum = 12;
  if (matlabpool('size'))<=0</pre>
62
       matlabpool('open','local',CoreNum);
63
  else
64
       disp('Parallel Computing Enabled')
65
  end
66
67 tstart = tic;
  %loop until maximum iteration have been met
68
69
  for k = 1:iter
70
71
       %create particles dictated by swarm size input
72
73
74
       % if this is the first iteration
75
       if k == 1
76
           rng('shuffle');
77
           x = random('unif',llim2,ulim2,[n,swarm]);
78
```

```
v = random('unif',-d2,d2,[n,swarm]);
79
80
           %if this is after the first iteration, update velocity and
81
               position
            %of each particle in the swarm
82
       else
83
            parfor h = 1:swarm
84
                c1 = 2.09;
85
                c2 = 2.09;
86
                phi = c1+c2;
87
                ci = 2/abs(2-phi - sqrt(phi<sup>2</sup> - 4*phi));
88
                cc = c1*random('unif',0,1);
89
                cs = c2*random('unif', 0, 1);
90
91
92
                vdum = v(:,h);
93
                %update velocity
94
                vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x
95
                    (:,h)));
96
97
                %check to make sure velocity doesn't exceed max velocity for
98
                     each
                %variable
99
                for w = 1:n
100
101
                    %if the variable velocity is less than the min, set it
102
                        to the min
                     if vdum(w) < -d(w)
103
                         vdum(w) = -d(w);
104
                         %if the variable velocity is more than the max, set
105
                             it to the max
```

```
387
```

```
elseif vdum(w) > d(w);
106
                          vdum(w) = d(w);
107
                      end
108
                 end
109
110
                 v(:,h) = vdum;
111
112
                 %update position
113
                 xdum = x(:,h) + v(:,h);
114
115
                 for r = 1:n
116
117
                      %if particle has passed lower limit
118
                      if xdum(r) < llim(r)</pre>
119
                          xdum(r) = llim(r);
120
121
                      elseif xdum(r) > ulim(r)
122
                          xdum(r) = ulim(r);
123
                      end
124
125
                      x(:,h) = xdum;
126
127
                 end
128
129
            end
130
131
        end
132
133
        % round variables to get finite precision
134
        parfor aa = 1:n
135
            x(aa,:) = round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
136
            v(aa,:) = round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
137
```

138		end
139		
140		%% ***********************************

141		
142		<pre>xmin = limits(2,1);</pre>
143		<pre>% rel_pos = zeros(3,length(T_prop));</pre>
144		<pre>% rel_pos_box = zeros(3,length(T_prop));</pre>
145		<pre>% temp1 = zeros(1,length(T_prop));</pre>
146		<pre>% temp2 = zeros(1,length(T_prop));</pre>
147		<pre>% temp3 = zeros(1,length(T_prop));</pre>
148	%	<pre>fclose('all');</pre>
149	%	<pre>fid=fopen('K-M.txt','w');</pre>
150	%	<pre>fprintf(fid,'\r\n\r\n\r\n%s %i','K',k);</pre>
151	%	<pre>fid2 = fopen('optvals.txt','w');</pre>
152	%	<pre>fprintf(fid2,'\r\n\r\n%s %i','K',k);</pre>
153		parfor m = 1:swarm
154		% ******************Cost function evaluation here

155		<pre>opt_vars = x(:,m);</pre>
156	%	<pre>fid2=fopen('optvals.txt','a');</pre>
157	%	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),
		<pre>opt_vars(6));</pre>
158	%	<pre>fclose(fid2);</pre>
159		<pre>[J(m)] = rel_shadow_cost_function2(opt_vars,chief_pos0,</pre>
		<pre>chief_vel0,chief_posf,chief_velf,alphavec,betavec,t_zone,Pc,</pre>
		<pre>T_prop,xmin,r_cyl,dep_params,</pre>
160		GEO_params,T_enter,
		T_exit,GMST0,

long_site,

```
lat_site,t_step,
                                                 el_val);
  %
161
  %
162
              fid=fopen('K-M.txt','a')
  %
163
              fprintf(fid,'\r\n%s %i %8.5f','M complete',m,J(m));
  %
164
              fclose(fid);
  %
165
166
      end
167
168
      169
         ******
     %%
170
         %%
171
172
173
     %round cost to nearest precision required
174
      J = round(J*10^{prec}(n+1))/10^{prec}(n+1);
175
176
      if k == 1
177
         count = 0;
178
         Jpbest(1:swarm) = J(1:swarm);
179
         pbest(:,1:swarm) = x(:,1:swarm);
180
181
         [Jgbest,IND] = min(Jpbest(:));
182
183
         gbest(:) = x(:,IND);
184
185
      else
186
187
```

```
for h=1:swarm
188
                  if J(h) < Jpbest(h)</pre>
189
                       Jpbest(h) = J(h);
190
                       pbest(:,h) = x(:,h);
191
                       if Jpbest(h) < Jgbest</pre>
192
193
                            Jgbest = Jpbest(h);
194
                            gbest(:) = x(:,h);
195
196
                       end
197
                  end
198
             end
199
        end
200
201
202
203
        diff = zeros(swarm,1);
204
        parfor y = 1:swarm
205
206
             diff(y) = Jgbest - Jpbest(y);
207
        end
208
209
        indcount = find(abs(diff)<10^(-prec(n+1)));</pre>
210
211
212
213
214
        JG(k) = Jgbest;
215
        JGmin = Jgbest;
216
217
   %
          kinf = 50;
218
          if k > kinf
219 %
```

```
220 %
            if Jgbest == Inf
   %
                   break
221
              end
   %
222
223
   %
          end
224
        if length(indcount) == swarm
225
             ex_flag = 0;
226
             break
227
        end
228
229
        if k > 1
230
             if JG(k) == JG(k-1)
231
                 count = count + 1;
232
             else
233
   %
                   MinCost = Jgbest*1000
234
                    k
   %
235
                 count = 0;
236
237
             end
        end
238
239
        if count > 1000
240
             ex_flag = 1;
241
             break
242
        end
243
   end
244
245
   if k == iter
246
        ex_flag = 2;
247
248 end
```

F.1.1.5 Cost Function Script

	GEO_params,
	T_enter,
	T_exit,
	GMST0,
	long_site
	,
	lat_site
	,t_step,
	el_val)
.000072921151467;	

```
4 OmegaEarth=0
5 RE=6378.137;
             variable definitions
  %
6
7 %
             nu0 = opt_vars(1);
  %
             TOF1 = opt_vars(2);
8
  %
             x_in = opt_vars(3);
9
             x_out = opt_vars(4);
10
  %
             coast3 = opt_vars(5);
  %
11
             nu_GE0 = opt_vars(6);
12
  %
             Tend = opt_vars(7);
  %
13
14
  alpha0 = alphavec(1);
15
16 beta0 = betavec(1);
17 alphaf = alphavec(end);
18 betaf = betavec(end);
19
20 a_dep = dep_params(1);
21 e_dep = dep_params(2);
22 i_dep = dep_params(3);
```

2

3

```
23 O_dep = dep_params(4);
24 o_dep = dep_params(5);
25
26
a_{GEO} = GEO_{params}(1);
e_{GEO} = GEO_{params}(2);
i_GEO = GEO_params(3);
30 \quad O\_GEO = GEO\_params(4);
o_GEO = GEO_params(5);
32
33
34 %determine entry and exit conditions of deputy in cylinder frame
35 \text{ box}_\text{vec}0 = \text{zeros}(3,1);
36 \text{ box}_vec0(1) = opt_vars(3);
37 box_vec0(2) = 0;
38 box_vec0(3) = 0;
39
  [deputy_pos0,rel_pos0] = box2cw(chief_pos0,chief_vel0,box_vec0,alpha0,
40
      beta0);
41
42 box_vecf = zeros(3,1);
43 box_vecf(1) = opt_vars(4);
44 box_vecf(2) = 0;
45 box_vecf(3) = 0;
46
  [deputy_posf,rel_posf] = box2cw(chief_posf,chief_velf,box_vecf,alphaf,
47
      betaf);
48
49 %determine required entry/exit velocities corresponding to entry/exit
      conditions
50 [v0_tilde,vf_tilde] = relative_velocity(t_zone,Pc,rel_pos0,rel_posf);
51
```

```
s2 %propogate (discretely) relative motion for time chief in contact with
53 %ground site
54 [rel_pos] = CW_Motion3(rel_pos0,v0_tilde,T_prop',Pc);
55
  %convert relative position from cw to cylinder frame
56
  temp1 = cos(betavec).*rel_pos(1,:) + sin(betavec).*rel_pos(2,:);
57
  temp2 = cos(betavec).*rel_pos(2,:) - sin(betavec).*rel_pos(1,:);
58
  temp3 = rel_pos(3,:);
59
60
61
  rel_pos_box = zeros(3,length(rel_pos));
62
  rel_pos_box(1,:) = cos(-alphavec).*temp1 - sin(-alphavec).*temp3;
63
64 rel_pos_box(2,:) = temp2;
  rel_pos_box(3,:) = cos(-alphavec).*temp3 + sin(-alphavec).*temp1;
65
66
  [T_out] = out_of_cylinder(rel_pos_box,T_prop',xmin,r_cyl);
67
68
  %propogate (discretely) relative motion for post inspection motion to
69
70 %ensure chaser doesn't intercept chief
  T_post_ci = [(0:opt_vars(7)) opt_vars(7)];
71
72
  [rel_pos2] = CW_Motion3(rel_posf,vf_tilde,T_post_ci,Pc);
73
74 rel_min_vec = sqrt(rel_pos2(1,:).*rel_pos2(1,:) + rel_pos2(2,:).*
      rel_pos2(2,:) + rel_pos2(3,:).*rel_pos2(3,:));
75
76 %closest approach must be more than 50 meters away
77 min_approach = min(rel_min_vec);
78
79 if T_out > 0 || min_approach < 0.05
     J = Inf;
80
81 else
82
```

