Improved Collocation Methods to Optimize Low-Thrust, Low-Energy Transfers in the Earth-Moon System

by

Jonathan F. C. Herman

B.Sc., Aerospace Engineering, Delft University of Technology, 2010

M.Sc., Aerospace Engineering, Delft University of Technology, 2012

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Aerospace Engineering Sciences

2015

This thesis entitled:
Improved Collocation Methods to Optimize Low-Thrust, Low-Energy Transfers in the Earth-Moon System
written by Jonathan F. C. Herman
has been approved for the Department of Aerospace Engineering Sciences

Assistant Professor Dr. Jeffrey S. Parker

Professor Dr. George Born

Assistant Research Professor Dr. Brandon A. Jones

Assistant Research Professor Dr. Jay McMahon

Professor Dr. Bengt Fornberg

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Herman, Jonathan F. C. (Ph.D., Aerospace Engineering Sciences)

Improved Collocation Methods to Optimize Low-Thrust, Low-Energy Transfers in the Earth-Moon System

Thesis directed by Assistant Professor Dr. Jeffrey S. Parker

Modern and near-future Solar Electric Propulsion capabilities enable many new missions that were inconceivable using chemical propulsion systems. Many of these involve highly complex trajectories that are very challenging to design. New tools are needed that effectively utilize the rapidly growing parallel processing capabilities of modern computers. This research improves Gauss-Lobatto collocation methods, which are known to perform very well for low-thrust trajectory optimization, by formulating them as massively parallel processes. The parallelized elements of the problem formulation execute up to 11 times faster, depending on what force model is used and when evaluated by themselves. When accounting for the operations of the nonlinear programming solver, this translates to up to 3.7 times faster performance for solving a complete trajectory optimization problem, again depending on the force model that is used. The remaining barriers to further performance improvements, and the conditions upon which these depend, are clearly identified.

The implemented methods are combined into an optimization tool named *Maverick*. More general improvements to the formulation of the Gauss-Lobatto collocation methods are also developed and included in *Maverick*, which permit a more flexible use of these optimization schemes and enable them to find more complex solutions. One example of this is *Maverick*'s ability to autonomously introduce gravity assists into trajectories, which greatly increases the utility and convergence radius of these methods.

In order to demonstrate the benefit of this work, three applications are studied. The first are transfers between halo-like orbits in the Earth-Moon system, which shows this is likely an unattractive region for missions like the New Worlds Observer. The second application investigates

stabilization maneuvers in lunar distant retrograde orbits. This work demonstrates the feasibility of these stabilization transfers for a variety of sample return missions, such as the upcoming Asteroid Redirect Mission. The final application discussed is a series of multi-body low-thrust transfers from the Earth to the Moon that efficiently utilize highly variable dynamics to reduce propellant consumption, which is relevant for a variety of future mission concepts. These are computed for a wide range of flight times, showing that reductions up to 45% of the transfer time can be achieved with a propellant consumption as little as 0.5% of the total spacecraft mass. Up to 90% of the flight time can be eliminated for a propellant cost of 4% of the total spacecraft mass, or up to 83% for a propellant cost of less than 2%. The developed algorithm seamlessly transitions its solutions from full low-thrust, low-energy trajectories to the 'pure' low-thrust trajectories that define the shortest transfer trajectories, validating its robust performance. Beyond these quantifiable results, these examples illustrate the complexity of the solutions that can be identified with these improved implementations of Gauss-Lobatto collocation methods, with many instances where the optimization method autonomously introduces powered gravity assists, an unusual capability that has the potential for useful application to many other trajectory optimization problems.

Dedication

To all the family and friends I have spent far too little time with for the sake of these pages, particularly Aline, Margreet, and Daan, as well as our dearly missed Gérard.

Acknowledgements

First and foremost, I want to thank my committee members, not just for their valuable and critical input over the years, but for tirelessly working their way through the various iterations of this dissertation. I want to thank George Born and Jeff Parker in particular, for welcoming me into the Colorado crowd years ago and for staunchly supporting me throughout my whole stay. Being a foreign national graduate student comes with more financial challenges and uncertainties than I would have expected, and I could not have asked for better guardians than George and Jeff. I want to give a special mention to Brad Cheetham as well, who had a big role in making my first year at Colorado possible.

I also want to thank and acknowledge the people who helped me find my way through the technical details of this research. Brandon Jones provided me with the initial spark to get interested in collocation methods, which was strengthened by discussions with many others. Dan Grebow deserves special mention, for patiently corresponding with me in person and by e-mail throughout the course of my research on a variety of topics. I also want to acknowledge David Garza, Jacob Englander, Mike Sekerak, and Martin Ozimek, all of whom provided valuable discussions and insights over the years. Similarly, I want to express my gratitude for having been a part of the Colorado Center for Astrodynamics Research, a department full of wonderful and interesting people that have enriched my time as a graduate student in more ways than I could have reasonably expected. I want to explicitly acknowledge the members of the relatively young low-thrust trajectory optimization research group, particularly Nathan Parrish and Jonathan Aziz, and especially Stijn De Smet, who has been the best 'partner-in-crime' I could have hoped for as a Ph.D. student. Under

Jeff's guidance, this group has clawed its way into this research field from practically nothing, and it has been inspiring, exciting, and tremendously educational to have been a part of this.

Of course, I also want to thank all my family and close friends who graciously tolerated my continuous absence for weeks, months, or even years. Not only have I had to settle for only a few weeks out of the year with all my friends and family back in the Netherlands, where I am always received by so many as though we had only seen each other yesterday when in fact it may have been years, I have also had to split my time equally between Pasadena and Boulder, two cities more than 1500 km apart. It would have been easy to have ended up with practically no friends in either place, so I consider myself very lucky to have made many friends in both places. I especially want to thank my brother Daan, who has somehow made my own habit of living abroad seem like a walk around the block, for being one of the people closest to me even when he is farther away than anyone I know. On the same note, I want to thank my mother, for not just tolerating but embracing both Daan's and my own meandering across an increasingly-smaller-but-still-pretty-big-for-a-single-family-to-scatter-across planet, usually in opposite directions. Any mother who not only endures an hour-long Ph.D. defense with full attention, but voluntarily suffers jet lag and a trans-Atlantic flight just to attend said ordeal, should be given an award. Even after nearly 30 years of knowing her, I continue to be amazed when she yet again gives me unusually keen professional advice, despite having no technical background. And finally, I want to thank Aline for, well, pretty much all of the various things I've thanked people for above. She has been more patient with my absence than anyone, she has been the greatest source of support during times of stress, and over the years she has given me an abundance of valuable and insightful technical advice. This work would not have been the same without her support.

Contents

Chapter

1 Introduction 1

2 Literature Survey 4

2.1 Electric Propulsion Overview 5

2.1.1 Electric propulsion systems 8

2.1.2 Trajectory Optimization 10

2.2 Low-Energy Trajectories 12

2.3 Significant Missions & Concept Studies 12

2.3.1 GRAIL 13

2.3.2 SMART-1 13

2.3.3 Dawn 14

2.3.4 Low-Thrust, Low-Energy Earth-Moon transfer concept studies 14

2.4 Parallel Computing 14

2.5 Chapter conclusion 17

3 Using Collocation Methods for Trajectory Optimization 18

3.1 Optimization through direct transcription 19

3.2 Gauss Lobatto collocation 23

3.2.1 4^{th}-order 23

3.2.2 12^{th}-order 25

3.3 Mesh refinement . . . 28
3.3.1 Equidistribution . . . 30
3.3.2 Error Reduction . . . 31
3.3.3 Mesh refinement process . . . 31
3.4 Alternative schemes . . . 33
3.5 Implementation . . . 35
3.5.1 Sparsity . . . 36
3.5.2 Scaling . . . 38
3.5.3 Force models . . . 41
3.5.4 Hyperbolic thrust constraints . . . 42
3.5.5 Objective function . . . 45
3.5.6 Additional constraints . . . 47
3.6 Differences from earlier implementations . . . 47
3.7 Chapter conclusion . . . 48

4 Parallel results 49
4.1 Overall performance . . . 49
4.2 Raw Performance results . . . 53
4.3 Full Performance Results . . . 58
4.4 Chapter summary . . . 60

5 Transfers between Halo-like orbits 62
5.1 Assumptions . . . 62
5.2 Results . . . 64
5.2.1 45-day transfer . . . 65
5.2.2 36-day transfer . . . 68
5.2.3 27-day transfer . . . 71
5.3 Discussion of results . . . 74

6 DRO stabilization transfers 75

6.1 Assumptions . . . 76

6.2 Results . . . 78

6.2.1 7-day transfer . . . 80

6.2.2 41-day transfer . . . 83

6.2.3 47-day transfer . . . 86

6.2.4 49-day transfer . . . 89

6.3 Discussion of results . . . 92

7 LEO to DRO transfers 93

7.1 Assumptions . . . 94

7.1.1 Accuracy of trajectories . . . 95

7.1.2 Additional modifications to problem formulation . . . 96

7.2 Results . . . 98

7.2.1 187-day transfer . . . 101

7.2.2 168-day transfer . . . 104

7.2.3 140-day transfer . . . 107

7.2.4 103-day transfer . . . 110

7.2.5 32-day transfer . . . 113

7.2.6 18-day transfer . . . 116

7.3 Results for lower power . . . 119

7.4 Discussion of results . . . 121

8 Conclusion 122

8.1 Significant contributions . . . 125

8.2 Future Work . . . 127

Bibliography 133

Appendix

A Code Architecture 139

B Performance Variations 144

Tables

Table

3.1 Node positioning on the interval [0,1] for the GL-12 scheme, reproduced from Reference [47]. 26

3.2 Constant coefficients for the GL-12 scheme, as found in Reference [47]. 28

3.3 Sparsity as a function of segments for the 4^{th} order Gauss-Lobattto scheme. 37

3.4 Sparsity as a function of segments for the 12^{th} order Gauss-Lobattto scheme. 38

4.1 Maximum achievable performance improvements for a variety of force models under current conditions. 50

4.2 Absolute and relative performance of constraint (Cons.) and derivative (Deriv.) calculations for 4^{th} order Gauss-Lobattto scheme using the two-body and 50x50 force models. Absolute performance refers to the total runtime of 10,000 constraint or 1,000 derivative evaluations, respectively. 56

4.3 Absolute and relative performance of constraint (Cons.) and derivative (Deriv.) calculations for 12^{th} order Gauss-Lobattto scheme using the two-body and 50x50 force models. Absolute performance refers to the total runtime of 10,000 constraint or 1,000 derivative evaluations, respectively. 56

4.4 Absolute and relative performance of full optimizations for the 50x50 force model and the 4^{th} order Gauss-Lobattto scheme. 60

4.5 Absolute and relative performance of full optimizations for the 50x50 force model and the 12^{th} order Gauss-Lobattto scheme. 60

Figures

Figure

2.1 A contour plot showing ΔV against mass fraction for a range of specific impulses. . 7

2.2 Trade-off between propellant mass and power source mass for electric propulsion, assuming a constant level of thrust. Reproduced from Reference [16]. 8

2.3 Performance for three different engines as a function of input power, showing thrust (left) and specific impulse (right). 10

2.4 35 years of microprocessor trend data, obtained from Reference [55] 15

3.1 Illustration of a 4^{th} order Gauss-Lobatto integration segment, adapted from Reference [48] 24

3.2 Illustration of a 12^{th} order Gauss-Lobatto integration segment, adapted from Reference [47] 25

3.3 Representation of a toy problem cost function with two unscaled input parameters. The arrow indicates the direction of the derivative at the innermost contour line of the cost function. 39

3.4 Representation of a toy problem cost function with two scaled input parameters. The arrow indicates the direction of the derivative at the innermost contour line of the cost function. 40

3.5 Comparison of a hyperbolic thrust constraint with $\epsilon_H = 0.225$ and a traditional thrust constraint ($T_y = T_z = 0$) 43

3.6 Comparison of a hyperbolic thrust derivative with $\epsilon_H = 0.225$ and a traditional thrust derivative ($T_y = T_z = 0$) 44

3.7 Comparison of a hyperbolic thrust constraints for various ϵ_H ($T_y = T_z = 0$) 46

4.1 Illustration of relative performance for the constraint evaluations, showing the 4^{th} and 12^{th} order methods with circular and triangular markers respectively. The GPU implementation is shown in a solid linestyle, with the OpenMP implementation in a dashed linestyle. The two-body force model is shown in black, with the 50x50 aspherical gravity force model shown in red. 53

4.2 Illustration of relative performance for the derivative evaluations, showing the 4^{th} and 12^{th} order methods with circular and triangular markers respectively. The GPU implementation is shown in a solid linestyle, with the OpenMP implementation in a dashed linestyle. The two-body force model is shown in black, with the 50x50 aspherical gravity force model shown in red. 54

4.3 Illustration of relative performance for the derivative evaluations for a low number of segments, showing the 4^{th} and 12^{th} order methods with circular and triangular markers respectively. The GPU implementation is shown in a solid linestyle, with the OpenMP implementation in a dashed linestyle. The two-body force model is shown in black, with the 50x50 aspherical gravity force model shown in red. 55

5.1 Final mass as a function of time of flight for the starshade transfers. The red dashed line indicates the initial mass, and the red marked points indicate those that are presented in more detail below. 64

5.2 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 45-day starshade transfer. 65

5.3 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 45-day starshade transfer. Both the Earth and the Moon are shown to scale. 66

5.4 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 45-day starshade transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 67

5.5 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 36-day starshade transfer. 68

5.6 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 36-day starshade transfer. Both the Earth and the Moon are shown to scale. 69

5.7 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 36-day starshade transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 70

5.8 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 27-day starshade transfer. 71

5.9 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 27-day starshade transfer. Both the Earth and the Moon are shown to scale. 72

5.10 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 27-day starshade transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 73

6.1 XY- and XZ-projections of the target Distant Retrograde Orbit. The upper figures show the Earth-Moon rotating frame, whereas the lower figures show the Sun-Earth rotating frame. All figures show the same 100 day propagation, starting at the arrival date used for the transfers in this chapter. 76

6.2 Final mass as a function of time of flight for the DRO stabilization transfers. The red dashed line indicates the initial mass, and the red marked points indicate those that are presented in more detail below. 78

6.3 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 7-day DRO stabilization transfer. 80

6.4 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 7-day DRO stabilization transfer. Both the Earth and the Moon are shown to scale. 81

6.5 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 7-day DRO stablization transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 82

6.6 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 41-day DRO stabilization transfer. 83

6.7 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 41-day DRO stabilization transfer. Both the Earth and the Moon are shown to scale. 84

6.8 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 41-day DRO stablization transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 85

6.9 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 47-day DRO stabilization transfer. 86

6.10 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 47-day DRO stabilization transfer. Both the Earth and the Moon are shown to scale. 87

6.11 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 47-day DRO stablization transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 88

6.12 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 49-day DRO stabilization transfer. 89

6.13 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 49-day DRO stabilization transfer. Both the Earth and the Moon are shown to scale. 90

6.14 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 49-day DRO stablization transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 91

7.1 Final mass as a function of time of flight for the LEO-DRO transfers. The red dashed line indicates the initial mass, and the red marked points indicate those that are presented in more detail below. 98

7.2 Launch energy as a function of time of flight for the LEO-DRO transfers. The red dashed line indicates the initial mass, and the red marked points indicate those that are presented in more detail below. 99

7.3 Normalized representation of radial distance to the Earth and Moon as a function of time of flight for the ballistic 187-day LEO-DRO transfer. 101

7.4 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for the ballistic 187-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale. 102

7.5 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for the ballistic 187-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 103

7.6 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 168-day LEO-DRO transfer. 104

7.7 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 168-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale. 105

7.8 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 168-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 106

7.9 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 140-day LEO-DRO transfer. 107

7.10 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 140-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale. 108

7.11 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 140-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 109

7.12 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 103-day LEO-DRO transfer. 110

7.13 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 103-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale. 111

7.14 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 103-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 112

7.15 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 32-day LEO-DRO transfer. 113

7.16 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 32-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale. 114

7.17 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 32-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 115

7.18 Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for an 18-day LEO-DRO transfer. 116

7.19 The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for an 18-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale. 117

7.20 The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for an 18-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray. 118

7.21 Final mass as a function of time of flight for the LEO-DRO transfers for an 1800 kg vehicle with 6 kW of power. 119

7.22 Launch energy as a function of time of flight for the LEO-DRO transfers for an 1800 kg vehicle with 6 kW of power. 120

A.1 The structure and flow of the code that defines *Maverick* when set to use the serial CPU version of the 12^{th} order Gauss-Lobatto scheme. Bold text within blocks indicates the general function, whereas plain text within blocks shows the filename of the actual file(s) used for this. Red blocks represent Python code, blue blocks represent C code, and the yellow block is the NLP solver, in whatever language the selected NLP solver uses. The dashed arrows indicate blocks that are essentially subroutines of the blocks that the dashed arrows originate from. The solid arrows indicate significant transitions from one phase in the program to another. The red dashed box outlines where more than 99% of the computational effort occurs. 140

A.2 The structure and flow of the code that defines *Maverick*, with all different methods included. Bold text within blocks indicates the general function, whereas plain text within blocks shows the filename of the actual file(s) used for this. Red blocks represent Python code, blue blocks represent C code, and the yellow block is the NLP solver, in whatever language the selected NLP solver uses. The dashed arrows indicate blocks that are essentially subroutines of the blocks that the dashed arrows originate from. The solid arrows indicate significant transitions from one phase in the program to another. The red dashed box outlines where more than 99% of the computational effort occurs. 143

Chapter 1

Introduction

Electric propulsion is a promising technology that is rapidly becoming mature enough for use in a number of space flight applications. While first tested before the human exploration of the Moon, its technological maturity was lacking for decades. Gradually, capabilities increased and applications became more widespread, but even today it is mainly used as a station-keeping system. Capabilities have now developed to a point where spacecraft like Dawn show that missions exist that can only be performed by the grace of the order of magnitude increase in efficiency provided by electric propulsion. Meanwhile, a number of studies show the wide variety of other opportunities within reach. Much like the turbofan enabled extreme performance at a manageable cost in aviation, electric propulsion offers a path for space flight to consider missions with requirements that were previously inconceivable, while also enabling a reduction in costs of present activities. However, with these benefits comes the burden of significantly more complex spacecraft trajectories. With the engines running for months or years on end, possibly having a failure at any moment throughout that time, there is more responsibility on the trajectory designer to prove that their trajectory is reliable and safe. This combines significantly more expensive analysis of a single trajectory with the need to study many more variations of that trajectory, resulting in a great deal more work than typically encountered for spacecraft trajectory design. While a number of tools exist that can navigate these problems, they are few in number, demanding in performance, and in many cases fail to make full use of the parallel processing capabilities of modern computers.

This research aims to enhance the field of low-thrust trajectory optimization by improving the use of massively parallel computing, which is key to taking advantage of the full computational power of modern hardware. Improvements are also made to collocation methods that allow them to be used more flexibly by the designer, and enables them to find more complicated solutions, such as those where the optimization method autonomously introduces gravity assists. Based on this work a new optimization tool is developed and named *Maverick*. Its capabilities are tested by studying three types of low-thrust, low-energy problems, spread across a range of dynamical regimes. Ultimately, this work develops capabilities that make it easier to study the challenging trajectory problems that come hand-in-hand with the new mission concepts enabled by electric propulsion, and studies some of these problems as a demonstration of what these newly developed capabilities can deliver.

In the following chapter, a literature survey is presented that summarizes the basic workings of electric propulsion, the most prevalent form of low-thrust propulsion. It also presents some basics on trajectory optimization, and provides some performance details of real electric propulsion systems. Several missions and concept studies that have particular relevance for the current work are also highlighted in this chapter, as well as a brief discussion of parallel computing with modern hardware. Next, Chapter 3 goes into much greater detail on trajectory optimization, and specifically the collocation schemes that are used for this work. It thoroughly discusses how these methods are implemented for this research in the *Maverick* optimization tool, and what general improvements are made with respect to earlier efforts. Chapter 4 then illustrates the performance improvements that can be achieved with the parallelized implementation of these methods, as well as what the most important remaining barriers are to further performance improvements. Chapter 5 presents a brief analysis on transfers between halo-like orbits in the Earth-Moon system, which has implications for missions like the New Worlds Observer. Next, a similar brief study is presented in Chapter 6 for stabilization maneuvers in lunar distant retrograde orbits. This work has implications for a variety of sample return missions, such as the upcoming Asteroid Redirect Mission. Chapter 7

then presents a detailed analysis of a numerically sensitive multi-body low-thrust transfer from the Earth to the Moon that utilizes highly variable dynamics to improve propellant efficiency, which is relevant for a variety of future mission concepts. Finally, Chapter 8 summarizes the conclusions of this work, highlights the significant contributions presented in this thesis, and provides suggestions for future work to effectively build on the work presented here.

Chapter 2

Literature Survey

This chapter provides the majority of the background that is relevant to this research. The brunt of this covers electric propulsion, in Section 2.1. While the optimization methods developed for this research apply to any type of low-thrust propulsion, and can be set up to do so fairly easily, electric propulsion is far more mature than other types, such as the different types of solar sails. Nonetheless, throughout this report it should be kept in mind that these optimization methods can easily be used for any of these. Unless explicitly mentioned otherwise, it is assumed throughout this report that propulsion is implemented through Solar Electric Propulsion (SEP). While nuclear power, rather than solar power, is highly compatible with existing engines, it is unlikely that nuclear reactors will be flying in space in the near future, and Radioisotope Thermoelectric Generators (RTGs) typically provide insufficient power. As a result, any other form of power generation than solar is going to supply only very limited amounts of power, and will not be considered in detail here. Some general discussion of low-thrust trajectory optimization is given in Section 2.1.2, but the specifics relevant for this research are saved for Chapter 3. Section 2.2 introduces some concepts for low-energy trajectory design. Section 2.3 highlights several missions and studies that are particularly relevant for the current research. Finally, Section 2.4 provides some details on modern trends in computational hardware, specifically their increasingly parallel nature, and the chapter is concluded in Section 2.5.

2.1 Electric Propulsion Overview

Electric propulsion (EP) has grown steadily over the last decades in the world of spaceflight. It is characterized by having a thrust that is several orders of magnitude smaller than that of traditional chemical propulsion, with a powered phase of the trajectory that is much longer to compensate for this. The main advantage of electric propulsion is the much higher energy efficiency of the system than that of traditional chemical propulsion, resulting in much less required propellant. Furthermore, these systems have proven to be extremely reliable, considering a recent NASA review which concluded that "over 240 Hall thrusters have been flown since 1971 with 100% success rate" [45]. However, using these systems often results in a longer trajectory as well as more complicated equations of motion.

Rocket propulsion can be considered a fundamentally different means of propulsion than any other application. Other ways of propulsion, such as walking, sailing, driving or flying, involve some sort of 'engine' (the term is loosely applied here) that generates a force on a surrounding medium. This medium can be the ground, water, air, etc. By applying this force to a medium, an equal and opposite reaction force is created that propels the 'vehicle'. However, rocket propulsion does not only generate the force itself, it also supplies the medium that is brought into motion with this force, in the form of the jet exhaust. This allows rocket propulsion to operate in a vacuum, where there is no medium available to generate a force on.

Another fundamental difference exists between chemical rocket propulsion and electrical rocket propulsion. Chemical propulsion has a propellant that both provides the power, through combustion, and also acts as the medium being expelled. Electric propulsion separates the power supply from the propellant. Power can be generated in a number of ways, such as a solar array or a nuclear reactor. This power is then used to generate a force which expels a propellant. This results in a much lower thrusting force, but much higher exhaust velocities can be attained, as is shown below. There is also the advantage that very safe propellants can be used, such as the chemically inert noble gases, since they are not required to combust for the engine to work.

A good measure for the efficiency of a means of rocket propulsion is the specific impulse:

$$I_{sp} = \frac{T}{\dot{m} g_0} = \frac{\dot{m} V_j}{\dot{m} g_0} = \frac{V_j}{g_0} \tag{2.1}$$

Where T is the thrust, $\dot{m}$ is the mass flow, V_j is the exhaust velocity of the propellant and g_0 is the standard acceleration of free fall. The I_{sp} is measured in seconds, which makes it a very universal unit. It is related to the efficiency of a rocket through the following relation, which is a variant of Tsiolkovsky's equation:

$$\frac{M_f}{M_0} = e^{-\frac{\Delta V}{V_j}} = e^{-\frac{\Delta V}{I_{sp} g_0}} \tag{2.2}$$

This equation is visualized in a contour plot in Figure 2.1. This figure shows contours for different specific impulses. The red curves show the performance range of the best chemical propulsion systems available. The black curve shows the performance of the NASA Evolutionary Xenon Thruster (NEXT) engine, one of the highest performing electric thrusters currently available, discussed in more detail in Section 2.1.1. The figure also identifies the typical ΔV to get to Geostationary Earth Orbit (GEO) from Low Earth Orbit (LEO) or a Geostationary Transfer Orbit (GTO), as well as two previously flown SEP missions that are discussed below, with Dawn quite literally off the chart. Clearly, for a given ΔV the higher the I_{sp} (or rather, the rocket exhaust velocity), the greater the fraction will be of the final mass M_f over the initial mass M_0. In other words, the greater the velocity at which a propellant is expelled, the less propellant will be required.

Although this exhaust velocity has no theoretical upper limit for electric propulsion (except for the speed of light), there is a rather severe limitation through the following equation:

$$P_j = \eta_j P_0 = \frac{1}{2} \dot{m} V_j^2 = \frac{1}{2} T V_j \tag{2.3}$$

Where P_j is the power of the exhaust jet, P_0 is the power supplied by the power source, η_j is the jet efficiency and T is the thrust. The jet efficiency is the fraction of the engine input power that is converted into thrust. Clearly, when limited power is assumed, this also implies a limit on the exhaust velocity (even when the thrust is very low). Equation 2.3 explains why electric propulsion

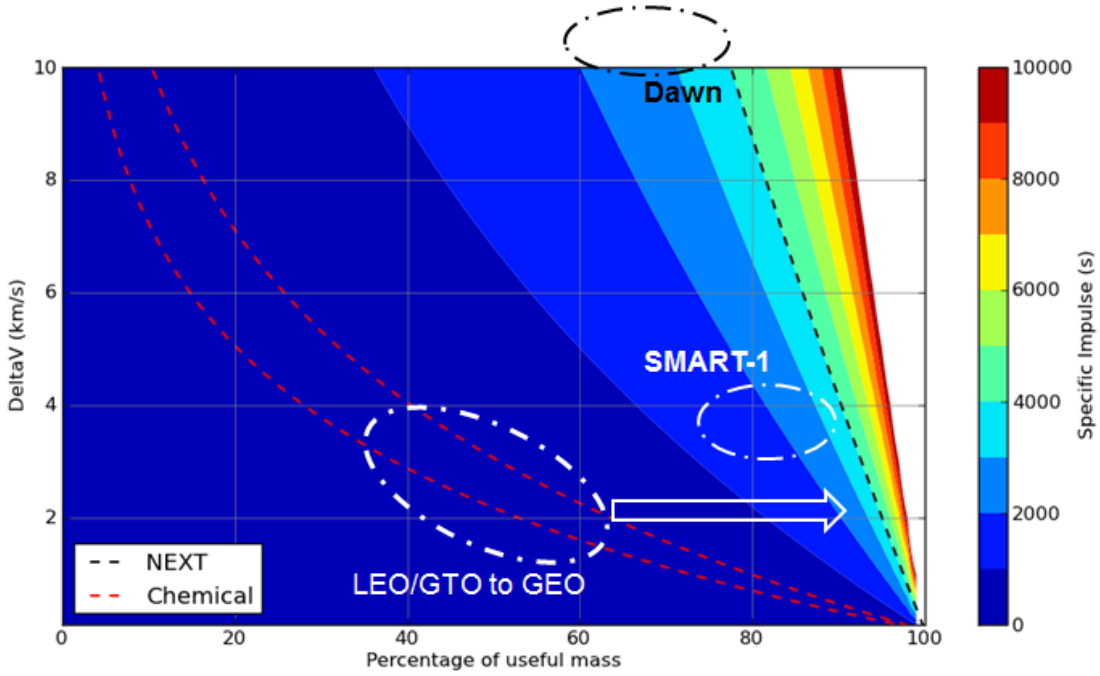


Figure 2.1: A contour plot showing ΔV against mass fraction for a range of specific impulses.

operates at a very low thrust: doing so allows for a high exhaust velocity at a given power, which results in a high efficiency.

The above discussion explains why electric propulsion systems are considered to be **power limited**: because the only factor theoretically limiting the exhaust velocity is available power. When more and more power is designed to be available, higher exhaust velocities become possible, but this will inevitably increase the mass of the power source, which results in a higher system mass. As can be seen in Figure 2.2, there is an optimum value for the specific impulse at a given thrust, and thus also the available power. By contrast, chemical systems are considered to be **energy limited**, resulting from the fact that they have a limited amount of energy per unit mass, which introduces its own limit on the achievable exhaust velocity.

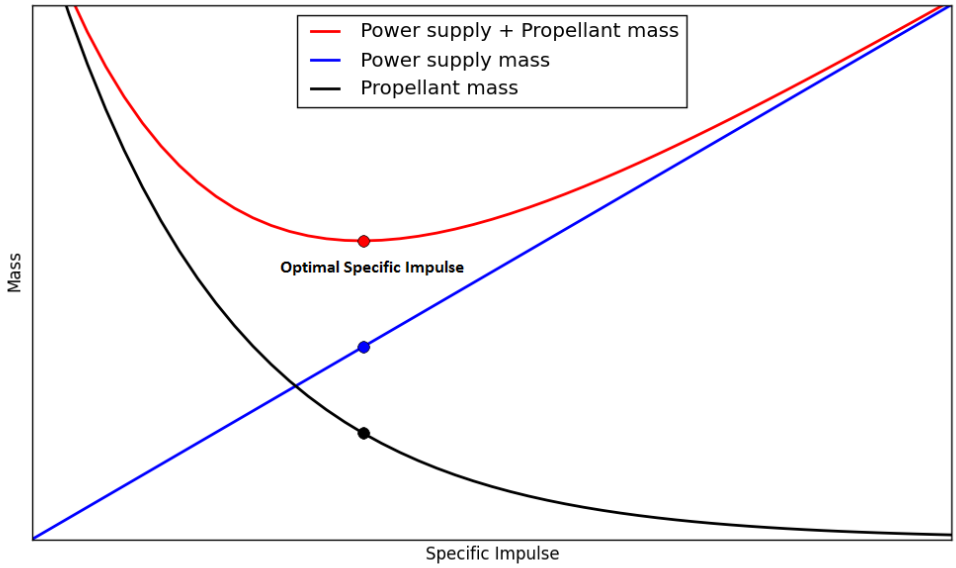


Figure 2.2: Trade-off between propellant mass and power source mass for electric propulsion, assuming a constant level of thrust. Reproduced from Reference [16].

2.1.1 Electric propulsion systems

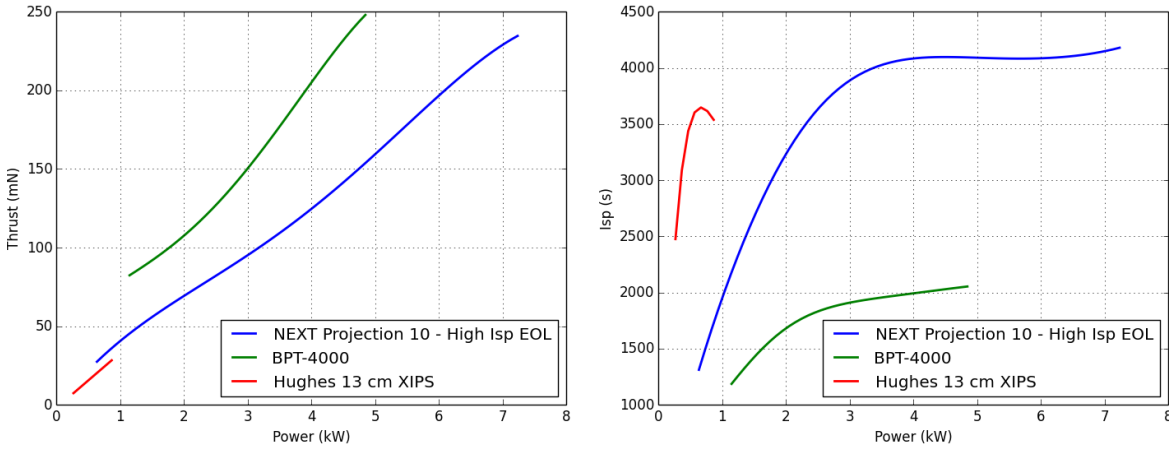
Reference [68] identifies three fundamental categories for electrical propulsion. A detailed discussion of their workings is not provided here, but Reference [22] is an excellent source for in-depth coverage of electric propulsion systems and the fundamentals behind them. The general types of electric propulsion systems can be globally summarized as follows:

- **Electrothermal propulsion**: These systems heat a propellant electrically and then thermodynamically expand and accelerate this propellant. This can be done through resistojets or arcjets. Resistojets heat high-resistance metal parts that heat a propellant that flows over them, arcjets heat the propellant by passing it directly through an electric arc discharge. This type of propulsion is mostly used for attitude control and station-keeping of satellites, and in some cases for orbit change maneuvers.

- **Electrostatic/Ion propulsion**: These systems ionize propellant atoms through electron bombardment, electron cyclotron resonance or direct extraction from the liquid phase at the ion source. Positive ions are accelerated to a high velocity by an electric field established between the ion source and an accelerating electrode. In order to avoid the spacecraft building up a negative electrical load, the exhaust jet is electrically neutralized by injecting a stream of electrons into the exhaust beam. This type of propulsion is mostly used for orbit transfer maneuvers and for planetary missions, because it can attain high exhaust velocities, provides a good exhaust power over mass ratio and because it is technologically mature.

- **Electromagnetic propulsion**: These systems create an acceleration by an interaction between electric and magnetic fields on a highly-ionized propellant plasma. Stationary plasma thrusters use the Hall effect to set up an electrostatic field, which accelerates the propellant ions. Magnetoplasmadynamic thrusters use an electrical arc discharge, like an arcjet. This creates an interaction between the arc and the self-induced and applied magnetic fields. Pulsed plasma thrusters accelerate the propellant plasma by the interaction of a generalized azimuthal current with a magnetic field from a coil current.

Beyond these general types, a lot of variety in performance exists within each category, depending on what the system has been designed for. Accurate performance data for an extensive list of both existing and conceptual electric propulsion systems is available through Reference [24]. Data is available for the thrust and massflow as a function of input power. Using these curves the specific impulse and jet efficiency at a given power setting can also be calculated. Thrust and specific impulse curves for three different engines are shown in Figures 2.3(a) and 2.3(b), respectively. The BPT-4000 is a Hall thruster, whereas NEXT and the Hughes Xenon Ion Propulsion System (XIPS) are electrostatic ion thrusters. These plots show the typical distinction of Hall thrusters having higher thrust than electrostatic counterparts, but a lower specific impulse. Despite being theoretically similar systems, there is also a clear difference among the NEXT and XIPS, particularly in



(a) (b)

Figure 2.3: Performance for three different engines as a function of input power, showing thrust (left) and specific impulse (right).

the specific impulse, with the XIPS clearly designed for operation at a much lower level of input power. This illustrates that even within the different categories of electric propulsion significant performance differences exist.

As a final note on engines, some electric propulsion systems can be scaled down to very small sizes. This opens up the possibility of providing highly efficient means of propulsion for vehicles as small as cubesats, with several parties already successful in developing these systems [50, 56], although none have flown to date. Often, these engines operate through slightly different principles than their larger counterparts, but the resulting high efficiency combined with very low thrust results in characteristically identical trajectories. These technologies enable dramatically new applications for the extremely mass- and volume-constrained cubesats. In the rest of this thesis, detailed specifics of engines are no longer discussed.

2.1.2 Trajectory Optimization

Trajectory design for electric propulsion has proven to be a challenging topic. It involves optimization of a trajectory with an engine that is active for a long time and continuously changing the trajectory, which is radically different from most historical mission design experience. Further-

more, for a true optimization, one cannot assume optimizing the trajectory to be sufficient, and the available spacecraft power as well as the specific impulse of the engine need to be included in the optimization, further complicating the process. For relatively simple trajectories, involving simple dynamics and a low number of revolutions about the central body, tools have matured well in the past years, and they are usually some sort of variation of direct optimization methods [4]. In particular, Sims-Flanagan based methods [61, 9, 29], such as MALTO [60], BOLTT [62] or EMTG [17], have proven to be very effective at solving these problems. A detailed example of the workings of a tool of this type as developed by the author, as well as its application, is given in Reference [29]. These methods apply particularly well to interplanetary transfers, which are well approximated (for mission design purposes) by two-body motion and rarely have more than 10 revolutions about the Sun before arriving at their destination. For more complex problems that require many revolutions or complex dynamics, a few tools also exist, such as Mystic [69] or ITOP [18]. However, these problems typically still require significant computational effort, and often suffer from needing accurate initial guesses to converge to an optimal solution. In fact, for these complex problems, ITOP relies on many runs of a separate optimization tool, HYTOP [34], just to get started. In addition, most existing methods do not utilize the parallel nature of modern computational hardware, and are often fundamentally limited in doing so, which leaves room for a lot of improvement considering the modern hardware trends described in Section 2.4.

All of this makes the thorough analysis of all but the simplest low-thrust trajectories cumbersome. With only a few tools available at all, that all take a significant time to run and may require much work for developing an initial guess just to get them started, there is a lot of room for improvement. These issues become particularly relevant when many small variations of a trajectory need to be studied, for example to convince the operators that the planned maneuver sequence does not have any catastrophic sensitivities. This can require significant computational expense, an obstacle that is undesirable for any space mission, but particularly relevant for smaller spacecraft. These vehicles will in the near future have access to low-thrust propulsion systems. Despite their budgets being orders of magnitude smaller than regular space missions, they will likely receive

no leniency in demonstrating that their trajectories pose no risk to other spacecraft, making the expense of performing these computations a much larger fraction of their mission budget.

2.2 Low-Energy Trajectories

Over the past two decades, it has become increasingly popular to design spacecraft trajectories that utilize the gravitational attraction of more than one body to create transfers with a superior efficiency, although typically with a long flight time. This effort has been aided by the increase in computational capability, making the study of these trajectories much more accessible. The term 'low-energy' refers to the low fuel and therefore low energy required to control the trajectory from a given starting condition to a targeted final condition [35]. The relevance to the presented work is that some of the trajectories discussed in later chapters will operate in low-energy regimes, where multi-body dynamics play a significant role. A basic but powerful formulation for low-energy trajectories is the Circular Restricted Three-Body Problem (CR3BP), which is extensively discussed in References [65] and [35], among many others. The details of these dynamics are not repeated here since they are not used directly, although knowledge of them can be helpful to fully appreciate the results presented in this thesis.