```
v0_rel = v0_tilde/Pc;
83
       [~,~,ic1,0c1,~,nuc01] = RV2COE(chief_pos0,chief_vel0);
84
       v0_arrive = chief_vel0 + rot3mat(-0c1)*rot1mat(-ic1)*rot3mat(-nuc01)
85
           *v0_rel;
86
                      %determine departure location of maneuvering satellite
       %
87
       nu_dep = opt_vars(1);
88
       [r0_d,v0_d] = COE2RV(a_dep,e_dep,i_dep,0_dep,o_dep,nu_dep);
89
90
       %% solve lambert's problem both ways to get from satellite to lobe
91
           entry condition
       [V1S, V2S] = lambert2(r0_d',deputy_pos0',( opt_vars(2))/(3600*24)
92
           ,0,398600.5);
93
       [V1L, V2L] = lambert2(r0_d',deputy_pos0',-(opt_vars(2))/(3600*24)
94
           ,0,398600.5);
95
       %Depature DV
96
       DV1S = V1S - v0_d';
97
       DV1L = V1L - v0_d';
98
99
       %arrival DV
100
       DV2S = v0_arrive' - V2S;
101
       DV2L = v0_arrive' - V2L;
102
103
       DV_shadeS = norm(DV1S) + norm(DV2S);
104
       DV_shadeL = norm(DV1L) + norm(DV2L);
105
106
       if DV_shadeS < DV_shadeL</pre>
107
           DV = DV_shadeS;
108
           V12_d = V1S';
109
           DV_depart1 = DV1S';
110
```

```
DV_arrive1 = DV2S';
111
       else
112
           DV = DV_shadeL;
113
            V12_d = V1L';
114
            DV_depart1 = DV1L';
115
            DV_arrive1 = DV2L';
116
       end
117
118
       %determine ground site inertial position vectors for duration of
119
           second maneuver
       %(coast0 to Tenter)
120
       Tvec12 = (T_enter-opt_vars(2):t_step:T_enter)';
121
       GMST12 = GMST0*ones(length(Tvec12),1) + OmegaEarth.*Tvec12;
122
       longvec12 = long_site*ones(length(Tvec12),1) + GMST12;
123
124
125
126
       %inertial coordinates of the ground site
127
       Rsite12 = zeros(3,length(Tvec12));
128
       Rsite12(1,:) = RE*cos(lat_site).*cos(longvec12);
129
       Rsite12(2,:) = RE*cos(lat_site).*sin(longvec12);
130
       Rsite12(3,:) = RE*sin(lat_site);
131
132
133
134
       %determine maneuvering spacecraft inertial position vectors for
135
           duration od
       %second maneuver
136
       Tvec12m = Tvec12 - Tvec12(1);
137
138
139
140
       [am12,em12,im12,Om12,om12,num012] = RV2COE(r0_d,V12_d);
```

141 142 if imag(num012) < 1e-6</pre> 143 num012 = real(num012);144 else 145 fid = fopen('error_data.txt','a'); 146 [~,ind_imag] = max(imag(num012)); 147 fprintf(fid,'%s %s','Real','Imag'); 148 fprintf(fid, '\n\r%10.8f %10.8f', real(num012(ind_imag)), imag(149 num012(ind_imag))); end 150 151 152 153 [val] = site_contact_vec(am12,im12,em12,Om12,om12,num012,long_site, 154 GMST12(1),Tvec12m(end),t_step,lat_site,Rsite12,el_val); 155 % T_out2 = length(val)/length(Tvec12m)*Tvec12m(end); 156 157 if isempty(val) == 0 158 J = Inf;159 else 160 161 %% 3rd and 4th Maneuver 162 163 vf_rel = vf_tilde/Pc; 164 [~,~,ic2,0c2,~,nuc02] = RV2COE(chief_posf,chief_velf); 165 vf_depart = chief_velf + rot3mat(-0c2)*rot1mat(-ic2)*rot3mat(-166 nuc02)*vf_rel; 167 %determine orbital parameters of satellite upon zone exit 168 [at,et,it,Ot,ot,nut0] = RV2COE(deputy_posf,vf_depart); 169

170	<pre>nu_tL = nuf_from_TOF(nut0,opt_vars(5),at,et);</pre>
171	<pre>[r_tL,V_tL] = COE2RV(at,et,it,Ot,ot,nu_tL);</pre>
172	
173	
174	%determine arrival location of maneuvering satellite
175	<pre>[r_GE0,V_GE0] = COE2RV(a_GE0,e_GE0,i_GE0,0_GE0,o_GE0,opt_vars(6)</pre>
);
176	
177	%% solve lambert's problem both ways to get from lobe exit
	condition to GEO
178	<pre>[V3S, V4S] = lambert2(r_tL',r_GEO',(opt_vars(7))/(3600*24)</pre>
	,0,398600.5);
179	
180	<pre>[V3L, V4L] = lambert2(r_tL',r_GEO',-(opt_vars(7))/(3600*24)</pre>
	,0,398600.5);
181	
182	%Depature DV
183	DV3S = V3S - V_tL';
184	$DV3L = V3L - V_tL';$
185	
186	%arrival DV
187	$DV4S = V_GEO' - V4S;$
188	$DV4L = V_GEO' - V4L;$
189	
190	DV_GEOS = norm(DV3S) + norm(DV4S);
191	DV_GEOL = norm(DV3L) + norm(DV4L);
192	
193	if DV_GEOS < DV_GEOL
194	$V_mL = V3S';$
195	$DV2 = DV_GEOS;$
196	DV_depart2 = DV3S';
197	DV_arrive2 = DV4S';

198	else
199	$V_mL = V3L';$
200	$DV2 = DV_GEOL;$
201	DV_depart2 = DV3L';
202	DV_arrive2 = DV4L';
203	end
204	
205	%determine ground site inertial position vectors for duration of
	second maneuver
206	%(Texit + coastf) to Tend
207	<pre>Tvec2 = (T_exit+opt_vars(5):t_step:T_exit+opt_vars(5)+opt_vars</pre>
	(7))';
208	<pre>GMST = GMST0*ones(length(Tvec2),1) + OmegaEarth.*Tvec2;</pre>
209	<pre>longvec = long_site*ones(length(Tvec2),1) + GMST;</pre>
210	
211	
212	
213	%inertial coordinates of the ground site
214	<pre>Rsite = zeros(3,length(Tvec2));</pre>
215	<pre>Rsite(1,:) = RE*cos(lat_site).*cos(longvec);</pre>
216	<pre>Rsite(2,:) = RE*cos(lat_site).*sin(longvec);</pre>
217	<pre>Rsite(3,:) = RE*sin(lat_site);</pre>
218	
219	%determine maneuvering spacecraft inertial position vectors for
	duration od
220	%second maneuver (Texit + coastf) to Tend
221	<pre>Tvec3 = Tvec2 - T_exit - opt_vars(5);</pre>
222	
223	[am,em,im,Om,om,num0] = RV2COE(r_tL,V_mL);
224	
225	<pre>if imag(num0) < 1e-6</pre>
226	num0 = real(num0);

```
else
227
                 fid = fopen('error_data.txt','a');
228
                 [~,ind_imag0] = max(imag(num0));
229
                 fprintf(fid,'%s %s','Real','Imag');
230
                 fprintf(fid, '\n\r%10.8f %10.8f', real(num0(ind_imag0)), imag(
231
                     num0(ind_imag0)));
            end
232
233
234
            [val2] = site_contact_vec(am, im, em, Om, om, num0, long_site, GMST(1),
235
                Tvec3(end),t_step,lat_site,Rsite,el_val);
236
237
            if isempty(val2) == 0
238
                 J = Inf;
239
            else
240
                 J = DV + DV2;
241
            end
242
        end
243
  end
244
```

F.1.1.6 Convert RSW Coordinates to Cylinder Frame

```
1 function [deputy_pos,rel_pos] = box2cw(chief_pos,chief_vel,box_vec,alpha
,beta)
2 %this function converts from safe zone coordinate frame to the cw
3 %coordinate frame
4
5 %INPUTS
6 % box_vec - (3x1) vector defining a coordinate in the box frame (km)
7 % alpha - rotation angle between fundamental plane in box frame and
8 % fundamental plane in cw frame (rad)
9 % beta - rotation angle between principal axis in cw frame and box frame
```

```
10
11 rel_pos = rot3mat(-beta)*rot2mat(alpha)*box_vec;
12
13 xhat = chief_pos/norm(chief_pos);
14 yhat = chief_vel/norm(chief_vel);
15 hvec = cross(chief_pos,chief_vel);
16 zhat = hvec/norm(hvec);
17
18 [~,~,ic,Oc,~,nuc0] = RV2COE(chief_pos,chief_vel);
19
20 deputy_pos = chief_pos + rot3mat(-Oc)*rot1mat(-ic)*rot3mat(-nuc0)*
rel_pos;
```

F.1.1.7 Determine Initial and Final Velocities for Inspection Segment

```
1 function [v0_tilde,vf_tilde] = relative_velocity(T,P,pos0,posf)
2 %relative velocity returns the required initial and final relative
3 %velocities to get the deputy satellite from the relative position pos0
4 %to the relative position posf (relative to the chief satellite) in T/P
      time units
5
6 %INPUTS
7 % T = actual time of trajectory (sec)
8 % P = period of the chief satellite (sec)
9 \% \text{ pos} = \text{relative position vector } (3x1) \text{ of lobe entry point } (km)
10 % posf = relative position vector (3x1) of lobe exit point (km)
11
12 %OUTPUTS
13 % v0_tilde = time scaled relative velocity vector (3x1) at pos0
14 % vf_tilde = time scaled relative velocity vector (3x1) at posf
15
16 %%
17
```

```
18 x0 = pos0(1);
19 y0 = pos0(2);
z_0 z_0 = pos_0(3);
xf = posf(1);
22 yf = posf(2);
23 	ext{ zf = posf(3);}
24
25 T_tilde = T/P;
26 S_tilde = sin(2*pi*T_tilde);
27 C_tilde = cos(2*pi*T_tilde);
28 delta_y = yf - y0;
29
30
31 %Initialize A Matrices to determine relative velocities at entry and
32 %arrival locations
A0 = zeros(3,5);
34 Af = zeros(3,5);
35
36 %A Matrix at lobe entry
37 AO(1,1) = (6*pi*T_tilde*C_tilde - 4*S_tilde)/(8 - 6*pi*T_tilde*S_tilde -
       8*C_tilde);
38 A0(1,3) = (4*S_tilde - 6*pi*T_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*
      C_tilde);
39 A0(1,5) = (2*C_tilde - 2)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
40 A0(2,1) = (-14 + 12*pi*T_tilde*S_tilde + 14*C_tilde)/(8 - 6*pi*T_tilde*
      S_tilde - 8*C_tilde);
41 A0(2,3) = (2 - 2*C_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
42 A0(2,5) = (S_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
43 AO(3,2) = -C_tilde/S_tilde;
44 AO(3,4) = 1/S_{tilde};
45
46 \ A0 = 2*pi*A0;
```

```
48 %A Matrix at lobe exit
49 Af(1,1) = (-4*S_tilde + 6*pi*T_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*
      C_tilde);
50 Af(1,3) = (4*S_tilde - 6*pi*T_tilde*C_tilde)/(8 - 6*pi*T_tilde*S_tilde -
       8*C_tilde);
s1 Af(1,5) = (2-2*C_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
52 Af(2,1) = (2 - 2*C_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
53 Af(2,3) = (-14 + 12*pi*T_tilde*S_tilde + 14*C_tilde)/(8 - 6*pi*T_tilde*
      S_tilde - 8*C_tilde);
54 Af(2,5) = S_tilde/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
55 \text{ Af}(3,2) = -1/S_{tilde};
56 Af(3,4) = C_tilde/S_tilde;
57
58 \text{ Af} = 2*pi*Af;
59
60 state_vec = [x0;z0;xf;zf;delta_y];
61
62 v0_tilde = A0*state_vec;
63 vf_tilde = Af*state_vec;
```

47

F.1.1.8 Propagate Motion of Chaser For Relative Inspection Phase

```
1 function [rel_pos] = CW_Motion3(deputy_rel0,v0_tilde,Tvec,P)
2 %CW Motion determines the position of a deputy satellite in a relative
3 %frame cenetred on a chief satellite given an initial relative position,
4 %velocity, the actual time of the motion and period of the chief
        satellite
5
6 %INPUTS
7 % deputy_rel0 = position vector (3x1) of deputy satellite (km)
8 % v0_tilde = velocity vector (3x1) of deputy satellite
9 % T = actual time of motion (sec)
```

```
404
```

```
10 % P = period of chief satellite (sec)
11
12
13
14 %OUTPUTS
15 % rel_pos = relative position vector (3xlength(Tvec)) of deputy (km)
16 % T_out = Amount of time spent outside of ellipse (sec/P)
  %%
17
18
19
20 Tvec = Tvec/P;
  Tmat1(1,:) = sin(2*pi*Tvec);
21
  Tmat1(2,:) = cos(2*pi*Tvec);
22
23
  Tmat1(3,:) = 1;
24
25 Tmat2(1,:) = sin(2*pi*Tvec);
26 Tmat2(2,:) = cos(2*pi*Tvec);
  Tmat2(3,:) = (-1/pi*v0_tilde(1) + deputy_rel0(2)).*Tmat1(3,:);
27
  Tmat2(3,:) = Tmat2(3,:) - (3*v0_tilde(2) + 12*pi*deputy_rel0(1))*Tvec;
28
29
30 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)];
31 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];
32
33
34 xpos = xvals*Tmat1;
35 ypos = yvals*Tmat2;
36 zpos = zvals*Tmat1;
37
38 rel_pos(1,:) = xpos;
39 rel_pos(2,:) = ypos;
40 rel_pos(3,:) = zpos;
```

```
i function [T_out,T_in,pos_out,pos_in,time_out] = out_of_cylinder(rel_pos,
      Tvec,xmin,r_cyl)
2
3 %% Determine if satellite leaves safe zone
4 time_out = zeros(length(Tvec),1);
5
6 r_vec = sqrt(rel_pos(2,:).^2 + rel_pos(3,:).^2);
8 %set of indices where deputy is less than xmin
9 ind_ex_xmin = find(rel_pos(1,:) < xmin);</pre>
  time_out(ind_ex_xmin(:)) = 1;
10
11
12 %set of indices where deputy is greater than ymax
ind_ex_cyl = find(r_vec(:) > r_cyl);
  time_out(ind_ex_cyl(:)) = 1;
14
15
ind_out = find(time_out > 0);
ind_in = find(time_out == 0);
18
  pos_out = rel_pos(:,ind_out(:));
19
  pos_in = rel_pos(:,ind_in(:));
20
21
22 T_out = length(ind_out)/length(time_out)*Tvec(end);
23 T_in = length(ind_in)/length(time_out)*Tvec(end);
```

F.1.1.10 Determine Maneuver Path is in Sight of Ground Site

```
1 function [val,rho_sez] = site_contact_vec(a,inc,ecc,Omega,omega,nu0,
lambda0,GMST0,tmax,tstep,lat_site,rgs,el_val)
2
3 %INPUTS
4 % a = satellite semimajor axis (km)
```

```
5 % inc = satellite inclination (rad)
6 % ecc = satellite eccentricity
7 % Omega = satellite RAAN (rad)
8 % omega = satellite argument of perigee (rad)
9 % nu0 = initial true anomaly (rad)
10 % lambda0 = initial GMST of ground site
11 % tmax = maximum scenario time (sec)
12 % tstep = time step (sec)
13
14 %OUTPUTS
15 % C_times = times satellite is in contact with the ground site
16 % C_ind = indices of satellite contat times
17 % rijk = position vectors of satellite at discretized times
18 % vijk = velocity vectors of satellite at discretized times
19 % rgs = position vectors of the ground site at discretized times
20 % rho_sez = vector from ground site to satellite in SEZ coordinates
21 % rho_RIC = vector from ground site to satellite in RIC coordinates
22 %
     _____
23 tvec = (0:tstep:tmax)';
24
25 % determine true anomaly of spacecraft at each time step
 [nu_vec] = nuf_from_TOF_vec(nu0,tvec,a,ecc);
26
27
_{28} if ecc == 1
      keyboard
29
30
 end
31
32 %determine inertial position and velocity vectors at each tiem step
33 [rijk] = COE2RV_vec(a,ecc,inc,Omega,omega,nu_vec);
34
```

```
35
  if size(rijk) ~= size(rgs)
36
      keyboard
37
  end
38
39 wgs84data
40
  global OmegaEarth
41
42
  long_site = lambda0 + GMST0 + OmegaEarth*tvec;
43
44
45 %vector from ground site to satellite
  rho_ijk = rijk - rgs;
46
47
48 %transform into sez coordinates
49 temp = zeros(3,length(tvec));
50 temp(1,:) = cos(long_site').*rho_ijk(1,:) + sin(long_site').*rho_ijk
      (2,:);
51 temp(2,:) = -sin(long_site').*rho_ijk(1,:) + cos(long_site').*rho_ijk
      (2,:);
52 \text{ temp}(3,:) = \text{rho}_{ijk}(3,:);
53
54 rho_sez = zeros(3,length(tvec));
  rho_mag = zeros(1,length(tvec));
55
56
57 rho_sez(1,:) = cos(pi/2 - lat_site)*temp(1,:) - sin(pi/2 - lat_site)*
      temp(3,:);
58 rho_sez(2,:) = temp(2,:);
59 rho_{sez}(3,:) = sin(pi/2 - lat_site)*temp(1,:) + cos(pi/2 - lat_site)*
      temp(3,:);
60
61 rho_mag = sqrt(rho_sez(1,:).^2 + rho_sez(2,:).^2 + rho_sez(3,:).^2);
62
```
```
63 val = find(asind(rho_sez(3,:)./rho_mag) > el_val);
```

F.1.2 PSO Driver Script

```
1 %%
2
3 %maximum number of entries into exclusion zone before maneuver is
4 %required
5 close all
6 % clear all
7 clc
8
9 el_val_pass = 0;
10 el_val_shadow = 1;
11 GMST0 = 0;
12 lat_site = pi/4;
13 long_site = 0;
14 t0 = 0;
15 tf_max = 36*3600;
16 tstep = 1;
17 r_cyl = 1;
18 \text{ xmin} = 1;
19 xmax = 3;
20
21 %% Determine Chief Satellite Entry/Exit over Exclusion Zone
22
23 %Initial COEs of chief satellite
24 a_chief_vec = [26581.76 7378 6878];
25 \text{ e_chief} = 0;
26 i_chief = 55*pi/180;
27 0_{chief} = 0;
28 o_chief = 0;
29 % nu_chief_vec = 0;
```