One key characteristic of low-energy trajectories is that they are an example of chaotic motion, meaning a small perturbation early in the trajectory can result in a large difference in the final state. This ties in very well with the low-thrust nature of SEP, since these low-energy trajectories allow for that low thrust to be put to maximum effect, if combined properly.

2.3 Significant Missions & Concept Studies

A number of missions are highlighted in detail below, due to their relevance to the proposed research.

2.3.1 GRAIL

The Gravity Recovery and Interior Laboratory (GRAIL) mission was launched by NASA in 2011, consisting of two spacecraft that flew in a tight formation in lunar orbit to produce a high-quality gravity model of the Moon [51]. In order to get to this orbit, the gravitational attraction of the Sun, Earth, and Moon were all utilized. This resulted in a transfer that took three months instead of a few days, but it greatly reduced the propellant requirements for the mission. In addition, applying these dynamics aided in operational complications with placing two spacecraft in a very close orbital formation at low lunar altitude, such as the timing of spacecraft maneuvers and the magnitude of the maneuvers on arrival. Both the prime and extended mission were a success, with both spacecraft intentionally crashing into the Moon in late 2012. During the prime mission GRAIL orbited at an average altitude of 55 km, but during the extended mission this was reduced to an average of 23 km, placing the vehicles within 8 km of some of the Moon's higher surface features [70] (which happens to be lower than the cruise altitude for aircraft on Earth).

2.3.2 SMART-1

Launched in 2003, the European Space Agency (ESA)'s Small Missions for Advanced Research in Technology-1 (SMART-1) spiraled up from GTO and entered a polar orbit at the Moon, all accomplished with a SEP system. At the end of the mission, it was purposefully crashed into the lunar surface. It essentially passed through the vicinity of L_1 and effectively navigated a low-thrust, low-energy transfer. SMART-1 used Snecma's PPS-1350-G thruster, a Hall effect thruster operating nominally at an I_{sp} of 1660 and a thrust of 90 mN [36]. Trajectory design largely used control laws to create a spiral out of the initial orbit, and similarly a spiral backwards in time from the target orbit. Optimization efforts focused on patching these orbits together [58]. This provided an effective approach to develop a successful and historic transfer trajectory, but it was improved upon later with more advanced optimization techniques [5].

2.3.3 Dawn

Dawn [52] was launched in 2007 and is still active. It operates three NASA Solar Technology Application Readiness (NSTAR) engines to deliver a total ΔV of more than 11 km/s, orbiting two separate planetary bodies (Vesta and Ceres), each of which has never been previously visited. Both this high ΔV and the orbiting of two separate celestial bodies on a single mission is unprecedented. Furthermore, with 10kW of power at 1 AU, Dawn is the most powerful SEP mission ever flown. Altogether, Dawn serves as an excellent example of how SEP can be used to fly spacecraft with extreme performance at a low cost.

2.3.4 Low-Thrust, Low-Energy Earth-Moon transfer concept studies

As mentioned in Sections 2.2 and 2.3.1, using low-energy transfers to the Moon is an extremely efficient way of performing this transfer. Designing these for low-thrust trajectories has proven to be tricky, however, in large part due to the large and sensitive optimization problem this poses. Nonetheless, some work in this area has been performed, most notably that presented in References [41, 40, 42]. These all involve a handful of point designs, rather than an expansive search of the parameter space, which was no doubt related to the very long computation times identified in Reference [41], which was on the order of 40 hours per trajectory. Despite that, these point designs do reveal that promising low-propellant transfers exist.

2.4 Parallel Computing

Throughout the majority of the history of computing, increased computational performance was achieved through putting more transistors on a single chip, with operations performed in a serial fashion, using a single 'thread' that queues competing processes. As shown in Figure 2.4, single-thread performance has shown a convergence to a maximum performance as of the mid-2000s. Since then, much of the improved performance for computational power has come from using multiple processors, and running different processes in parallel rather than serial. This has been true for

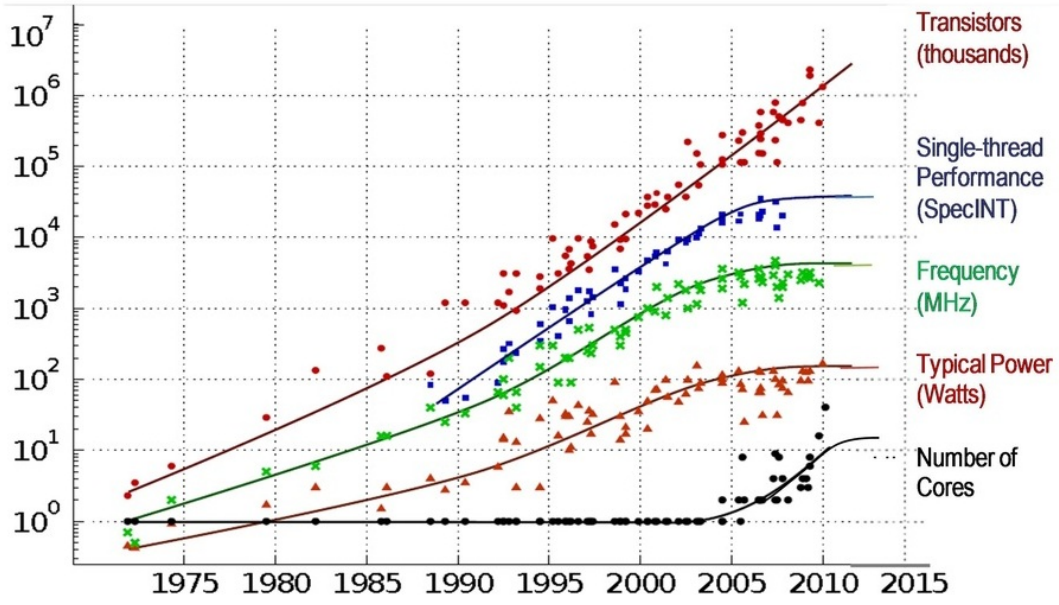


Figure 2.4: 35 years of microprocessor trend data, obtained from Reference [55]

the Central Processing Unit (CPU), which today frequently has 4 separate cores or more, but is particularly true for the Graphics Processing Unit (GPU). Deriving their name from their origin in the video gaming industry, GPUs today can have many hundreds or thousands of individual processors. Until the late 2000s, using these for anything other than graphical applications was extremely cumbersome. It essentially required researchers to trick the GPU into thinking it was computing a color for a surface, whereas instead it was perhaps computing the temperature of a cell in a mesh of a mechanical structure. A great insight into this early history of GPU computing is given in the introductory pages of Reference [57]. As of the late 2000s, it has become much more straightforward to program for GPUs, and while still more complicated than writing code for a CPU, the similarities between the two are now very high. This opens the door to more widespread use of these devices, but it does necessarily require that the application can be run as a massively parallel process. Assuming this is the case, it is not uncommon for GPUs to improve runtime by a factor of 10 or more [43]. It should be noted that GPUs typically require less power for a given amount of work than CPUs do. This has made them very attractive for supercomputers, where cost is dominated not by the cost of the hardware, but by the cost of the power used throughout its life.

It is interesting to keep in mind that spacecraft have an extremely competitive power budget as well. Combined with the robust nature of a GPU by having a large number of processors, it could be concluded that in the future these devices may help to significantly boost on-board computing power for spacecraft. It should be noted that at the time of writing the author is not aware of any GPU being flight-rated for space missions. While this potential boost in on-board computing power may not be relevant in the very near future, it is nonetheless interesting to note that both for energy efficiency and redundancy reasons, the latter due to having many individual processors, GPUs theoretically make for very attractive on-board components, which has implications for algorithms that can take advantage of their architecture.

In any case, there is strong appeal for any community that handles computationally intensive tasks to transition to algorithms and formulations that can take advantage of parallel computing. The astrodynamics community has developed several useful GPU applications in an effort to keep up with the developing industry, but there are many areas where more development is needed. Some great examples of parallel computing in astrodynamics can be found in References [2, 54, 37, 12, 11, 3, 43]. Typically applications have used parallel computing for processes that are inherently parallel, such as a Monte Carlo analysis. Yet if (some of) the inherently serial processes in the community can be reformulated as parallel processes, a greater set of tasks could take advantage of future improvements in hardware. As Figure 2.4 shows, no noteworthy growth in computational performance can be expected from individual processors in the near future, locking in the runtime of any algorithm that relies solely on serial performance for many years to come. Algorithms that can utilize parallel performance, on the other hand, may already see a performance improvement today, which through no further coding effort would continue to grow due to the improvements of hardware over time, increasing the gap between serial and parallel formulations with every year that goes by.

2.5 Chapter conclusion

This Chapter contains a brief overview of a variety of concepts that are important to the work performed in this thesis. All topics in this Chapter could be discussed to much greater length, but for the sake of brevity no attempt to do so is made, since all key concepts are already covered. The exception to this is trajectory optimization, which is discussed in far greater detail in the next chapter.

Chapter 3

Using Collocation Methods for Trajectory Optimization

Earlier work has shown great potential for a variety of collocation-based low-thrust trajectory optimization techniques [28, 48, 47, 26, 5]. Specifically, techniques using direct transcription of Gauss-Lobatto collocation of varying orders have been successfully applied to a number of problems such as lunar south-pole orbiters [48, 26], or a SMART-1 type transfer to a lunar polar orbit [5]. These schemes do not solve the original low-thrust trajectory optimization problem, but instead solve a piece-wise continuous polynomial that satisfies the original equations of motion at discrete points. This formulation of the problem has shown to be very robust, allowing for application to very complex scenarios such as many-revolution trajectories, or those subjected to complex dynamics.

With direct transcription through collocation, instead of propagating a trajectory that provides outputs for an optimization problem, a collection of positions and velocities are part of the optimization state, which are used to compute pre-determined defect equations that only depend on a small subset of the complete optimization state. Some general optimization definitions and discussion are provided in Section 3.1, including how collocation-based methods fit into this. This approach is then presented in detail for Gauss-Lobatto collocation in Section 3.2. Section 3.3 details a critical extension of collocation methods, mesh refinement, which allows simultaneously for error control and variable step size numerical integrations. Section 3.4 elaborates on how Gauss-Lobatto collocation compares to other collocation schemes. Based on the theory presented in this chapter, Section 3.5 presents the implementation of these methods into a tool called *Maverick*. Sec-

tion 3.6 then summarizes noteworthy differences between earlier implementations of Gauss-Lobatto collocation methods and the one presented here, with the Chapter concluded in Section 3.7.

It should be noted that all the trajectory optimization studies presented in this work focus on solving what is sometimes called the 'inner loop', or the 'local' optimization problem. This means that one, or a small number of versions of a given problem is computed, and the expected quality of the solution is high. In contrast to that, the 'outer loop', or 'global' optimization problem, solves each problem many times over, accepting that some of them will be of low quality, and picking the best version in each case. This often involves the use of some stochastic processes to initialize these iterations. The 'outer loop' solution process is nonetheless benefited by the quality of the 'inner loop' solver, since it may lead to fewer iterations for a good solution, or it will converge to the desired solution with a poorer initial guess. A great example of a powerful combination of solving the 'inner' and 'outer' loop is EMTG [17]. Again, all work in this thesis focuses on using collocation methods as a better 'inner loop' solver, which could then be enhanced with an 'outer loop' solver in much the same way as EMTG has demonstrated.

3.1 Optimization through direct transcription

For the remainder of this chapter, it is useful to introduce some formulations and definitions related to optimization problems. Reference [4] provides an excellent summary of trajectory optimization methods, a topic that has produced an enormous body of literature over the past decades. For the sake of brevity, only specific elements are discussed here, as one could easily double the length of this chapter just by briefly describing all the alternatives. In a very general form, a constrained optimization problem can be described as:

$$\text{Min} \quad J\left(\bar{X}\right) \tag{3.1}$$

$$\text{subject to} \quad \bar{F}\left(\bar{X}\right) = \bar{0} \tag{3.2}$$

$$\text{and} \quad \bar{X}_l \leq \bar{X} \leq \bar{X}_u \tag{3.3}$$

Where J is the objective function, $\bar{F}$ is the constraint vector, $\bar{X}$ is the optimization state, and $\bar{X}_l, \bar{X}_u$ are the lower and upper bounds for the optimization state, respectively. It should be noted that despite having a constraint vector equal to zero, this formulation does not exclude inequality constraints, which can be included through:

$$F_i = F_{ineq,i} - \rho_i = 0 \tag{3.4}$$

Where $F_{ineq,i}$ is an inequality constraint, and ρ_i is a slack variable that is included in the optimization state $\bar{X}$, with bounds corresponding to the original inequality constraint. In this formulation, the derivatives of the objective function and constraint vector can be written as:

$$DJ = \frac{dJ}{d\bar{X}} \tag{3.5}$$

$$D\bar{F} = \frac{d\bar{F}}{d\bar{X}} \tag{3.6}$$

Where the derivatives of the constraint vector $D\bar{F}$ are a Jacobian matrix. The necessary conditions for a minimum can then be given as:

$$\frac{dJ}{d\bar{X}} = 0 \tag{3.7}$$

$$\frac{d^2J}{d\bar{X}^2} \geq 0 \tag{3.8}$$

Equation 3.7 identifies a stationary point, with the added condition of $\frac{d^2J}{d\bar{X}^2} > 0$ identifying a local minimum. More details about the contents of the state and constraint vectors, as well as the objective function, are given in the following sections. At this stage, the formulation is very general, and the optimization problem could be solved with any number of methods. One decision to be made is whether to use a direct or indirect method, a trade-off that is discussed at length in Reference [4]. Indirect methods rely on the calculus-of-variations to solve the optimization problem in terms of the adjoint variables, often also called Lagrangian multipliers [1]. Without going into the derivation of these methods, one problem with these methods is that these adjoint variables often are not physically intuitive, making it difficult to provide an accurate initial guess. This is a poor match with the tendency of indirect methods to be very sensitive, with a small convergence

basin. For the interested reader, this is thoroughly discussed in Reference [4]. Other problems this reference identifies is that indirect methods are not very flexible, due to requiring analytical expressions that can require extensive work to derive when constraints are added or the formulation is altered in some other way, and a poor handling of inequality constraints. Before being able to iterate over solutions, indirect methods require the user to guess the sequence of subarcs where the inequality constraint is active or not. One of the attractive elements of an indirect method is that they do give a fully continuous solution of the problem.

Indirect methods have historically been widely applied, and are still in frequent use today, but as computational capabilities have improved they have lost popularity with respect to direct methods. Direct methods avoid the use of adjoint variables, and instead solve the optimization state variables directly. This is done by discretizing the optimization problem in some way, and solving this as a Non-Linear Programming (NLP) problem. All that is required is a set of formulations that calculate the objective function, constraint vector, and their respective derivatives, the latter of which could be done with finite differencing if analytical expressions are cumbersome. These values can be provided to an existing optimization library, such as Sparse Nonlinear OPTimizer (SNOPT) [21], or the Interior Point OPTimizer IPOPT [67]. SNOPT uses Sequential Quadratic Programming (SQP) for the solution of NLP problems, which is fully detailed in Reference [21]. In short, minor iterations are used to establish the search direction from the current optimization state, and in major iterations a step is then taken in that direction. This process is conceptually similar to a minimum-norm Newton method, except that the major iterations scale these steps based on the proximity to the optimal solution, as judged by the values of $\frac{dJ}{dX}$. This is repeated until the constraints are satisfied and the convergence towards an optimal solution has decreased below a user-defined threshold. The workings of SNOPT or IPOPT are not detailed here, since these operations are considered to be out of the scope of this research. For a thorough description of SNOPT's or IPOPT's workings, the reader is referred to Reference [21] or Reference [67], respectively.

One approach to formulating the problem is a single shooting method, which includes an initial state vector and a control history in the optimization state, and some arbitrary set of constraints. A multiple shooting method is very similar to this, but includes additional state vectors in the optimization state along with the control history. It necessarily includes additional constraints to guarantee continuity. The advantage of a multiple shooter is that, with respect to a single shooter, it is less sensitive to small variations early in the trajectory. In some ways, direct transcription through collocation is an extension of this approach, although it relies on using an implicit integration scheme. Instead of taking a state and propagating it for a number of time steps, characteristic of a shooting method, collocation methods consider all grid points of the implicit integration scheme, either as optimization states or as problem constraints. To ensure the trajectory is continuous, defect constraints are added to the existing problem formulation at selected points of the implicit integration scheme. Due to the added optimization state elements and constraints, this results in a very large problem formulation, but also a very sparse one. Both of these properties are quantified in the following sections. This large problem size is not without benefit, since by giving the optimization process direct control over a large number of states along the trajectory, an increased convergence radius emerges. This can be compared to a multiple shooter being less sensitive to small variations early in the trajectory than a single shooter. In a similar manner, due to the direct control over the full state and control vector on many points along the trajectory, collocation methods significantly reduce sensitivity to any single phase of the trajectory. An interesting illustration of the strength of collocation methods is provided in Reference [41], where a collocation scheme is compared directly with a multiple shooting scheme to optimize low-thrust, low-energy transfers. In the results presented there, collocation finds the same solution in a significantly shorter amount of time than a multiple shooter. Along similar lines, it is not inconceivable that, for other problems, a multiple shooter would not converge at all in an acceptable time frame.

After this qualitative discussion on optimization through direct transcription, the remainder of this chapter goes into detail on how it can be applied to low-thrust trajectory optimization.

3.2 Gauss Lobatto collocation

An excellent description of Gauss-Lobatto collocation is given in Reference [47], which is based on the earlier work in Reference [27], a discussion that is largely followed in this section. The first step is to compose the trajectory of n nodes, that bound $n-1$ total segments. These segments are the path that connect the two neighboring nodes, and each segment can have a unique length in both distance and time. Every node can be identified by a time $\{t_1, t_2, ..., t_i, ..., t_{n-1}, t_n\}$, where t_1 is the initial time and t_n is the final time. The length in time Δt_i of segment i can be found through $\Delta t_i = t_{i+1} - t_i$. All times within a segment can then be normalized through $\{0, \tau_1, \tau_2, ..., \tau_{j-1}, \tau_j, 1\} = \frac{1}{\Delta t_i}(\{t_{i,0}, t_{i,1}, t_{i,2}, ..., t_{i,j-1}, t_{i,j}, t_{i+1,0}\} - t_{i,0})$, where $t_{i,j}$ is some time within segment i and $t_{i,0} = t_i$. At every time, a node is defined by a state vector $\bar{x}_i$ and a control vector $\bar{u}_i$, with m state elements and k control elements. An interpolating polynomial is constructed using the Gauss-Lobatto quadrature rules that ensures the segments connecting the nodes lie on a continuous trajectory, as defined by the governing dynamics. This is accomplished by making sure the interpolating polynomial satisfies constraints known as the defects $\bar{\Delta}_i$. The following two sections present these systems for a third-degree and seventh-degree method, of integration orders four and twelve, respectively.

3.2.1 4^{th}-order

The third degree Gauss-Lobatto scheme, with integration order four, is also known as the Hermite-Simpson method [27], yielding a quadratic polynomial interpolant. Figure 3.1 illustrates one of the $n-1$ segments. The node points are those at the endpoints of the segments, shown in the figure as $\bar{x}_i$, $\bar{u}_i$, and $\bar{x}_{i+1}$, $\bar{u}_{i+1}$. Force model evaluations are identified with $\bar{f}_i = \bar{f}(t_i, \bar{x}_i, \bar{u}_i)$, representing a force model evaluation at time t_i, subject to state $\bar{x}_i$ and controls $\bar{u}_i$. The state $\bar{x}_{i,c}$ and controls $\bar{u}_{i,c}$ at the center are labeled as a defect point. The state vector can be calculated as follows

$$\bar{x}_{i,c} = \frac{1}{2}(\bar{x}_i + \bar{x}_{i+1}) + \frac{\Delta t_i}{8}\{\bar{f}_i - \bar{f}_{i+1}\} \tag{3.9}$$

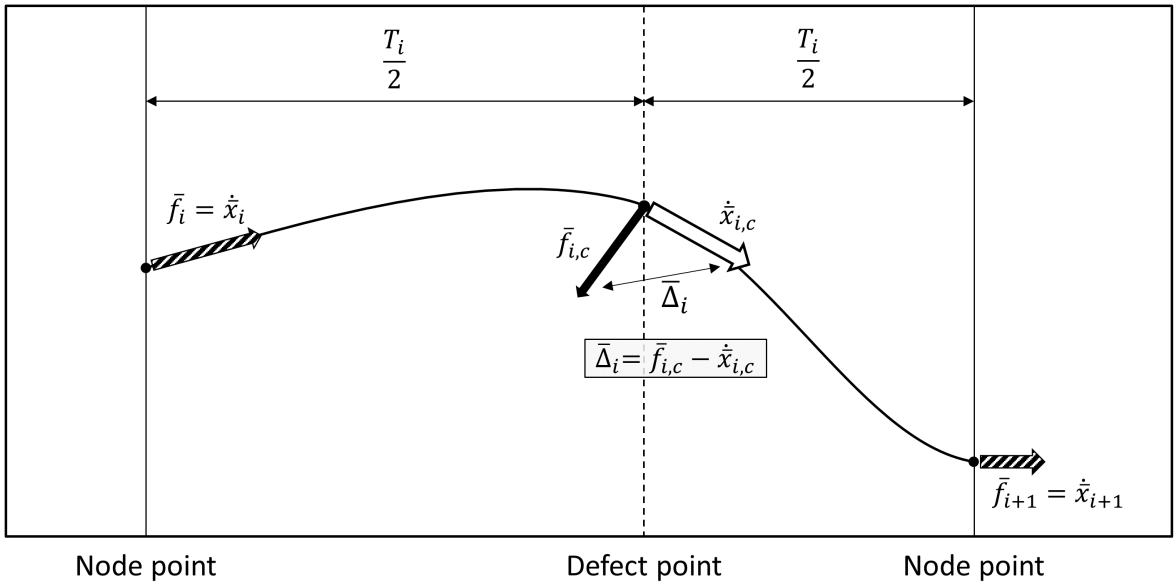


Figure 3.1: Illustration of a 4th order Gauss-Lobatto integration segment, adapted from Reference [48]

The control vector at the defect point can be constructed in a number of ways. One of the simplest approaches is a linear interpolation

$$\bar{u}_{i,c} = \frac{1}{2}(\bar{u}_i + \bar{u}_{i+1}) \tag{3.10}$$

While this may seem appealing, since it always results in a control history without any abrupt changes in direction, it should be kept in mind that optimal solutions always follow a bang-bang structure, meaning that for a truly optimal solution control is either on at full throttle, or off. However, if controls are linearly interpolated between two control vectors, a bang-bang structure is impossible by definition. Instead, the control vector can always be kept constant for each segment, which allows for control histories that follow bang-bang control more closely. On the downside, in the middle of thrust arcs (as opposed to their edges) the linearly interpolated thrust vectors can follow optimal thrust directions more closely, whereas constant thrust directions will only point in the average optimal direction for that segment. This is considered an acceptable trade-off to enable bang-bang solutions more freely in this work, and constant thrust vectors for each segment

are used. The defect constraint for this segment is then computed with:

$$\left|\bar{\Delta}_{i,c}\right| = \bar{x}_i - \bar{x}_{i+1} + \frac{\Delta t_i}{6}\{\bar{f}_i + 4\bar{f}_{i,c} + \bar{f}_{i+1}\} \leq \bar{\epsilon}_\Delta \tag{3.11}$$

Where $\bar{\epsilon}_\Delta$ is a numerical tolerance defined by the user, set to zero if infinite precision were available.

The 4$^{\text{th}}$ order Gauss-Lobatto scheme, hereafter abbreviated as GL-4, allows for a quick implementation due to its relatively few equations. More discussion on the implementation is given below, but as a fourth order integration scheme the GL-4 method is not ideal for high-fidelity problems. To that end, the equations for the 12$^{\text{th}}$ order scheme, GL-12, are presented next.

3.2.2 12$^{\text{th}}$-order

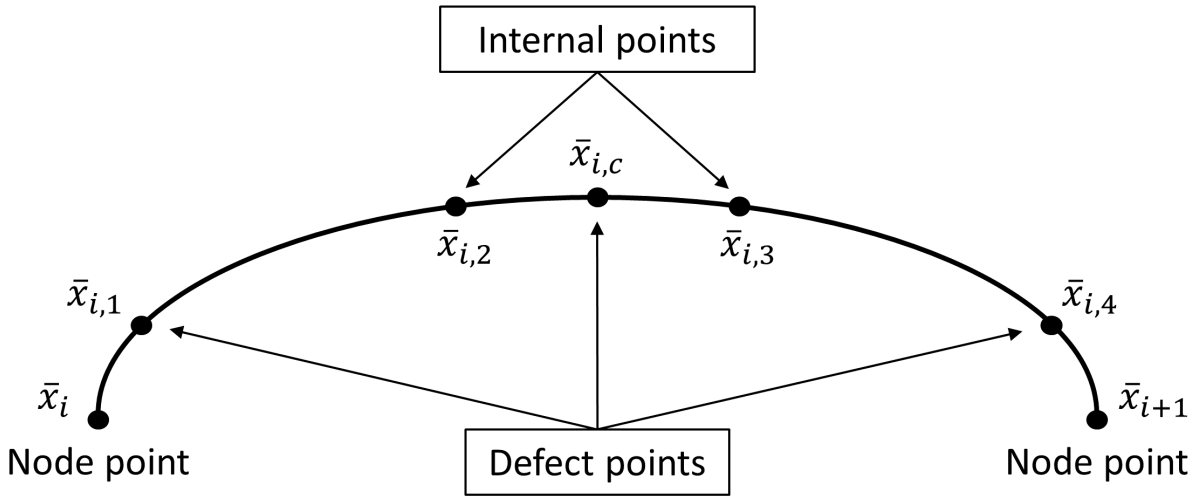


Figure 3.2: Illustration of a 12$^{\text{th}}$ order Gauss-Lobatto integration segment, adapted from Reference [47]

The approach for setting up the seventh degree Gauss-Lobatto scheme, with integration order twelve, is very similar to the one presented in the previous section. This scheme is thoroughly detailed in Reference [47], and again this approach and notation is followed here due to its similarity of the GL-4 equations and definitions. Using the normalized time τ as defined above, a polynomial representation of the segment can be given by

$$\bar{x}(\tau) = \bar{L} \times \{1 \quad \tau \quad \tau^2 \quad \tau^3 \quad \tau^4 \quad \tau^5 \quad \tau^6 \quad \tau^7\}^T \tag{3.12}$$

where $\bar{L}$ is a matrix of coefficients that is defined below. A trajectory segment for GL-12 is presented in Figure 3.2. For GL-12, there are four input states per segment (instead of two), namely the node points $\bar{x}_i$ and $\bar{x}_{i+1}$, as well as the internal points $\bar{x}_{i,2}$ and $\bar{x}_{i,3}$, with corresponding controls. In addition, there are three defect points, $\bar{x}_{i,1}$, $\bar{x}_{i,c}$, and $\bar{x}_{i,4}$, again with corresponding controls. These points are positioned in normalized time along the segment according to the values in Table 3.1.

Table 3.1: Node positioning on the interval [0,1] for the GL-12 scheme, reproduced from Reference [47].

τ_i	=	0.0	$\tau_{i,3}$	=	0.734424396735357
$\tau_{i,1}$	=	0.0848880518607166	$\tau_{i,4}$	=	0.915111948139283
$\tau_{i,2}$	=	0.265575603264643	τ_{i+1}	=	1.0
$\tau_{i,c}$	=	0.5			

Using Equation 3.12, the input states $\bar{x}_i$, $\bar{x}_{i,2}$, $\bar{x}_{i,3}$, and $\bar{x}_{i+1}$ can be derived to satisfy

$$\{\bar{x}_i \quad \bar{x}'_i \quad \bar{x}_{i,2} \quad \bar{x}'_{i,2} \quad \bar{x}_{i,3} \quad \bar{x}'_{i,3} \quad \bar{x}_{i,4} \quad \bar{x}'_{i,4}\} = \bar{L}\bar{B} \tag{3.13}$$

where $\bar{x}'_j$ represents a derivative with respect to normalized time τ, which means it can be computed with

$$\bar{x}'_j = \Delta t_j \bar{f}(t_j, \bar{x}_j, \bar{u}_j) \tag{3.14}$$

and

$$\bar{B} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & \tau_{i,2} & 1 & \tau_{i,3} & 1 & 1 & 1 \\ 0 & 0 & \tau_{i,2}^2 & 2\tau_{i,2} & \tau_{i,3}^2 & 2\tau_{i,3} & 1 & 2 \\ 0 & 0 & \tau_{i,2}^3 & 3\tau_{i,2}^2 & \tau_{i,3}^3 & 3\tau_{i,3}^2 & 1 & 3 \\ 0 & 0 & \tau_{i,2}^4 & 4\tau_{i,2}^3 & \tau_{i,3}^4 & 4\tau_{i,3}^3 & 1 & 4 \\ 0 & 0 & \tau_{i,2}^5 & 5\tau_{i,2}^4 & \tau_{i,3}^5 & 5\tau_{i,3}^4 & 1 & 5 \\ 0 & 0 & \tau_{i,2}^6 & 6\tau_{i,2}^5 & \tau_{i,3}^6 & 6\tau_{i,3}^5 & 1 & 6 \\ 0 & 0 & \tau_{i,2}^7 & 7\tau_{i,2}^6 & \tau_{i,3}^7 & 7\tau_{i,3}^6 & 1 & 7 \end{bmatrix} \tag{3.15}$$

It should be noted that while $\bar{B}$ is presented in References [25] and [46], a typo exists in those presentations that is corrected in Equation 3.15. Using the left-hand side of Equation 3.13, as well

as the fact that $\bar{B}$ is a matrix of known constants, the polynomial efficients $\bar{L}$ can now be computed. These coefficients can then be used to derive equations for the defect points of the GL-12 method as a function of the input states $\bar{x}_i$, $\bar{x}_{i,2}$, $\bar{x}_{i,3}$, and $\bar{x}_{i+1}$. These equations are similar to those presented for the GL-4 method, albeit with more involved expressions

$$\bar{x}_{i,1} = a_i^1\bar{x}_i + a_{i,2}^1\bar{x}_{i,2} + a_{i,3}^1\bar{x}_{i,3} + a_{i+1}^1\bar{x}_{i+1} + \Delta t_i(v_i^1\bar{f}_i + v_{i,2}^1\bar{f}_{i,2} + v_{i,3}^1\bar{f}_{i,3} + v_{i+1}^1\bar{f}_{i+1}) \tag{3.16}$$

$$\bar{x}_{i,c} = a_i^c\bar{x}_i + a_{i,2}^c\bar{x}_{i,2} + a_{i,3}^c\bar{x}_{i,3} + a_{i+1}^c\bar{x}_{i+1} + \Delta t_i(v_i^c\bar{f}_i + v_{i,2}^c\bar{f}_{i,2} + v_{i,3}^c\bar{f}_{i,3} + v_{i+1}^c\bar{f}_{i+1}) \tag{3.17}$$

$$\bar{x}_{i,4} = a_i^4\bar{x}_i + a_{i,2}^4\bar{x}_{i,2} + a_{i,3}^4\bar{x}_{i,3} + a_{i+1}^4\bar{x}_{i+1} + \Delta t_i(v_i^4\bar{f}_i + v_{i,2}^4\bar{f}_{i,2} + v_{i,3}^4\bar{f}_{i,3} + v_{i+1}^4\bar{f}_{i+1}) \tag{3.18}$$

With all the constant coefficients shown in Table 3.2. Using these defect points, the defect equations can then be calculated as follows

$$\begin{aligned}\left|\bar{\Delta}_{i,1}\right| =& b_i^1\bar{x}_i + b_{i,2}^1\bar{x}_{i,2} + b_{i,3}^1\bar{x}_{i,3} + b_{i+1}^1\bar{x}_{i+1} \\ &+ \Delta t_i(w_i^1\bar{f}_i + w_{i,1}^1\bar{f}_{i,1} + w_{i,2}^1\bar{f}_{i,2} + w_{i,3}^1\bar{f}_{i,3} + w_{i+1}^1\bar{f}_{i+1}) \leq \bar{\epsilon}_\Delta\end{aligned} \tag{3.19}$$

$$\begin{aligned}\left|\bar{\Delta}_{i,c}\right| =& b_i^c\bar{x}_i + b_{i,2}^c\bar{x}_{i,2} + b_{i,3}^c\bar{x}_{i,3} + b_{i+1}^c\bar{x}_{i+1} \\ &+ \Delta t_i(w_i^c\bar{f}_i + w_{i,2}^c\bar{f}_{i,2} + w_{i,c}^c\bar{f}_{i,c} + w_{i,3}^c\bar{f}_{i,3} + w_{i+1}^c\bar{f}_{i+1}) \leq \bar{\epsilon}_\Delta\end{aligned} \tag{3.20}$$

$$\begin{aligned}\left|\bar{\Delta}_{i,4}\right| =& b_i^4\bar{x}_i + b_{i,2}^4\bar{x}_{i,2} + b_{i,3}^4\bar{x}_{i,3} + b_{i+1}^4\bar{x}_{i+1} \\ &+ \Delta t_i(w_i^4\bar{f}_i + w_{i,2}^4\bar{f}_{i,2} + w_{i,3}^4\bar{f}_{i,3} + w_{i,4}^4\bar{f}_{i,4} + w_{i+1}^4\bar{f}_{i+1}) \leq \bar{\epsilon}_\Delta\end{aligned} \tag{3.21}$$

With all constant coefficients presented in Table 3.2.

An important thing to note, for both the GL-4 and GL-12 methods, is that the defect constraints depend only on states and controls of nodes that are either within the segment or at its boundary. Regardless of the number of segments in the entire problem, the set of defect constraints have a dependency on just a small set of input states, with input states at segment boundaries influencing both adjacent segments and providing a continuity to the full trajectory as a result. One outcome from this is that there is no reason to evaluate the defect constraints of the first segment before those of the last segment. All segments could theoretically be evaluated in a random order, or all at the same time. It is this characteristic of these formulations that makes these methods highly parallelizable, as discussed further in Chapter 4.

Table 3.2: Constant coefficients for the GL-12 scheme, as found in Reference [47].

a_i^1	=	+6.18612232711785d-1	b_i^1	=	+8.84260109348311d-1
$a_{i,2}^1$	=	+3.34253095933642d-1	$b_{i,2}^1$	=	-8.23622559094327d-1
$a_{i,3}^1$	=	+1.52679626438851d-2	$b_{i,3}^1$	=	-2.35465327970606d-2
a_{i+1}^1	=	+3.18667087106879d-2	b_{i+1}^1	=	-3.70910174569208d-2
v_i^1	=	+2.57387738427162d-2	w_i^1	=	+1.62213410652341d-2
			$w_{i,1}^1$	=	+1.38413023680783d-1
$v_{i,2}^1$	=	-5.50098654524528d-2	$w_{i,2}^1$	=	+9.71662045547156d-2
$v_{i,3}^1$	=	-1.53026046503702d-2	$w_{i,3}^1$	=	+1.85682012187242d-2
v_{i+1}^1	=	-2.38759243962924d-3	w_{i+1}^1	=	+2.74945307600086d-3
a_i^c	=	+1.41445282326366d-1	b_i^c	=	+7.86488731947674d-2
$a_{i,2}^c$	=	+3.58554717673634d-1	$b_{i,2}^c$	=	+8.00076026297266d-1
$a_{i,3}^c$	=	+3.58554717673634d-1	$b_{i,3}^c$	=	-8.00076026297266d-1
a_{i+1}^c	=	+1.41445282326366d-1	b_{i+1}^c	=	-7.86488731947674d-2
v_i^c	=	+9.92317607754556d-3	w_i^c	=	+4.83872966828888d-3
$v_{i,2}^c$	=	+9.62835932121973d-2	$w_{i,2}^c$	=	+1.00138284831491d-1
			$w_{i,c}^c$	=	+2.43809523809524d-1
$v_{i,3}^c$	=	-9.62835932121973d-2	$w_{i,3}^c$	=	+1.00138284831491d-1
v_{i+1}^c	=	-9.92317607754556d-3	w_{i+1}^c	=	+4.83872966828888d-3
a_i^4	=	+3.18667087106879d-2	b_i^4	=	+3.70910174569208d-2
$a_{i,2}^4$	=	+1.52679626438851d-2	$b_{i,2}^4$	=	+2.35465327970606d-2
$a_{i,3}^4$	=	+3.34253095933642d-1	$b_{i,3}^4$	=	+8.23622559094327d-1
a_{i+1}^4	=	+6.18612232711785d-1	b_{i+1}^4	=	-8.84260109348311d-1
v_i^4	=	+2.38759243962924d-3	w_i^4	=	+2.74945307600086d-3
$v_{i,2}^4$	=	+1.53026046503702d-2	$w_{i,2}^4$	=	+1.85682012187242d-2
$v_{i,3}^4$	=	+5.50098654524528d-2	$w_{i,3}^4$	=	+9.71662045547156d-2
			$w_{i,4}^4$	=	+1.38413023680783d-1
v_{i+1}^4	=	-2.57387738427162d-2	w_{i+1}^4	=	+1.62213410652341d-2

3.3 Mesh refinement

While the schemes presented in the preceding sections may successfully converge with all user tolerances on the defect constraints satisfied, this does not guarantee the trajectory is physically accurate. It may be that the time step for one or more segments is too large for these defect constraints to provide a sufficiently accurate representation of the governing dynamics. It may also be that the time step is too small, which can lead to errors in extreme cases, but more generally means an inefficient use of memory and computational time, since apparently fewer segments could have been used. Both of these problems can be solved by providing an estimate of the error on

each segment, using this estimate to change each segment length, and possibly adding or removing segments in the process. This is typically called mesh refinement, where the mesh is the spacing in time of the segments. Below, an approach is presented for mesh refinement for the GL-12 method, as found in Reference [47]. A similar approach for the GL-4 method is presented in References [6] and [7], but is not repeated here.

Considering that GL-12 is 7^{th} degree, the error on the i^{th} segment e_i can be represented as follows:

$$e_i = C\Delta t_i^8 ||\bar{x}^{(8)}||_i + O(\Delta t_i^9) \tag{3.22}$$

Where $||\bar{x}^{(8)}||$ is the eighth time derivative of x, and $O(\Delta t_i^9)$ is the residual error, proportional to a 9^{th} degree polynomial term. Reference [25] uses Reference [53] to determine the constant C to be:

$$C = 2.93579395141895\text{d-}9 \tag{3.23}$$

However, in a 7^{th} degree polynomial, the eighth time derivative is zero, providing no information on what the actual error might be. Reference [15] provides an approximation for this derivative, with the following equations

$$||\bar{x}^{(8)}|| \approx \theta(t) \triangleq \begin{cases} max\left[2\frac{|\bar{y}_1 - \bar{y}_2|}{\Delta t_1 + \Delta t_2}\right] & \text{on } (t_1, t_2) \\ max\left[\frac{|\bar{y}_{i-1} - \bar{y}_i|}{\Delta t_{i-1} + \Delta t_i} + \frac{|\bar{y}_{i+1} - \bar{y}_i|}{\Delta t_{i+1} + \Delta t_i}\right] & \text{on } (t_i, t_{i+1}), i = 2, ..., n-2 \\ max\left[2\frac{|\bar{y}_{n-2} - \bar{y}_{n-1}|}{\Delta t_{n-2} + \Delta t_{n-1}}\right] & \text{on } (t_{n-1}, t_n) \end{cases} \tag{3.24}$$

where

$$\bar{y}_i = \frac{\bar{x}^{(7)}(\tau)}{\Delta t_i^7} \qquad \text{on } (t_i, t_{i+1}) \tag{3.25}$$

and where $\bar{x}^{(7)}(\tau)$ is the seventh time derivative of $\bar{x}$. Combining Equations 3.12 and 3.13, this can be written as

$$\begin{aligned} \bar{x}^{(7)}(\tau) &= \left(\bar{L} \times \{1 \quad \tau \quad \tau^2 \quad \tau^3 \quad \tau^4 \quad \tau^5 \quad \tau^6 \quad \tau^7\}^T\right)^{(7)} \\ &= \{\bar{x}_i \quad \bar{x}'_i \quad \bar{x}_{i,2} \quad \bar{x}'_{i,2} \quad \bar{x}_{i,3} \quad \bar{x}'_{i,3} \quad \bar{x}_{i,4} \quad \bar{x}'_{i,4}\} \times \bar{B}^{-1}\{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 7!\}^T \\ &= 7!\ \{\bar{x}_i \quad \bar{x}'_i \quad \bar{x}_{i,2} \quad \bar{x}'_{i,2} \quad \bar{x}_{i,3} \quad \bar{x}'_{i,3} \quad \bar{x}_{i,4} \quad \bar{x}'_{i,4}\} \times \bar{b} \end{aligned} \tag{3.26}$$

where $\bar{b}$ is the last column of $\bar{B}^{-1}$. Conceptually, Equation 3.24 essentially finite-differences the 7^{th} derivative of the polynomials of adjacent segments to approximate the 8^{th} derivative, which is then used to compute the error through Equation 3.22. Although the 7^{th} derivative is constant for each individual 7^{th}-degree polynomial, which is also clearly visible in Equation 3.26, this constant value does change for each segment, since every segment represents a completely independent polynomial. These variations are what Equation 3.24 uses to approximate the 8^{th} derivative.

3.3.1 Equidistribution

While the approximation for the error presented above could be used to find places to add or remove nodes in a solution, essentially the same is achieved by taking the existing number of nodes and re-distributing them to equally distribute the error across the mesh. This puts the current number of nodes to the best possible use. The new points are selected so that the following equation holds for all i:

$$I(t_{i+1}) = \frac{iI(t_n)}{n-1} \quad , \quad i = 1, ..., n-2 \tag{3.27}$$

$$I(t) = \int_{t_1}^{t} \theta(s)^{1/8} ds \tag{3.28}$$

Since $\theta(t)$ has a constant value for each segment, the integral $I(t_{i+1})$ amounts to summing up these constants for each segment that falls fully in the $[t_1, t_{i+1}]$ domain, except for the final segment, where the appropriate fraction of that segment must be taken, at the end of which the new node is placed.

Using the old mesh and corresponding solution, the polynomial expression can be used to interpolate the states at the new times. The problem can now be solved again with the new mesh, using the very good initial guess provided by the polynomial interpolation of the old solution, which will have a more equidistributed error. However, since the solution likely changed as a result of the new mesh, the equidistribution may not yet be perfect. As a result, several equidistribution iterations are typically required to create a mesh that successfully equidistributes the error.

3.3.2 Error Reduction

When the mesh has sufficiently equidistributed the error, the number of nodes n can be updated with the following equation[1] :

$$n_{new} = round\left(n_{old}\left(\frac{10e_{mean}}{\epsilon_{tol}}\right)^{1/8} + 5\right) \tag{3.29}$$

Where e_{mean} is the mean error of the equidistributed mesh, and ϵ_{tol} is a user-defined tolerance for the error. Since only an approximation of the error exists, this approximation is multiplied by 10 to add ample margin for any inaccuracies in the error approximation. Furthermore, a minimum of 5 nodes is added per error reduction iteration, to avoid the mesh growing too slowly. With the new number of nodes, a new mesh can be constructed with Equation 3.27. The states can again be interpreted from the polynomial expressions, and a new solution can be found. If necessary, the mesh of the new solution can be further equidistributed, but typically the error is equidistributed and below the desired tolerance after a single iteration.

3.3.3 Mesh refinement process

The mesh refinement process as described in this section is an iterative process. First, the problem is formulated, an initial guess and mesh are created, and these are then solved by an NLP solver like IPOPT. This solution is then evaluated for local errors, and the mesh is re-distributed to have equal local errors along the whole solution. While these iterations move all the nodes around, they do not add or remove nodes, and as such do not change the problem size. This new mesh is then handed back to the NLP solver, which solves the problem again. This process continues until the error is sufficiently equidistributed. Typically, the mean error still violates the desired tolerance, which results in an error reduction iteration, where additional nodes are added, creating a larger version of the same trajectory problem that is solved again by the NLP solver. At the conclusion of this, a fully converged solution is found that meets all accuracy standards. A few remaining points are clarified below.

[1] This equation was suggested through personal communication by Daniel J. Grebow

Node reduction Technically, it is conceivable that at (or towards) the end of the equidistribution iterations, the maximum error is well below the desired tolerance. In theory, a node reduction step could be taken, that is essentially the opposite of the steps described in Section 3.3.2. However, all this would achieve is a longer runtime, since the converged optimal solution that satisfies the desired tolerance already exists. It is unlikely that the user would want the program to keep running, considering that fact, but in theory such a step could easily be implemented.

Additional verification In the code used for this research, an additional verification is done by taking the very first input state provided by the optimal solution, and the entire control history, and propagating the trajectory with TurboProp's Dormand-Prince 8^{th} order numerical integrator [33]. Both local errors and the global error are provided to the user. The result of this is not used at any stage by the mesh refinement algorithm, but it provides another metric to allow the user to be confident in the accuracy of the solution. If desired a mesh refinement algorithm could be built around this process, but since the GL-12 method has numerical integration order 12, it would be preferable to use an integrator with a numerical order greater than itself, rather than lower. As discussed in Chapter 7, the results from this propagation are not always very valuable due to the unstable nature of low-energy transfers, although a similar approach as described here could be used with a multiple shooter propagation to improve this.

Eliminating equidistribution iterations In theory, the equidistribution process could be moved into the optimization performed by the NLP solver. This is not possible for the error reduction steps, which add nodes and thereby reformulate the problem, but since the equidistribution iterations do not affect problem size there is no obstacle to doing this. However, as Reference [6] points out, this could increase the problem sensitivity, by not only having to satisfy the defect equations of a given grid, but simultaneously improving the accuracy of that grid. As a result, it is expected that any gains received from a less iterative approach are lost by a longer runtime spent in the NLP solver. This research does not investigate a trade-off between these formulations.

3.4 Alternative schemes

So far, specific attention has only been given to Gauss-Lobatto Implicit Runge-Kutta (IRK) schemes. However, a wide variety of IRK schemes exist, and all of them could be applied in a similar manner to that described in Section 3.2. In Reference [30], the author compares Gauss-Legendre IRK to Band-Limited Collocation IRK (BLC-IRK) for the orbit propagation problem. As discussed there, Gauss-Legendre can be shown to be superconvergent, meaning it has the fewest function evaluations for a given numerical integration order. BLC-IRK, however, does not have a numerical integration order, due to the definition of such an order breaking down for BLC-IRK. This means that superconvergence does not guarantee Gauss-Legendre IRK to perform better. Reference [30] compares these methods for the initial value problem, for a series of representative test cases for orbit propagation. It concludes that while there can be up to a 40% performance increase for BLC-IRK when 200 node points are used per integration step, this difference is only marginally present for as few nodes as 32 per integration step. Fewer nodes per integration step are not compared, but the performance difference is expected to effectively disappear there.

Some important conclusions can be inferred from the study in Reference [30]. For the approaches discussed in Section 3.2, only 3 or 7 nodes per integration step are used for GL-4 and GL-12, respectively. While Gauss-Legendre is superconvergent, and is therefore able to take slightly larger time steps than an equal degree Gauss-Lobatto method, this difference is very small for the polynomial degrees under consideration. It likely would not be measurable under all but the most ideal circumstances, and even then it would be small. This would be a different matter if a much higher polynomial degree would be used, but there is no reason to believe this would be attractive. The reason a large number of nodes is potentially appealing for the initial value problem is that it allows the parallel evaluation of the force model for a single integration step, and this is why the study in Reference [30] was performed. For the formulation discussed in the present thesis, no such effort is needed to make the problem massively parallel. The approach presented in this thesis is already massively parallel, by virtue of allowing the evaluation of all integration steps at

one time. This removes the necessity to have many nodes on a single integration step for a parallel formulation. Furthermore, Reference [30] observes that total performance suffers when the number of nodes is increased, for both Gauss-Legendre IRK and BLC-IRK. For the initial value problem this performance reduction may be compensated by the improved performance of a parallel formulation, but since the optimization problem already has a parallel formulation this is not the case, and it appears likely that total performance would suffer by going to a very high polynomial degree.

As a quick sidestep, it should be noted that if the control vector and objective function are removed from the optimization methods discussed in Section 3.2, these formulations essentially reduce to an initial value problem. This means that this formulation could be an alternative way of giving the initial value problem a massively parallel formulation, without having to resort to introducing a large number of node points and the associated performance reductions. It is unclear if this formulation would be competitive with modern numerical integration techniques, but it could be an interesting avenue to pursue based on the results obtained from the optimization problem.

Having established that the expected performance increase of either Gauss-Legendre or BLC-IRK with respect to Gauss-Lobatto is very limited for the polynomial degree under consideration, this still does not explain why Gauss-Lobatto is used. The justification for this is that Gauss-Lobatto has been extensively applied for these types of low-thrust trajectory optimization problems, and is very well documented as a result. Considering that a much larger performance gain is expected from a proper parallel implementation than from a switch to Gauss-Legendre or BLC-IRK, it is decided to focus effort on the most promising performance gain, rather than spending considerable effort establishing the defect constraints, mesh refinement equations, and other aspects of these other methods. Once a parallel implementation of the Gauss-Lobatto scheme exists, it would significantly reduce the time required to create a similar implementation for a Gauss-Legendre or BLC-IRK scheme, since there is great similarity among them all. Potentially, this could be an extension of the work discussed here, as discussed in Section 8.2.

3.5 Implementation

The previous section described the GL-4 and GL-12 methods in their general form, without specifying how these methods might be implemented, which is discussed here. The optimization tool that is developed for this research, based on the methods described above, is referred to as *Maverick*. Both the GL-4 and GL-12 methods are implemented in very similar ways. All CPU code is written in the C programming language, with all parallel CPU performance achieved through OpenMP. All C code is compiled with the GCC compiler, version 4.8.2. The GPU code is written in the CUDA-C programming language [57, 44] , and compiled using the NVCC compiler driver, version 6.5.12. All results are produced on the same machine, which has an Intel i7-4810 MQ CPU, and an NVIDIA GeForce GTX 870M GPU. The nonlinear programming (NLP) solver that is used is IPOPT [67]. The SNOPT [21] NLP solver was also used for comparison, but provided inferior performance almost universally, with only a handful of scenarios leading to comparable or slightly improved performance. Results using SNOPT are therefore left out in this thesis.

For the formulation of the problem, the state consists of $m = 7$ elements, namely three position, three velocity, and mass. Position and velocity are given in Cartesian coordinates in the J2000 reference frame, which is also the convention for the position and velocity defect constraints. The control vector contains $k = 4$ elements, namely the three directions of the control vector $\bar{P}_0$, expressed in engine power, as well as the specific impulse I_{sp}. Both for GL-4 and GL-12 the control vector is only defined at the beginning of the segment, and kept constant in both magnitude and direction throughout the segment. The directional elements of the control vector are represented as power, which is converted to thrust as follows

$$\bar{T} = \frac{2 \cdot \eta_j \cdot \bar{P}_0}{I_{sp} \cdot g_0} \tag{3.30}$$

Where η_j is the jet efficiency and $\bar{P}_0$ are the first three entries of the control vector, representing available power. The direction of the control vector can optionally be given in a Velocity, Normal, Co-Normal (VNC) reference frame, since this often has a more constant direction for the control

as it orbits the central body, enabling a simpler expression of the optimal solution that is sought. This reference frame is defined using the following transformation

$$\bar{Q}_{VNC} = [i_V,\ i_N,\ i_C] = \left[\frac{\bar{V}}{||\bar{V}||},\ \frac{(\bar{V}\times\bar{R})\times\bar{R}}{||\bar{V}\times\bar{R}||||\bar{R}||},\ \frac{\bar{V}\times\bar{R}}{||\bar{V}\times\bar{R}||}\right] \tag{3.31}$$

Where $\bar{R}$ and $\bar{V}$ are the current position and velocity of the spacecraft, respectively. Effectively, this rotates the thrust vector with respect to the central body along each segment. Besides the defect constraints discussed above, there are also power constraints on all node points to ensure that the presented control history is feasible, as follows

$$0 \leq ||\bar{P}_0|| \leq P_{max} \tag{3.32}$$

The power constraint derivatives are computed analytically, whereas all defect constraint derivatives are acquired through sparse finite differencing. Due to the super-sparse nature of the GL-4 and GL12 methods, providing the derivatives in this manner is very computationally efficient, and it allows for very quickly introducing new equations of motion without having to derive additional analytical derivatives.

The above details of the implementation, along with the general theory discussed in Section 3.1, provide a general understanding of the methods used for this research. The combined body of code that forms the entire optimization tool has been named *Maverick*. The following sections give more specific details on *Maverick*. Additionally, Appendix A discusses very specific details related to the actual structure and use of *Maverick*. The discussion in Appendix A is not critical to the understanding of the methods or the results presented in this dissertation, but is instead mainly directed at those trying to actually use the code.

3.5.1 Sparsity

Using the above definitions, some simple equations demonstrate the size of the optimization state and the constraint vector, and as a result also the Jacobian. As before, this relates to a trajectory with n node points, $n-1$ segments, m elements in each state vector, and k elements in

each control vector. For all work in this thesis, $m = 7$ and $k = 4$. For GL-4, these equations are the following

$$\text{len}(\bar{X}_{GL-4}) = (n) \cdot (m + k) = 11n \tag{3.33}$$

$$\text{len}(\bar{F}_{GL-4}) = (n-1) \cdot m + (n) = 8n - 7 \tag{3.34}$$

$$\text{size}(D\bar{F}_{GL-4})_{dense} = len(\bar{X}_{GL-4}) \cdot len(\bar{F}_{GL-4}) = 88n^2 - 77n \tag{3.35}$$

$$\text{size}(D\bar{F}_{GL-4})_{sparse} = 2(n-1) \cdot m \cdot (m + k) + \cdot k = 158n - 154 \tag{3.36}$$

For GL-12, the equations are similar

$$\text{len}(\bar{X}_{GL-12}) = n \cdot (m + k) + (n-1) \cdot (2m) = 25n - 14 \tag{3.37}$$

$$\text{len}(\bar{F}_{GL-12}) = 3(n-1) \cdot m + n = 22n - 21 \tag{3.38}$$

$$\text{size}(D\bar{F}_{GL-12})_{dense} = len(\bar{X}_{GL-12}) \cdot len(\bar{F}_{GL-12}) = 550n^2 - 833n + 294 \tag{3.39}$$

$$\text{size}(D\bar{F}_{GL-12})_{sparse} = 3(n-1) \cdot m \cdot (4m + 2k) + (n) \cdot k = 760n - 756 \tag{3.40}$$

For both these cases, the sparse size of the Jacobian is calculated using the knowledge that only a small subset of the total optimization state affects any defect (or power) constraint. It is important to note that while Equations 3.35 and 3.39 have a quadratic dependence on the number of segments n, Equations 3.36 and 3.40 are linear in the number of segments. As a result, the Jacobian for both formulations becomes sparser as the number of segments increases. This is illustrated for the GL-4 method in Table 3.3, and for the GL-12 method in Table 3.4. These tables clearly show how for every order of magnitude increase in segments, the sparsity is increased by an order of magnitude as well. Anything above 100 segments has a Jacobian that is very sparse, which is highly relevant because the problems of interest frequently require on the order of hundreds of segments [5]. For

Table 3.3: Sparsity as a function of segments for the 4^{th} order Gauss-Lobattto scheme.

Number of segments	10	100	1 000	10 000	100 000
Full Jacobian	9801	8.9e5	8.8e7	8.8e9	8.8e11
Non-zero elements	1584	1.6e4	1.6e5	1.6e6	1.6e7
Density (%)	16	1.8	0.18	0.018	0.0018

Table 3.4: Sparsity as a function of segments for the 12^{th} order Gauss-Lobattto scheme.

Number of segments	10	100	1 000	10 000	100 000
Full Jacobian	57681	5.5e6	5.5e8	5.5e10	5.5e12
Non-zero elements	7604	7.6e4	7.6e5	7.6e6	7.6e7
Density (%)	13	1.4	0.14	0.014	0.0014

large cases, using the full Jacobian would create problems with computer memory. For example, if every value in the Jacobian has double precision they take up 8 bytes each. As a result, for 1000 segments in the GL-4 method the full Jacobian would need approximately 700 megabytes of memory. Only 1.26 megabyte of this memory would contain non-zero entries, with almost the entire memory populated with values that are known to be zero. For 5000 segments the contrast is 17 gigabytes for the full Jacobian, compared to just 6.3 megabytes for the sparse Jacobian. Clearly, it is of great importance that sparsity is properly utilized, or the size of problems that can be considered is severely and unnecessarily constrained. This importance is magnified when GPUs are used for computation, where memory transfer between the CPU and GPU can be a dominant factor in total runtime. The implementation of these methods for this research takes the sparsity of the Jacobian into account, which is also fully supported by IPOPT.

3.5.2 Scaling

Finally, all aspects of the problem are scaled to the [-1,1] domain, which is important for the NLP solver to properly interpret the problem formulation, as well as to ensure the proper use of the numerical tolerances for the integration of the trajectory. This requires a unique scaling for position and velocity elements, with more unique values for other types of elements, as well as a scaling that is adapted based on the problem under consideration. For example, a GEO problem requires quite different scaling than a heliocentric problem.

As an example, consider Figure 3.3, which shows the cost function $J = X^2 + (3Y)^2$, where $X = [-1, 1]$ and $Y = [-0.33, 0.33]$. Clearly, anyone can see that even though the total magnitude of Y is smaller than X, the role of Y in the cost function is identical as the role of X. Unfortunately,

if this problem is fed into an NLP solver, or indeed any numerical routine, it has no way of knowing that Y should be weighted in this manner. When it computes what direction to move the cost function, it may try to move in a direction perpendicular to the contour of the current cost (imagine a ball rolling into a bowl), which would result in a derivative pointing in the direction of the arrow shown in Figure 3.3. This clearly does not point straight at the minimum cost at $[0, 0]$, and several iterations are needed to arrive there.

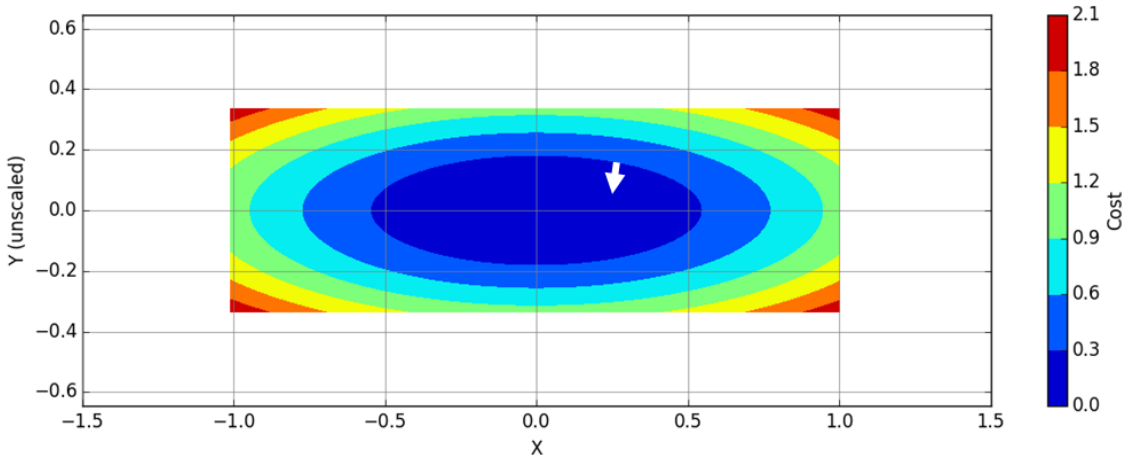


Figure 3.3: Representation of a toy problem cost function with two unscaled input parameters. The arrow indicates the direction of the derivative at the innermost contour line of the cost function.

Instead, the user could choose to scale Y by a factor of 3, which would result in $Y_{scaled} = [-1, 1]$ and $J_{scaled} = X^2 + (Y_{scaled})^2$. This would reformulate the problem to look like that presented in Figure 3.4, which clearly shows the derivative pointing straight at the true minimum. It is easy to see that this formulation has a more straightforward path to the optimal solution. Essentially, when scaling trajectory optimization problems, the above behavior is mimicked, but to a much more complicated degree. Looking at Figures 3.3 and 3.4, this transformation may look trivial to the point it should be automated by the NLP solver. To some extent this is possible, but keep in mind that the NLP solver cannot distinguish between variables that either have (1) radically different scalings, or (2) radically different constraints. For example, a common scaling problem in astrodynamics is that of position and velocity. Position is typically expressed in thousands of

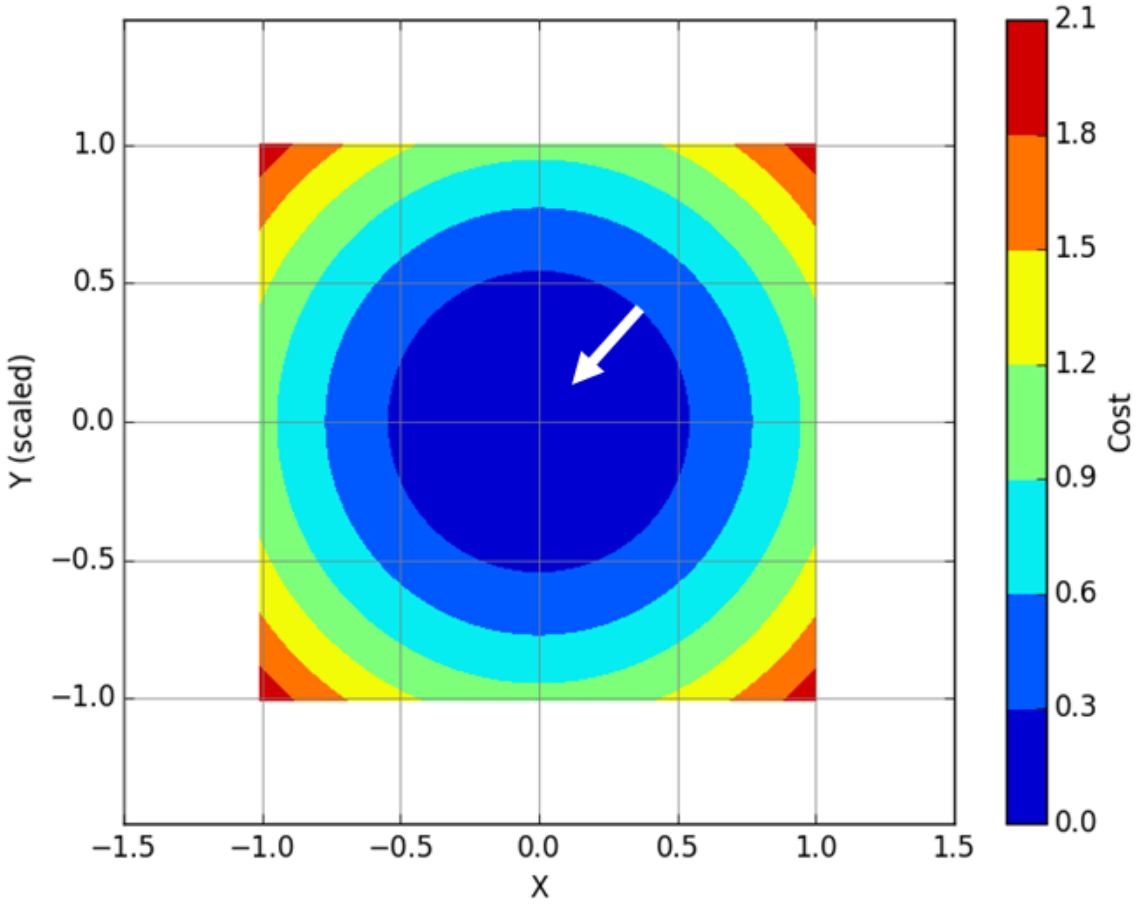


Figure 3.4: Representation of a toy problem cost function with two scaled input parameters. The arrow indicates the direction of the derivative at the innermost contour line of the cost function.

kilometers, whereas velocity is expressed in at most tens of kilometers per second. It is likely that the user will want to know both values to the same precision, for example six digits, and for ease of use scales everything to magnitudes of 1. Position would then be scaled by a factor of $\frac{1}{1000}$, and velocity by a factor of $\frac{1}{10}$ to achieve this. Now let's say a specific constraint exists to hit some position within 1 kilometer. If an automated practice were used, it likely would consider that here, too, it requires six digits of accuracy, just like the rest of the problem, and with position and velocity already scaled to 1, this constraint is left as is. Of course, this makes no sense. If position elements ranging in the thousands of kilometers are known to six digits, that equates millimeter

accuracy. The position constraint of 1 km, known to six digits, is instead known to micrometers. There is no meaning to this level of accuracy, when every other aspect of position is only known to millimeter accuracy. Instead, the constraint should have been scaled by a factor of $\frac{1}{1000}$, to ensure the accuracy matches. While the NLP solver could probably solve the problem in both cases, it would likely require significantly more iterations to achieve the greater, meaningless accuracy of a micrometer. In any case, there would be no way for the NLP solver to understand that this 1 km tolerance is of a very different nature, and does not need to be subjected to the same accuracy as other variables. This can not be achieved with automation, and can only be achieved with an actual understanding of the different elements of the problem.

This is only a rough example, and there are other ways that this could be avoided. But first and foremost, the user understands the problem they are formulating, and needs to convey this understanding to the NLP solver not just with all relevant equations, but also a proper weighting of each individual element. Any lack of effort in this part leads to inferior convergence, and often even a failed convergence altogether. This is not specific to collocation methods, or even trajectory optimization methods, it is inherent to numerical operations performed on any complex system.

3.5.3 Force models

A variety of force models are used for this research. Chapter 4 chiefly uses either a two-body force model or an asymmetric gravity field. For the asymmetric gravity field, the model described in Reference [23] is used, along with the GGM02C gravity model [66]. For the research described in the remaining chapters, the code is set up to link to the TurboProp [33] force models easily. This research uses the DE and DE_SRP force models from TurboProp, but any other TurboProp force model can also be plugged in with very little effort.

3.5.4 Hyperbolic thrust constraints

Reference [39] discusses, among other modeling effects, a reformulation of thrust or ΔV constraints. Normally thrust would be computed and constrained through

$$T = \sqrt{T_x^2 + T_y^2 + T_z^2} \tag{3.41}$$

where T is the total thrust magnitude with components $T_{x,y,z}$. Instead, Reference [39] suggests computing the following value

$$T_H = \sqrt{T_x^2 + T_y^2 + T_z^2 + \epsilon_H} \tag{3.42}$$

where ϵ_H is identified as a "fudge factor". Reference [38] refers to the same practice, where it's also called a "mass leak". To be perfectly accurate, the approach above is applied to ΔV (instead of Thrust) in both Reference [39] and [38], but this difference is trivial. In both references, the value computed in Equation 3.42 is used not only to provide the NLP solver with the necessary constraints, but also for the propagation of the trajectory, which results in an apparent mass leak, since the thrust is not actually zero, even when all components are. However, this is not actually necessary, especially with collocation methods, which do not have a continuous propagation but instead rely on many piece-wise continuous polynomials. Equation 3.42 can be used by the NLP solver to appropriately constrain the thrust, while Equation 3.41 is used to determine properties such as the mass flow. This completely avoids the presence of a mass leak, while maintaining the favorable convergence behavior provided by the formulation in Equation 3.42. In the remaining text, Equation 3.42 is referred to as a hyperbolic thrust constraint, since the formulation is identical to a hyperbola. Considering that, $\sqrt{\epsilon_H}$ is in fact the semi-major axis of the resulting hyperbola. To avoid confusion with the typical semi-major axis described in astrodynamics (that of the orbit), and to remain consistent with the notation presented in Reference [39], ϵ_H is called the "fudge factor".

The appeal of introducing a fudge factor, resulting in a hyperbolic thrust constraint, is illustrated in Figure 3.5. In this figure, it is assumed that the y- and z-components of the thrust are 0, and that total thrust can range from 0 to 1. For the regular constraint, it can easily be seen

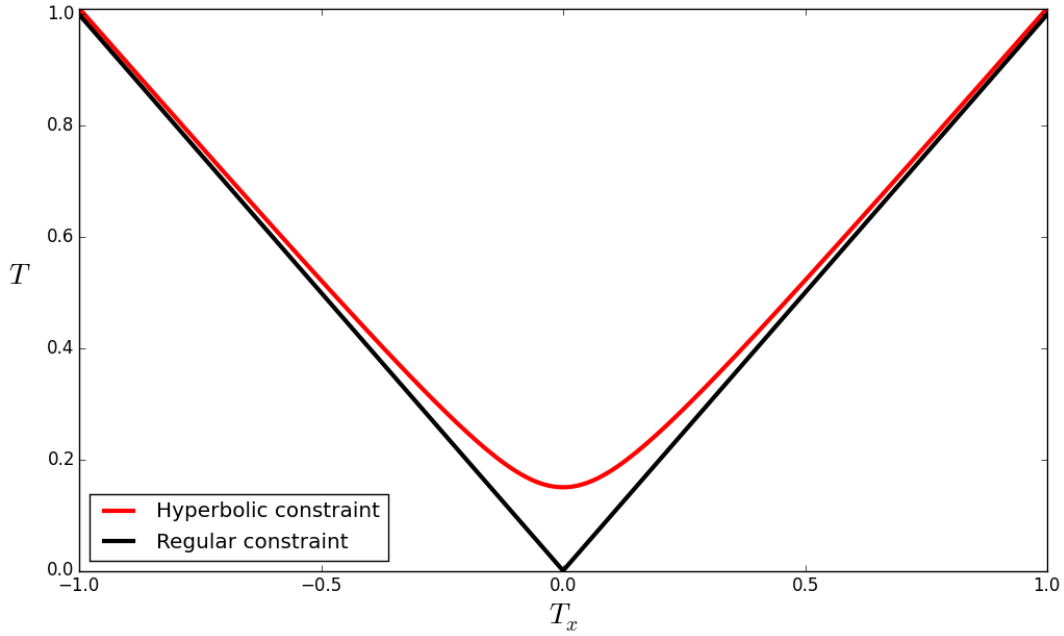


Figure 3.5: Comparison of a hyperbolic thrust constraint with $\epsilon_H = 0.225$ and a traditional thrust constraint ($T_y = T_z = 0$)

that the second derivative of the thrust with respect to the x-component of the thrust is zero for the majority of the domain, except for $T = T_x = 0$. At this point, the second derivative is infinite, and the first derivative instantaneously changes sign. This can also be seen clearly in Figure 3.6, which shows the first derivative over the applicable domain. In contrast, for the hyperbolic thrust constraints, such a discontinuity never occurs. Both first and second derivatives are always properly defined. Considering that NLP solvers like SNOPT and IPOPT benefit greatly from well-behaved, smooth derivatives, it should be no surprise that using these hyperbolic thrust constraints can have a great impact on convergence. Especially for the constraint functions, where the only importance is that the value is between a lower and upper bound, and no priority is given to how far from either bound the thrust is, the value for ϵ_H can be chosen quite freely, so long as the constraint bounds are adjusted accordingly. For the objective function, some additional considerations apply, which are discussed in Section 3.5.5. Reference [39] studies the impact of different values for ϵ_H on the convergence of Sims-Flanagan low-thrust trajectory optimization methods. Significant performance

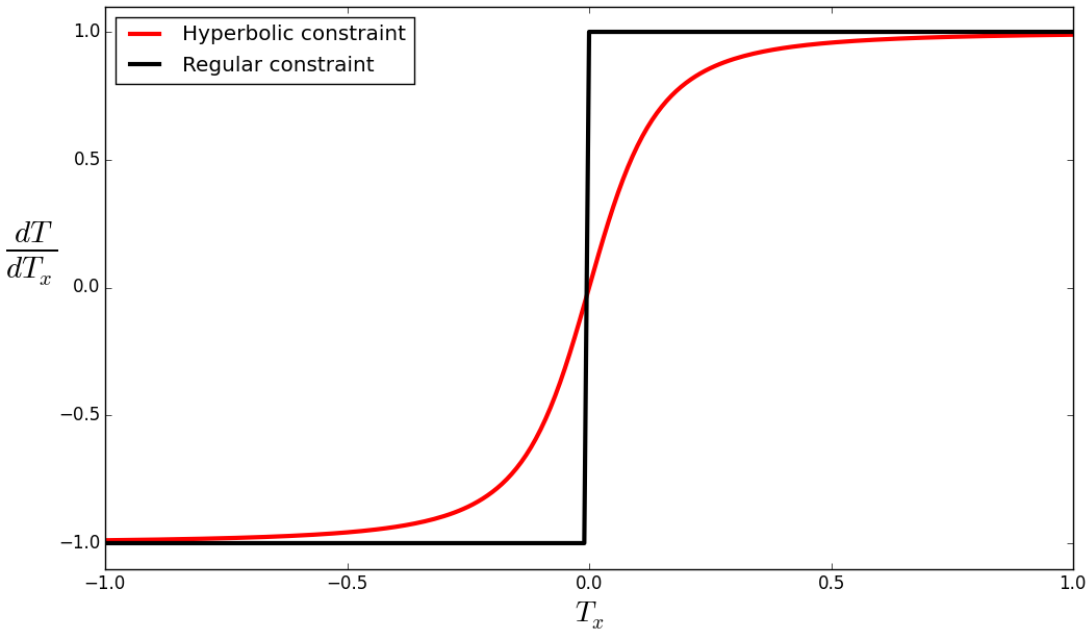


Figure 3.6: Comparison of a hyperbolic thrust derivative with $\epsilon_H = 0.225$ and a traditional thrust derivative ($T_y = T_z = 0$)

gains are achieved by increasing the value of ϵ_H, although this comes at the cost of an inaccuracy in the optimal solution due to the mass leak discussed above. For the research described in this dissertation, such a mass leak is not created while still maintaining the favorable gradients, meaning significant performance improvements are achieved with no negative side effects on account of the constraint functions. For this research, the value of ϵ_H is typically set in the range of 10^{-6} to 10^{-10}, although in the case of the constraints it is sometimes set as high as 10^{-2}. Of course, care must be taken to adjust the upper bound on the constraint value along with the fudge factor, since the maximum allowable constraint value now occurs not at T_{max} but at $\sqrt{T_{max} + \epsilon_H}$. No formal study of an optimal value for ϵ_H is performed for collocation methods in this research, though such a study may be valuable in the future.

3.5.5 Objective function

For the majority of this research, the objective function J (as defined in Equation 3.1) is geared towards minimizing propellant consumption. At times alternative objective functions are used, which is then called out specifically in the text, but in general the discussion provided here applies. At least two ways can be imagined to minimize propellant consumption, the simplest of which is simply subtracting the final mass from the initial mass, such as

$$J(\bar{X}) = M_f - M_0 \tag{3.43}$$

Where $\bar{X}$ is the optimization state vector and $M_{0/f}$ are the initial and final mass, respectively. While this objective function is very simple with very few and well-behaved derivatives, it does omit a lot of information, such as the obvious role that the control vector components (thrust and specific impulse) have in propellant consumption. This role is only implicitly defined, through the defect constraints that relate mass consumption to the thrust components. Furthermore, the NLP solver is now incentivized to try and violate the defect constraints for mass by as much as the tolerance will allow, since that directly benefits the objective function as it is defined in Equation 3.43, which is an unnecessary and fictional incentive. Instead, the objective function could be defined more explicitly through

$$J(\bar{X}) = \sum_{i=0}^{n-1} \frac{T_i \, \Delta t_i}{I_{sp,i} \, g_0} \tag{3.44}$$

where T_i is the thrust on segment i, Δt_i is the length of segment i, $I_{sp,i}$ is the specific impulse on segment i, and g_0 is the universal constant of gravitational acceleration, which is summed over all $n-1$ segments. This objective function depends directly on all thrust components, as well as the specific impulse, and therefore more explicitly models the cost as a function of the variables that physically influence it, which is thus also reflected in its derivatives. Indeed, the latter formulation leads to an improved convergence as a result of this.