```
30 nu_chief = 0;
31
32 chief_params = [e_chief;i_chief;0_chief;o_chief;nu_chief];
33 %Initial COEs of deputy satellite
a_{dep} = 6578;
35 e_dep = 0;
_{36} i_dep = 55*pi/180;
37 \text{ 0}_{dep} = 0;
38 o_dep = 0;
39 nu_dep0 = 0;
40
  dep_params = [a_dep;e_dep;i_dep;0_dep;o_dep];
41
42
a_{43} = 42164.14;
44 e_GE0 = 0;
45 i_GE0 = 0;
46 O_GEO = 0;
47 O_GEO = 0;
48
  GEO_params = [a_GEO;e_GEO;i_GEO;O_GEO;o_GEO];
49
50
51 \text{ swarm} = 15;
52 iter = 10;
53 prec = [0;0;6];
54 % if kinf \tilde{} = 0, inner loop PSO assigned inifinite cost to categorical
55 %variables if inner loop PSO has infinite cost after kinf iterations
56 \text{ kinf} = 50;
57
58 load('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
      Article_Data\Rev2\ThreePassEnumData');
59
60 \text{ for } aa = 1:3
```

```
410
```

```
C_times_c = ThreePassEnumData(aa).times;
61
       [max_ind(aa), ~] = size(C_times_c);
62
63
  end
64
65 maxP = max(max_ind);
66
  for bb = 16:30
67
68
       if bb == 1 || bb == 0
69
           total_repPSOinf = zeros(length(a_chief_vec),maxP);
70
           save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative
71
              Motion\Article_Data\Rev2\total_repPSOinf.mat','
              total_repPSOinf');
72
       end
73
       tstart = tic;
74
75
76
       [JGmin, Jpbest, gbest_tot, x, k, k_tot, JG, rep_mat, pop_mat] =
77
          PS0_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);
78
79
80
       tend = toc(tstart);
81
82
       load('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
83
          Article_Data\Rev2\total_repPS0inf');
       fid = fopen('C:\Users\Dan Showalter\Documents\MATLAB\PS0\Relative
84
          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
85
           %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
86
               (5),gbest_tot(6),gbest_tot(7),gbest_tot(8),gbest_tot(9),
               JGmin,k_tot,tend);
87
       for ee = 1:length(a_chief_vec)
88
           for ff = 1:maxP
89
                Jrep = rep_mat(ee,ff);
90
                Jtot = total_repPSOinf(ee,ff);
91
                if Jrep < Jtot || Jtot == 0
92
                    if Jtot ~= Inf
93
                    total_repPSOinf(ee,ff) = Jrep;
94
                    end
95
                end
96
           end
97
       end
98
99
100
101
       save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
102
           Article_Data\Rev2\total_repPS0inf.mat','total_repPS0inf');
       clear total_repPS0inf
103
104
  end
```

F.1.2.1 Outer Loop PSO

```
1 function [JGmin, Jpbest, gbest_tot, x, k, k_tot, JG, rep_mat, pop_mat] =
PS0_MULTISAT_COOP_WRAPPER(n, limits, prec, iter, swarm, GMST0, lat_site,
long_site, t_step, a_chief_vec, dep_params, GE0_params, xmin, xmax, r_cyl
,...
2 el_val_shadow, max_ind, DataStruct, kinf)
3
```

```
4 %Author: Dan Showalter 23 Sep 2013
5
6 %Purpose: PSO inside of a PSO
7
8 %generic PSO inputs
      n: # of design variables
  %
9
  %
      limits: bounds on design variables (n x 2 vector) with first element
10
           in row n being lower bound for element n and 2nd element in row
11 %
      n being
           upper bound for element n
12 %
      iter: number of iterations
13 %
      swarm: swarm size
14
  %
      prec: defines the number of decimal places to keep for each design
  %
15
          variable and the cost function evalution size: (n+1,1)
16
  %
17
18 %Problem specific PSO inputs
19 % GMSTO = initial Greenwich mean standard time (rad)
20 % lat_site = ground site latitude
21 % long_site = ground site longitude
22 % chief_params = vector (1x5) of fixed orbital elements of chief
      satellite
23 % nu_chief_vec = vector of potential initial true anomalies for chief
24 % dep_params = vector (1x6) of initial orbital elements of deputy
      satellite
25 % GEO_params = vector of (1x5) of fixed orbital elements of GEO
      satellite
26 % Coast_time_d = matrix (2xm) of allowed maneuver windows
                   (1,m) = start time of mth window
27 %
28 %
                   (2,m) = end time of mth window
29 % tf_max = maximum scenario time (sec)
30 % tstep = discrete time step (sec)
31 % xmin = minimum x distance from deputy to satellite in CW frame (km)
```

```
413
```

```
32 % xmax = maximum x distance from deputy to satellite in CW frame (km)
33 % Pc = period of chief satellite (sec)
34 % r_cyl = cylinder radius (km)
35 %
     _____
36
37 %%
38
  [N,~] = size(limits);
39
40
41 llim = limits(:,1);
42 ulim = limits(:,2);
43
44 if N^{\sim}=n
      fprintf('Error! limits size does not match number of variables')
45
      stop
46
47
 end
48
49 gbest = zeros(n,1);
50 x = zeros(n, swarm);
51 v = zeros(n, swarm);
52 pbest = zeros(n,swarm);
53 Jpbest = zeros(swarm,1);
54 x_inside = zeros(7,swarm);
55 d = (ulim - llim);
56 JG = zeros(iter,1);
57 J = zeros(swarm,1);
58 rep_mat = zeros(ulim(1),ulim(2));
59 pop_mat = struct('pop', zeros(n, swarm), 'J', zeros(swarm, 1));
60
61 llim2 = ones(n,swarm);
```

```
62 ulim2 = ones(n,swarm);
63 % CoreNum = 12;
64 % if (matlabpool('size')) <=0</pre>
65 %
         matlabpool('open','local',CoreNum);
66 % else
         disp('Parallel Computing Enabled')
67 %
68 % end
69
  parfor aa = 1:n
70
       llim2(aa,:) = llim(aa)*llim2(aa,:);
71
       ulim2(aa,:) = ulim(aa)*ulim2(aa,:);
72
73
  end
74
75 d2 = ulim2 - llim2;
76
77
  xrep(ulim(1),max_ind) = struct('xinsidevals',zeros(1,7));
78
  %loop until maximum iteration have been met
79
  for k = 1:iter
80
       t_inside = tic;
81
      %create particles dictated by swarm size input
82
83
84
      % if this is the first iteration
85
       if k == 1
86
           x = unidrnd(ulim2);
87
           v = random('unif',-d2,d2,[n,swarm]);
88
89
           %if this is after the first iteration, update velocity and
90
               position
           %of each particle in the swarm
91
92
       else
```

93 for h = 1:swarm 94 95 c1 = 2.09;96 c2 = 2.09;97 phi = c1+c2; 98 $ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));$ 99 100 cc = c1*random('unif', 0, 1);101 cs = c2*random('unif', 0, 1);102 103 104 vdum = v(:,h);105 106 %update velocity 107 vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + % 108 cs*(gbest - x(:,h))); 109 vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x 110 (:,h))); %check to make sure velocity doesn't exceed max velocity for 111 each %variable 112 for w = 1:n113 114 %if the variable velocity is less than the min, set it 115 to the min if vdum(w) < -d(w)116 vdum(w) = -d(w);117 %if the variable velocity is more than the max, set 118 it to the max 119 elseif vdum(w) > d(w);

```
vdum(w) = d(w);
120
121
                      end
                 end
122
123
                 v(:,h) = vdum;
124
125
                 %update position
126
                 xdum = x(:,h) + v(:,h);
127
128
                 for r = 1:n
129
130
                      %if particle has passed lower limit
131
                      if xdum(r) < llim(r)</pre>
132
                           xdum(r) = llim(r);
133
134
                      elseif xdum(r) > ulim(r)
135
                           xdum(r) = ulim(r);
136
                      end
137
138
                      x(:,h) = xdum;
139
140
                 end
141
142
             end
143
144
        end
145
146
        % round variables to get finite precision
147
        for aa = 1:n
148
            x(aa,:) = round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
149
            v(aa,:) = round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
150
151
        end
```

152			
153		pop_	_mat(k).pop = x;
154			
155			
156		%%	*********************************Cost Function

157			
158		for	m = 1:swarm
159			MU = 398600.5;
160			
161			% ********************** Cost function evaluation here

162			<pre>opt_vars = x(:,m);</pre>
163			% variable definitions
164			<pre>satellite = opt_vars(1);</pre>
165			<pre>min_ind = opt_vars(2);</pre>
166			
167			<pre>C_times_c = DataStruct(satellite).times;</pre>
168			<pre>C_ind_c = DataStruct(satellite).ind;</pre>
169			<pre>Rijk_c = DataStruct(satellite).Rc;</pre>
170			<pre>Vijk_c = DataStruct(satellite).Vc;</pre>
171			<pre>rho_vec_cw_c = DataStruct(satellite).rho_c;</pre>
172			<pre>Tvec_c = DataStruct(satellite).Tc;</pre>
173			<pre>max_ind = DataStruct(satellite).max_ind;</pre>
174			
175			<pre>if min_ind > max_ind</pre>
176			J(m) = Inf;
177			else
178	%		<pre>if rep_mat(satellite,min_ind) == Inf</pre>
179	%		J(m) = Inf;
180			<pre>if rep_mat(satellite,min_ind) ~= 0</pre>
181			<pre>J(m) = rep_mat(satellite,min_ind);</pre>