Similar to the discussion in Section 3.5.4, the thrust on a segment T_i in Equation 3.44 can be replaced with

$$T_{H,J} = \sqrt{T_x^2 + T_y^2 + T_z^2 + \epsilon_{H,J}} \tag{3.45}$$

where the additional subscript J is used to indicate this hyperbolic equation is used for the objective function J, with a fudge factor $\epsilon_{H,J}$ that may be set to a different value than ϵ_H. Similar to the discussion in Section 3.5.4, increasing the value of $\epsilon_{H,J}$ leads to smoother gradients throughout the domain of the thrust components, and the figures demonstrating that are not reproduced here. Unlike the case of hyperbolic thrust constraints, increasing the value of $\epsilon_{H,J}$ also has a negative effect on the optimal solution that must be balanced with the smoother convergence. Figure 3.7

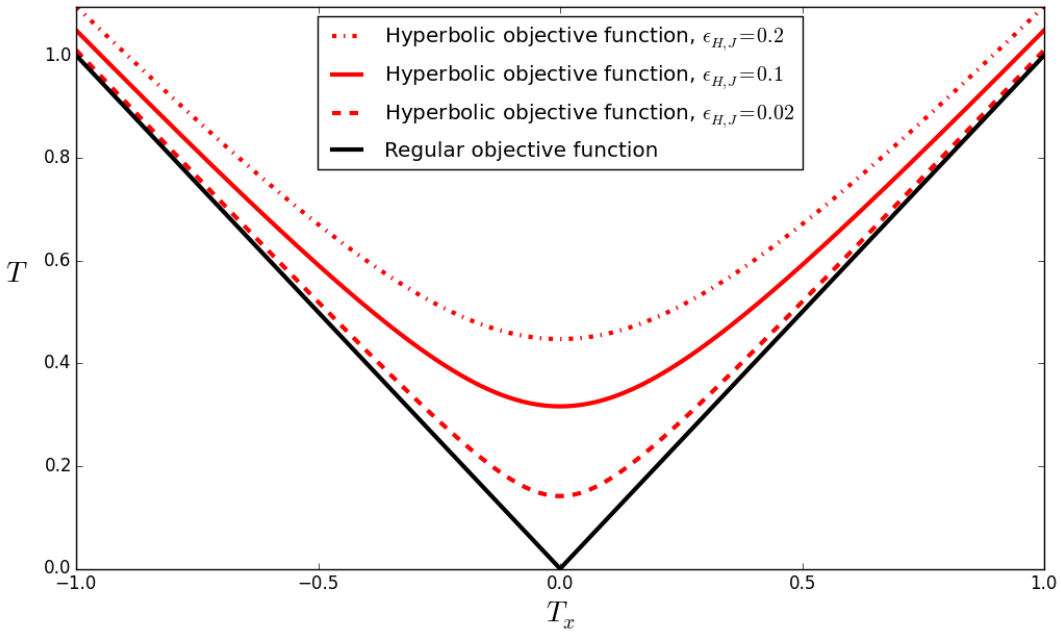


Figure 3.7: Comparison of a hyperbolic thrust constraints for various ϵ_H ($T_y = T_z = 0$)

reproduces the hyperbolic thrust curves for various $\epsilon_{H,J}$, which clearly shows how the basin where the first derivative is close to zero increases with $\epsilon_{H,J}$. If the NLP solver determines that a coast arc can be introduced, and moves towards $T_x = 0$, it encounters this region earlier when this basin is wider. Whereas for small $\epsilon_{H,J}$ the derivatives remain steep until very close to zero, for larger $\epsilon_{H,J}$ the NLP solver may determine that the gradients no longer warrant further improvement, leaving the thrust at (for example) 10% of the maximum value, instead of introducing a true coast arc. As a result, while no mass leak is modeled as it is in Reference [39], using these hyperbolic

objective functions nonetheless leads to sub-optimal results. The values for the fudge factor in the cost function are typically set in the range of 10^{-6} to 10^{-10}. With these values *Maverick* technically does produce sub-optimal results, since the thrust often does not go entirely to zero, but the impact of this on the cost function is negligible.

3.5.6 Additional constraints

Between the defect constraints and power constraints, the most basic requirements for any low-thrust trajectory optimization problem are in place. Of course, additional constraints could easily be added to suit the problem at hand. Furthermore, any desirable objective function can easily be combined with the formulation so far. Any modifications of that nature do not significantly affect the sparsity of the problem as discussed above, and typically require a very small number of additional computations. When additional constraints specific to the problem under discussion are added, this is called out specifically in the remaining text.

3.6 Differences from earlier implementations

The implementation of Gauss-Lobatto collocation in *Maverick* differs from existing implementations. One of the most significant differences is covered in Chapter 4, namely the focus on parallel computing. Another difference that is worth stressing is the fact that *Maverick* does not require the user to predefine coast and thrust arcs for the solution, as is the case in the implementations presented in References [25] and [5]. While such a user input does greatly benefit the convergence to some solution, it also limits the optimizer from truly searching the full solution space. As shown in the rest of this thesis, Chapter 7 in particular, for sufficiently complex transfers it is unlikely that a human would intuitively guess the correct sequence of thrust and coast arcs. After *Maverick* is given free reign to introduce its own coast and thrust arcs and finds a solution, the user could always enforce the newly found thrust/coast structure for additional runs to aid convergence, in case some degree of refinement is required. In any case, removing the requirement for predefining the thrust and coast sequence gives *Maverick* greater freedom but also makes it

harder to find and converge on a solution. This greater freedom for *Maverick* is only effective due to the collection of smaller improvements described in this chapter, such as the extensive control of the problem through a range of scaling factors, described in Section 3.5.2, and the use of hyperbolic thrust constraints and cost function, described in Sections 3.5.4 and 3.5.5, respectively. Collectively, these alterations result in an optimization tool that can be used more freely to explore the full solution space of low-thrust trajectory optimization problems.

3.7 Chapter conclusion

This chapter provides further discussion of trajectory optimization methods, specifically those using Gauss-Lobatto collocation. A thorough explanation is given of the optimization tool that is developed for this research, which is named *Maverick*. Both the 4^{th} and 12^{th} order Gauss-Lobatto methods, GL-4 and GL-12, are described and explained. For the rest of this research, GL-4 is only used for the performance comparison of parallel implementations in Chapter 4. For all other results, GL-12 is used due to its greater numerical order, leading to a more efficient formulation of the optimization problems under consideration.

Chapter 4

Parallel results

As discussed in Section 3.2, one desirable outcome of direct transcription through collocation is the resulting super-sparse Jacobian. It also means most of the problem can be computed in parallel, allowing for distributing the workload across dozens or even hundreds of processors. The combination of the super-sparse nature of the problem and the high parallelization potential allows for impressive computational performance, which is discussed in this chapter. Section 4.1 discusses the relative performance between the code that is actually parallelized in this research and the NLP solver. Section 4.2 then assesses the achieved performance improvements specifically for individual elements of *Maverick* as they are executed in parallel. Section 4.3 then puts this all together to look at the overall performance improvements for *Maverick* for a range of conditions. Section 4.4 then concludes the Chapter.

Most of the results and discussion in this Chapter were first presented in Reference [32].

4.1 Overall performance

At the outset of this research, one of the primary assumptions was that the majority of the runtime would be spent in the "user code" as opposed to the NLP solver. This was based on private communication with others working on similar methods. Unfortunately, after extensive testing, that could not be reproduced for this research. A summary of the changes attempted is presented in Appendix B. This discrepancy may largely be explained by the differences with respect to earlier implementations of the same collocation schemes that are discussed in Section 3.6. However,

since all these changes represent improvements, it is considered counterproductive to remove these improvements in order to attempt to reproduce the behavior in runtime. The limitations this presents are illustrated for a variety of force models in Tables 4.1, with some explanation below.

Table 4.1: Maximum achievable performance improvements for a variety of force models under current conditions.

	Force model						
	Two Body Earth	10x10	30x30	50x50	100x100	DE430 Earth	DE430 S-E-M
R_{serial} (%)	11.61	36.98	77.99	87.88	95.98	31.16	40.24
Performance improvement							
$Z(K = 83\%)$	1.11	1.45	2.86	3.76	5.00	1.35	1.51
$Z(K = 90\%)$	1.12	1.50	3.36	4.83	7.37	1.39	1.57
$Z(K \approx 100\%)$	1.13	1.59	4.56	8.53	26.16	1.45	1.68

Table 4.1 shows theoretically achieveable performance improvements for parallelized constraint and derivative evaluations. It is produced by optimizing the problem described in Section 4.3 under the influence of a variety of force models, using code that performs in serial fashion. The DE430 force model is the same one used by TurboProp [33]. Table 4.1 includes a distinction between a two-body force model of Earth which simply assumes the point mass to be at [0,0,0] in the reference frame, and one that computes Earth's actual position from the DE430 ephemeris, since there is a noticeable difference in required computational effort. Table 4.1 also shows a force model that includes the Sun, Earth, and Moon (S-E-M) from the DE430 ephemeris. In both cases of the DE430 force model, only point mass gravitational attraction is used for the included bodies. All aspherical gravity models assume Earth to be at [0,0,0].

The ratio R_{serial} is computed by dividing the serial runtime of the constraint and derivative evaluations by the total serial runtime of the optimization, as follows

$$R_{serial} = \frac{(\text{Constraint+Derivative Evaluation Runtime})_{serial}}{(\text{Total Runtime})_{serial}} \tag{4.1}$$

A potential performance improvement Z is then estimated based on an improved runtime of the constraint and derivative evaluations. The degree to which the performance is improved is set by a speed-up factor K

$$Z = \frac{1}{1 - K \cdot R_{serial}} \tag{4.2}$$

As stated above, R_{serial} is determined from actual optimization runs, and is a number that does not significantly change for either the 4^{th} or 12^{th} order methods, nor does it change significantly for a different number of segments. It is mostly affected by the force model being used. It also varies somewhat among IPOPT's different linear solvers, or when a very unconventional scaling of the optimization parameters is used, but in both cases not greatly so. On the other hand, K is a fictional number as it appears in this table, but it essentially sets an assumption of how great a performance improvement is expected to be achieved by parallelizing the constraint and derivative evaluations. The best performance improvement demonstrated in this study was a factor of approximately 11, as is shown in the following sections, which roughly corresponds to $K = 90\%$, since 90% of the serial runtime is eliminated. A more typical performance improvement demonstrated in this study is a factor of approximately 6, resulting in $K = 83\%$, which is also included. Finally, a highly optimistic estimate is included, which assumes the parallel improvement can be made so fast as to have an insignificant runtime, or $K \approx 100\%$. This is merely included to set an upper bound on the performance improvement that can be achieved by parallellizing the constraint and derivative evaluations.

Considering these definitions, Table 4.1 shows that no matter how effective a parallel implementation of the constraint/derivative evalutations is, it will not translate into a very large performance improvement for a problem defined by pure two-body motion, under current conditions. On the other hand, in the case of very expensive force models, notable performance improvements are feasible. Although as will be discussed, performance improvements on a GPU can get complicated for these very memory intensive force models, which means actually achieving such performance improvements is challenging. It may also be that alternative formulations or scalings of the problem can improve R_{serial} for a given force model, without sacrificing the total performance, thus allowing for a greater impact of a parallel implementation. However, as discussed in Appendix B a wide

range of aspects were varied extensively, and the presented results reflect the best outcome of these many alternatives.

If it seems unlikely that a single LEO case could accurately summarize a ratio like R_{serial}, as described above, keep in mind that this ratio makes no assumptions about the total runtime of the optimization problem. If the problem is exceptionally tough, this is reflected by the optimizer requiring many iterations before converging to a solution. However, the division of labor within a single iteration between the optimization library and the user-written constraint/derivative evaluations is fairly constant throughout these iterations, regardless of how many iterations occur. As mentioned before, some aspects outside of the force model do affect R_{serial}. Table 4.1 is therefore not meant as a definitive statement on the utility of parallelizing a problem that relies on any particular force model, but it can be a useful guide nonetheless. Finally, to avoid the impression that IPOPT is being blamed for limiting the achieved performance, it is worth reiterating that IPOPT was compared extensively to SNOPT during this study, and performed significantly better. There is clearly a significant overhead from IPOPT, especially when an optimization problem only requires a very cheap force model, but despite that it is the best performing NLP solver that the authors have found. There remains the possibility that this overhead is an artifact from IPOPT (or SNOPT) being incorrectly utilized by the authors, but a great deal of effort has been invested to experiment with alternative configurations, with no significant change to the results presented here.

In the face of these findings, it is clear that "merely" improving the runtime of the "user code" will have a limited impact on the total runtime, especially for cheap force models. Nonetheless, it is considered relevant to know if significant performance improvements can be achieved in the "user code". First of all, these improvements would provide a notable runtime reduction when expensive force models are used, which happen to be the problems in greatest needs of improvement to begin with. Second of all, if performance improvements in the "user code" are demonstrated, this creates greater incentive for NLP solvers like IPOPT to be improved, in order to fully leverage the improvements to the "user code". As recently presented in Reference [20], attempts to

provide such performance improvements by using parallel computing within the NLP solver itself shows promising early results. With all this in mind, the achieved performance improvements are presented in the remainder of this chapter.

4.2 Raw Performance results

Before presenting the results for full optimizations, the performance of some individual elements is presented. As mentioned, to improve overall performance of the optimization process, it is critical to efficiently compute the constraint derivatives, and to some extent the constraints

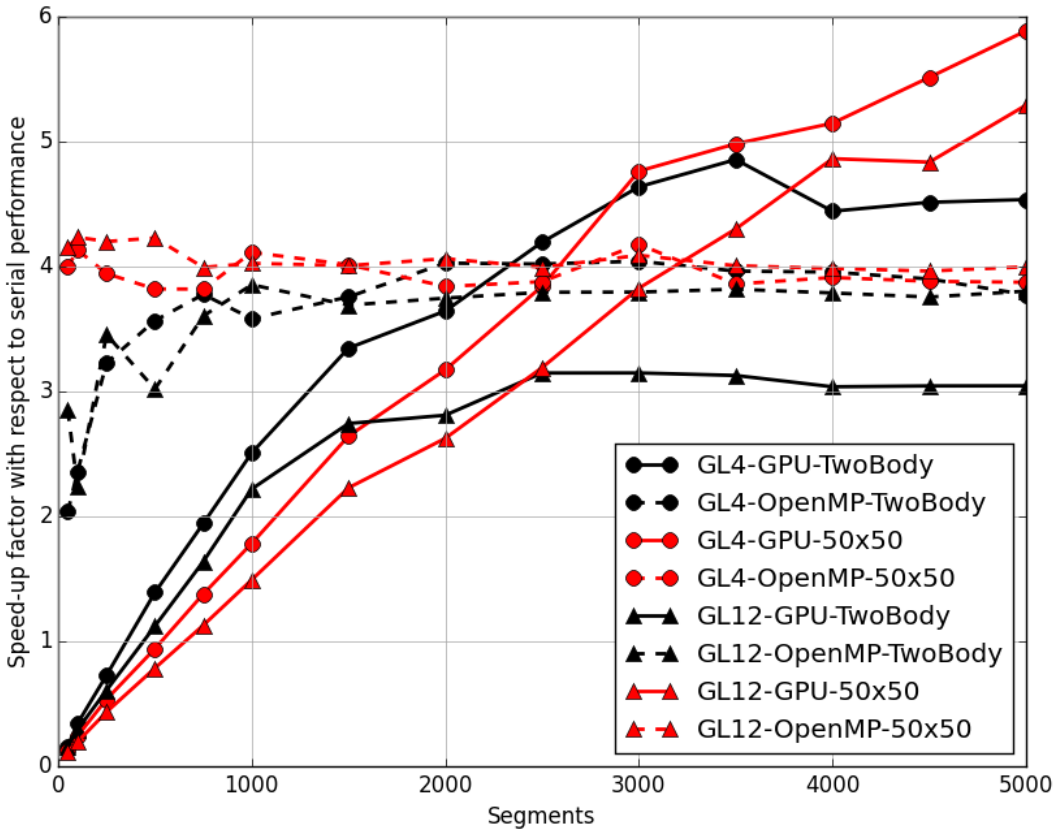


Figure 4.1: Illustration of relative performance for the constraint evaluations, showing the 4^{th} and 12^{th} order methods with circular and triangular markers respectively. The GPU implementation is shown in a solid linestyle, with the OpenMP implementation in a dashed linestyle. The two-body force model is shown in black, with the 50x50 aspherical gravity force model shown in red.

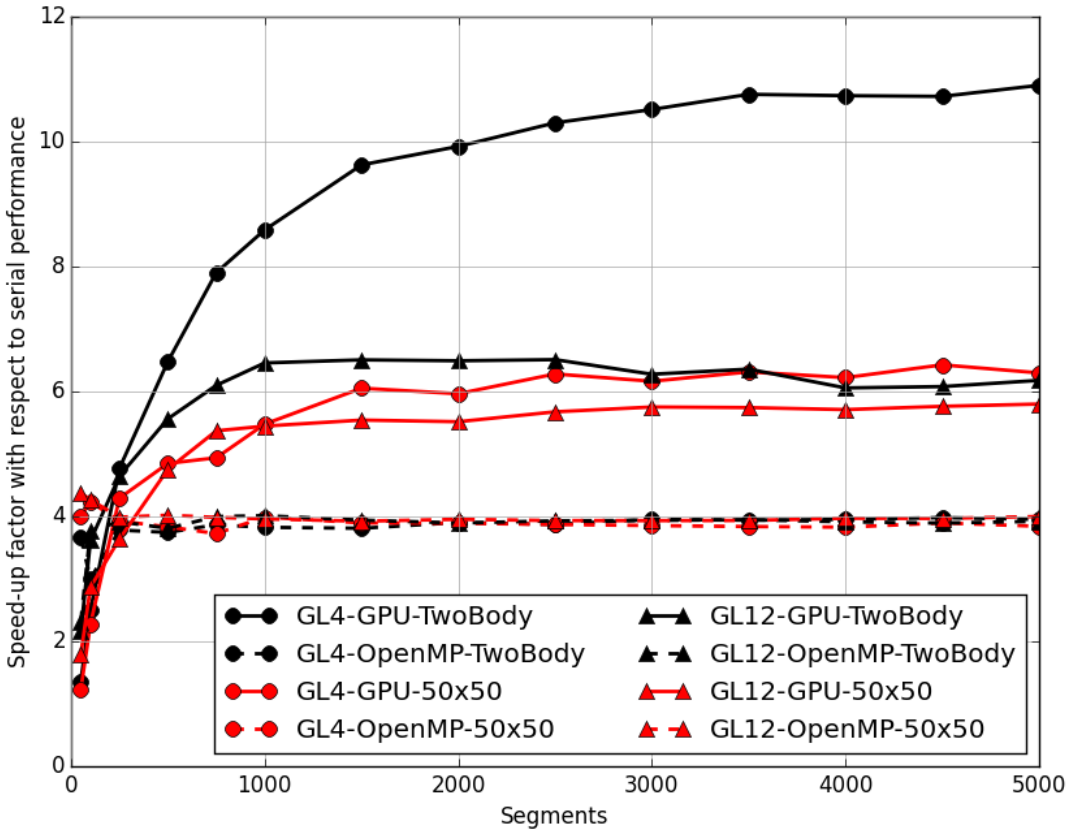


Figure 4.2: Illustration of relative performance for the derivative evaluations, showing the 4^{th} and 12^{th} order methods with circular and triangular markers respectively. The GPU implementation is shown in a solid linestyle, with the OpenMP implementation in a dashed linestyle. The two-body force model is shown in black, with the 50x50 aspherical gravity force model shown in red.

themselves. This is achieved by writing parallel implementations of these functions that utilize either the CPU or the GPU. The relative performance of these functions is shown in Figures 4.1 and 4.2 for the constraint and derivative evaluations, respectively. For the derivative evaluations, Figure 4.3 shows the trends for low numbers of segments in greater detail. Additionally, the absolute and relative performance of these functions is presented for a selected number of segments in Tables 4.2 and 4.3.

To produce these results, these functions are called outside of the optimization algorithm, but perform the exact same steps. To filter out noise in the performance that can occur for any

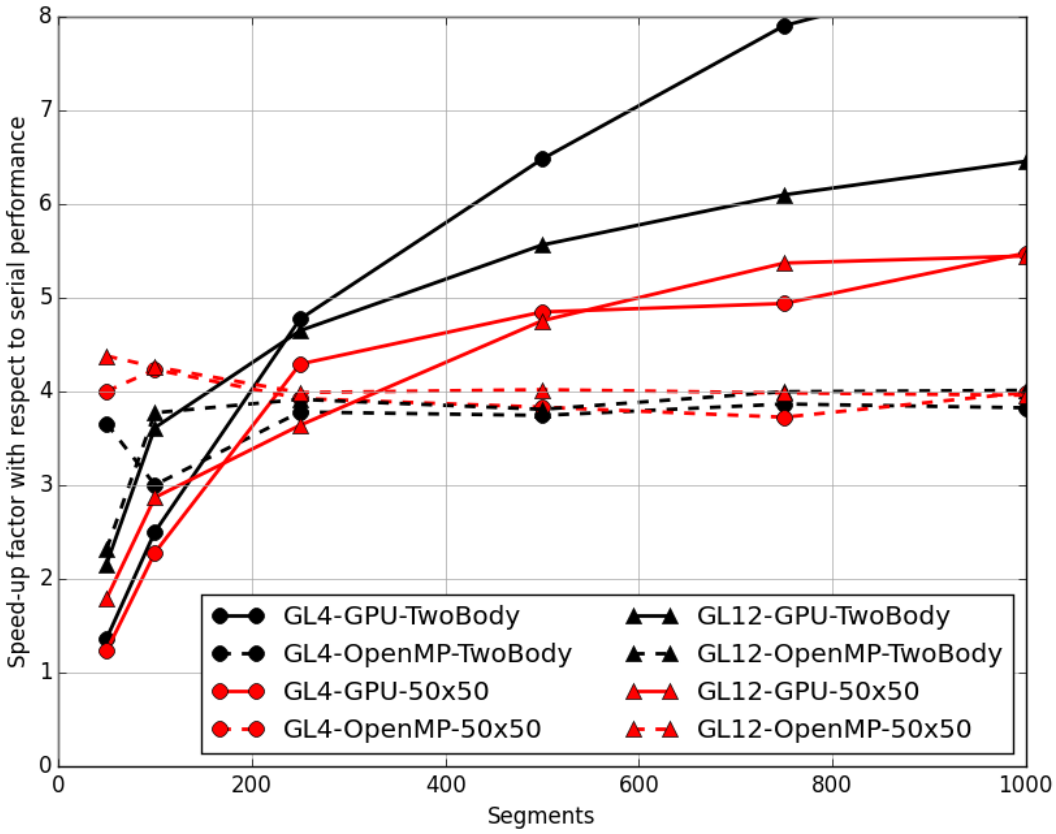


Figure 4.3: Illustration of relative performance for the derivative evaluations for a low number of segments, showing the 4^{th} and 12^{th} order methods with circular and triangular markers respectively. The GPU implementation is shown in a solid linestyle, with the OpenMP implementation in a dashed linestyle. The two-body force model is shown in black, with the 50x50 aspherical gravity force model shown in red.

single function call, the constraint functions are called 10,000 times each. The derivative functions are called 1,000 times each, due to their much larger computational cost. The OpenMP results use both threads on all 4 CPU cores, whereas the serial results use a single thread on a single CPU core. The GPU attempts to utilize as many of the GPU cores as it can, but it can not always utilize the full GPU, especially in the case of the constraint evaluations. For these constraint evaluations parallelization occurs at the segment level, both for the GL-4 and GL-12 methods. For both methods, this leads to a total of $(n-1)$ parallel processes, where $(n-1)$ is the total number

Table 4.2: Absolute and relative performance of constraint (Cons.) and derivative (Deriv.) calculations for 4^{th} order Gauss-Lobattto scheme using the two-body and 50x50 force models. Absolute performance refers to the total runtime of 10,000 constraint or 1,000 derivative evaluations, respectively.

Number of segments	100		1,000		5,000	
Two-body force model	Cons.	Deriv.	Cons.	Deriv.	Cons.	Deriv.
Serial runtime (s)	0.29	0.66	2.78	6.67	13.89	33.03
OpenMP runtime (s)	0.12	0.22	0.78	1.74	3.68	8.36
GPU runtime (s)	0.85	0.26	1.11	0.78	3.06	3.03
Serial runtime / OpenMP runtime (-)	2.36	3.00	3.58	3.83	3.77	3.95
Serial runtime / GPU runtime (-)	0.34	2.50	2.51	8.59	4.54	10.90
50x50 force model	Cons.	Deriv.	Cons.	Deriv.	Cons.	Deriv.
Serial runtime (s)	19.49	45.18	191.71	441.40	960.58	2212.91
OpenMP runtime (s)	4.71	10.67	46.60	110.64	247.95	575.96
GPU runtime (s)	80.63	19.81	107.45	80.58	163.17	351.40
Serial runtime / OpenMP runtime (-)	4.13	4.23	4.11	3.99	3.87	3.84
Serial runtime / GPU runtime (-)	0.24	2.28	1.78	5.48	5.89	6.30

Table 4.3: Absolute and relative performance of constraint (Cons.) and derivative (Deriv.) calculations for 12^{th} order Gauss-Lobattto scheme using the two-body and 50x50 force models. Absolute performance refers to the total runtime of 10,000 constraint or 1,000 derivative evaluations, respectively.

Number of segments	50		500		2,000	
Two-body force model	Cons.	Deriv.	Cons.	Deriv.	Cons.	Deriv.
Serial runtime (s)	0.34	1.33	3.46	13.05	13.77	52.31
OpenMP runtime (s)	0.12	0.57	1.14	3.43	3.68	13.43
GPU runtime (s)	2.18	0.62	3.08	2.35	4.90	8.06
Serial runtime / OpenMP runtime (-)	2.85	2.32	3.02	3.81	3.75	3.89
Serial runtime / GPU runtime (-)	0.16	2.16	1.12	5.56	2.81	6.49
50x50 force model	Cons.	Deriv.	Cons.	Deriv.	Cons.	Deriv.
Serial runtime (s)	25.21	92.44	234.24	858.63	941.84	3430.15
OpenMP runtime (s)	6.07	21.11	55.39	213.60	231.79	867.48
GPU runtime (s)	214.65	51.65	298.07	180.57	358.75	621.80
Serial runtime / OpenMP runtime (-)	4.15	4.38	4.23	4.02	4.06	3.95
Serial runtime / GPU runtime (-)	0.12	1.79	0.79	4.76	2.63	5.52

of segments for the current problem, conforming to the notation in Chapter 3. This explains the poor performance of the GPU for a low number of segments, due to only a fraction of the full GPU being utilized. For the derivative evaluations, the parallelization occurs at the level of every individual optimization state element, per defect constraint. For the 4^{th} order scheme this results

in a total of $22(n-1)$ parallel processes, and for the 12^{th} order scheme this results in $36(n-1)$ parallel processes, again with $(n-1)$ the total number of segments. As a result, for a given number of segments, the derivative evaluations achieve a much greater utilization of the GPU. It should be noted that the evaluation of the constraint functions could be reformulated to run across more cores for a given number of segments, but at this time this has not been pursued, largely due to the fact that the constraint function requires a much lower absolute runtime than the derivative functions: The constraint evaluations are more than 20 times faster than the derivative evaluations in the case of GL-4, and more than 30 times faster in the case of GL-12.

Focusing on the derivative evaluations, the OpenMP implementation runs approximately 4 times faster than the serial implementation. Considering the CPU that is used for these results has 4 cores, this is to be expected. It is very likely that if a CPU with more cores were utilized, the relative performance would improve proportionally, up to some point. The GPU improves on this factor 4 considerably, although for a low number of segments it can perform somewhat worse than the OpenMP implementation, again due to the full GPU not being utilized. For large problem sizes however, the GPU can be between 1.5 and 2.5 times faster than the OpenMP implementation. The OpenMP implementation has a fairly steady relative performance regardless of force model and method used, but for the GPU a large difference exists between the GL-4 two-body results and all remaining results. Inevitably, both the 50x50 force model and the GL-12 method in general involve the use of more memory when assessing the constraints or derivatives than the GL-4 two-body results. When the CPU is used, a large amount of on-chip memory is accessible by the computational cores. However, a GPU is characterized by many small cores with a small amount of on-chip memory. In the case of complex force models, or the GL-12 method, much more reliance on off-chip memory occurs, and this noticeably affects performance. Despite this, the GPU still provides the best performance for moderate and large problems, but it does benefit significantly when the necessary computations have a low memory footprint.

It should be noted that the largest GPU performance improvement for the constraint evaluations is for the more expensive force models, as can be seen in Figure 4.1, which does not agree with

the explanation given above for the derivative evaluations. In the case of the constraint function, the complete evaluation is sufficiently cheap that there is a greater influence of overhead involved with the CPU-GPU memory transfers, in both directions. Due to the larger runtime of the more computationally expensive force models, this is less of a factor here. As result, the latter cases converge to the same performance improvement of approximately 6 times faster as seen with the derivative evaluations, although many more segments are required, as explained above. For the cheaper force models, the overhead involved with using the GPU limits the performance improvement for the constraint function more dramatically, but there is still a noticeable difference for the GL-4 and GL-12 methods, similar to what is observed for the derivative evaluations. As mentioned before, the relative runtime of the constraint function is sufficiently low that it is not considered a priority to further improve performance, which may have been accomplished by using the ability of modern GPUs to simultaneously perform computations while also transferring memory to or from the CPU.

4.3 Full Performance Results

Next, results are presented from a full optimization run, which means the runtime includes operations by the IPOPT library. Throughout this study, a variety of test cases were considered in varying Earth orbits, as well as some heliocentric orbits, mostly Earth-Mars transfers. For the purposes of the results presented here, no significant difference existed between these widely varying scenarios. Even when total runtime significantly varied, the relative performance that is highlighted below remained identical. For this reason, the results shown here are limited to a single orbit problem, to enable the most effective comparison of the different results. The presented results are for a 45 degree inclined circular Earth orbit at a radius of 6600 km. The time of flight is fixed at 7.5 hours, which corresponds to roughly 5 revolutions. The spacecraft initial mass is 3035 kg, which includes a 49kW power supply, 60% of which can be used for thrust. The specific impulse is fixed at 2000s. These numbers are not meant to mimic a specific vehicle or thruster, but these values are realistic for SEP performance in the near future, although certainly on the high end of that range.

The objective function for this test case is to maximize the two-body orbital energy at Earth, or in other words, to move towards escape velocity. There is of course an analytical solution to this problem, or at least in the case of two-body motion, which is to thrust in the velocity direction at all times. However, the purpose is to have some non-linear objective function that can easily be checked for erroneous output for these test cases.

In all cases presented here, the initial guess for the optimizer is the propagated state history of a ballistic coast in the starting orbit, which can be obtained very simply and cheaply, relative to the optimization itself. Without exception, the optimal solution is found, and agrees among all methods to the tolerance set within IPOPT. This tolerance is left at the default value of 10^{-8}. The linear solver ma86 is used for IPOPT in all presented results, although many of IPOPTs other solvers were tested with comparable results. Finally, IPOPT is set to use a limited-memory Hessian for all presented results.

The results of the problem formulation described above are presented for both the 4^{th} and 12^{th} order schemes, using the 50x50 force model, in Tables 4.4 and 4.5. The presented runtimes are the time required to perform a single optimization run, although this number is acquired by averaging the runtime of running the same problem 100 times. It should immediately be apparent that both the OpenMP and GPU implementations offer performance improvements up to 3.7 times faster with respect to the serial implementation. However, it should also be noted that compared to Tables 4.2 and 4.3, the performance of both the OpenMP and GPU implementations offer less improvement with respect to serial performance than they do when purely assessing the functions by themselves. The reason for this is the additional computational effort performed by IPOPT to iteratively solve the optimization problem, which is unaffected by the parallelizations implemented in the constraint and derivative evaluations. As explained in Section 4.1, this unfortunately presents a barrier to fully utilizing the performance improvements enabled by the implemented parallelizations. As explained there, this is even more readily apparent for the two-body force model, where the runtime is dominated by the NLP solver. As a result, those minimal performance improvements for the full optimization are not detailed here.

Table 4.4: Absolute and relative performance of full optimizations for the 50x50 force model and the 4^{th} order Gauss-Lobattto scheme.

Number of segments	100	1,000	5,000
Serial runtime (s)	1.36	25.16	135.91
OpenMP runtime (s)	0.49	8.07	44.58
GPU runtime (s)	0.93	6.98	36.83
Serial runtime / OpenMP runtime (-)	2.80	3.12	3.05
Serial runtime / GPU runtime (-)	1.46	3.61	3.69

Table 4.5: Absolute and relative performance of full optimizations for the 50x50 force model and the 12^{th} order Gauss-Lobattto scheme.

Number of segments	50	500	2,000
Serial runtime (s)	3.34	38.23	144.33
OpenMP runtime (s)	1.07	17.11	46.45
GPU runtime (s)	2.76	19.21	41.42
Serial runtime / OpenMP runtime (-)	3.12	2.23	3.11
Serial runtime / GPU runtime (-)	1.21	1.99	3.48

4.4 Chapter summary

The results presented in this chapter show significant performance improvements for low-thrust trajectory optimization problems by parallelizing the constraint and derivative evaluations. The largest performance improvements are achieved by using a GPU, but notable performance improvements are also feasible by using all the cores on a CPU. These performance improvements have the greatest impact when the force model is computationally expensive. For cheaper force models, the runtime is dominated by the NLP solver, and parallelizing the constraint and derivative evaluations has a limited effect on the overall performance. This limitation may not exist in the future, either through better use of existing NLP solvers, or the development of improved NLP solvers, which perhaps use parallel computing for improved performance themselves. As discussed in Reference [20], this is a promising avenue of research. At any rate, if this limitation is removed or reduced, performance improvements of an order of magnitude or more are readily accessible for trajectory problems with cheap force models, as demonstrated in this chapter.

For the time being, the performance improvements demonstrated here apply to those prob-

lems that suffered from the greatest computational challenges to begin with. These problems can now be solved up to 3.7 times faster, enabling more efficient analysis of this class of low-thrust trajectory optimization problems. Furthermore, as computer architecture shifts more towards parallel execution, the performance improvements of the methods demonstrated in this chapter are likely to grow over time as the hardware itself becomes more capable.

Chapter 5

Transfers between Halo-like orbits

The transfers described in this chapter are in the vicinity of halo orbits in the Earth-Moon system. Their primary application is for the New Worlds Observer (NWO) mission [13, 59], which requires a telescope and a starshade to fly in formation at several thousand kilometers of separation. In the discussed scenario, the telescope is in a halo orbit around L_2 in the Earth-Moon system, performing no maneuvers other than orbit maintenance. The starshade is displaced from the halo orbit in such a manner that it is at the correct distance and angular position with respect to the telescope to perform an observation. The method for placing the starshade is not discussed here, but it is repeated for several observations. No ballistic transfer typically exists between these starshade orbits, instead a low-thrust system is used to move the starshade from one observation to another. The problem formulation for *Maverick* is to solve this transfer for some time of flight while minimizing propellant consumption. A range of transfers between two such observation orbits is discussed here, chiefly as an example of *Maverick*'s capabilities, but in addition to illustrate why the Earth-Moon system is not favorable to the NWO mission.

5.1 Assumptions

For the presented transfers, the gravitational forces of all planets, the Sun, and the Moon are included, using the DE405 JPL ephemeris [63]. In addition, solar radiation pressure is accounted for, with an area-to-mass ratio of 0.01 and a coefficient of reflectivity of 0.2. The starshade has a mass of 500 kg and a power supply of 5 kW. The propulsion system has a fixed specific impulse of

2000 s, and a jet efficiency of 60%. The time between observations is 45 days, which serves as an upper limit to the time of flight. The initial guess for each transfer is a ballistic propagation from the departure orbit. All transfers use the same arrival time and state, which means the departure date is varied to achieve shorter times of flight, along with the departure state. To get the new departure state, a ballistic propagation of the original departure state is performed.

For those interested in reproducing these results, the Earth-centered departure state for the 45-day transfer is

$$X_0 = \{ -286837.6928846151,\ 296780.8421312311,\ 145441.2263636905,$$
$$-0.8612818763804879,\ -0.7051009078841147,\ -0.2195552680179014\}$$

in km and km/s, at a Julian Date of 2458861.634249815. The above state can be used to reproduce all other departure states. The Earth-centered arrival state (for all transfers) is

$$X_f = \{479029.4198407893,\ 55636.24675689182,\ -31564.86323503604,$$
$$-0.06654409392372701,\ 0.9311436484842787,\ 0.3948527827705729\}$$

in km and km/s, at a Julian Date of 2458906.634249815.

5.2 Results

Figure 5.1 presents the final mass as a function of flight time. Each of the markers represent a fully optimized low-thrust trajectory, which are spaced 1 day apart. Figure 5.1 clearly shows that a minimum of approximately 15 kg of propellant is required, which gradually increases as the time of flight decreases. A minimum time of flight of 27 days can be achieved, though at significant additional propellant cost. Below, three transfers are presented in greater detail, to illustrate the evolution of this solution family.

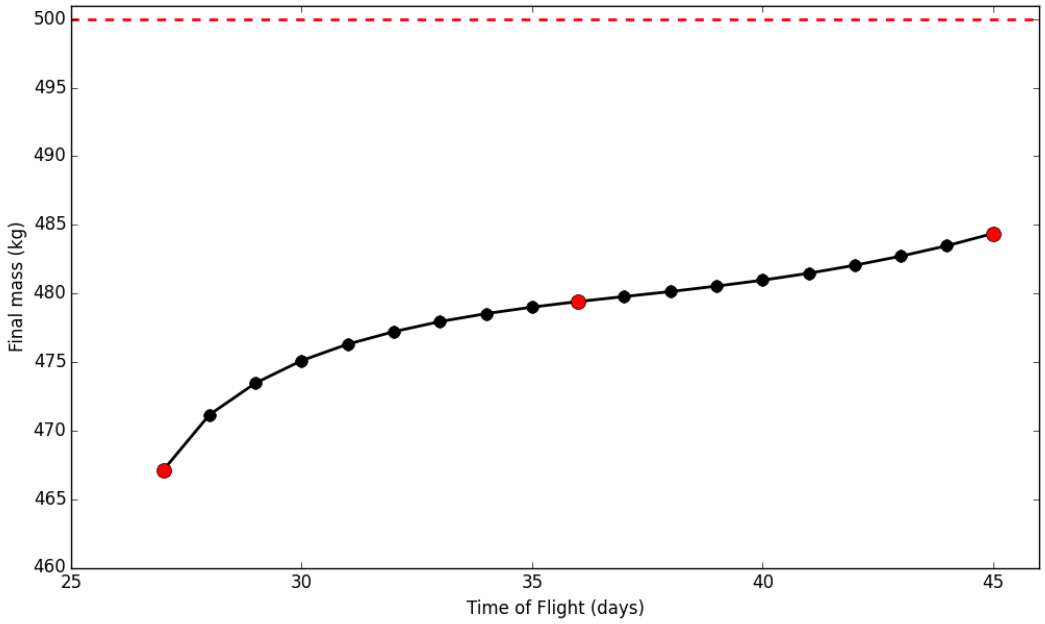


Figure 5.1: Final mass as a function of time of flight for the starshade transfers. The red dashed line indicates the initial mass, and the red marked points indicate those that are presented in more detail below.

5.2.1 45-day transfer

The maximum time transfer of 45 days is presented more thoroughly in Figures 5.2 through 5.4. Figure 5.2 clearly shows that there are two distinct burn arcs at the start and end of the transfer. An additional small burn occurs at 36 days into the transfer, which is likely an artifact of the low number of segments used (40 total, which provides acceptable accuracy). This would disappear entirely with the use of additional segments, or a stricter optimization tolerance, but the impact would likely only be several grams of propellant, and is thus considered insignificant. Figures 5.3 and 5.4 show the transfer in the Earth-Moon and Sun-Earth rotating frame, respectively.

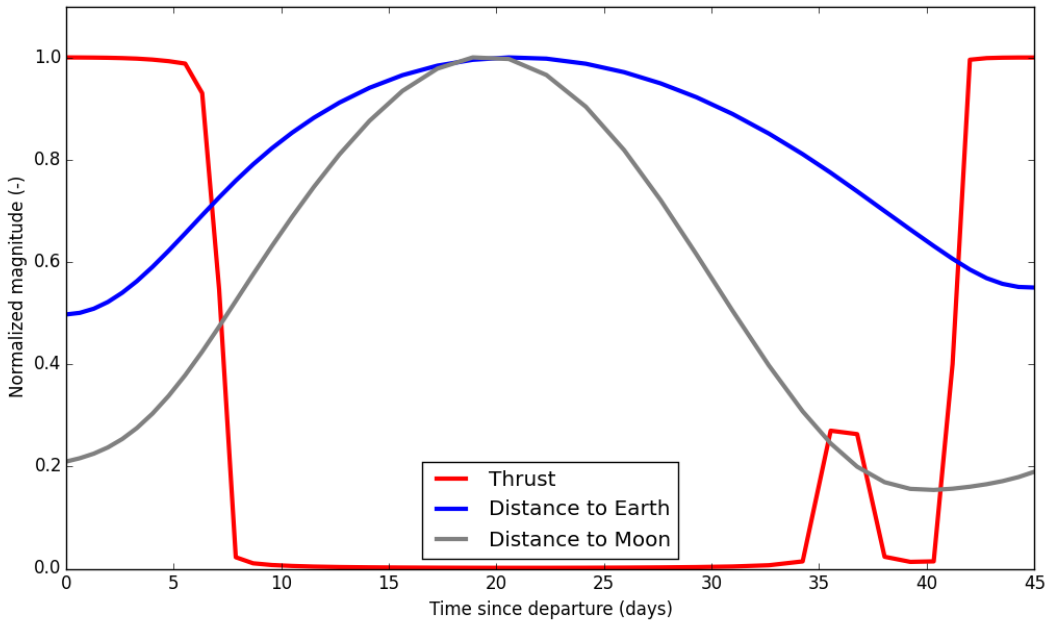


Figure 5.2: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 45-day starshade transfer.

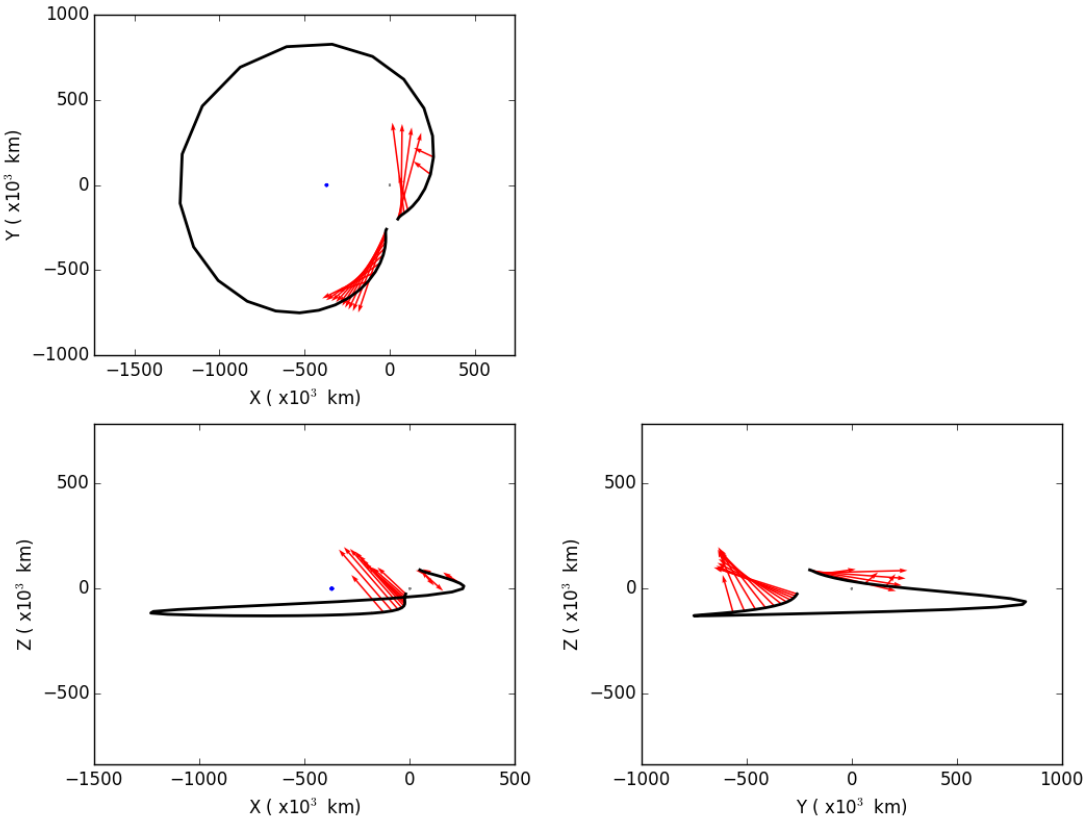


Figure 5.3: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 45-day starshade transfer. Both the Earth and the Moon are shown to scale.

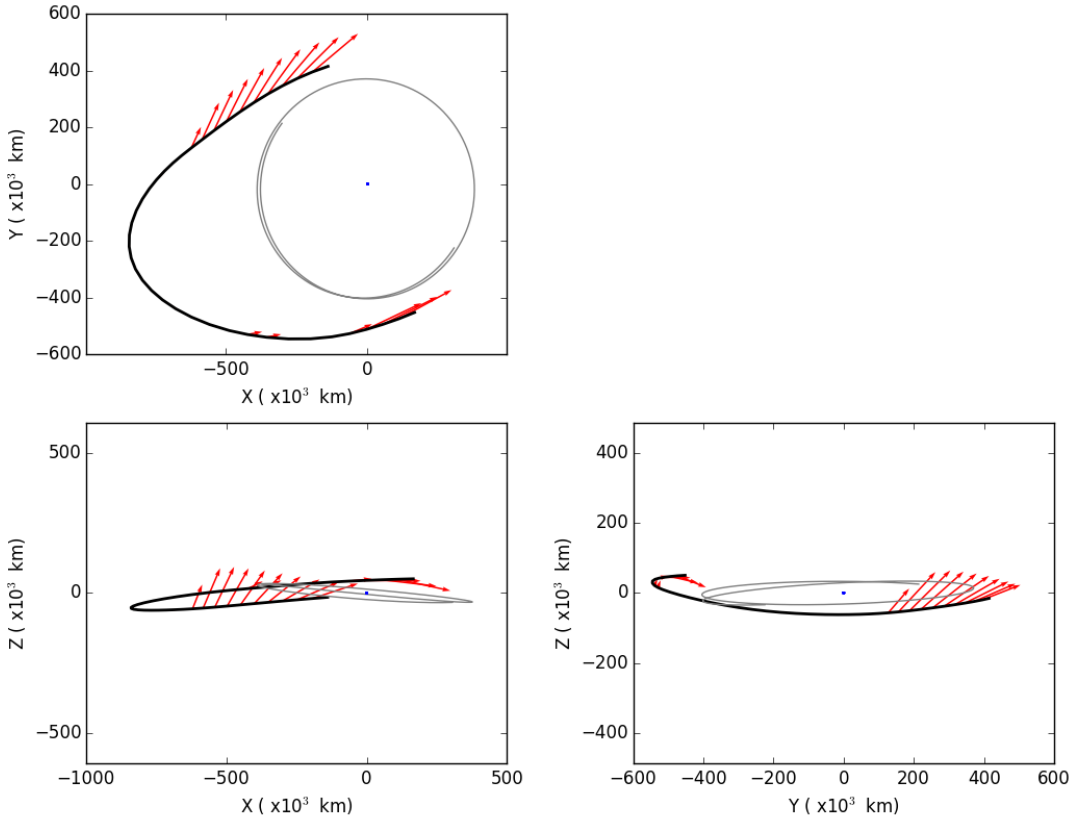


Figure 5.4: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 45-day starshade transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

5.2.2 36-day transfer

The 36-day starshade transfer is presented more thoroughly in Figures 5.5 through 5.7. Figure 5.7 clearly shows the starshade has drifted quite far from the halo-region it originally occupied, but the other figures show that overall a very similar transfer is still feasible, albeit at roughly 30% additional required propellant.

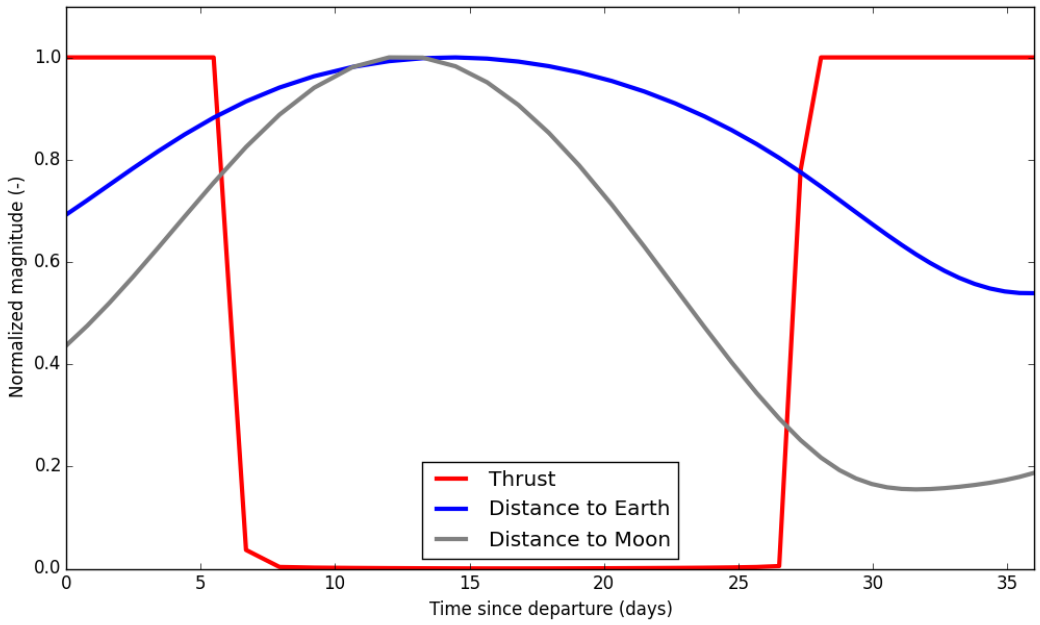


Figure 5.5: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 36-day starshade transfer.

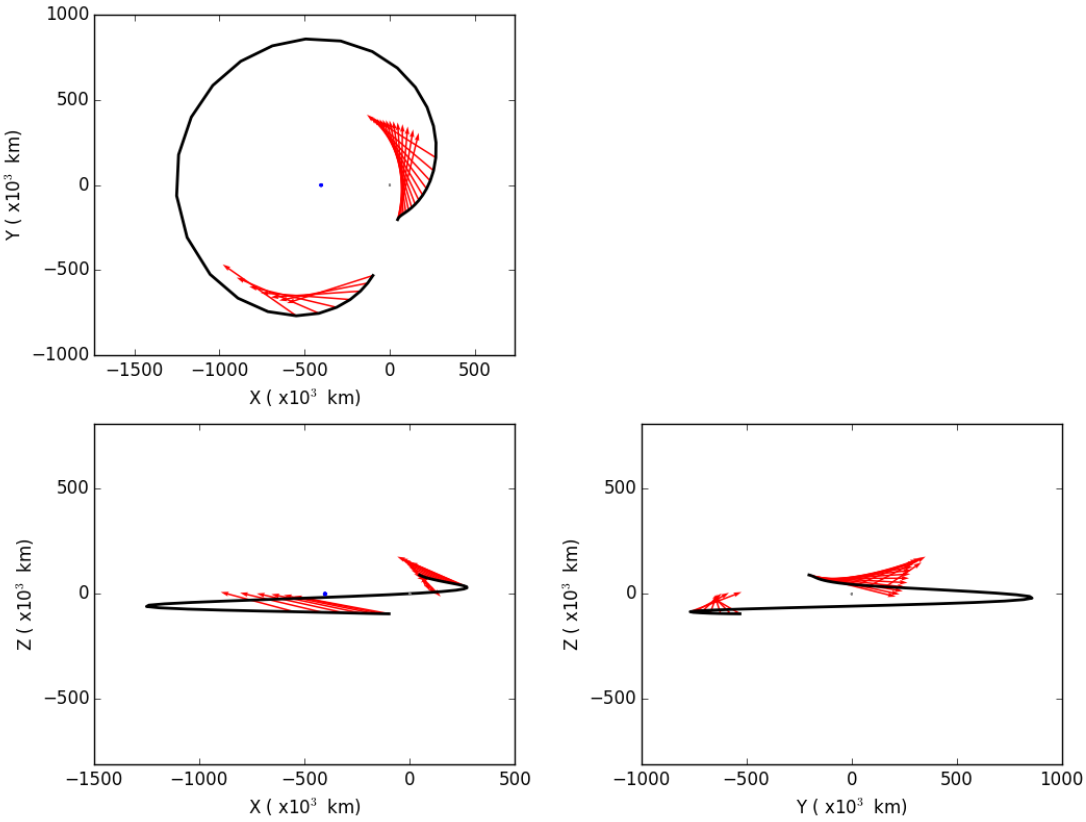


Figure 5.6: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 36-day starshade transfer. Both the Earth and the Moon are shown to scale.

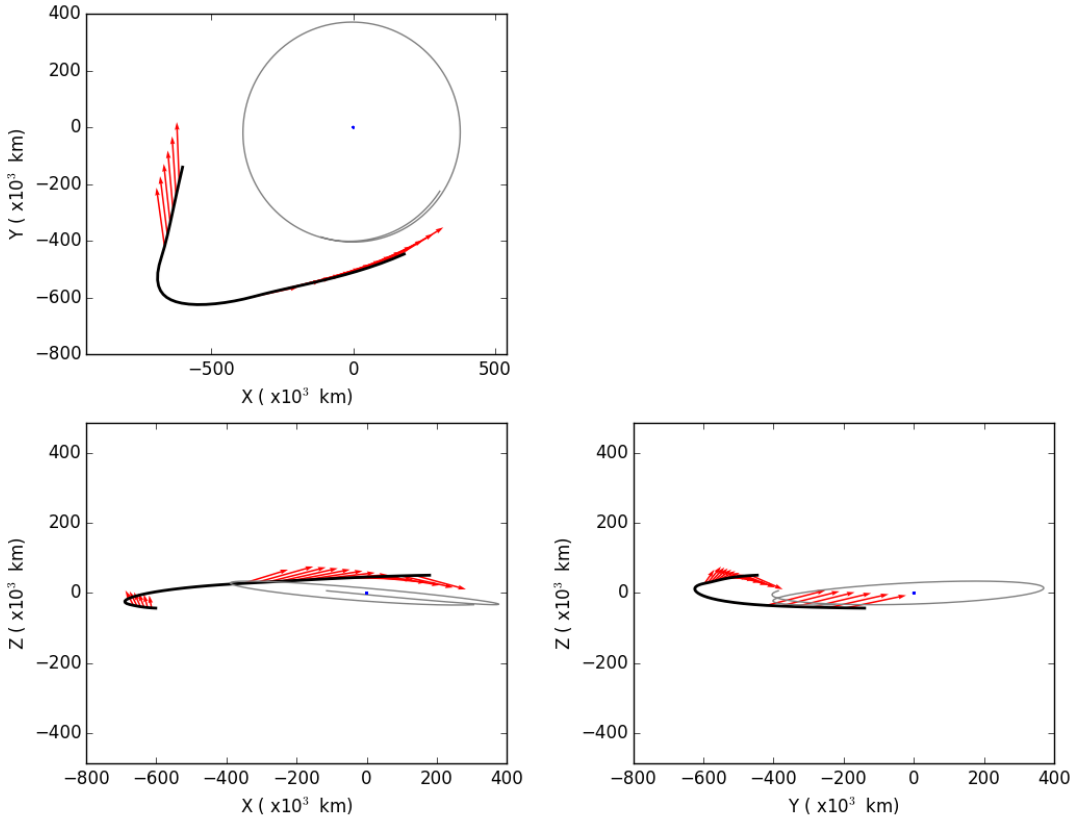


Figure 5.7: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 36-day starshade transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

5.2.3 27-day transfer

The minimum-time, 27-day starshade transfer is presented more thoroughly in Figures 5.8 through 5.10. Figure 5.8 clearly shows that almost all time is required to thrust in order to make the transfer succeed. The other figures also show how far the starshade has drifted, and how aggressive the transfer is as a result. It is conceivable that a few more hours could be cut from the transfer time, but a 26-day transfer proved entirely infeasible.

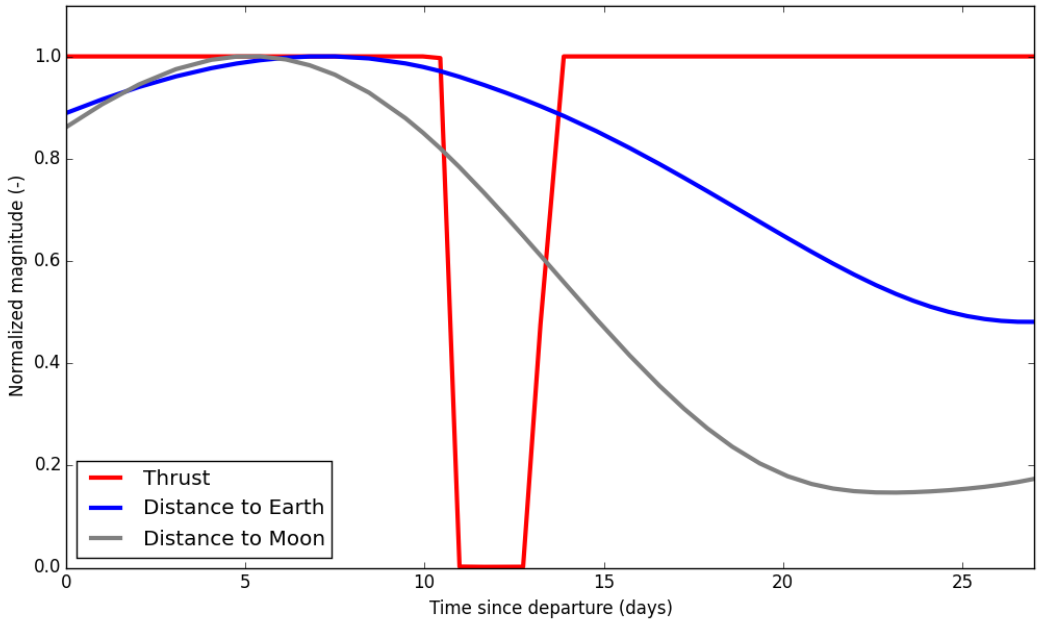


Figure 5.8: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 27-day starshade transfer.

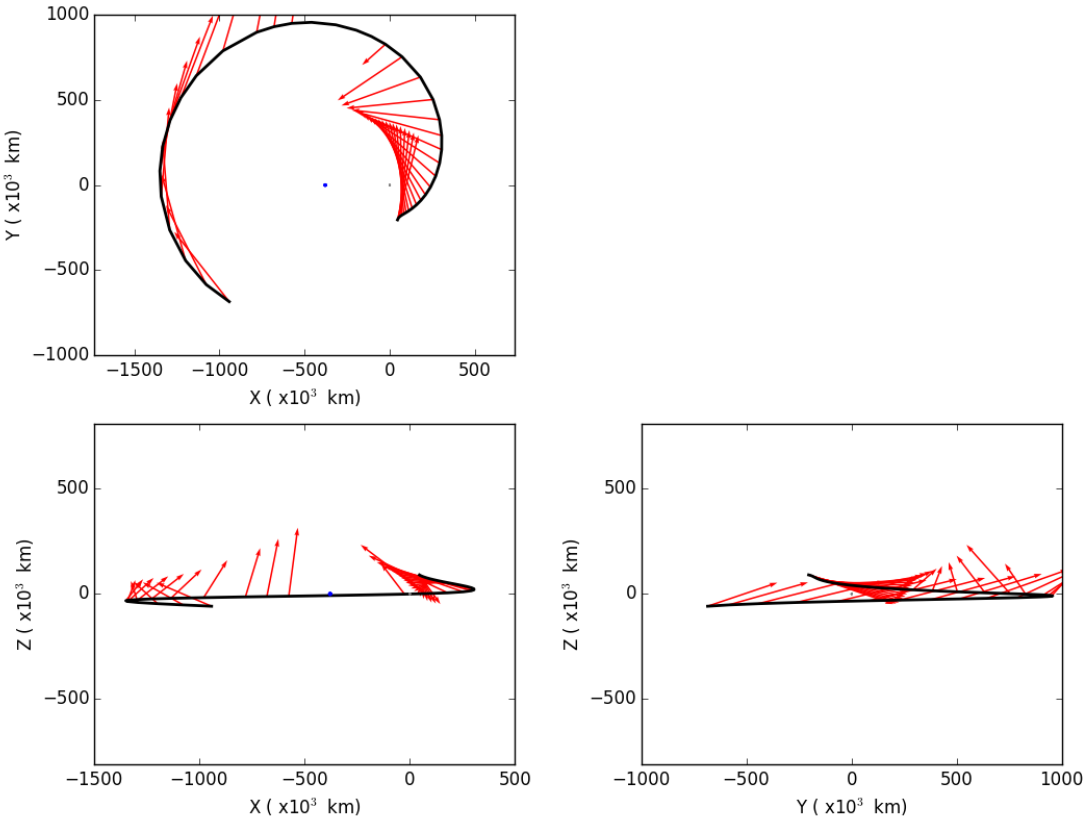


Figure 5.9: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 27-day starshade transfer. Both the Earth and the Moon are shown to scale.

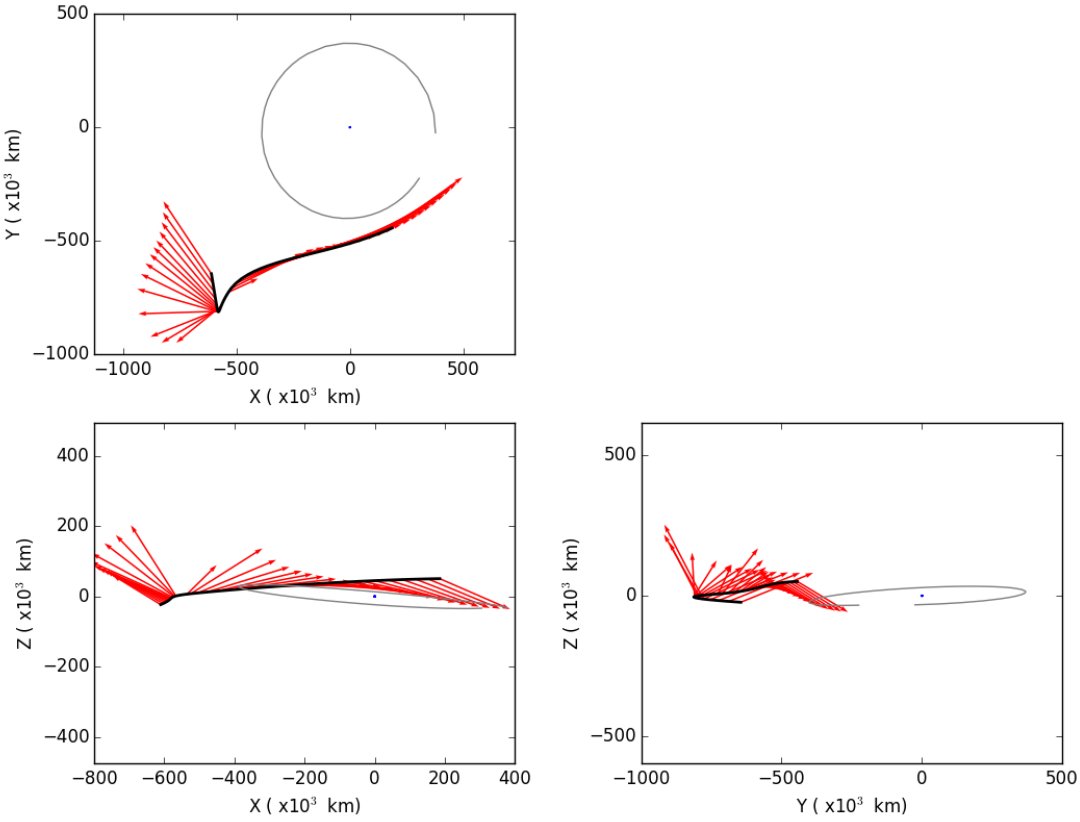


Figure 5.10: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 27-day starshade transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

5.3 Discussion of results

The main purpose of the results presented in this chapter is to illustrate that *Maverick* can consistently find solutions in a larger family. The dynamical regime that characterizes halo orbits is known to be relatively chaotic, as compared to typical two-body orbits. Despite this, *Maverick* displays the ability to find solutions for a wide range of flight times, all of which are clearly related to each other. This serves as a useful capability demonstration before more complex cases are discussed.

An additional conclusion from these results is that the Earth-Moon system appears to be unfavorable to the NWO mission. A single transfer requires 15 kg of propellant for a 500 kg vehicle, which is not dramatic by itself, but NWO aims to perform dozens of these transfers, perhaps more than 100. This is clearly infeasible. Furthermore, the results presented here rank among the best that were identified in the Earth-Moon system. Many other transfers were studied, and most proved completely infeasible. Simultaneously, comparable transfers were studied for the Sun-Earth halo region, which is characterized by similar but much slower dynamics. The Sun-Earth system proved much more attractive for NWO, with transfers requiring as little as 1 kg of propellant, or even less. Since this is an on-going study that is out of the scope of this thesis, those results will not be presented here, but from the results presented here it should be clear that using halo orbits in the Earth-Moon system for the NWO mission is unlikely to produce a feasible mission concept.

Chapter 6

DRO stabilization transfers

The transfers discussed in this chapter are between two Distant Retrograde Orbits (DROs). DROs have recently gained popularity as interesting targets, in large part due to their existence at the edge of the gravitational influence of celestial bodies. Lunar DROs have gained particular attention, for their possible application to the Asteroid Redirect Robotic Mission (ARRM) concept [64], since they do not require a large change in energy to reach from a heliocentric orbit [8]. This makes them promising targets for capturing a large mass. Furthermore, despite lunar DROs existing in a region significantly affected by the gravitational effects of the Sun, Earth, and Moon, it has been shown that these orbits can be stable for long periods of times [10]. While these orbits can be stable, it has recently been demonstrated that DROs can be found that are stable only forward or backward in time, for a given epoch [49]. This means that a DRO may be stable for a long period of time, but even so can be arrived at through a ballistic transfer that utilizes the backward unstable nature of the DRO. Chapter 7 discusses arriving directly into a forward stable DRO from LEO. In the current chapter, a transfer from an unstable DRO to a stable DRO is discussed. The goal of this chapter is not to provide an elaborate analysis of stabilizing DROs, which is ongoing research that is considered out of the scope of this thesis. Instead, it is meant as a demonstration of the feasibility of doing so, and as an initial indication of the costs involved. Additionally, it highlights some interesting characteristics of *Maverick*.

6.1 Assumptions

For the presented transfers, the gravitational forces of Earth, the Sun, and the Moon are included, using the DE430 JPL ephemeris [19]. The spacecraft has a mass of 1800 kg and a power supply of 18 kW. The propulsion system has a fixed specific impulse of 2000 s, and a jet efficiency of 60%. This serves as a rough indication of characteristics that a vehicle might have that would perform a sample return mission of a destination beyond Earth, such as Mars or an asteroid. Both the unstable and stable DRO used in this work were produced for Reference [8] and shared by

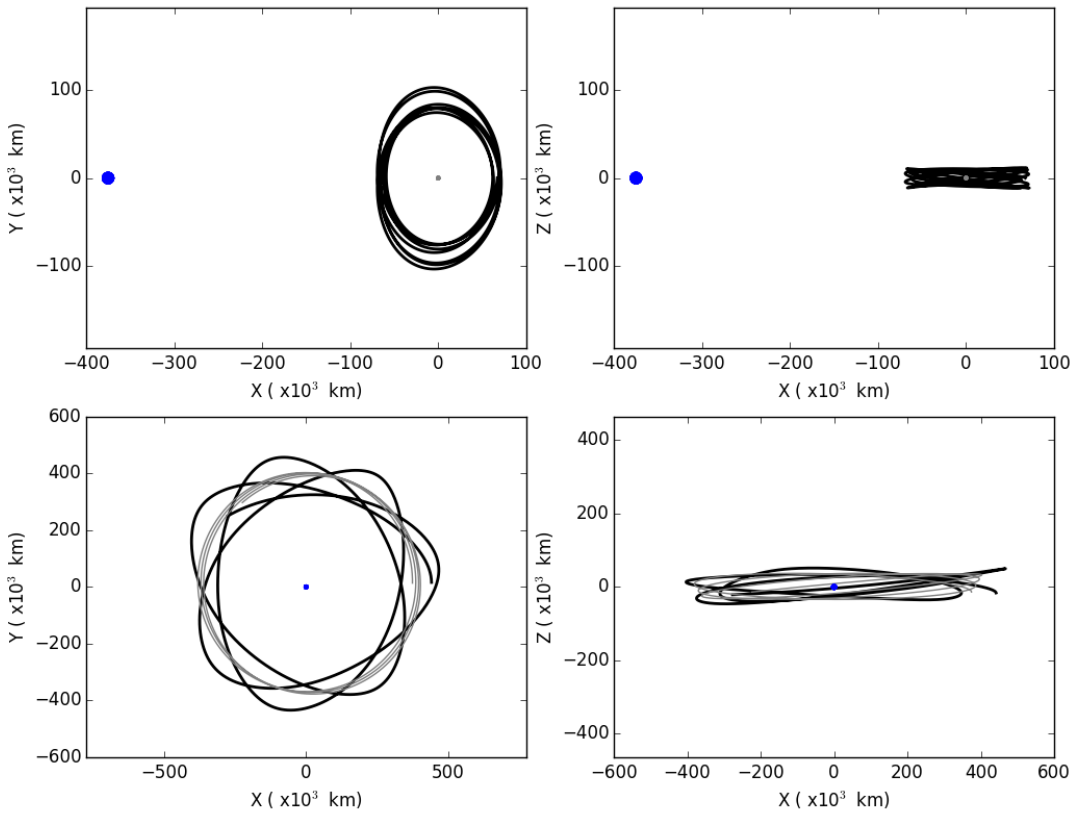


Figure 6.1: XY- and XZ-projections of the target Distant Retrograde Orbit. The upper figures show the Earth-Moon rotating frame, whereas the lower figures show the Sun-Earth rotating frame. All figures show the same 100 day propagation, starting at the arrival date used for the transfers in this chapter.

the author. The target (stable) DRO is shown in Figure 6.1, in a 100 day propagation after the arrival time used in this study. As can be seen, this DRO orbits at approximately 70 000 km from the Moon, revolving about the Moon approximately every 2 weeks, tracing out a similar (but not identical) pattern every revolution.

The initial guess for each transfer is a ballistic propagation from the departure orbit. All transfers use the same arrival time and state, which means the departure date is varied to achieve shorter times of flight, along with the departure state. To get the new departure state, a ballistic propagation of the original departure state is performed.

For those interested in reproducing these results, the Moon-centered reference state for the unstable (departure) DRO is

$$X_0 = \{746.1256604413,\ -23702.4977719188,\ -18502.3986489793,$$
$$0.655162506091504,\ 0.016451280481766,\ -0.0250772203418204\}$$

in km and km/s, at a Julian Date of 2441080.67191458. The above state can be used to reproduce all departure states through a ballistic propagation to the correct time. The Moon-centered arrival state (for all transfers) is

$$X_f = \{-13994.399297515731,\ 59360.696879827112,\ 28388.816363975184,$$
$$0.321433864382,\ 0.0322270328225,\ 0.00331006020682\}$$

in km and km/s, at a Julian Date of 2441317.50048823.

6.2 Results

Figure 6.2 presents the final mass as a function of flight time. Each of the markers represent a fully optimized low-thrust trajectory, which are spaced 2 days apart. The arrival time is always the same, so for shorter transfers the departure date is altered, along with the departure state. To get the new departure state, a ballistic propagation of the original departure state is performed. Figure 6.2 clearly shows that a minimum-time solution of approximately 7 days exists that requires 5 kg of propellant. As the time of flight increases, the propellant requirement slightly decreases (other than some slight differences in convergence between solutions), but doesn't dramatically change. Indeed, these transfers are largely the same as the 7-day transfer. Then for flight times greater than 30 days, problems occur. Some points still return the 5 kg of solution of before, but others are significantly worse.

While there are scenarios where a larger flight time can actually lead to greater propellant

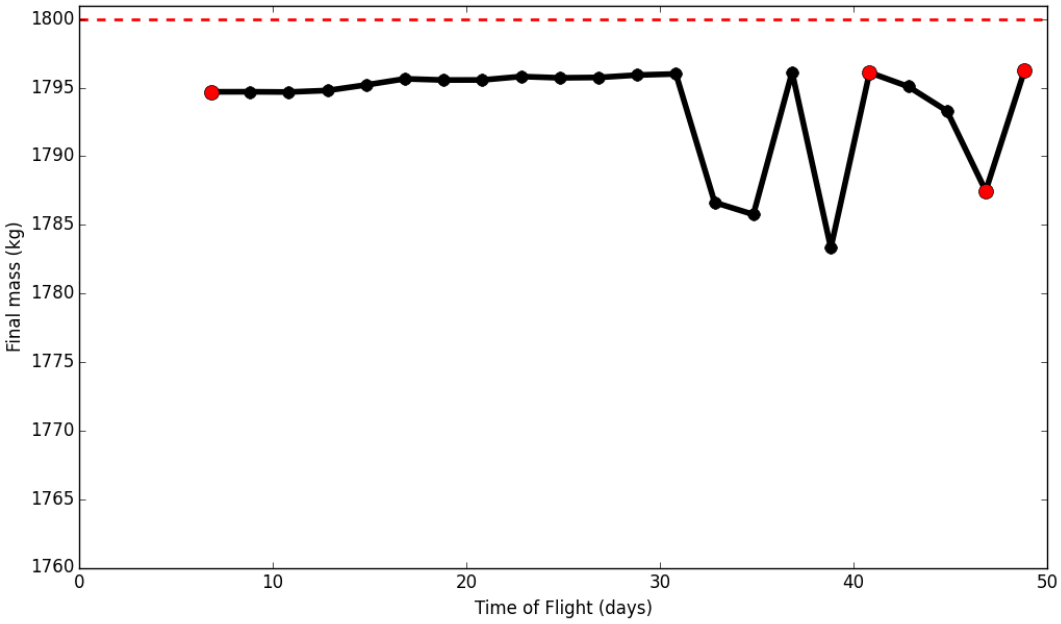


Figure 6.2: Final mass as a function of time of flight for the DRO stabilization transfers. The red dashed line indicates the initial mass, and the red marked points indicate those that are presented in more detail below.

requirements (such as when phasing is an issue), in the current setup there is clearly no reason for that to be the case. The starting points of all transfers are on the exact same ballistic orbit, so a longer transfer could just leave the engine off and take advantage of the 7-day, 5 kg transfer. Instead, the following is happening: DROs are notoriously sensitive, defined by highly chaotic motion. As *Maverick* gets larger flight times, it also gets the opportunity to attempt more complex solutions, utilizing these very sensitive dynamics. Unfortunately, in this case it never results in a solution superior to the original 5 kg solution, and instead *Maverick* gets caught in a local optimum.

Although this is unfortunate, it is not an unknown problem in low-thrust trajectory optimization, and it is typically resolved by running the same optimization problem several times, with differing initial guesses or by tweaking some of the relevant optimization variables. Ideally, some logic is applied to this 'outer loop' as well, a good example of this is the monotonic basin hopping of EMTG [17]. The only downside of this is additional runtime, or alternatively greater computational demands. In any case, the scope of this thesis was never to improve these 'outer loop' or 'global search' solving strategies, focusing instead strongly on improving the 'inner loop' solver. Figure 6.2 clearly shows that an excellent transfer exists for as little as 7 days, and that little is gained by increasing time of flight. This is not surprising, since 5 kg already represents less than 0.3% of the initial mass. If for some reason there is need to extend Figure 6.2 beyond 30 days with perfectly consistent results, it is recommended that a "global search" algorithm is implemented on top of these optimizations.

To illustrate the local optima that *Maverick* can get trapped in, several solutions are detailed below, including the 47-day transfer that requires approximately 15 kg of propellant due to having attempted to improve the solution by introducing a lunar gravity assist.

6.2.1 7-day transfer

Figures 6.3 through 6.5 detail the minimum-time 7-day transfer. Figure 6.3 shows a significant thrust arc on the final day of the transfer, with some additional thrusting on the first day. At higher convergence tolerances, the first thrust arc would likely be taller and more narrow (shorter in time), but as mentioned before the reduction in propellant would likely be insignificant. Since the trajectory spends the majority of its time coasting, this seems to suggest that shorter transfers are possible, but a 6-day transfer proved infeasible. It is likely that the minor thrust arc at the start of the transfer, which occurs almost exactly opposite of the target state with respect to the Moon as clearly shown in Figure 6.4, is critical to enabling this low-cost transfer. Starting the transfer any later likely rules out this type of transfer, resulting in much more costly (and infeasible) alternatives.