```
x_inside(:,m) = xrep(satellite,min_ind).xinsidevals;
182
                else
183
184
185
186
                    %Period of Chief satellite's orbit
187
                    Pc = 2*pi*sqrt(a_chief_vec(satellite)^3/MU);
188
189
190
191
                    t_enter = C_times_c(min_ind,1);
                    t_exit = C_times_c(min_ind,2);
192
                    t_zone = t_exit - t_enter;
193
194
                    %determine indices of minimum duration contact
195
                    C_ind_contact = C_ind_c(min_ind,:);
196
197
                    %find unit vector pointing towards the deputy that puts
198
                        chief between
                    %ground site and deputy
199
                    rho_unit_cw = rho_vec_cw_c(:,C_ind_contact(1):
200
                        C_ind_contact(2));
201
                    %determine alpha and beta angles during contact times
202
                    [alphavec,betavec] = alphabeta(rho_unit_cw);
203
204
                    %Vector of times for propogation
205
                    T_prop = Tvec_c(C_ind_contact(1):C_ind_contact(2)) -
206
                        Tvec_c(C_ind_contact(1))*ones(length(Tvec_c(
                        C_ind_contact(1):C_ind_contact(2))),1);
207
                    %Determine position/velocity vectors of chief satellite
208
                        upon intial/final
```

```
%contact
209
                    chief_pos0 = Rijk_c(:,C_ind_c(min_ind,1));
210
                    chief_vel0 = Vijk_c(:,C_ind_c(min_ind,1));
211
                    chief_posf = Rijk_c(:,C_ind_c(min_ind,2));
212
                    chief_velf = Vijk_c(:,C_ind_c(min_ind,2));
213
214
                    if min_ind < max_ind</pre>
215
                         max_coastf = C_times_c(min_ind+1,1) - C_times_c(
216
                            min_ind,2);
                    else
217
                         max_coastf = DataStruct(satellite).max_coastf;
218
                    end
219
220
                    %time variables have precision to .1 second. Others
221
                        have
                    %precision to 0.001 units (km,rad)
222
                    prec2 = [2;0;3;3;0;2;0;6];
223
224
                    x(:,m);
225
226
                    [J(m), ~, x_inside_dum, ~, k_inside, ~] =
227
                        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,GE0_params,alphavec,betavec,
228
                            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
229
                         rep_mat(satellite,min_ind) = J(m);
230
                         xrep(satellite,min_ind).xinsidevals = x_inside_dum;
231
232
                    else
```

if J(m) < rep_mat(satellite,min_ind)</pre> 233 rep_mat(satellite,min_ind) = J(m); 234 end 235 end 236 J(m); 237 x_inside(:,m) = x_inside_dum; 238 out_loop = m; 239 if k == 1240 k_tot = k_inside; 241 else 242 k_tot = k_inside + k_tot; 243 end 244 end 245 end 246 end 247 248 249 250 [minJ, ind_minJ] = min(J); 251 x_inside(:,ind_minJ) 252 253 ****** %% 254 %% 255 256 257 if k == 1258 259 Jpbest(1:swarm) = J(1:swarm);

pbest(:,1:swarm) = x(:,1:swarm); 261

260

```
262
             [Jgbest,IND] = min(Jpbest(:));
263
264
             gbest(:) = x(:,IND);
265
             g_inside_best = x_inside(:,IND);
266
267
        else
268
             parfor h=1:swarm
269
                 if J(h) < Jpbest(h)</pre>
270
                      Jpbest(h) = J(h);
271
                      pbest(:,h) = x(:,h);
272
                 end
273
             end
274
275
             [Jit_min,min_ind] = min(Jpbest);
276
             if Jit_min < Jgbest</pre>
277
278
                 Jgbest = Jpbest(min_ind);
279
                 gbest(:) = x(:,min_ind);
280
                 g_inside_best = x_inside(:,min_ind);
281
282
             end
283
        end
284
285
286
287
        %round cost to nearest precision required
288
        J = round(J*10^{prec}(n+1))/10^{prec}(n+1);
289
        pop_mat(k).J = J;
290
291
        JG(k) = Jgbest;
292
        JGmin = Jgbest;
293
```

```
294
        iter_complete = k
295
        iter_time = toc(t_inside)
296
        format long g
297
        gbest
298
        g_inside_best
299
        JGmin
300
   end
301
302
   gbest_tot(1:n) = gbest;
303
   gbest_tot(n+1:n+length(g_inside_best)) = g_inside_best;
304
```

F.1.2.2 Inner Loop PSO Algorithm with Infeasible Cutoff

```
1 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
2
                                                                      ,GMST0,
                                                                      lat_site
                                                                      long_site
                                                                      ,t_step,
                                                                      el_val,
                                                                      kinf)
3
4
5
6 %Author: Dan Showalter 18 Oct 2012
7
  %Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem
8
9
10 %generic PSO variable
```

```
11 %
      n: # of design variables
       limits: bounds on design variables (n x 2 vector) with first element
12 %
           in row n being lower bound for element n and 2nd element in row
13 %
      n being
14 %
           upper bound for element n
       iter: number of iterations
  %
15
       swarm: swarm size
  %
16
       prec: defines the number of decimal places to keep for each design
17
  %
  %
           variable and the cost function evalution size: (n+1,1)
18
19
20 %Problem specific PSO variables
21
22
23
24 %Specific Problem Variables
25
26
27
28 %%
29
30 [N,~] = size(limits);
31
32 llim = limits(:,1);
33 ulim = limits(:,2);
34
35 if N^{\sim}=n
       fprintf('Error! limits size does not match number of variables')
36
37
       stop
  end
38
39
40
41
```

```
42 gbest = zeros(n,1);
43 \mathbf{x} = \mathbf{zeros}(n, \mathbf{swarm});
44 v = zeros(n, swarm);
45 pbest = zeros(n,swarm);
46 Jpbest = zeros(swarm,1);
47 d = (ulim - 1lim);
48 JG = zeros(iter,1);
49 J = zeros(swarm, 1);
50
51 llim2 = ones(n,swarm);
52 ulim2 = ones(n,swarm);
53
  for aa = 1:n
54
       llim2(aa,:) = llim(aa)*llim2(aa,:);
55
       ulim2(aa,:) = ulim(aa)*ulim2(aa,:);
56
  end
57
58
  d2 = ulim2 - llim2;
59
60
  CoreNum = 12;
61
  if (matlabpool('size'))<=0</pre>
62
       matlabpool('open','local',CoreNum);
63
  else
64
       disp('Parallel Computing Enabled')
65
  end
66
67 tstart = tic;
  %loop until maximum iteration have been met
68
69
70
  for k = 1:iter
71
       %create particles dictated by swarm size input
72
73
```

```
74
       % if this is the first iteration
75
       if k == 1
76
            rng('shuffle');
77
            x = random('unif',llim2,ulim2,[n,swarm]);
78
            v = random('unif',-d2,d2,[n,swarm]);
79
80
            %if this is after the first iteration, update velocity and
81
               position
            %of each particle in the swarm
82
       else
83
            parfor h = 1:swarm
84
                c1 = 2.09;
85
                c2 = 2.09;
86
                phi = c1+c2;
87
                ci = 2/abs(2-phi - sqrt(phi<sup>2</sup> - 4*phi));
88
                cc = c1*random('unif', 0, 1);
89
                cs = c2*random('unif', 0, 1);
90
91
92
                vdum = v(:,h);
93
                %update velocity
94
                vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x
95
                    (:,h)));
96
97
                %check to make sure velocity doesn't exceed max velocity for
98
                     each
                %variable
99
                for w = 1:n
100
101
```

102	%if the variable velocity is less than the min, set it
	to the min
103	<pre>if vdum(w) < -d(w)</pre>
104	vdum(w) = -d(w);
105	%if the variable velocity is more than the max, set
	it to the max
106	<pre>elseif vdum(w) > d(w);</pre>
107	vdum(w) = d(w);
108	end
109	end
110	
111	v(:,h) = vdum;
112	
113	%update position
114	xdum = x(:,h) + v(:,h);
115	
116	for $r = 1:n$
117	
118	%if particle has passed lower limit
119	<pre>if xdum(r) < llim(r)</pre>
120	xdum(r) = llim(r);
121	
122	<pre>elseif xdum(r) > ulim(r)</pre>
123	<pre>xdum(r) = ulim(r);</pre>
124	end
125	
126	x(:,h) = xdum;
127	
128	end
129	
130	end
131	

```
end
132
133
      % round variables to get finite precision
134
      parfor aa = 1:n
135
         x(aa,:) = round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
136
         v(aa,:) = round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
137
      end
138
139
      140
         *************
141
      xmin = limits(2,1);
142
143
      parfor m = 1:swarm
144
         % *********************** Cost function evaluation here
145
            *****
         opt_vars = x(:,m);
146
147
         [J(m)] = rel_shadow_cost_function2(opt_vars, chief_pos0,
148
            chief_vel0, chief_posf, chief_velf, alphavec, betavec, t_zone, Pc,
            T_prop,xmin,r_cyl,dep_params,...
                                               GEO_params, T_enter,
149
                                                  T_exit,GMST0,
                                                  long_site,
                                                  lat_site,t_step,
                                                  el_val);
150
151
      end
152
      153
         ****
```