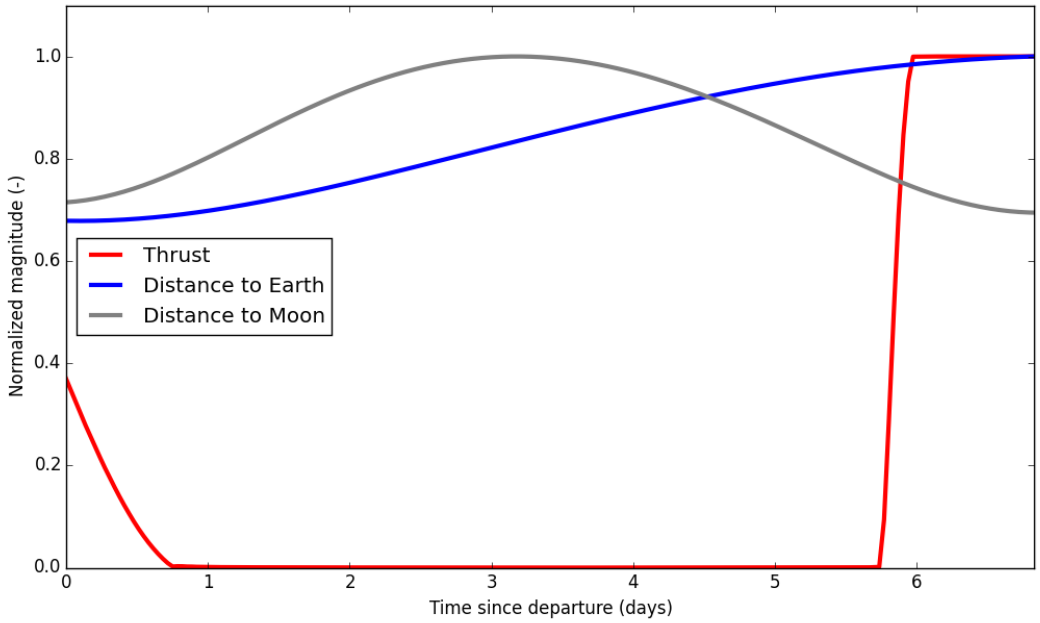


Figure 6.3: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 7-day DRO stabilization transfer.

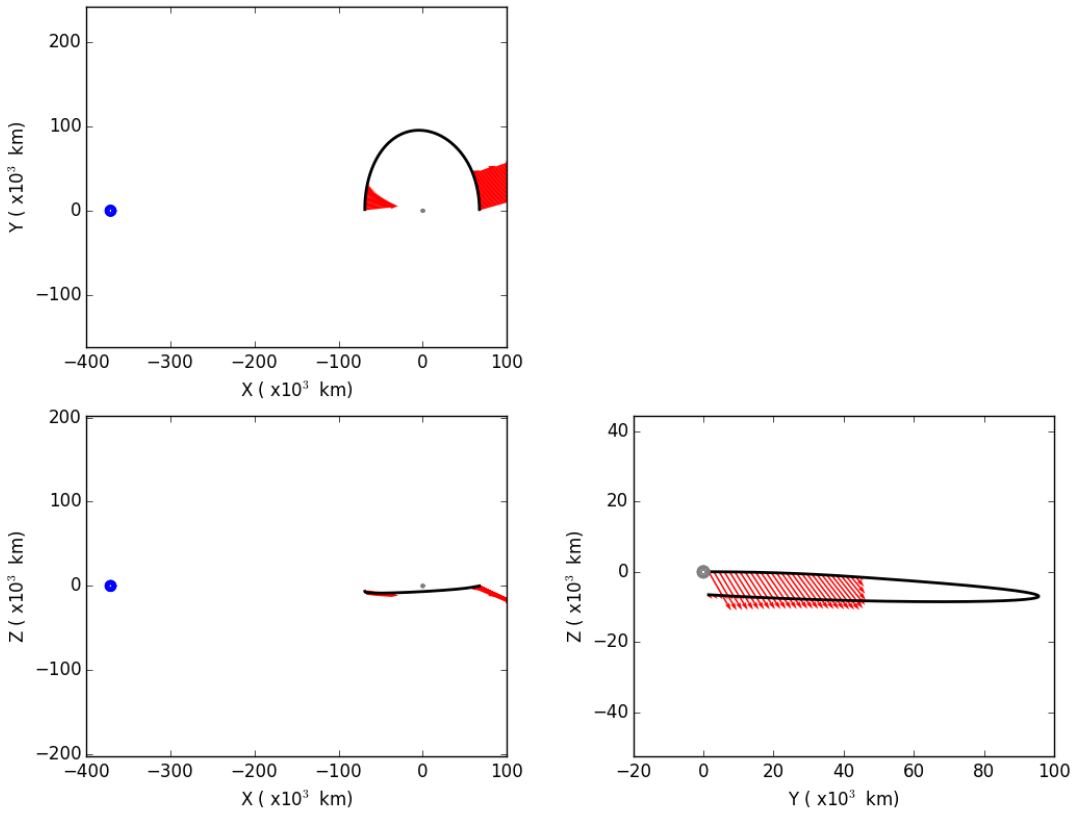


Figure 6.4: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 7-day DRO stabilization transfer. Both the Earth and the Moon are shown to scale.

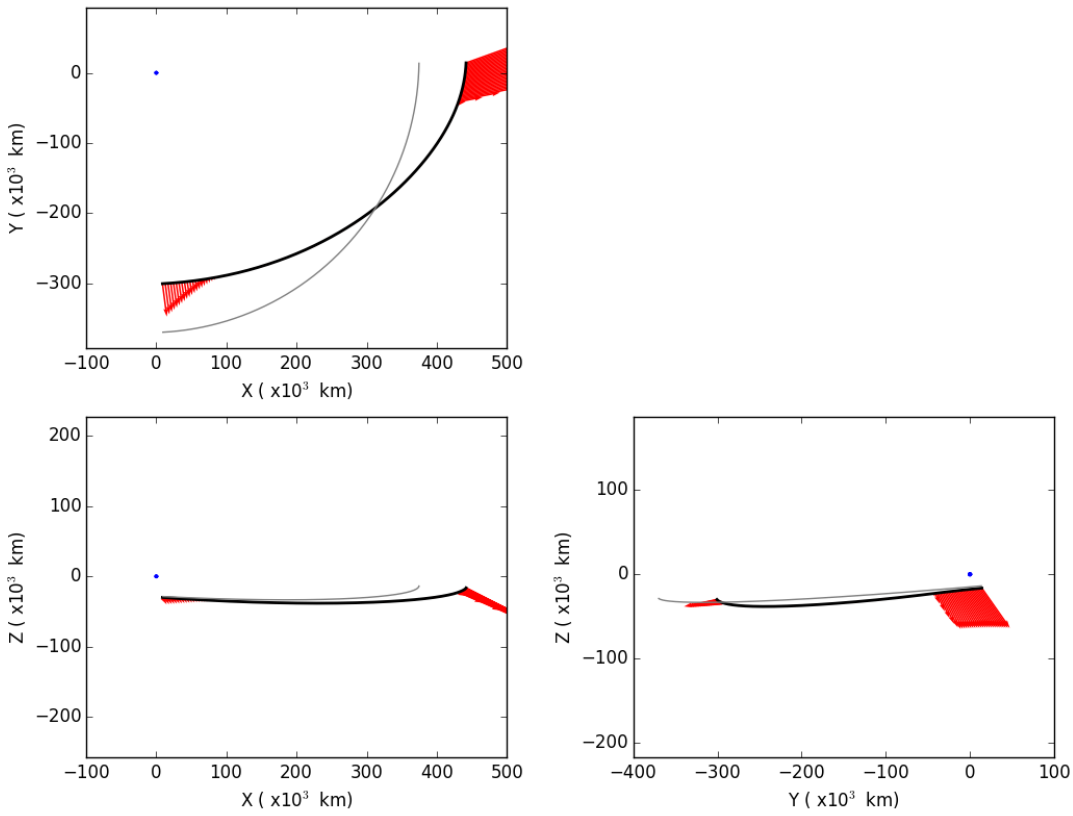


Figure 6.5: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 7-day DRO stablization transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

6.2.2 41-day transfer

Figures 6.6 through 6.8 detail the 41-day transfer. These figures clearly show this transfer is extremely similar to the 7-day transfer. The main thrust arc is still right at the very last day, and there is a small thrust arc remaining 7-days earlier, but it is much less defined now, and is instead split across 4 different small arcs. At a greater optimization tolerance, these would likely be shorter in time and greater in magnitude, perhaps even merging into a single thrust arc at one of these locations, but again at an insignificant gain. Ultimately, by having additional time of flight this transfer ends up with slightly less consumed propellant, as shown in Figure 6.2, but the difference is minimal.

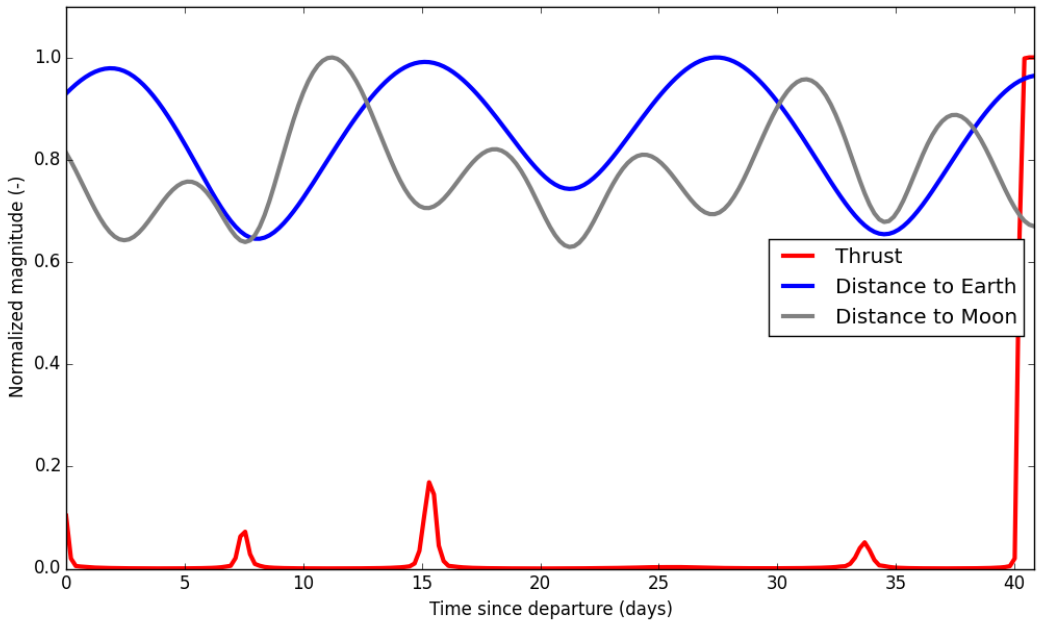


Figure 6.6: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 41-day DRO stabilization transfer.

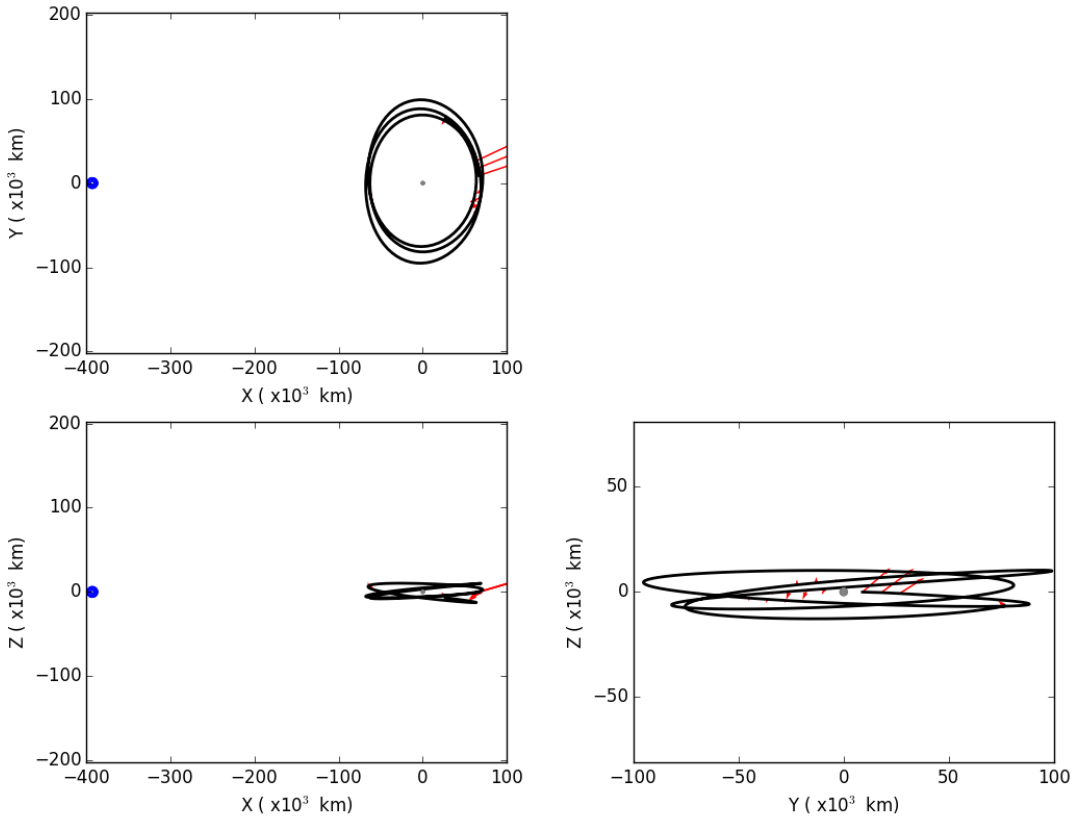


Figure 6.7: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 41-day DRO stabilization transfer. Both the Earth and the Moon are shown to scale.

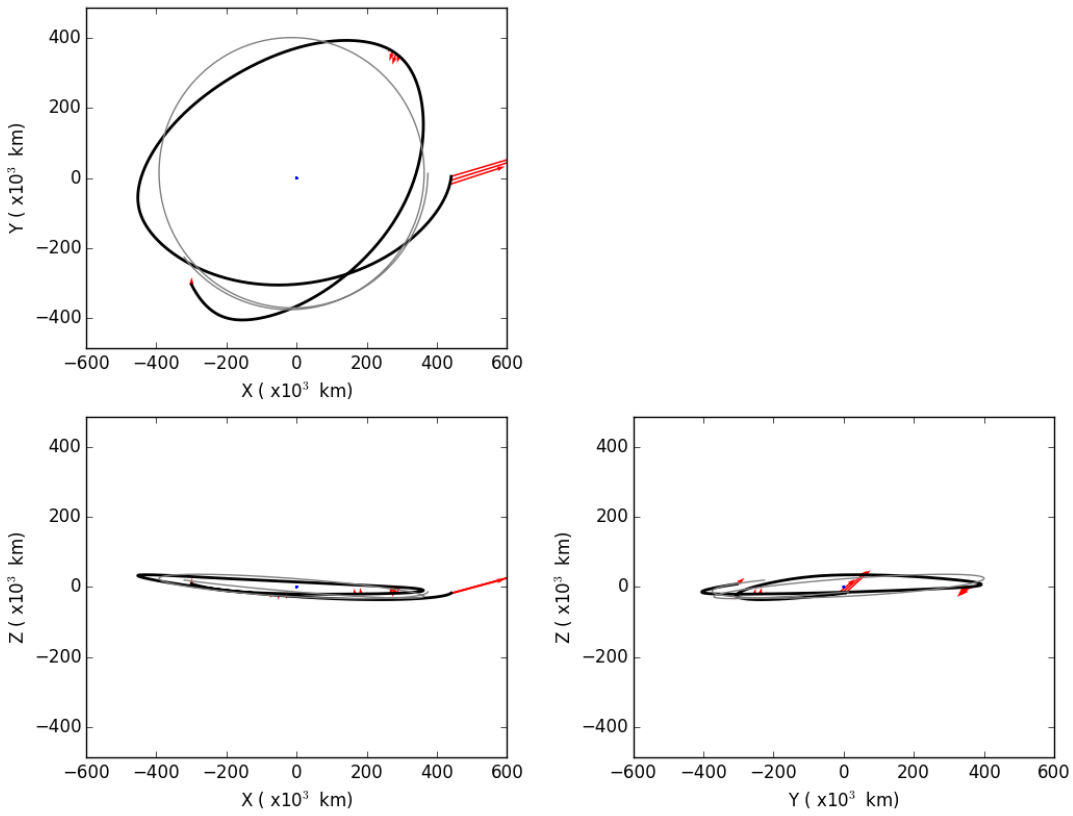


Figure 6.8: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 41-day DRO stablization transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

6.2.3 47-day transfer

Figures 6.9 through 6.11 detail the 47-day transfer, which has clearly gotten trapped in a local optimum that is part of a very different solution family. The optimizer sees an opportunity to introduce a lunar gravity assist early in the transfer, with just minimal thrusting. The optimizer turns this into a powered fly-by at a lunar altitude just under 5000 km. Figure 6.9 clearly shows how little thrusting is required before the lunar altitude plummets. Just a few hours of thrust from an electric propulsion system is enough to reduce the lunar close approach distance from approximately 60 000 km to less than 5000 km. First of all, this is a testament to how sensitive this DRO is. Second of all, it should be stressed that *Maverick* is in no way directed to introduce this gravity assist, or to attempt to do so. Merely the presence of the Moon in the force model indicates to *Maverick* that it could be used to accomplish part of the maneuver, and it successfully introduces this low altitude gravity assist autonomously. This is a very rare quality for trajectory optimization

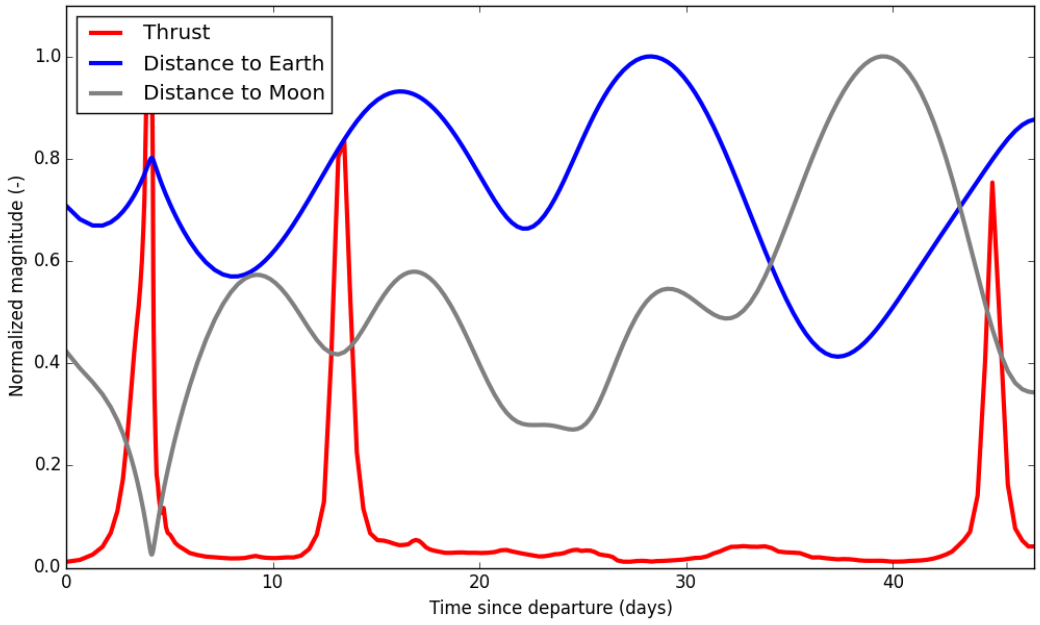


Figure 6.9: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 47-day DRO stabilization transfer.

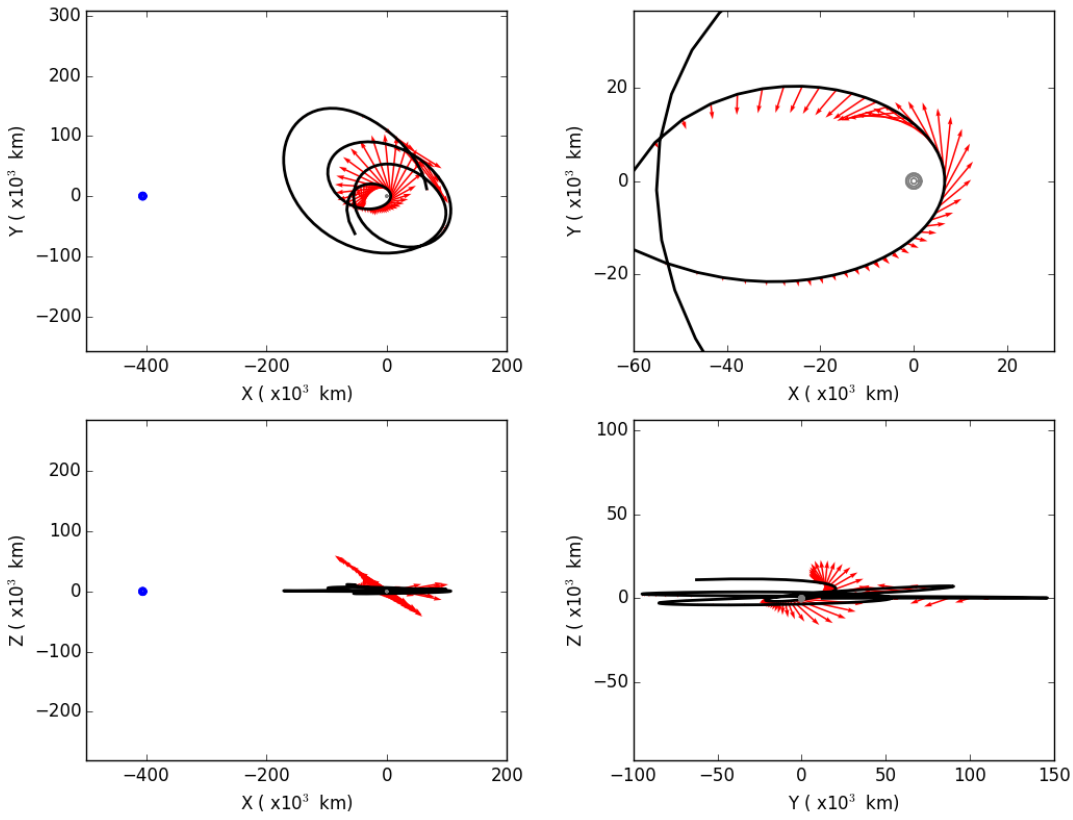


Figure 6.10: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 47-day DRO stabilization transfer. Both the Earth and the Moon are shown to scale.

tools, and a very useful feature. That being said, in this example it leads to a local optimum that *Maverick* is unable to recover from, which is unfortunate. However, in Chapter 7 examples are provided where *Maverick* uses gravity assists to much greater effect. As mentioned at the start of this chapter, the occurrence of local optima can be avoided through known ways. Considering that, it is preferable to have an optimizer that can introduce gravity assists in an attempt to improve the solution, as it will ultimately aid in the variety of solutions that are considered.

As a final note, the thrust history in Figure 6.9 is already close to bang-bang, but there is still clearly some small level of thrust throughout the transfer, and two of the peaks don't quite scale all the way to maximum thrust. As usual, with a higher convergence tolerance this could

be avoided, but it would present an insifignificant degree of savings in propellant, which is why *Maverick* ended the convergence where it did.

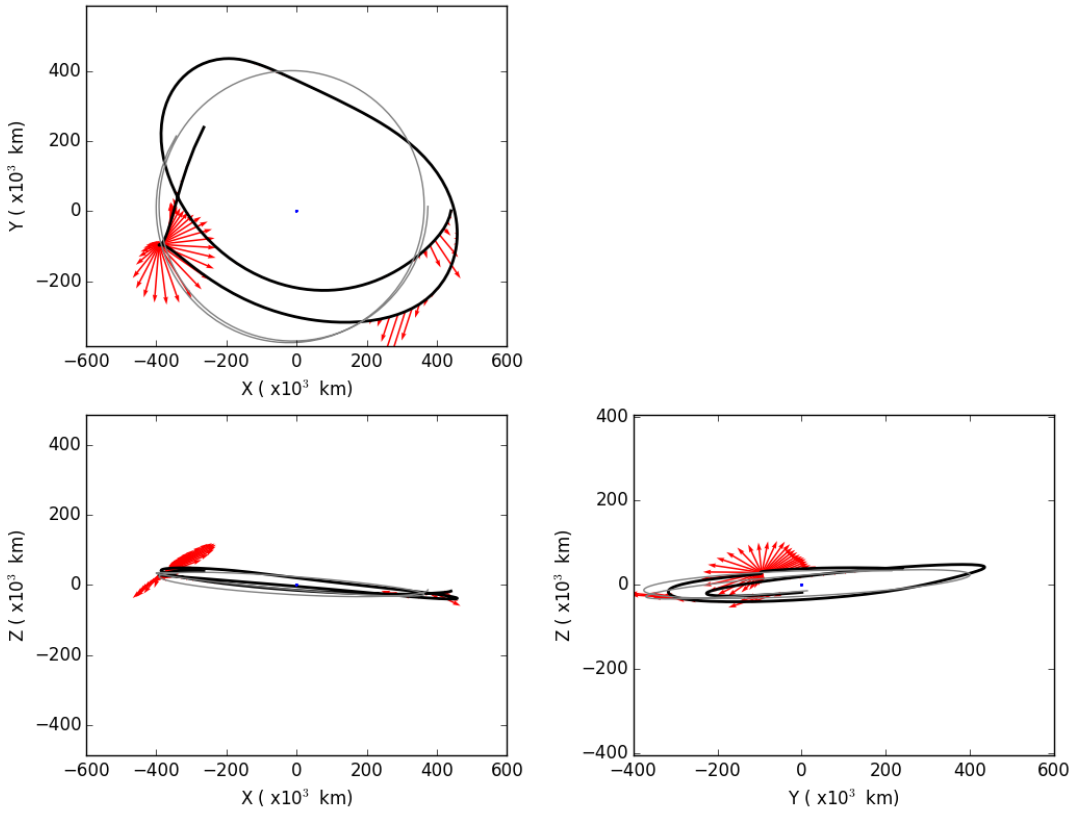


Figure 6.11: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 47-day DRO stablization transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

6.2.4 49-day transfer

Figures 6.12 through 6.14 detail the 49-day transfer, which is mainly included to show that *Maverick* can recover to the superior solution even for longer times of flight. Clearly, this transfer is very similar to the 7-day and 41-day transfers, only 8 days longer.

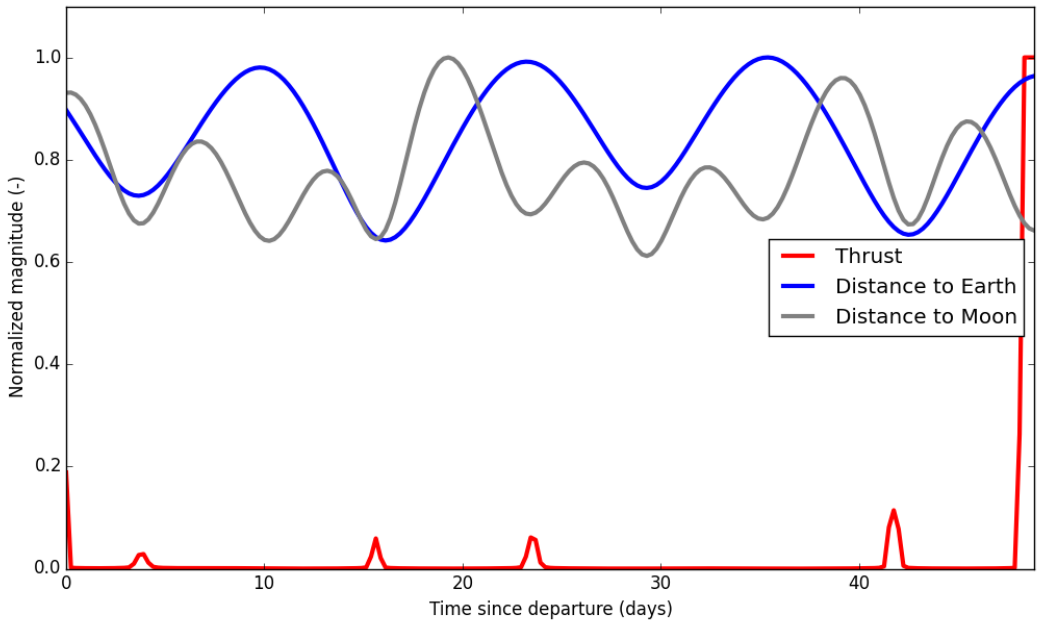


Figure 6.12: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 49-day DRO stabilization transfer.

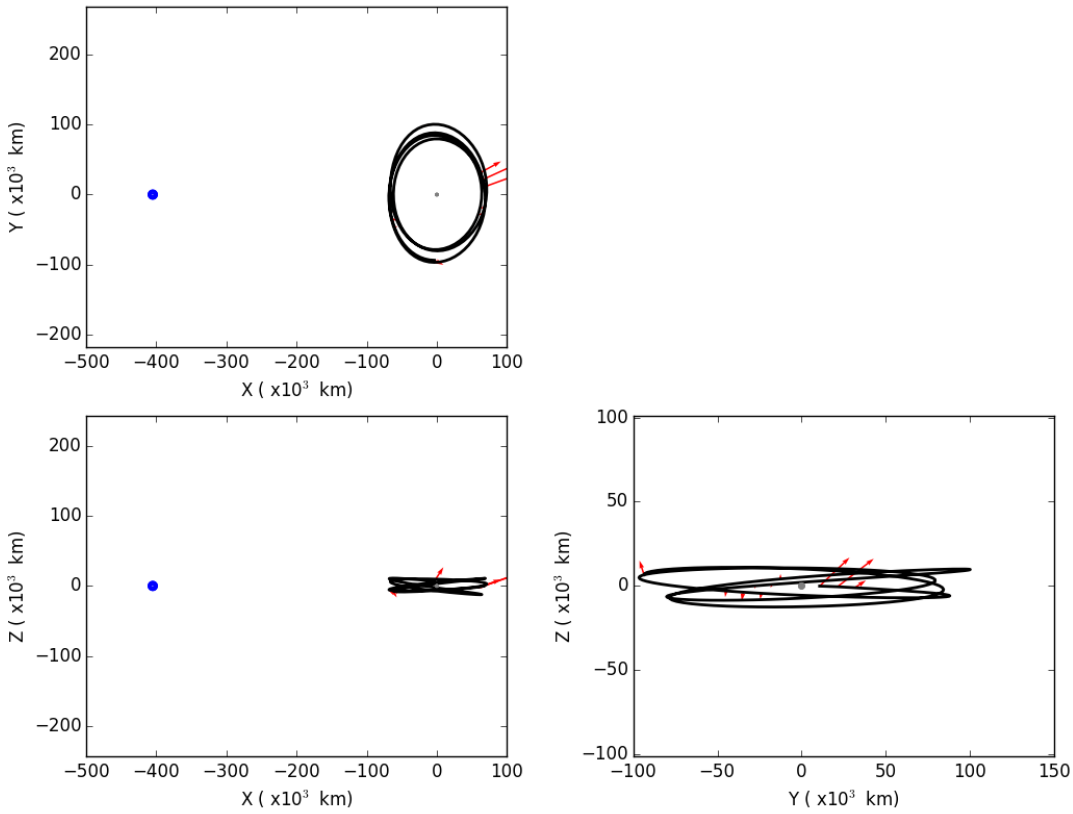


Figure 6.13: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 49-day DRO stabilization transfer. Both the Earth and the Moon are shown to scale.

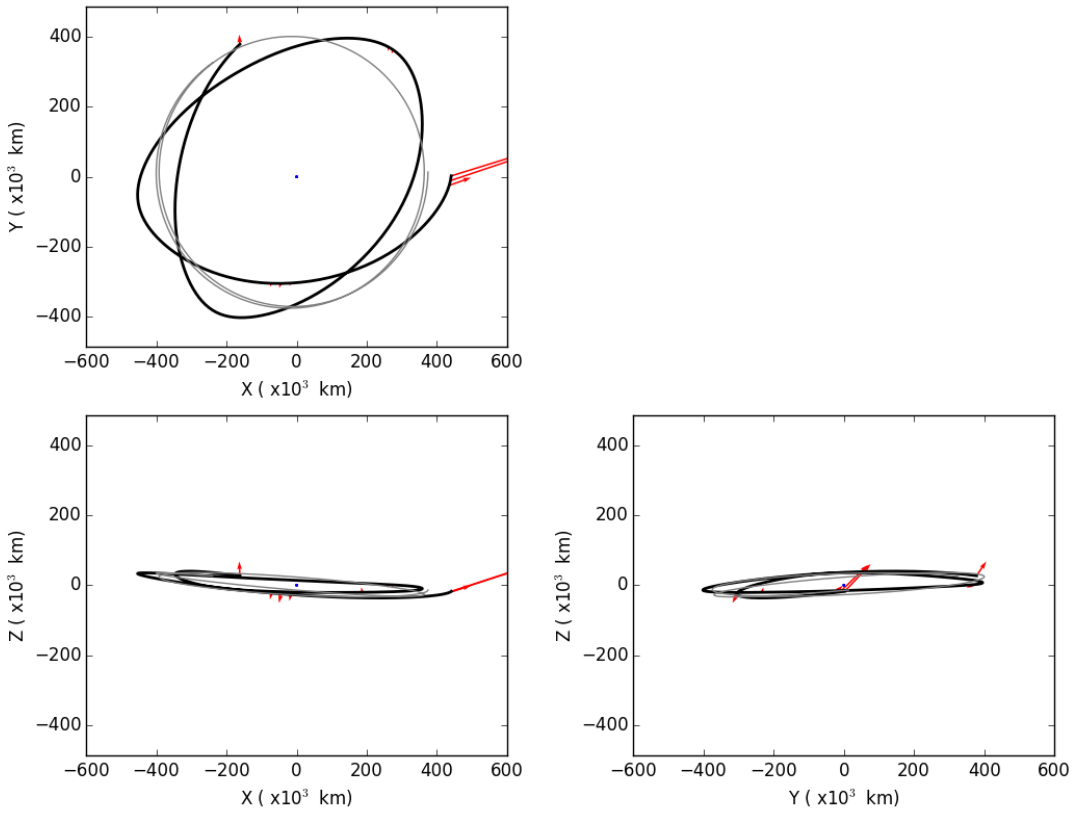


Figure 6.14: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 49-day DRO stablization transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

6.3 Discussion of results

The results presented here show a number of elements about low-thrust, low-energy transfers that return in the next chapter, and provide an indication of the costs involved in stabilizing a DRO. First of all, it is shown that stabilizing a DRO can be done at an extremely low cost in time and propellant. In the best case, merely 7 days and less than 0.3% of the initial mass is required to stabilize the orbit in the example presented here. It seems that significantly superior solutions do not exist for greater flight times, which is not entirely surprising considering the low cost of the minimum-time transfer.

Second of all, the 47-day transfer illustrates *Maverick*'s ability to autonomously introduce gravity assists as part of the optimization process. This is an extremely rare capability for trajectory optimization tools, and speaks to the complexity of solutions that can be found with collocation methods.

Finally, and unfortunately, the same highly chaotic dynamics that make DROs appealing targets also complicate trajectory optimization problems involving them. In this case, the gravity assists *Maverick* introduced never manage to outperform the simpler solution family that derives from the 7-day transfer. This comes as no surprise, considering the unstable nature of the dynamics in this region, and when necessary can easily be managed by the use of known 'global search' methods at no cost other than additional runtime.

Chapter 7

LEO to DRO transfers

The previous Chapter discussed stabilizing a DRO after arriving on one. This Chapter focuses on transferring to a DRO straight from a launch from low Earth orbit. As mentioned in the previous Chapter, Reference [49] identifies DROs that are, for a given epoch, stable forward in time but unstable backwards in time. Reference [49] discusses low-energy transfers that capitalize on this opportunity, utilizing gravitational accelerations from the Sun, Earth, and Moon to enable trajectories that require no maneuvers between the Earth departure and the DRO arrival, and in fact, arrive at the DRO without requiring an insertion maneuver. These low-energy transfers have a relatively low propellant requirement, but require a long time of flight, typically on the order of 4-6 months. While this is not necessarily a barrier to their utilization, it is nonetheless interesting to investigate opportunities to lower the time of flight. One such opportunity is presented by incorporating Solar Electric Propulsion (SEP) to improve some aspect of these transfers, but as can be seen from the ballistic transfers presented in Reference [49], these trajectories experience dramatically different dynamical regimes throughout their transfer, and by utilizing low-energy dynamics are expected to be very sensitive to small perturbations. As discussed in Section 3.1, collocation methods are expected to be especially appropriate for problems this sensitive in nature, and this chapter illustrates that by successfully designing SEP transfers for a range of flight times. Section 7.1 provides the relevant assumptions and constraints for these transfers. Section 7.2 discusses a wide range of results in great detail. Section 7.3 builds on this to present results for a lower power level, and Section 7.4 concludes the Chapter.

An early version of the work discussed in this Chapter was first presented in Reference [31]. However, with many improvements to *Maverick* since then, the results presented there are entirely obsolete, and will not be repeated here.

7.1 Assumptions

For the presented transfers, the gravitational forces of Earth, the Sun, and the Moon are included, using the DE430 JPL ephemeris [19]. The spacecraft has a mass of 1800 kg and a power supply of 18 kW. The propulsion system has a fixed specific impulse of 2000 s, and a jet efficiency of 60%. These parameters are typical for modern SEP vehicles. As a baseline, a ballistic 6 month transfer from Reference [49] is used. All transfers must have a launch energy that is the same or lower as this transfer, and they all target an arrival on the same DRO at the same time. To vary flight time, the departure date is varied. The departure state is constrained to be at the same altitude on Earth, and with a departure velocity identical to (or lower than) the departure velocity. Although the departure altitude is constrained, this altitude may occur anywhere on Earth, allowing the spacecraft to depart in any direction. Finally, the spacecraft is not allowed to thrust for 2 days after launch, which serves as a period to check out the vehicle and determine the actual trajectory after launch.

The initial guess for the ballistic transfer is the known ballistic transfer. For subsequent (shorter flight time) solutions, in the presented results the initial guess is the most recent converged solution. Runs were also performed that always use the original ballistic transfer as the initial guess, regardless of how short the transfer, which produced similar results as those presented here although typically at longer runtimes.

For those interested in reproducing these results, the Moon-centered departure state for the ballistic, 187-day transfer is

$$X_0 = \{280408.00080112100,\ -248944.94497254601,\ -79700.692212164606,$$
$$-0.95803600525177302,\ -9.3189009639297797,\ -3.3028031354517600\}$$

in km and km/s, at a Julian Date of 2436812.92130433. The Moon-centered arrival state (for all transfers) is

$$X_f = \{105000.92411397818,\ -1590.5185005757955,\ -11838.606503861754,$$
$$-0.0043980475188152013,\ -0.042478207536745256,\ -0.055634728121663417\}$$

in km and km/s, at a Julian Date of 2437000.30990386.

7.1.1 Accuracy of trajectories

It has been repeatedly stated that low-energy transfers in general, and DROs in particular, represent highly sensitive dynamical environments, where small discrepancies can lead to very different outcomes, also known as chaotic motion. This can be shown very clearly with the 187-day ballistic reference orbit, from Reference [49]. If the corresponding state in LEO is taken and propagated with TurboProp's Dormand-Prince 8^{th} order numerical integrator [33] for the correct transfer time, the final state will be on the desired DRO. However, TurboProp allows the user to specify how many output states to provide. While this does affect the number of integration steps that are taken, since TurboProp must take care to stop and record the state at the points requested by the user, the impact on the values coming out of the propagation should be negligible, with up to 10-14 digits accuracy, the remainder lost due to numerical precision. TurboProp is a mature and reliable piece of software and the above indeed holds for the vast majority of orbit propagations. However, for these LEO-DRO transfers, if the number of output states is altered, the final state in the DRO differs up to the 3^{rd} or 4^{th} digit. The same is observed with an independent numerical integrator developed by the author. This is a testament to how sensitive these transfers are. It is not a real problem for the work performed here, since at least the first 2-3 digits consistently agree. Furthermore, if the trajectory were to be flown in real life there would be all sorts of errors that would dwarf the error in the numerical integration, for example the error caused by the uncertainty in the launch vehicle performance (the launch vehicle injection error). The solution to this is to re-optimize the trajectory as it is flown to match the exact trajectory that the spacecraft is on.

However, when comparing the outcome from the 12^{th} order numerical integration with the Gauss-Lobatto scheme to other integrators like a Dormand-Prince 8^{th} order numerical integrator, it should be kept in mind that only the first few digits can be expected to agree.

7.1.2 Additional modifications to problem formulation

A few additional constraints to the general ones, described in Chapter 3, are added to help convergence for the complex LEO-DRO transfers. All these constraints are implemented into *Maverick* with the appropriate scaling factors and analytical derivatives. They do not significantly affect the sparsity or runtime balances described in Chapters 3 and 4.

The first set is alluded to in Section 7.1, constraining departure altitude to be constant, but only in magnitude and not direction. Similarly, the departure velocity has an upper bound equal to the departure velocity of the ballistic transfer, but an open lower bound. In reality, *Maverick* is allowed to increase the departure velocity by up to 0.01% of the original value (which equates to approximately 0.1 m/s in this study), to aid convergence. The direction for the velocity is kept completely free with this constraint.

An addition to that set of constraints is one that forces the dot product of the departure velocity and radius vector (with respect to the Earth) to be zero or greater. This ensures that the spacecraft departs at (or after) perigee. If this constraint is not in place, *Maverick* will often attempt to launch inward, passing through the Earth's surface. Since the first 2 days are an enforced coast, the point after passing through the Earth could be taken as the new departure date, which would only be a few minutes later than the intended departure time. However, it turns out that *Maverick* would frequently send the spacecraft straight through the Earth's core. At this distance, the small errors that are normally considered acceptable numerical tolerances for the defect constraints have a noticeable effect on the trajectory. As a result, when the spacecraft emerges from the other side of the Earth, the launch energy has changed! This is clearly 'cheating', besides being an absurd solution that requires a significant effort from the mesh refinement process to produce an accurate transcription. To avoid this, *Maverick* is required to depart the Earth

directly, without any detours of the undoubtedly fascinating terrestrial core, by enforcing launch conditions to be at or after perigee.

Along similar lines, *Maverick* needs to be encouraged to keep a distance from the Moon for these LEO-DRO transfers. As is discussed later in this chapter, *Maverick* is able to autonomously introduce (powered) gravity assists. This is very convenient, but in many cases the optimal gravity assist is as close to the fly-by body as possible. In those situations *Maverick* would fly straight through the lunar core to achieve an enormous shift in the trajectory. In some cases, it does this mere hours after performing a powered gravity assist through the Earth's core, for an impressive but physically irrelevant optimal solution. The downside of using a minimum altitude constraint is that, for a gradient based optimizer like IPOPT, encountering the Moon at some altitude can seem like a solid wall. In reality, it could actually move to the far side of the Moon, and again be feasible. There is no perfect answer to this problem, but in this study the cost function is altered, instead of using a constraint. The cost function $J(\bar{X})$ that computes propellant (as described in Section 3.5.5) is modified as follows:

$$J_{mod}(\bar{X}) = J(\bar{X}) + \sum_{i=0}^{n} \frac{C_J}{R_{i,L}^G} \tag{7.1}$$

where C_J is a constant, and $R_{i,L}$ is the lunar radius of segment i, raised to the power G. For most results, $C_J = 0.01$ and $G = 4$ is used, which enables *Maverick* to perform very close lunar gravity assists, but never has it violating the surface. The advantage of this formulation is that, even at moderately close distances to the Moon, these additional terms to the cost (and their derivatives) are entirely insignificant and hence invisible to the optimizer. It is only once the optimizer zeros in on a specific gravity assist that these terms come into play and serve as a counterweight to the ever-decreasing lunar altitude. This avoids the 'brick wall' effect of a constraint described above. With that said, no extensive testing has been performed to compare the implementation of this term as independent constraints versus the additions to the cost function described here. The addition to the cost function was implemented, proved satisfactory, and was then used for the remainder of this work. It should also be noted that once powered gravity assists of the lunar core are eliminated,

the powered gravity assists of Earth's core also stop occurring, since they would simply result in the spacecraft shooting far away from the Moon when not balanced with a similar gravity assist of the lunar core.

7.2 Results

The LEO to DRO scenario described in the previous sections is optimized for a range of flight times, maximizing final mass for each case. Figure 7.1 presents the final mass as a function of flight time. Figure 7.2 presents the corresponding launch energies as a function of time of flight. Each of the markers represent a fully optimized low-thrust trajectory, which are spaced approximately 4.5 days apart. The first thing to note is that the flight time can be significantly reduced, down to 18 days, or 10% of the original flight time. The cost of this is up to 75 kg of propellant, or approximately 4% of the total spacecraft mass. Perhaps more attractively, the time of flight can

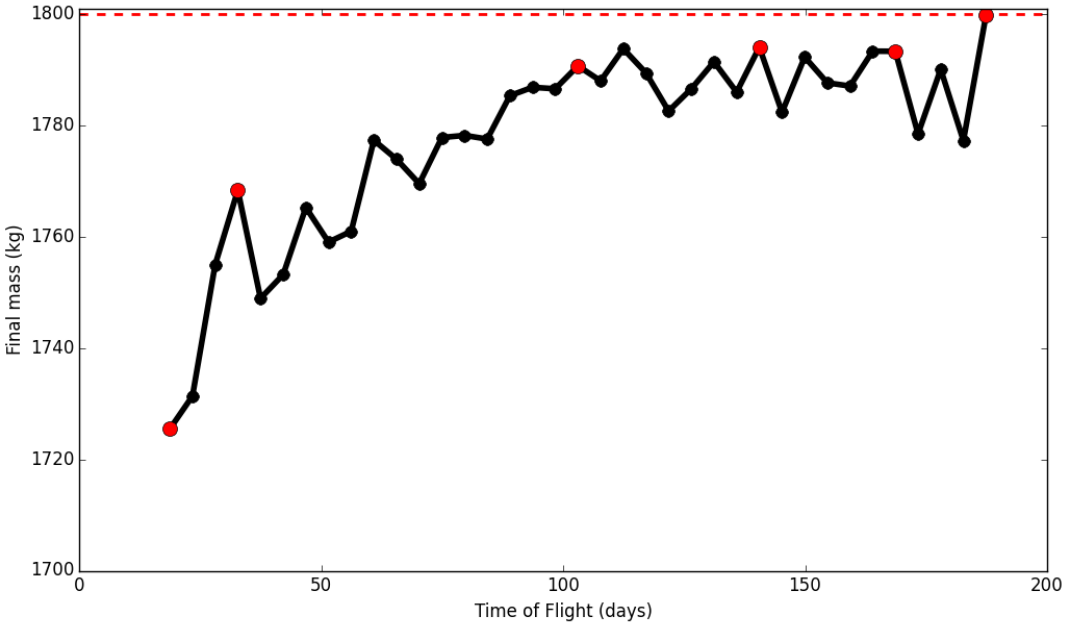


Figure 7.1: Final mass as a function of time of flight for the LEO-DRO transfers. The red dashed line indicates the initial mass, and the red marked points indicate those that are presented in more detail below.

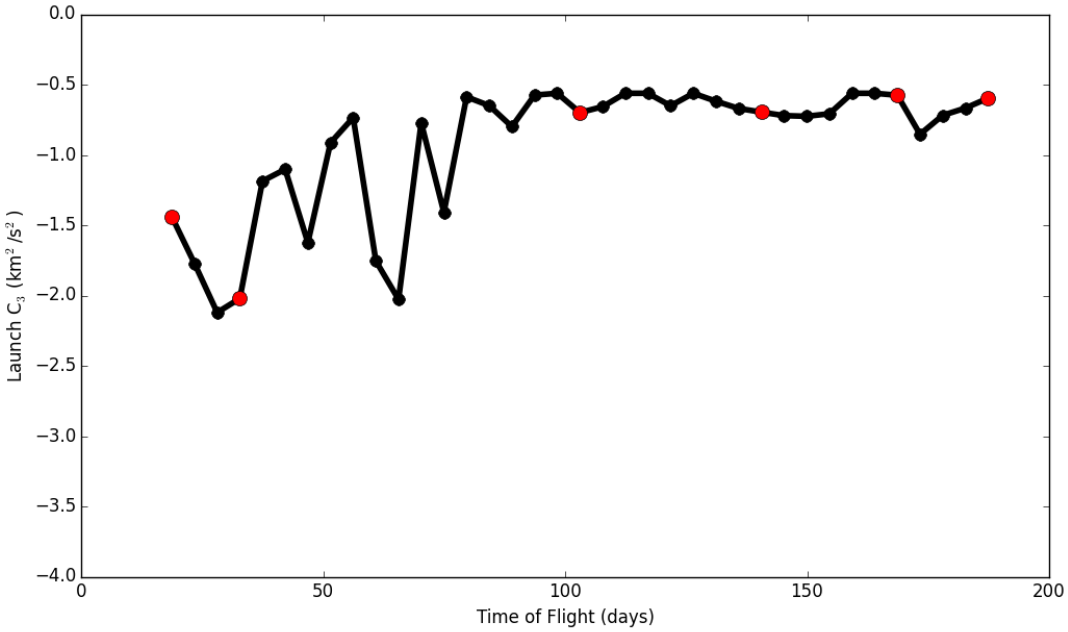


Figure 7.2: Launch energy as a function of time of flight for the LEO-DRO transfers. The red dashed line indicates the initial mass, and the red marked points indicate those that are presented in more detail below.

be reduced to 32 days, a reduction of approximately 83%, for merely 32 kg of propellant, less than 2% of the total spacecraft mass. Depending on the value of a reduced time of flight, this could be a very reasonable propellant cost. It should also be noted that the launch energy has decreased for these shorter flight times, which could account for part of the additional cost in SEP propellant, further reducing the cost of these shorter transfers.

Another interesting outcome is that the flight time can be reduced to as little as 103 days for approximately 9 kg of propellant, approximately 0.5% of the total spacecraft mass. This is nearly a free reduction of flight time by 45%.

Altogether, the solutions reasonably approximate a Pareto front [14], where the 32-day transfer could be identified as the Pareto optimum. It should nonetheless be noted how scattered neighboring solutions are. For example, the 168-day transfer requires only 7 kg of propellant, but when that time of flight is increased by just 4.5 days the cost is 22 kg, roughly three times greater. This

is reminiscent of the local optima that are found in Chapter 6 for flight times greater than 30 days. It should be stressed that, unlike the DRO stabilization transfers, for these LEO-DRO transfers a longer time of flight does not guarantee a cost that is smaller than or equal to a cost found for a shorter time of flight. For the DRO stabilization transfers, the spacecraft could coast and then start the shorter, cheaper transfer when a certain amount of time had passed. For the LEO-DRO transfers the spacecraft is forced to launch at the initial time, which locks in the relative positions of the Moon and the Sun. If either of these is not favorably aligned, this may eliminate certain types of transfers. As a result, longer flight times may actually require more propellant. Nonetheless, it is likely that at least some of the valleys in Figure 7.1 are merely local optima, but it should be expected that at least some of these variations are the result of the actual dynamics involved. It is impossible to completely test this statement, since it is impossible to know the global optimum, but to eliminate the most dramatic local optima, the results presented here are gathered from a set of different optimization runs that used slightly different scaling parameters and initial guesses. To better approximate the global optimum the use of a thorough 'global' search strategy could be applied, as discussed at the beginning of Chapter 3, but this is considered out of the scope for this work. Similarly, attempting to solve a transfer for a fixed time of flight for a range of departure dates, which would vary the relative positions of the Sun, Earth, and Moon, may smooth out the curve shown in Figure 7.1, but is not attempted due to the significant computational effort required (at least an order of magnitude increase) for only a minor improvement to the Pareto front.

Each of the transfers captured in Figure 7.1, in particular the longer ones, is a very complex trajectory. Many of them involve one or even several lunar gravity assists that are introduced completely autonomously by *Maverick*. Some of the more interesting solutions are detailed below, which also underlines that even though two solutions may appear similar in Figures 7.1 and 7.2, their trajectories may actually be significantly different. This further underlines the chaotic nature of this solution space that contributes to the uneven nature of the mass and C3 curves.

7.2.1 187-day transfer

Figures 7.3 through 7.5 detail the ballistic 187-day LEO-DRO transfer. This transfer is presented in Reference [49], but is fed back into *Maverick* to confirm it is able to identify the same ballistic transfer as an optimal solution. If scaling parameters or tolerances were set particularly poorly, or some error was present in the code, this would not be the case, as such it serves as a good test to confirm proper functioning. The figures clearly show a large arc into heliocentric space, where the Sun's gravity pulls up the perigee, and then a complex series of interactions with the Moon to enter the DRO. The closest lunar approach is just under 35 000 km for the ballistic transfer. As the following trajectories show, the interactions with the Moon differ noticeably for the shorter time of flight solutions that *Maverick* finds. In some instances, *Maverick* is able to find solutions that more closely resemble the ballistic transfer's lunar interactions (but of course with a lower time of flight), but these consistently under-perform with respect to the more unique

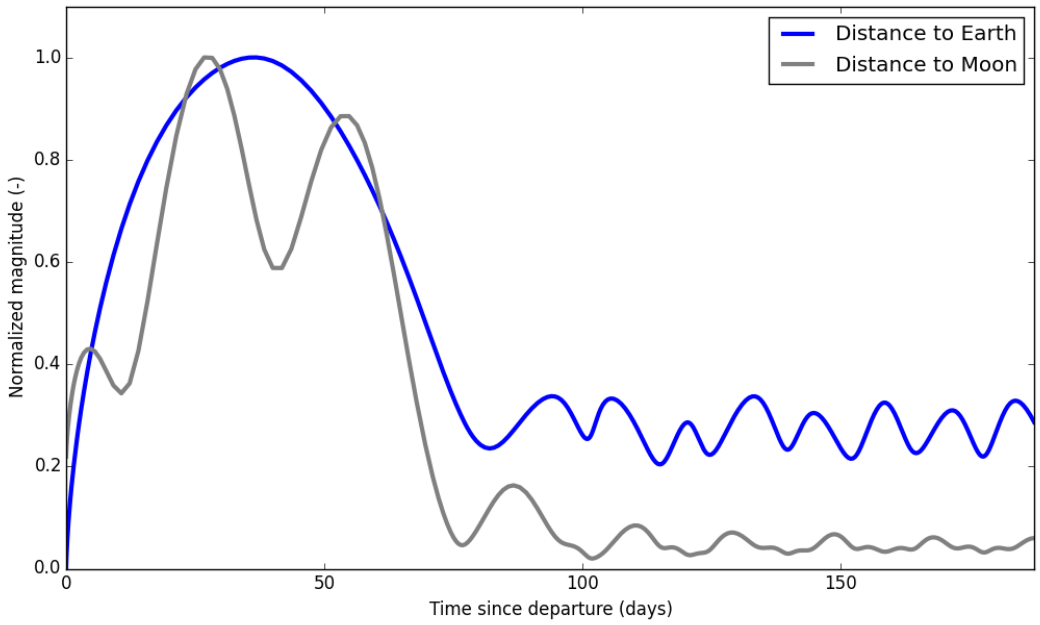


Figure 7.3: Normalized representation of radial distance to the Earth and Moon as a function of time of flight for the ballistic 187-day LEO-DRO transfer.

solutions. A great example of this is the 160-day solution presented in Reference [31], one of the early results in the work leading up to this dissertation, that clearly displays a lot of similarity to the ballistic transfer, but requires significantly more propellant than the results presented here, which are all found with the more advanced versions of *Maverick*.

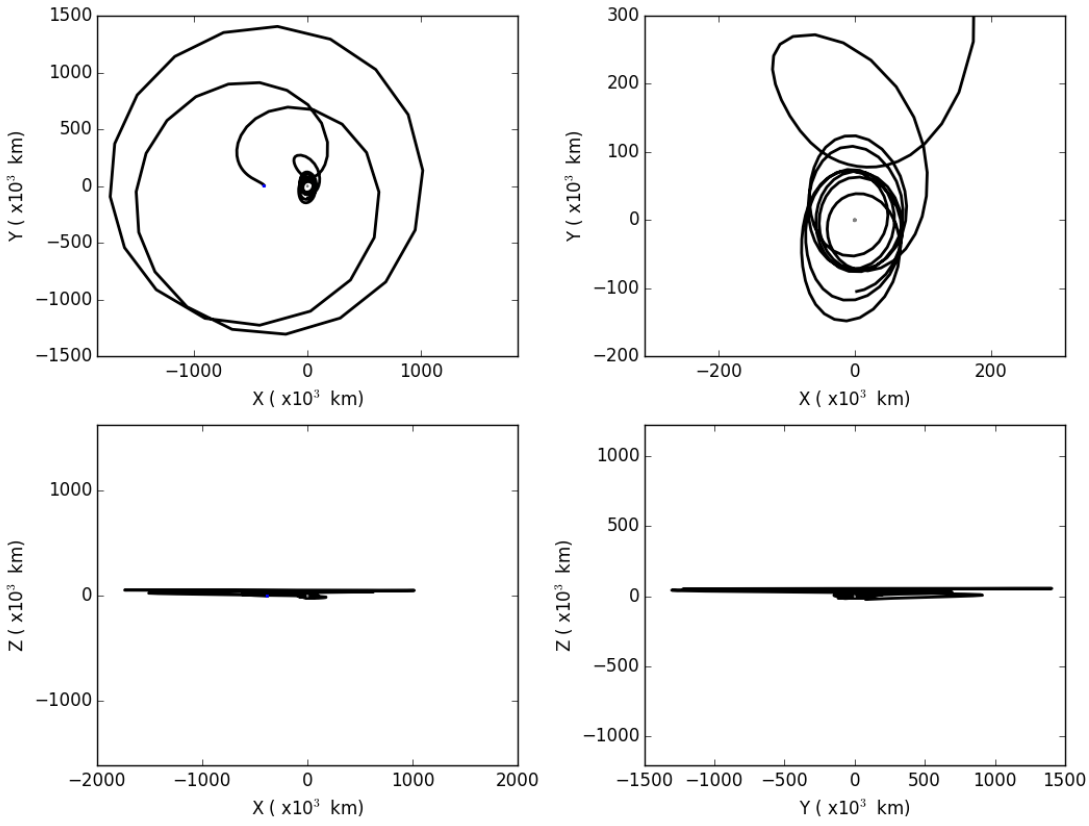


Figure 7.4: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for the ballistic 187-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale.

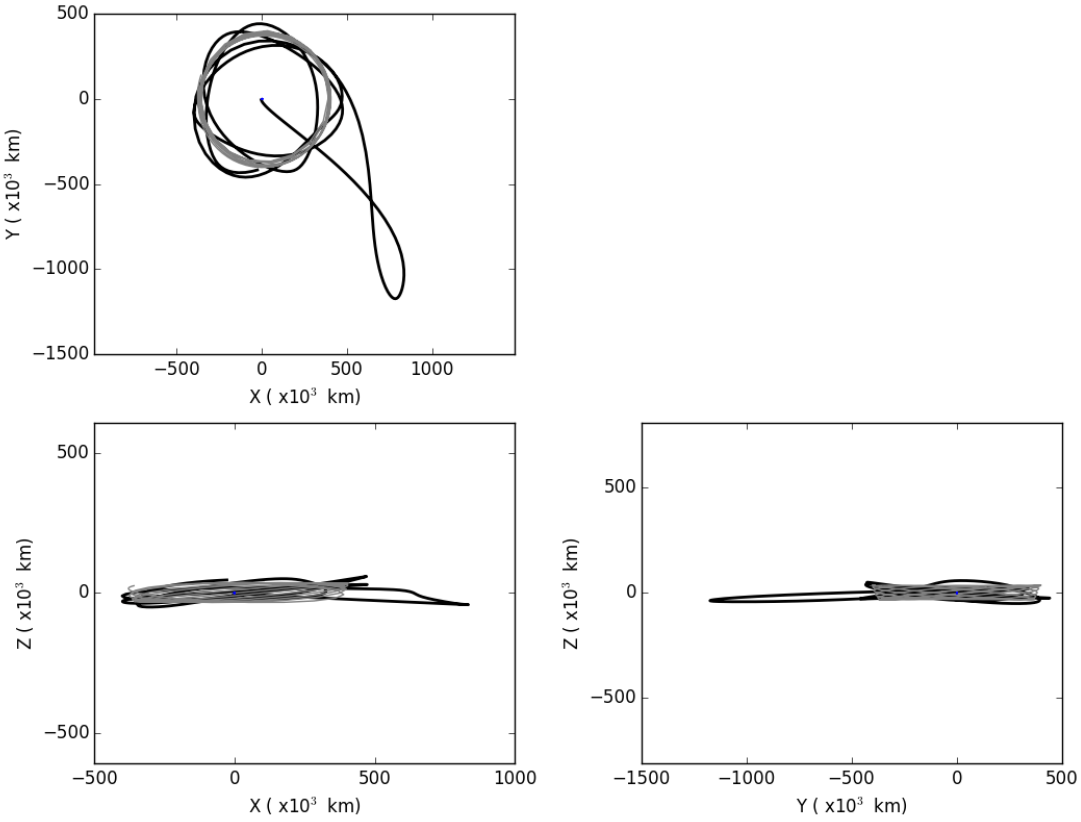


Figure 7.5: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for the ballistic 187-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

7.2.2 168-day transfer

Figures 7.6 through 7.8 detail the 168-day LEO-DRO transfer. This transfer is characterized by a 4 month arc out into heliocentric space, which is essentially entirely a coast. It then spends a month in the vicinity of the Moon, which is where it is briefly thrusting for a total propellant consumption of less than 7 kg. The closest lunar approach is 63 000 km, much farther than the ballistic transfer. It is clear that this transfer relies on gravity from the Sun to perform a lot of the work, whereas the ballistic transfer seems to rely much more on interactions with the Moon. Nonetheless, the 168-day transfer critically depends on those few days of thrusting to supplement the transfer. It may also be noted that the blocky nature of the trajectory in Figure 7.7 is caused by the very long time steps of the numerical integration. The integration does not require more steps for a proper model in inertial space, but the relatively rapid rotation of the Earth-Moon frame gives the trajectory this discontinuous nature. It is worth remembering that each segment is plotted as a straight line, but is in reality a 7^{th} degree polynomial arc.

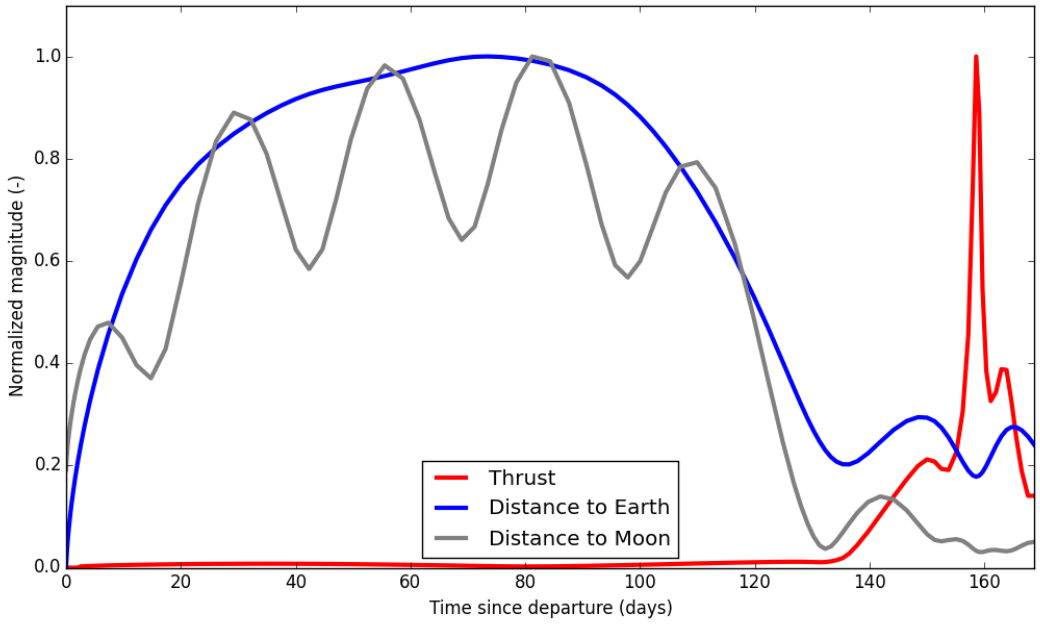


Figure 7.6: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 168-day LEO-DRO transfer.

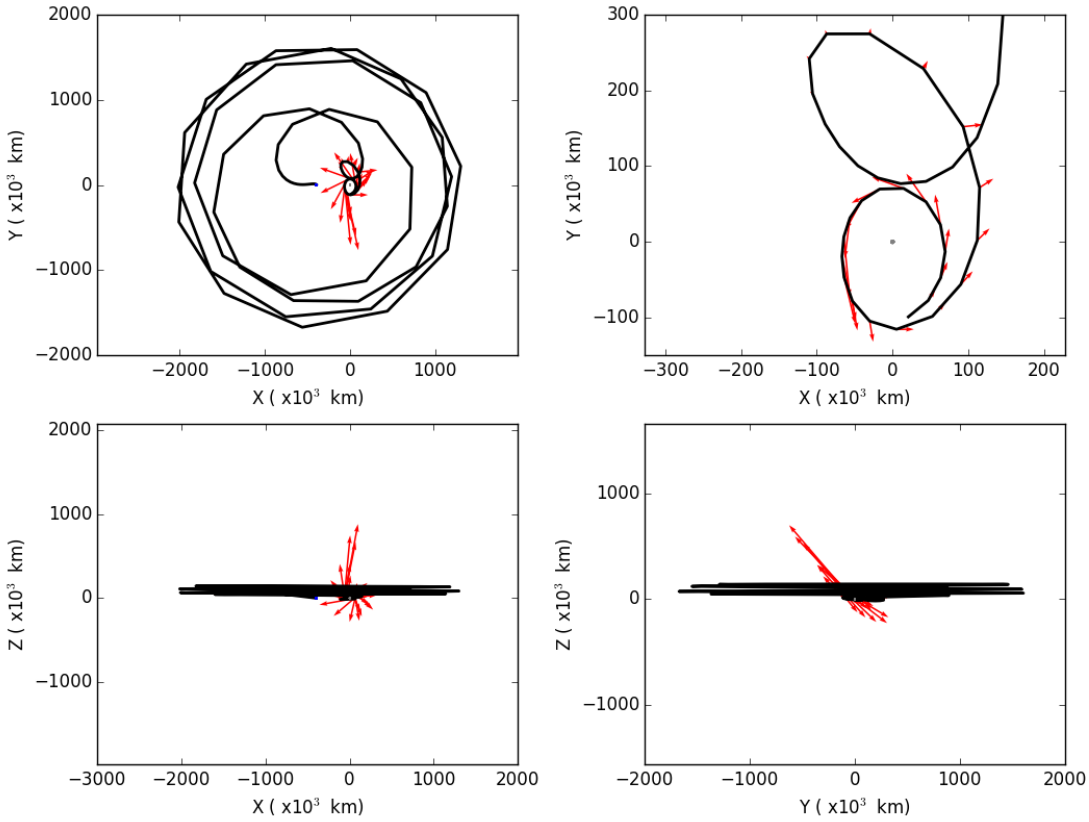


Figure 7.7: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 168-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale.

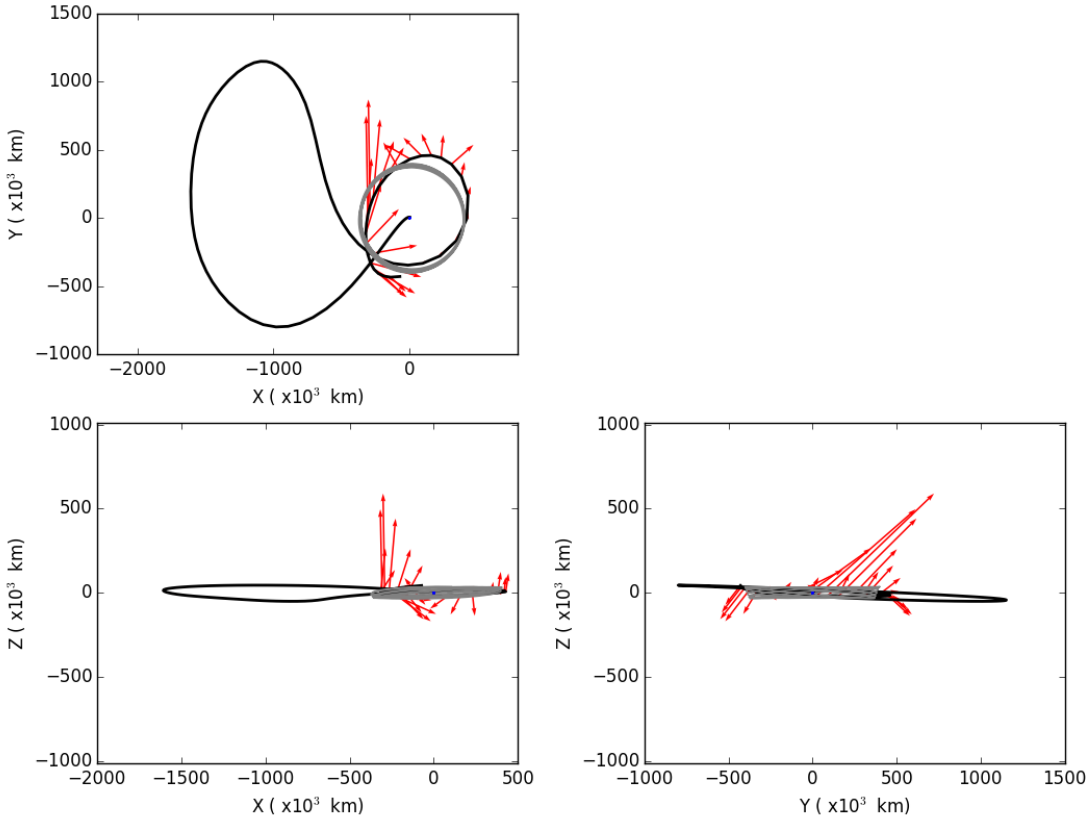


Figure 7.8: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 168-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

7.2.3 140-day transfer

Figures 7.9 through 7.11 detail the 140-day LEO-DRO transfer. It is interesting to note that the 168-day and 140-day transfers appear very similar in Figures 7.1 and 7.2, since they have almost identical final mass and departure energy, and are approximately 28 days apart in time of flight. This means that the Moon has almost exactly the same relative position throughout the transfer, and that the Sun has not moved much yet either. Despite this similar geometry, time of flight, propellant consumption, and departure energy, the trajectory is significantly different. The 140-day transfer starts on a similarly long heliocentric arc, about 3.5 months instead of the 4 months of the 168-day transfer, which is a ballistic coast throughout, and is also further from the Moon's orbital plane than for the 168-day transfer. It then performs a nearly polar lunar gravity assist, at 654 km altitude. It makes full use of this by thrusting throughout this encounter, for maximum leverage on the trajectory. The trajectory then mostly coasts, with some minor thrusting, until the last day

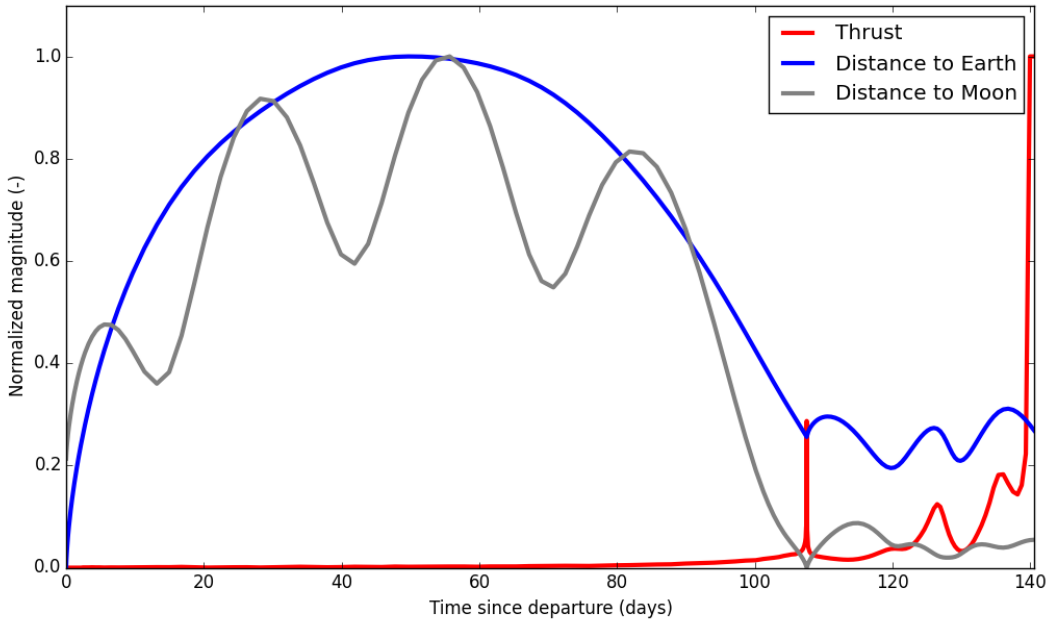


Figure 7.9: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 140-day LEO-DRO transfer.

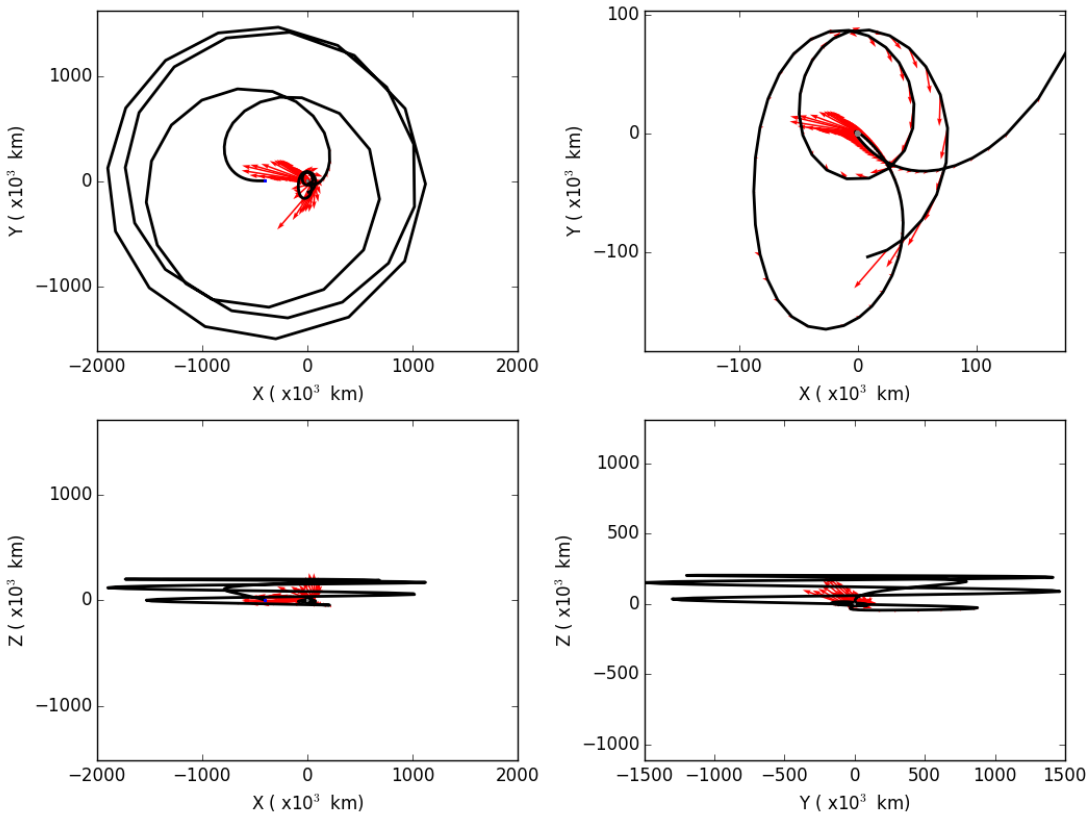


Figure 7.10: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 140-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale.

or so of the transfer, not dissimilar to the optimal solution of the DRO stabilization maneuvers presented in Chapter 6, for a total propellant consumption of less than 6 kg.

Two things stand out in this trajectory. The first is the previously mentioned dissimilarity to the 168-day solution. The departure shares some similarity, but the conclusion of the trajectory is significantly different, with an extremely close powered fly-by of the Moon. This is a testament to the chaotic solution space that defines these problems. Even though there is great similarity between two transfers, these then break off into significant differences to find a feasible and locally-optimal solution. The second thing to note is that *Maverick* introduced a very close lunar fly-by fully autonomously, and similarly it correctly identified this as an ideal time to run the thruster.

It achieves this by considering the full dynamics at all times, undoubtedly aided by the robust convergence of the collocation transcription discussed in Chapter 3, detecting the vicinity of the Moon as an opportunity to lower the value of the cost function and introducing the gravity assist accordingly. This was seen in Chapter 6, where it led to an inferior local minimum. While it is of course possible that even this 140-day transfer is not the globally optimal transfer, it is in any case very close to it, since it only uses approximately 0.3% of its total mass for propellant, which is very close to the lowest amount physically possible. This underlines the utility of a low-thrust trajectory optimization tool that is able to autonomously introduce these types of complicated maneuvers.