```
%%
154
          155
       %%
156
157
       %round cost to nearest precision required
158
       J = round(J*10^{prec}(n+1))/10^{prec}(n+1);
159
160
       if k == 1
161
           count = 0;
162
           Jpbest(1:swarm) = J(1:swarm);
163
           pbest(:,1:swarm) = x(:,1:swarm);
164
165
           [Jgbest,IND] = min(Jpbest(:));
166
167
           gbest(:) = x(:,IND);
168
169
       else
170
171
           for h=1:swarm
172
               if J(h) < Jpbest(h)</pre>
173
                   Jpbest(h) = J(h);
174
                   pbest(:,h) = x(:,h);
175
                   if Jpbest(h) < Jgbest</pre>
176
177
                       Jgbest = Jpbest(h);
178
                       gbest(:) = x(:,h);
179
180
                   end
181
               end
182
183
           end
```

```
end
184
185
186
187
        diff = zeros(swarm,1);
188
        parfor y = 1:swarm
189
190
             diff(y) = Jgbest - Jpbest(y);
191
        end
192
193
        indcount = find(abs(diff)<10^(-prec(n+1)));</pre>
194
195
196
197
198
        JG(k) = Jgbest;
199
        JGmin = Jgbest;
200
201
        if kinf \tilde{} = 0;
202
             if k > kinf
203
                  if Jgbest == Inf
204
                       break
205
                  end
206
             end
207
        end
208
209
        if length(indcount) == swarm
210
             ex_flag = 0;
211
             break
212
        end
213
214
        if k > 1
215
```

```
if JG(k) == JG(k-1)
216
                 count = count + 1;
217
            else
218
                 count = 0;
219
            end
220
        end
221
222
        if count > 1000
223
            ex_flag = 1;
224
            break
225
        end
226
   end
227
228
  if k == iter
229
      ex_flag = 2;
230
231 end
```

F.1.3 GA Driver Script

```
1 clc
2 close all
3
4 for h =1:10
5
6
7
       el_val_pass = 0;
       el_val_shadow = 1;
8
       GMST0 = 0;
9
       lat_site = pi/4;
10
       long_site = 0;
11
       t0 = 0;
12
       tf_max = 36*3600;
13
      tstep = 1;
14
```

```
r_cyl = 1;
15
16
       xmin = 1;
       xmax = 3;
17
       %% Determine Chief Satellite Entry/Exit over Exclusion Zone
18
19
       %Initial COEs of chief satellite
20
       a_chief = [26581.76 7378 6878];
21
       e_chief = 0;
22
       i_chief = 55*pi/180;
23
       0_{chief} = 0;
24
       o_chief = 0;
25
       % nu_chief_vec = 0;
26
       nu_chief_vec = [0 90 180 270]*pi/180;
27
28
       chief_params = [e_chief;i_chief;0_chief;o_chief;nu_chief_vec(1)];
29
       %Initial COEs of deputy satellite
30
       a_{dep} = 6578;
31
32
       e_dep = 0;
       i_dep = 55*pi/180;
33
       0_{dep} = 0;
34
       o_dep = 0;
35
       nu_dep0 = 0;
36
37
       dep_params = [a_dep;e_dep;i_dep;0_dep;o_dep];
38
39
       a_{GEO} = 42164.14;
40
       e_GEO = 0;
41
       i_GEO = 0;
42
       O_GEO = 0;
43
       o_GEO = 0;
44
45
46
       GEO_params = [a_GEO;e_GEO;i_GEO;O_GEO;o_GEO];
```

```
load('C:\Users\Dan Showalter\Documents\MATLAB\PS0\Relative Motion\
48
          Article_Data\Rev3\ThreeTarget\ThreePassEnumData');
49
       kinf = 50;
50
51
  for aa = 1:3
52
       C_times_c = ThreePassEnumData(aa).times;
53
       [max_ind(aa), ~] = size(C_times_c);
54
       clear C_times_c
55
  end
56
       maxP = max(max_ind);
57
58
       if h == 1
59
           total_repGAinf = zeros(3,maxP);
60
           save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative
61
               Motion\Article_Data\Rev3\ThreeTarget\total_repGAinf.mat','
               total_repGAinf');
       end
62
63
       llim = [1 \ 1];
64
       ulim = [length(a_chief) maxP];
65
66
       PopSize = 15;
67
       ulim2 = zeros(PopSize,2);
68
       ulim2(:,1) = ulim(1);
69
       ulim2(:,2) = ulim(2);
70
       EliteSize = 1;
71
72
       rep_mat = zeros(length(a_chief),maxP);
73
       fid3 = fopen('GA_Jmin.txt','w');
74
       fprintf(fid3,'%f',10000);
75
```

```
433
```

```
fclose(fid3);
76
77
       fid4 = fopen('GA_iters.txt','w');
78
       fprintf(fid4,'%i',0);
79
       fclose(fid4);
80
81
       fid5 = fopen('repository.txt','w');
82
       fprintf(fid5,'%7.5f %7.5f %7.5f',rep_mat);
83
       fclose(fid5);
84
85
       tstart = tic;
86
87
88
       rng('shuffle');
89
       PopInit = unidrnd(ulim2);
90
91
       options = gaoptimset('InitialPopulation', PopInit, 'PopulationSize',
92
           PopSize,'UseParallel','never','CrossoverFraction',0.8,...
            'StallGenLimit',9,'Generation',9,'TolFun',1e-6,'EliteCount',
93
               EliteSize, 'Display', 'diagnose', 'Vectorized', 'off');
94
       [gbest, ], exflag, output] = ga(@(x)GA_Hybrid_Cost_082014(x, GMST0,
95
           lat_site,long_site,tstep,a_chief,dep_params,GEO_params,xmin,xmax
           ,r_cyl,...
            el_val_shadow,ThreePassEnumData,rep_mat,max_ind,kinf)
96
               ,2,[],[],[],[],llim,ulim,[],[1,2],options);
97
       fid = fopen('GA_intermediate_vals.txt');
98
       x_inside = fscanf(fid,'%f',7);
99
       J_inside = fscanf(fid, '%d', 1);
100
       k_tot = fscanf(fid,'%d',1);
101
       fclose(fid);
102
```

```
434
```

```
103
       fid_int = fopen('GA_opt_int.txt');
104
       min_sat = fscanf(fid_int,'%i',1);
105
       min_pass = fscanf(fid_int,'%i',1);
106
       fclose(fid_int);
107
108
       fid_iters = fopen('GA_iters.txt');
109
       iters = fscanf(fid_iters,'%d');
110
111
       J
112
       J_inside
113
       tend = toc(tstart)
114
115
       load('C:\Users\Dan Showalter\Documents\MATLAB\PS0\Relative Motion\
116
           Article_Data\Rev3\ThreeTarget\total_repGA');
       load('C:\Users\Dan Showalter\Documents\MATLAB\PS0\Relative Motion\
117
           Article_Data\Rev3\ThreeTarget\rep_mat_out');
118
       for ee = 1:length(a_chief)
119
            for ff = 1:maxP
120
                Jrep = rep_mat_out(ee,ff);
121
                Jtot = total_repGAinf(ee,ff);
122
                if Jrep < Jtot || Jtot == 0</pre>
123
                     if Jtot ~= Inf
124
                         total_repGAinf(ee,ff) = Jrep;
125
                     end
126
                end
127
128
            end
       end
129
130
       save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
131
           Article_Data\Rev3\ThreeTarget\total_repGA.mat','total_repGA');
```

```
435
```

F.1.3.1 GA Cost Function

```
1 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)
2
3
4
5 %
     ______
6 % This function evaluates the cost for the MATLAB genetic algorithm
     routine
7 %Inputs:
     x: 2x1 vector of design variables
  %
8
         x(1) defines the satellite that will be shadowed
9
 %
 %
         x(2) is the pass of x(1) or the specified ground site to
10
     accomplish
11 %
         the shadow
12 %Outputs:
```

```
13 % Global Variables
14
15
16 %
     ______
MU = 398600.5;
18
19 satellite = x(1);
20 \min_{x(2)} = x(2);
21
  [rows,cols] = size(rep_mat);
22
23
24 fid_rep = fopen('repository.txt');
25 rep_mat = fscanf(fid_rep,'%g', [rows cols]);
26 fclose(fid_rep);
27
 if min_ind > max_ind(satellite)
28
      J = Inf;
29
      rep_mat(satellite,min_ind) = J;
30
      fid_rep = fopen('repository.txt','w');
31
      %number of elements must equal number of satellites
32
      fprintf(fid_rep,'%g %g %g ',rep_mat);
33
      fclose(fid_rep);
34
 else
35
      %
            if rep_mat(satellite,min_ind) == Inf
36
                J = Inf;
      %
37
      %
           else
38
39
      if rep_mat(satellite,min_ind) ~= 0
40
          J = rep_mat(satellite,min_ind);
41
42
      else
```

```
C_times_c = DataStruct(satellite).times;
44
           C_ind_c = DataStruct(satellite).ind;
45
           Rijk_c = DataStruct(satellite).Rc;
46
           Vijk_c = DataStruct(satellite).Vc;
47
           rho_vec_cw_c = DataStruct(satellite).rho_c;
48
           Tvec_c = DataStruct(satellite).Tc;
49
           max_ind = DataStruct(satellite).max_ind;
50
51
           %Period of Chief satellite's orbit
52
           Pc = 2*pi*sqrt(a_chief_vec(satellite)^3/MU);
53
54
55
           t_enter = C_times_c(min_ind,1);
56
           t_exit = C_times_c(min_ind,2);
57
           t_zone = t_exit - t_enter;
58
59
           %determine indices of minimum duration contact
60
           C_ind_contact = C_ind_c(min_ind,:);
61
62
           %find unit vector pointing towards the deputy that puts chief
63
              between
           %ground site and deputy
64
           rho_unit_cw = rho_vec_cw_c(:,C_ind_contact(1):C_ind_contact(2));
65
66
           %determine alpha and beta angles during contact times
67
           [alphavec,betavec] = alphabeta(rho_unit_cw);
68
69
           %Vector of times for propogation
70
           T_prop = Tvec_c(C_ind_contact(1):C_ind_contact(2)) - Tvec_c(
71
              C_ind_contact(1))*ones(length(Tvec_c(C_ind_contact(1):
              C_ind_contact(2))),1);
```

```
72
73
           %Determine position/velocity vectors of chief satellite upon
               intial/final
           %contact
74
           chief_pos0 = Rijk_c(:,C_ind_c(min_ind,1));
75
           chief_vel0 = Vijk_c(:,C_ind_c(min_ind,1));
76
           chief_posf = Rijk_c(:,C_ind_c(min_ind,2));
77
           chief_velf = Vijk_c(:,C_ind_c(min_ind,2));
78
79
           if min_ind < max_ind</pre>
80
               max_coastf = C_times_c(min_ind+1,1) - C_times_c(min_ind,2);
81
           else
82
               max_coastf = DataStruct(satellite).max_coastf;
83
           end
84
85
           %time variables have precision to .1 second. Others have
86
           %precision to 0.001 units (km,rad)
87
           prec2 = [2;0;3;3;0;2;0;6];
88
89
           [J, ,x_inside_dum, ,k_inside, ] = PSO_REL_SHADOW_DV4inf(7,[0 2*
90
              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,GE0_params,alphavec,betavec,t_zone,Pc,
91
                   t_enter,t_exit,r_cyl,T_prop,GMST0,lat_site,long_site,
                   t_step,el_val_shadow,kinf);
92
           %determine lowest cost so far
93
           fid1 = fopen('GA_Jmin.txt');
94
           Jmin = fscanf(fid1,'%f');
95
           fclose(fid1);
96
97
```

```
%Update inside loop iterations
98
            fid4 = fopen('GA_iters.txt');
99
            iters = fscanf(fid4,'%d');
100
            iters = iters + k_inside;
101
            fclose(fid4);
102
            fid5 = fopen('GA_iters.txt','w');
103
            fprintf(fid5,'%i',iters);
104
            fclose(fid5);
105
106
107
            Jrep = rep_mat(satellite,min_ind);
108
            if Jrep == 0 || J < Jrep;</pre>
109
                rep_mat(satellite,min_ind) = J;
110
111
            end
            %update repository
112
            fid_rep = fopen('repository.txt','w');
113
            %number of elements must equal number of satellites
114
            fprintf(fid_rep,'%g %g %g ',rep_mat);
115
            fclose(fid_rep);
116
117
            %If current cost is better than lowest cost so far, update inner
118
                 loop
            %variables
119
            if J < Jmin
120
                fid3 = fopen('GA_Jmin.txt','w');
121
                fprintf(fid3,'%g',J);
122
                fclose(fid3);
123
124
                fid6 = fopen('GA_opt_int.txt','w');
125
                fprintf(fid6,'%i %i',satellite,min_ind);
126
                fclose(fid6);
127
128
```

```
fid2=fopen('GA_intermediate_vals.txt','w');
129
                fprintf(fid2,'%3.2f\t %i\t %4.3f\t %4.3f\t %i\t %3.2f\t %i\t
130
                    %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);
131
132
            end
133
       end
134
135
   end
136
137
   fid7 = fopen('GA_opt_int.txt');
138
   min_sat = fscanf(fid7,'%i',1);
139
   min_pass = fscanf(fid7,'%i',1);
140
   fclose(fid7);
141
142
  rep_mat_out = rep_mat;
143
  save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
144
       Article_Data\Rev2\ThreeTarget\rep_mat_out.mat','rep_mat_out');
```

F.2 Fifteen Target GTMEI

F.2.0.2 Large Outer Loop PSO

```
1 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)

4 %Author: Dan Showalter 23 Sep 2013
5
6 %Purpose: PSO inside of a PSO
```

```
8 %generic PSO inputs
      n: # of design variables
9
  %
      limits: bounds on design variables (n x 2 vector) with first element
10
  %
11 %
           in row n being lower bound for element n and 2nd element in row
      n being
  %
          upper bound for element n
12
      iter: number of iterations
  %
13
      swarm: swarm size
  %
14
      prec: defines the number of decimal places to keep for each design
15
  %
  %
          variable and the cost function evalution size: (n+1,1)
16
17
18 %Problem specific PSO inputs
19 % GMSTO = initial Greenwich mean standard time (rad)
20 % lat_site = ground site latitude
21 % long_site = ground site longitude
22 % chief_params = vector (1x5) of fixed orbital elements of chief
      satellite
23 % nu_chief_vec = vector of potential initial true anomalies for chief
24 % dep_params = vector (1x6) of initial orbital elements of deputy
      satellite
25 % GEO_params = vector of (1x5) of fixed orbital elements of GEO
      satellite
  % Coast_time_d = matrix (2xm) of allowed maneuver windows
26
                   (1,m) = start time of mth window
27 %
                   (2,m) = end time of mth window
 %
28
29 % tf_max = maximum scenario time (sec)
30 % tstep = discrete time step (sec)
31 % xmin = minimum x distance from deputy to satellite in CW frame (km)
32 % xmax = maximum x distance from deputy to satellite in CW frame (km)
33 % Pc = period of chief satellite (sec)
```

34 % r_cyl = cylinder radius (km)

```
______
36
37 %%
38
 [N,~] = size(limits);
39
40 llim = limits(:,1);
41 ulim = limits(:,2);
42
43 if N^{\sim}=n
      fprintf('Error! limits size does not match number of variables')
44
      stop
45
46
  end
47
48 gbest = zeros(n,1);
49 x = zeros(n, swarm);
50 v = zeros(n,swarm);
51 pbest = zeros(n,swarm);
52 Jpbest = zeros(swarm,1);
53 x_inside = zeros(7, swarm);
54 d = (ulim - llim);
55 JG = zeros(iter,1);
56 J = zeros(swarm,1);
57 rep_mat = zeros(ulim(1),ulim(2));
58 pop_mat = struct('pop', zeros(n, swarm), 'J', zeros(swarm, 1), 'gbest', zeros(n
     ,1));
59
60 llim2 = ones(n,swarm);
61 ulim2 = ones(n,swarm);
62 % CoreNum = 12;
63 % if (matlabpool('size')) <=0</pre>
```

35 %

```
64 %
         matlabpool('open','local',CoreNum);
65 % else
         disp('Parallel Computing Enabled')
  %
66
  % end
67
68
  parfor aa = 1:n
69
       llim2(aa,:) = llim(aa)*llim2(aa,:);
70
       ulim2(aa,:) = ulim(aa)*ulim2(aa,:);
71
  end
72
73
74 d2 = ulim2 - llim2;
75
76 tstart = tic;
77
rep(ulim(1),max_ind) = struct('xinsidevals',zeros(1,7));
  %loop until maximum iteration have been met
79
  for k = 1:iter
80
       t_inside = tic;
81
      %create particles dictated by swarm size input
82
83
84
      % if this is the first iteration
85
       if k == 1
86
           x = unidrnd(ulim2);
87
           v = random('unif',-d2,d2,[n,swarm]);
88
89
           %if this is after the first iteration, update velocity and
90
              position
           %of each particle in the swarm
91
       else
92
93
           for h = 1:swarm
94
```
95 c1 = 2.09;96 c2 = 2.09;97 phi = c1+c2; 98 $ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));$ 99 100 cc = c1*random('unif',0,1); 101 cs = c2*random('unif', 0, 1);102 103 104 vdum = v(:,h);105 106 %update velocity 107 vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + 108 % cs*(gbest - x(:,h))); 109 vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x 110 (:,h))); %check to make sure velocity doesn't exceed max velocity for 111 each %variable 112 for w = 1:n113 114 %if the variable velocity is less than the min, set it 115 to the min if vdum(w) < -d(w)116 vdum(w) = -d(w);117 %if the variable velocity is more than the max, set 118 it to the max elseif vdum(w) > d(w); 119 vdum(w) = d(w);120 121 end

```
end
122
123
                 v(:,h) = vdum;
124
125
                 %update position
126
                 xdum = x(:,h) + v(:,h);
127
128
                 for r = 1:n
129
130
                      %if particle has passed lower limit
131
                      if xdum(r) < llim(r)</pre>
132
                           xdum(r) = llim(r);
133
134
                      elseif xdum(r) > ulim(r)
135
                          xdum(r) = ulim(r);
136
                      end
137
138
                      x(:,h) = xdum;
139
140
                 end
141
142
            end
143
144
        end
145
146
        % round variables to get finite precision
147
        for aa = 1:n
148
            x(aa,:) = round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
149
            v(aa,:) = round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
150
        end
151
        pop_mat(k).pop = x;
152
```

```
153
          ***********
154
       for m = 1:swarm
155
           MU = 398600.5;
156
157
           % *********************** Cost function evaluation here
158
              ******
           opt_vars = x(:,m);
159
                     variable definitions
160
           %
           satellite = opt_vars(1);
161
           min_ind = opt_vars(2);
162
163
           C_times_c = DataStruct(satellite).times;
164
           C_ind_c = DataStruct(satellite).ind;
165
           Rijk_c = DataStruct(satellite).Rc;
166
           Vijk_c = DataStruct(satellite).Vc;
167
           rho_vec_cw_c = DataStruct(satellite).rho_c;
168
           Tvec_c = DataStruct(satellite).Tc;
169
           max_ind = DataStruct(satellite).max_ind;
170
171
           if min_ind > max_ind
172
               J(m) = Inf;
173
           else
174
175
               if rep_mat(satellite,min_ind) ~= 0
176
                   J(m) = rep_mat(satellite,min_ind);
177
                   x_inside(:,m) = xrep(satellite,min_ind).xinsidevals;
178
               else
179
180
181
182
```

```
%Period of Chief satellite's orbit
183
                    Pc = 2*pi*sqrt(a_chief_vec(satellite)^3/MU);
184
185
186
                    t_enter = C_times_c(min_ind,1);
187
                    t_exit = C_times_c(min_ind,2);
188
                    t_zone = t_exit - t_enter;
189
190
                    %determine indices of minimum duration contact
191
                    C_ind_contact = C_ind_c(min_ind,:);
192
193
                    %find unit vector pointing towards the deputy that puts
194
                        chief between
195
                    %ground site and deputy
                    rho_unit_cw = rho_vec_cw_c(:,C_ind_contact(1):
196
                        C_ind_contact(2));
197
                    %determine alpha and beta angles during contact times
198
                    [alphavec,betavec] = alphabeta(rho_unit_cw);
199
200
                    %Vector of times for propogation
201
                    T_prop = Tvec_c(C_ind_contact(1):C_ind_contact(2)) -
202
                        Tvec_c(C_ind_contact(1))*ones(length(Tvec_c(
                        C_ind_contact(1):C_ind_contact(2)),1);
203
                    %Determine position/velocity vectors of chief satellite
204
                        upon intial/final
                    %contact
205
                    chief_pos0 = Rijk_c(:,C_ind_c(min_ind,1));
206
                    chief_vel0 = Vijk_c(:,C_ind_c(min_ind,1));
207
                    chief_posf = Rijk_c(:,C_ind_c(min_ind,2));
208
209
                    chief_velf = Vijk_c(:,C_ind_c(min_ind,2));
```

if min_ind < max_ind</pre> 211 max_coastf = C_times_c(min_ind+1,1) - C_times_c(212 min_ind,2); else 213 max_coastf = DataStruct(satellite).max_coastf; 214 end 215 216 %time variables have precision to .1 second. Others 217 have %precision to 0.001 units (km,rad) 218 prec2 = [2;0;3;3;0;2;0;6];219 220 x(:,m); 221 222 [J(m), ~, x_inside_dum, ~, k_inside, ~] = 223 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,GE0_params,alphavec,betavec, 224 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) == 0 225 rep_mat(satellite,min_ind) = J(m); 226 xrep(satellite,min_ind).xinsidevals = x_inside_dum; 227 else 228 if J(m) < rep_mat(satellite,min_ind)</pre> 229 rep_mat(satellite,min_ind) = J(m); 230 end 231 end 232 233 J(m);

210

234	<pre>x_inside(:,m) = x_inside_dum;</pre>						
235	<pre>out_loop = m;</pre>						
236	if k == 1						
237	<pre>k_tot = k_inside;</pre>						
238	else						
239	<pre>k_tot = k_inside + k_tot;</pre>						
240	end						
241	end						
242	end						
243	end						
244							
245	[minJ,ind_minJ] = min(J);						
246	<pre>x_inside(:,ind_minJ);</pre>						
247	%% ********************************Constraint Equations						
	* * * * * * * * * * * * * * * * * * * *						
248	%%						
	**********	* * *					

```
%%
249
250
251
        if k == 1
252
253
            Jpbest(1:swarm) = J(1:swarm);
254
            pbest(:,1:swarm) = x(:,1:swarm);
255
256
            [Jgbest,IND] = min(Jpbest(:));
257
258
            gbest(:) = x(:,IND);
259
            g_inside_best = x_inside(:,IND);
260
            stall = 0;
261
262
```

```
else
263
264
             for h=1:swarm
265
                  if J(h) < Jpbest(h)</pre>
266
                       Jpbest(h) = J(h);
267
                       pbest(:,h) = x(:,h);
268
                       if Jpbest(h) < Jgbest</pre>
269
270
                            Jgbest = Jpbest(h);
271
                            gbest(:) = x(:,h);
272
                            g_inside_best = x_inside(:,h);
273
274
                       end
275
                  end
276
             end
277
278
279
280
        end
281
        count = 0;
282
283
        for y = 1:swarm
284
285
             diff = Jgbest - Jpbest(y);
286
287
             if abs(diff) < 10^{(-prec(n+1)+1)}
288
                  count = count+1;
289
             end
290
291
        end
292
293
294
        %round cost to nearest precision required
```

```
J = round(J*10^{prec}(n+1))/10^{prec}(n+1);
295
        pop_mat(k).J = J;
296
        pop_mat(k).gbest = gbest;
297
        JG(k) = Jgbest;
298
        JGmin = Jgbest;
299
300
        if k > 1
301
            if (JG(k) - JG(k-1)) == 0
302
                 stall = stall + 1;
303
             else
304
                 stall = 0;
305
             end
306
        end
307
308
        if count == swarm
309
            break
310
        end
311
312
        if stall == stall_lim
313
             break
314
        end
315
        tend = toc(tstart);
316
317
        iter_complete = k
318
        iter_time = toc(t_inside)
319
        format long g
320
        gbest
321
        g_inside_best
322
        JGmin
323
   end
324
325
326 gbest_tot(1:n) = gbest;
```

327 gbest_tot(n+1:n+length(g_inside_best)) = g_inside_best;

Bibliography

- 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 5th 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

The public reporting maintaining the data suggestions for reduc Suite 1204, Arlingtor of information if it de	burden for this collect needed, and completii ing this burden to Dep I, VA 22202–4302. Res bes not display a curre	ion of information is es ng and reviewing the c partment of Defense, W spondents should be av ntly valid OMB contro	timated to average 1 hour per re ollection of information. Send co /ashington Headquarters Services vare that notwithstanding any ot I number. PLEASE DO NOT F	esponse, including the mments regarding this s, Directorate for Infor her provision of law, r RETURN YOUR FOR	time for revie s burden estir mation Oper- no person sha CM TO THE	ewing instructions, searching existing data sources, gathering and mate or any other aspect of this collection of information, including ations and Reports (0704–0188), 1215 Jefferson Davis Highway, II be subject to any penalty for failing to comply with a collection ABOVE ADDRESS .			
1. REPORT DA	TE (DD-MM-)	YYYY) 2. REPO		3. DATES COVERED (From — To)					
15-09-2014 Doctoral Dissertation						Oct 2011–Sep 2014			
4. TITLE AND	SUBTITLE	Doctor			5a. CON				
Optimal Autonomous Spacecraft Resiliency Maneuvers Using Metaheuristics						5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER			
6. AUTHOR(S)						Sa. PROJECT NUMBER			
Showalter, Daniel J., Captain, USAF						5e. TASK NUMBER			
7. PERFORMIN	IG ORGANIZAT	ION NAME(S)	AND ADDRESS(ES)			8. PERFORMING ORGANIZATION REPORT			
Air Force Ins	titute of Tech	nology				NUMBER			
Graduate Sch	nool of Engine	eering and Ma		AFIT-ENY-DS-14-S-29					
VDAFR OF	way [45422 7765								
	C (MONITOD			SC/FC)					
9. SPONSORIN	IG / MONITOR	ING AGENCY N	AME(S) AND ADDRES	55(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)			
Intentionally	Left Blank			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 natural and presilience by correspondin the extent will the viability and continuo require only and optimization target satelling and near-opt to be defined transfer from this dissertat utilized to op	congestion in nan-made haz altering their g to increased nich resiliency of metaheuris us-thrust resil meters per sec of resiliency n ces while en-re- imal results for . Results indi low Earth or ion demonstra- timize future	space has inc zards. Equipp arrival condit maneuverabit can be presentics to enable liency maneuvers cond of delta-vertice maneuvers whoute to geostate or the problem cate that the bit to geostate ate the viability classes of resi	reased the need for ing satellites with in ions as they enter p- lity requires these m rved. This research their autonomous of rers. The results den- velocity. Additionall ich require a maneu- tionary orbit. The r is investigated and c inspection requires a ionary orbit. The m ty of the autonomou- liency maneuvers.	spacecraft to acreased mane otentially haz aneuvers be a introduces ma ptimization. ' nonstrate tha y, bi-level ever vering spacec nethods devel can be applied an increase of aneuvers and us optimizatio	develop euvering cardous i optimize aneuvers Techniqu t impuls blutionar raft to p loped ar d to futu c only fiv optimiz on of spa	resilience capabilities in response to capability has the potential to enhance regions. The propellant expenditure ed to minimize fuel expenditure and to is to enhance resiliency and investigates uses are developed to optimize impulsive sive and low-thrust resiliency maneuvers cy algorithms are explored in the perform an inspection of one of several e shown to consistently produce optimal are classes of resiliency maneuvers yet re percent of the propellant needed to cation techniques developed throughout acceraft resiliency maneuvers and can be			
15. SUBJECT Optimization	, Metaheurist	ics, Astrodyna	amics						
16. SECURITY CLASSIFICATION OF: 17. LIMITATION OF 18. NUMBER 19a. NAME OF RESPONSIBLE PERSON									
a. REPORT	b. ABSTRACT	c. THIS PAGE	ABSTRACT	OF	Dr. Jo	nathan T. Black (ENY)			
-				PAGES	19b. TE	LEPHONE NUMBER (include area code)			
U	U	U	UU	486	(937) 2	255-3636 x4578 jonathan.black@afit.edu			