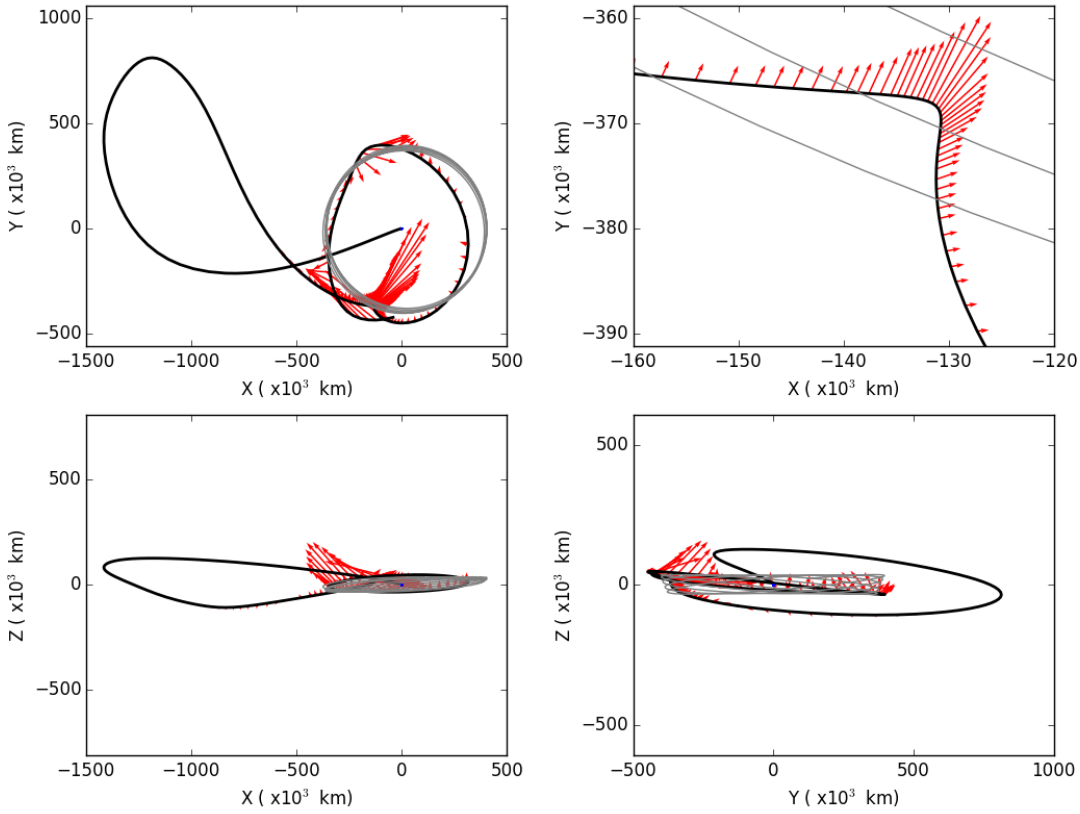


Figure 7.11: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 140-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

7.2.4 103-day transfer

Figures 7.12 through 7.14 detail the 103-day LEO-DRO transfer. This represents one of the shortest of the extremely low-cost transfers that is identified in this study. It is more than a month shorter than the 140-day solution, nearly three months shorter than the 187-day ballistic solution, and requires only 9 kg of propellant, 0.5% of the total spacecraft mass. It begins with a familiar arc into heliocentric space, then encounters the Moon at 11 000 km altitude, enough to noticeably bend the trajectory, but apparently it is not necessary to run the engine through this encounter. It then thrusts for most of the last week of the transfer, but only the last 3 days of thrust appear especially critical. Altogether, this transfer is not dissimilar from the 168-day transfer, albeit with fewer lunar interactions, which likely explains the additional expenditure of propellant.

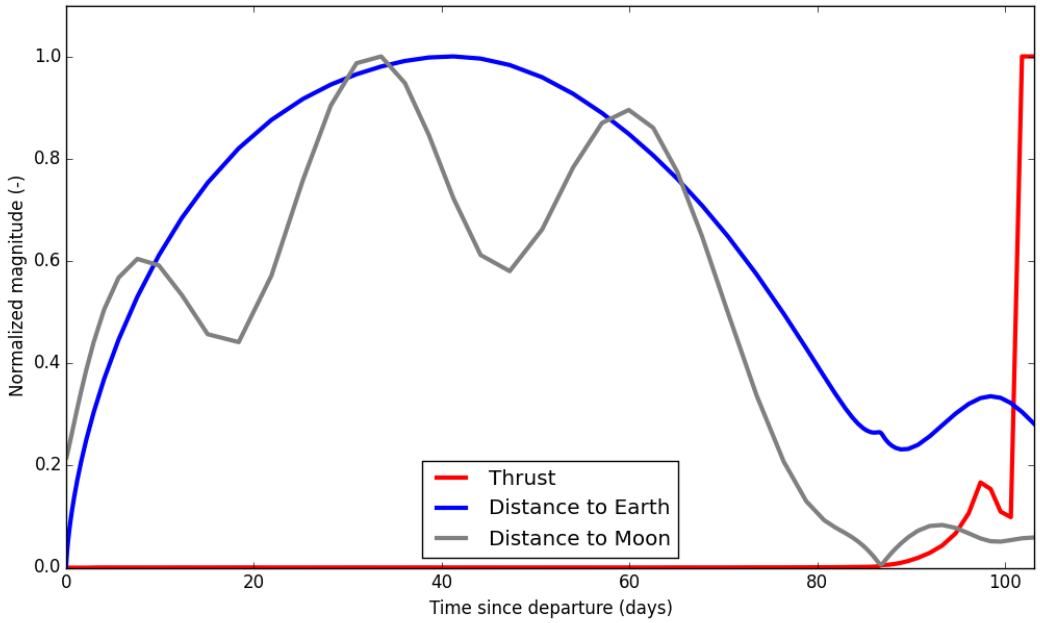


Figure 7.12: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 103-day LEO-DRO transfer.

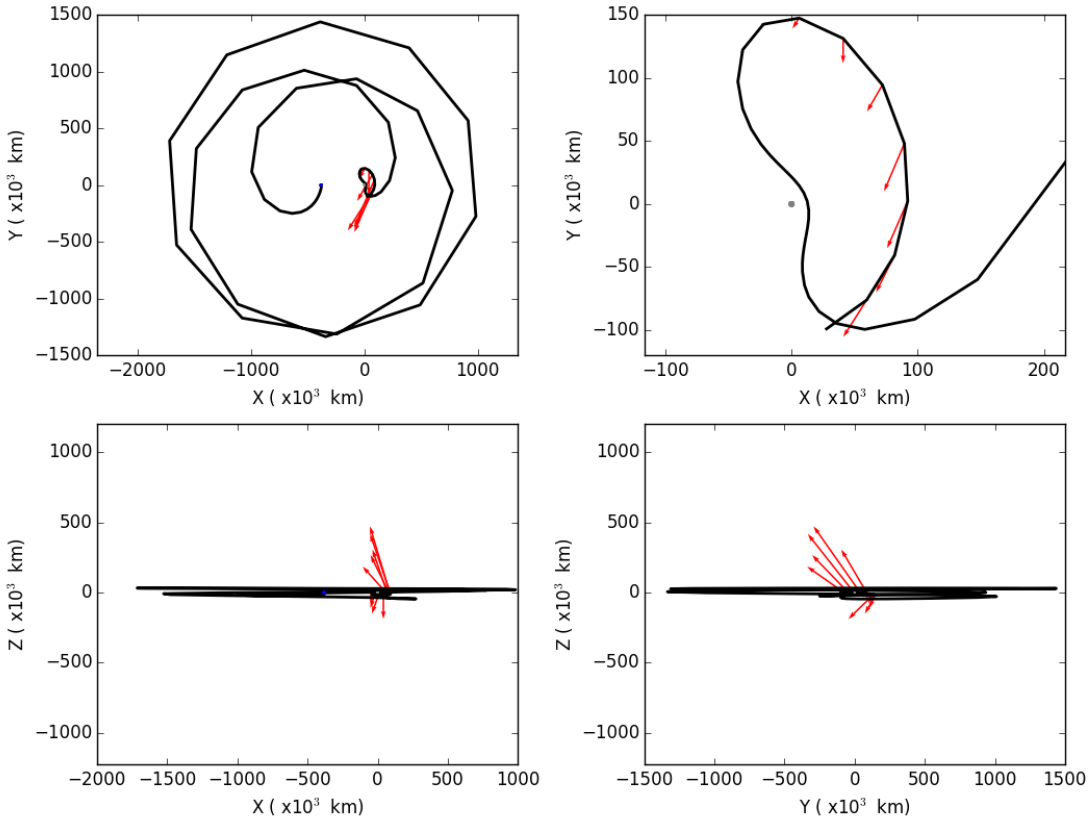


Figure 7.13: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 103-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale.

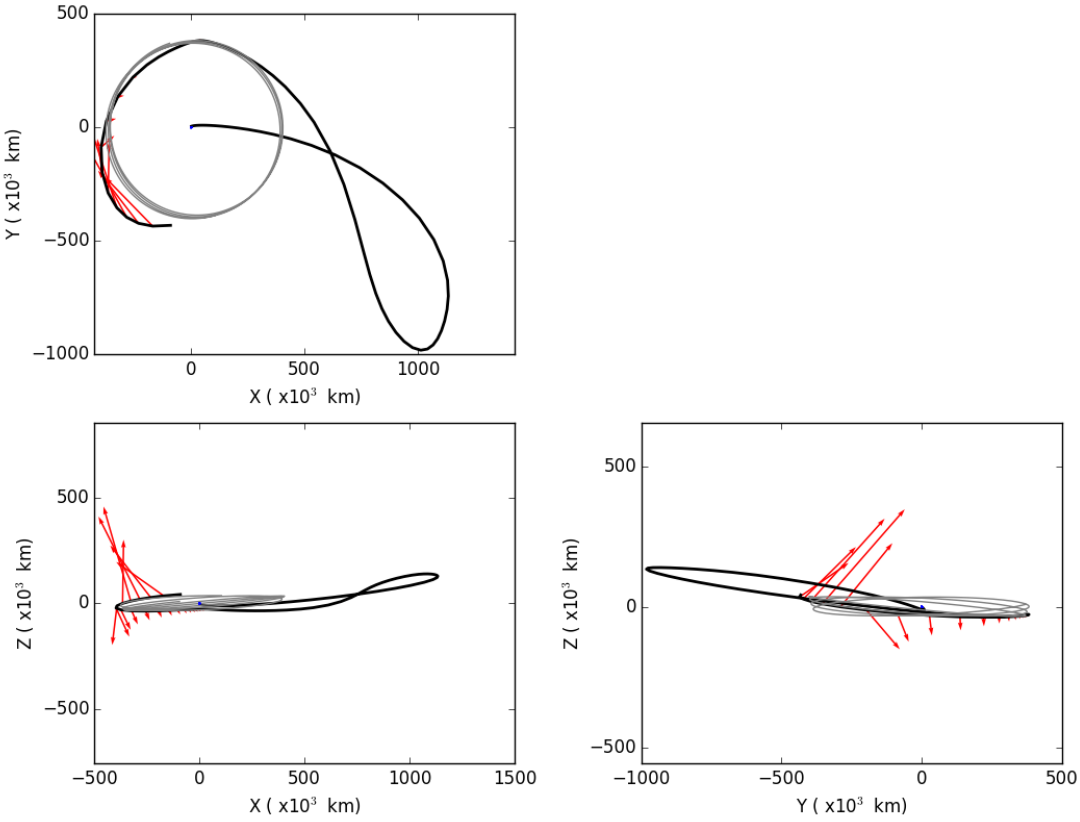


Figure 7.14: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 103-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

7.2.5 32-day transfer

Figures 7.15 through 7.17 detail the 32-day transfer, which has an unusually low propellant requirement of 32 kg for such a low time of flight. This transfer performs a fairly distant powered fly-by of the Moon at 13 000 km altitude, although it is close enough to noticeably bend the trajectory. This is then followed up by two distinct thrust arcs to arrive at the target DRO. In Figure 7.1, this transfer stands out as an unusual peak, which might suggest that the adjacent solutions represent unfavorable local optima. However, both the shorter and longer immediately adjacent transfers have extremely similar transfers, with a similar lunar fly-by. But due to some aspect of the phasing, these transfer end up needing significant thrusting in addition to the powered gravity assist. In fact, the same thing can be observed for the two distinct peaks to the right of the 32-day transfer in Figure 7.1, at 47 days and 62 days time of flight, respectively. The same thing can also be observed for the drop in the final mass from the plateau that ends at 75 days time of

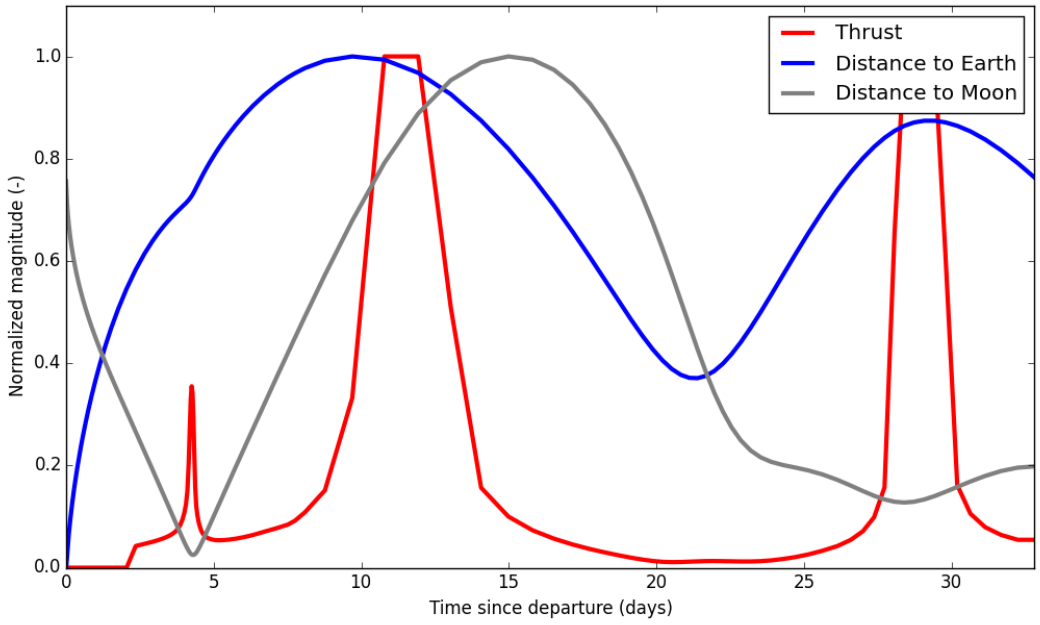


Figure 7.15: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for a 32-day LEO-DRO transfer.

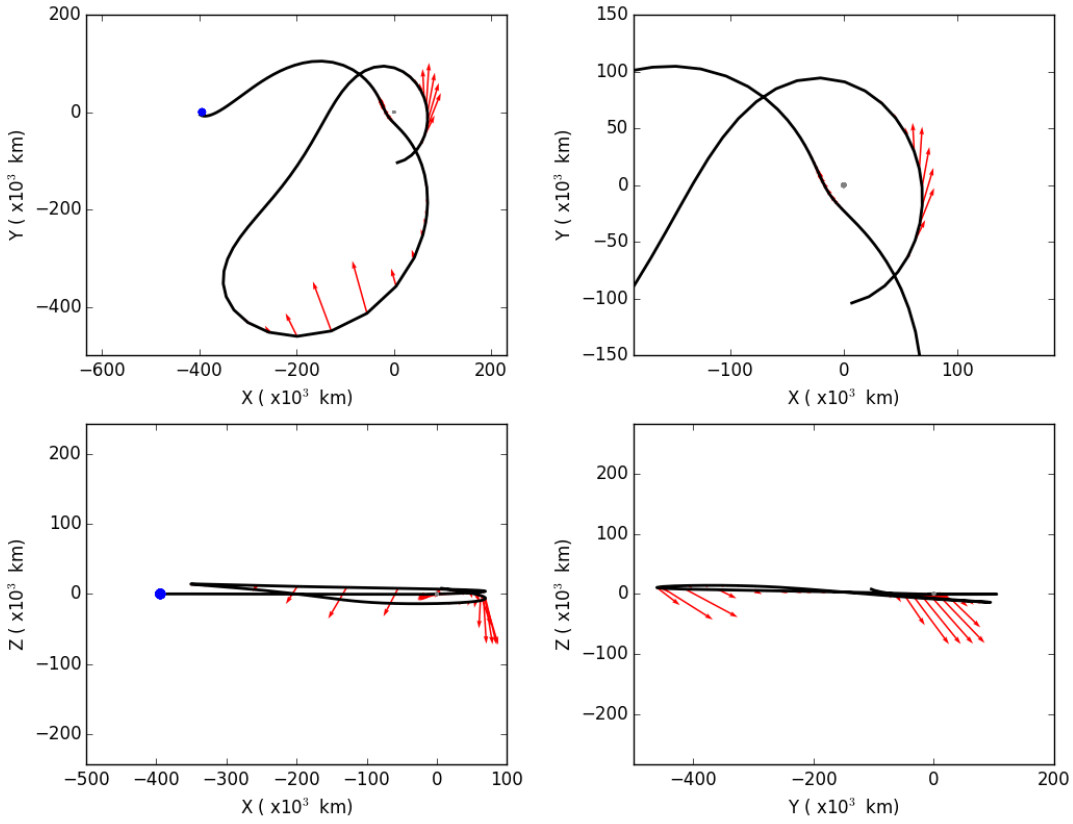


Figure 7.16: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for a 32-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale.

flight. In all these cases, the solution has an outbound lunar fly-by and then converges to the DRO in very similar fashion as the 32-day transfer. Adjacent solutions have very similar characteristics, but fail to achieve the transfer with an equally low cost. This is likely related to the phasing of the whole Sun-Earth-Moon system, and how it influences this family of solutions. When the phasing does not work out exactly, a significant cost in performance is suffered. This is of course all under the condition of a fixed combination of arrival date and time of flight. If the full 27-day period of the Moon (and perhaps also the full 365-day period of the Earth) were sampled for each time of flight, instead of a single combination, this behavior may be a lot more smooth. Unfortunately, that constitutes a momentous computational task for a marginal improvement to the results, and is not considered practical for this research.

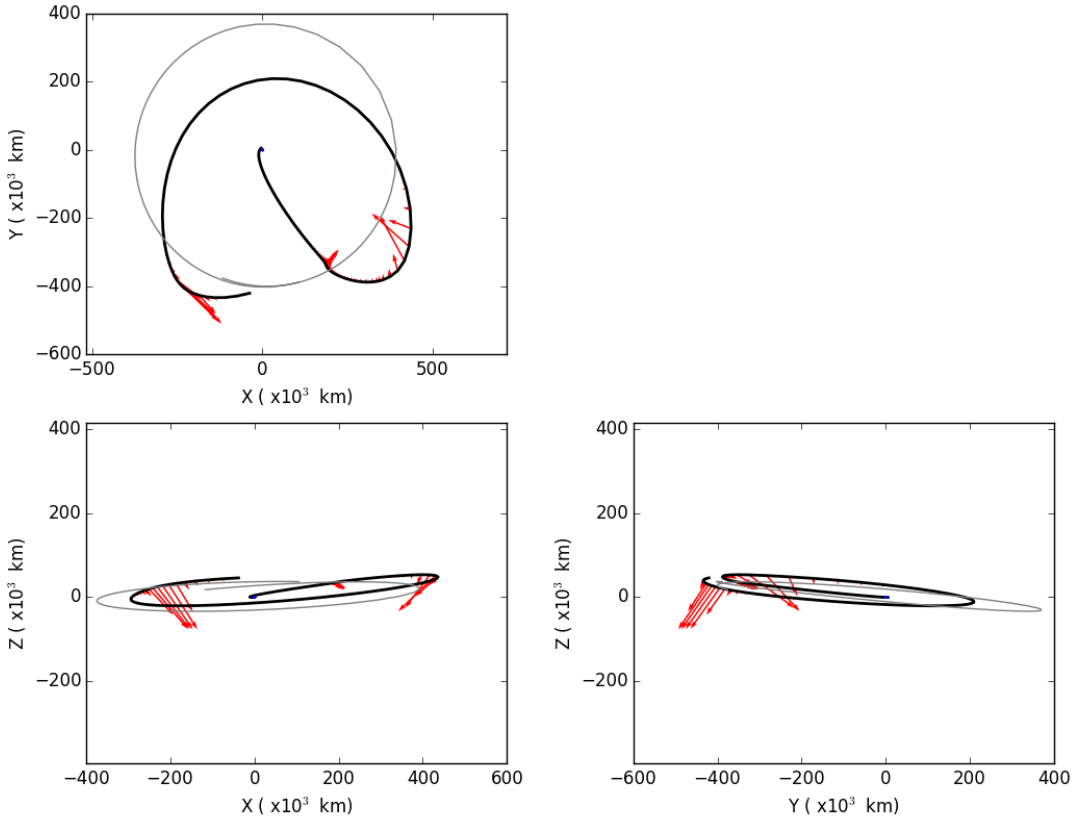


Figure 7.17: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for a 32-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

7.2.6 18-day transfer

Figures 7.18 through 7.20 detail the minimum-time 18-day transfer. Considering that the first 2 days of the trajectory are constrained to be a coast, as explained at the start of this chapter, this is clearly very close to the absolute minimum time possible. The trajectory is thrusting continuously almost immediately after the 2 day coast, for a total propellant consumption of nearly 75 kg. There are no close lunar encounters, and the trajectory essentially shoots straight up into an arc that approximates the DRO, and manages to adjust its trajectory just in time for arrival. Beyond the previously discussed 103-day transfer, as the transfers get shorter the heliocentric arcs decrease in altitude and length in time. All transfers overshoot the Moon nonetheless, as seen in the 18-day transfer, likely in an attempt to match the near-escape energy of the target DRO. Interestingly, the shorter transfers tend to have a significantly lower launch energy, likely to avoid shooting too far past the Moon, which would prevent entry into the DRO in the limited time available, or otherwise

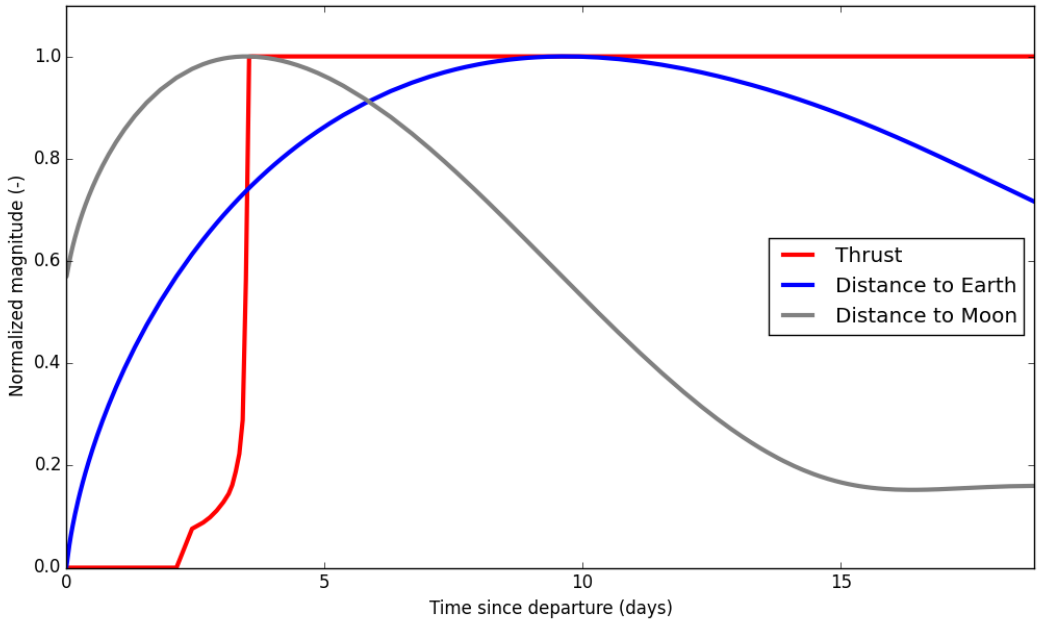


Figure 7.18: Normalized representation of Thrust and radial distance to the Earth and Moon as a function of time of flight for an 18-day LEO-DRO transfer.

negatively affect the required thrusting. It should be pointed out that it is debatable if these very short transfers are appropriately described as low-thrust, low-energy transfers, but nonetheless all dynamics are active upon them at all times. Similarly, it would be difficult to point to a specific time of flight where the transfers definitively stop being low-thrust, low-energy transfers and transition to 'purely' low-thrust transfers.

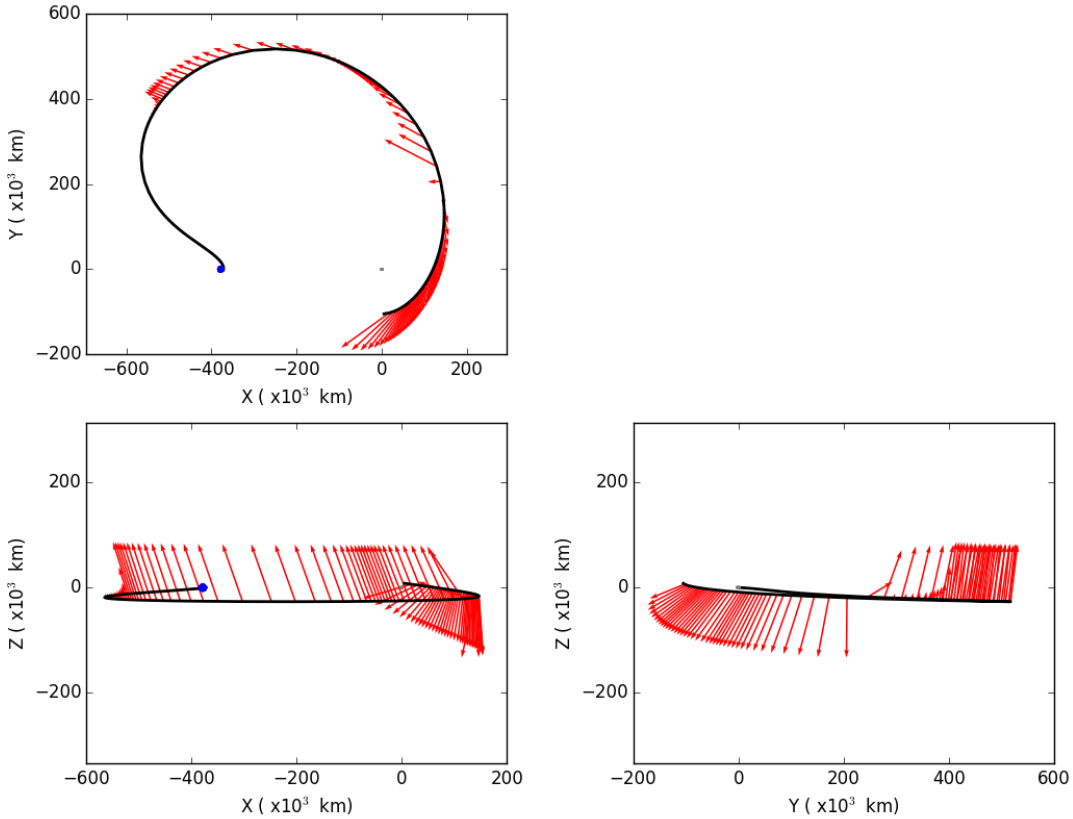


Figure 7.19: The three projections of the trajectory and thrust history in the Earth-Moon rotating frame for an 18-day LEO-DRO transfer. Both the Earth and the Moon are shown to scale.

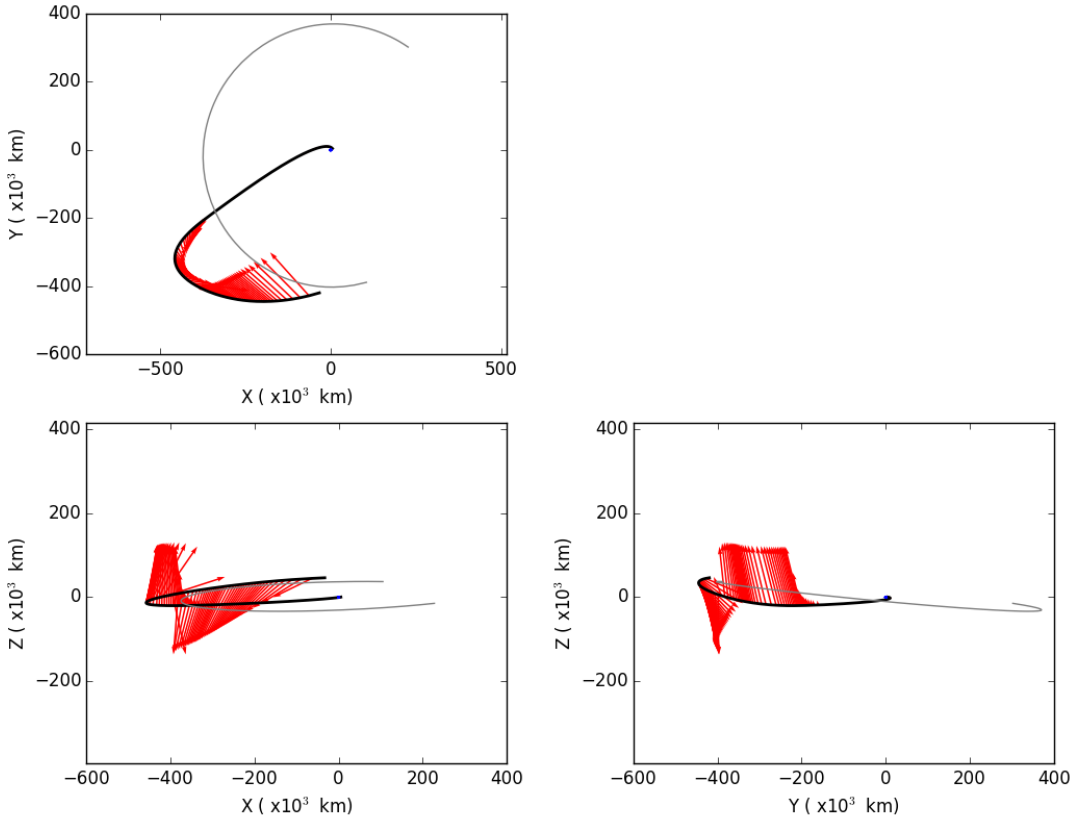


Figure 7.20: The three projections of the trajectory and thrust history in the Sun-Earth rotating frame for an 18-day LEO-DRO transfer. The Earth is placed at the origin, and the Moon's position is shown over time in gray.

7.3 Results for lower power

The results presented in Section 7.2 assume a vehicle with a mass of 1800 kg and 18 kW of engine power. While this is entirely representative of modern vehicles, it is nonetheless an unusually powerful SEP vehicle. To provide an idea of how these results scale to vehicles with more modest propulsion capability, Figures 7.21 and 7.22 present a similar study for an 1800 kg vehicle with 6 kW of engine power. For these figures, a 9-day step size in time of flight is used, instead of the 4.5-day step size that is used in Section 7.2. All other assumptions and constraints are identical. Figure 7.21 clearly shows that a very similar trend exists for the lower power vehicle. The long time of flight transfers exist almost unaltered, which is not surprising considering how little thrusting is required for these. As time of flight goes down, a similar downward curve in the delivered mass occurs, but for the lower power vehicle the shortest feasible transfer is on the order of 46 days. Some coasting exists for this transfer, so perhaps a few more days could be shaved

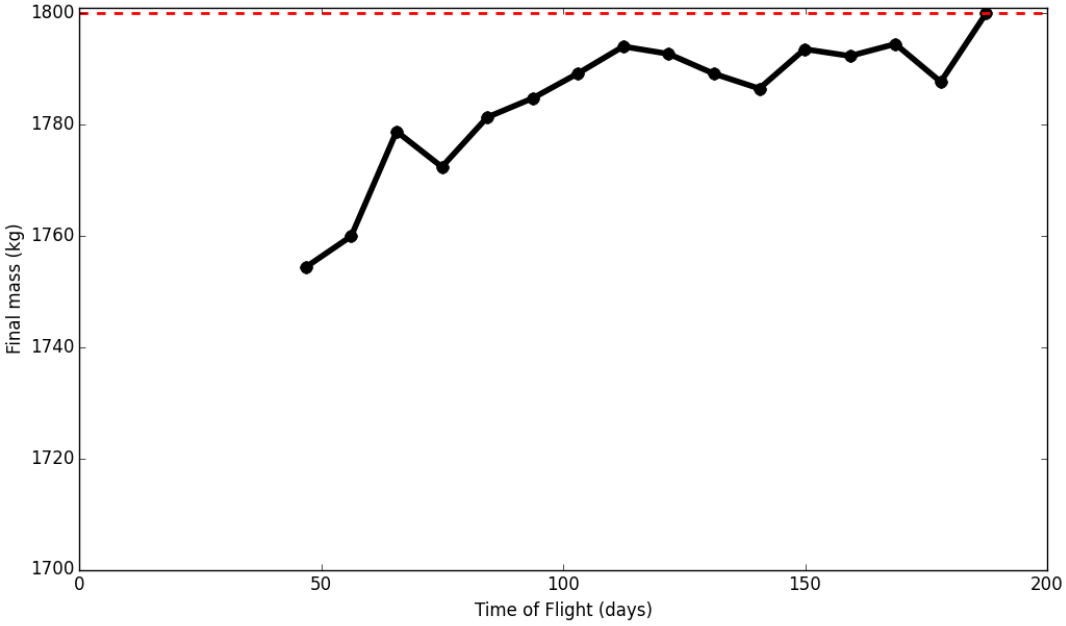


Figure 7.21: Final mass as a function of time of flight for the LEO-DRO transfers for an 1800 kg vehicle with 6 kW of power.

off, but a 37-day transfer proved infeasible. In general, as long as the lower power vehicle can fly a LEO-DRO transfer, the required propellant consumption is very similar, which is again not surprising but a positive result. Since the characteristics of the individual transfers are very similar to those presented in Section 7.2, none are presented in detail here.

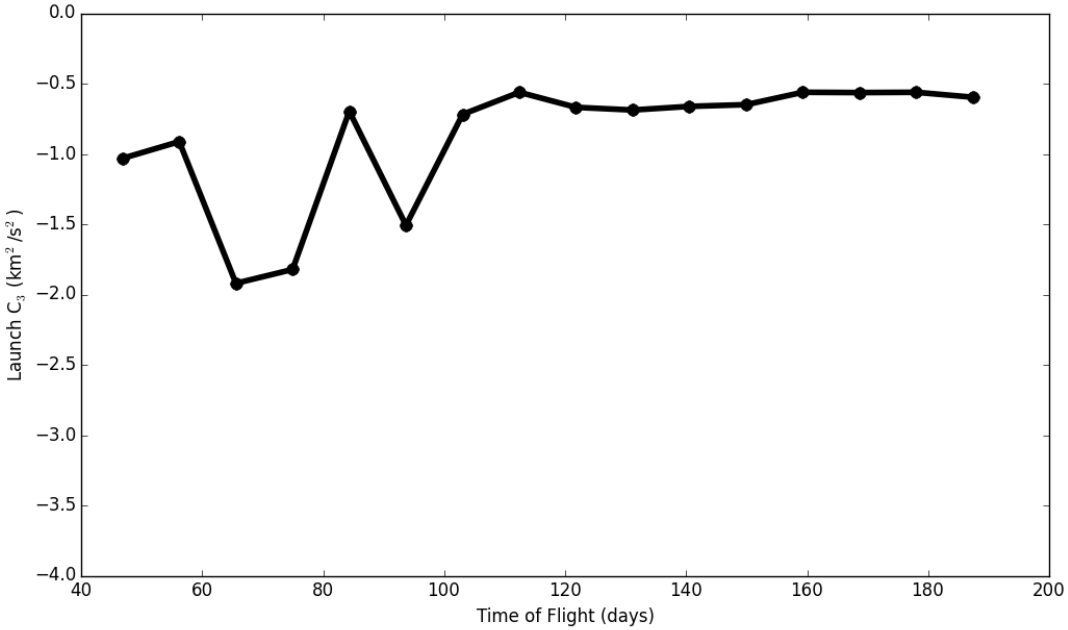


Figure 7.22: Launch energy as a function of time of flight for the LEO-DRO transfers for an 1800 kg vehicle with 6 kW of power.

7.4 Discussion of results

The results presented in this chapter show how trajectories evolve from a 187-day ballistic LEO-DRO transfer down to an 18-day low-thrust LEO-DRO transfer. Transfers are found throughout this whole domain, for two distinct power levels, with varying degrees of complexity. It appears that the lowest cost solutions are not available at all times, with noticeable fluctuations as a function of flight time. Part of this may be remaining local optima that could be improved with additional computational effort, but the variable positioning of the Sun and Moon with respect to the Earth, and each other, likely plays a part in this as well. Nonetheless, significant reductions in flight time can be achieved for a small fraction of the total mass of the spacecraft.

Additionally, the results presented in this chapter highlight the complexity of the solutions that *Maverick* can produce. At no point through the range of flight times did it fail to converge. It gradually evolves the trajectories through the region of low-thrust, low-energy transfers into 'pure' low-thrust transfers, as time of flight goes down. Furthermore, it was able to autonomously add gravity assists to individual solutions, in some cases several per transfer, and identify these as ideal times to run the thruster. This capability typically does not exist in a low-thrust trajectory optimization tool, indeed the author is not aware of earlier publications of similar capabilities, and enables a significant increase in the complexity of solutions that can be considered by trajectory designers.

Chapter 8

Conclusion

The work presented in this thesis presents a number of improvements to Gauss-Lobatto collocation methods, and collocation methods in general, for the use of low-thrust trajectory optimization. After describing the general setup of the problem formulation in Chapter 3, as well as the improvements that are made with respect to earlier implementations, this new implementation is then named *Maverick*. The improvements made specifically with the use of parallel computing are discussed in Chapter 4. All improvements are made exclusively to the computation of the constraints and the Jacobian matrix, in other words, the nonlinear programming (NLP) solver is left untouched. Without accounting for the runtime of the NLP solver, the results are promising. The parallelized elements of the problem formulation execute up to 11 times faster, depending on what force model is used. Even in the worst case, a performance improvement of at least a factor of 4-6 is feasible with Graphics Processing Units (GPUs), and a factor of 4 when a quad-core Central Processing Unit (CPU) is used. Unfortunately, the operations of the NLP solver are often a significant, or even dominant fraction of the runtime, and present an obstacle to delivering these performance improvements to the full optimization process. When the force model evaluations are very computationally expensive, such as a gravity field of high degree and order, performance is improved by a factor of 3.7 when also accounting for the operations of the nonlinear programming solver. For problems that use two-body dynamics, or even three- or four-body dynamics, the NLP solver represents 70-90% of the total runtime, and effectively eliminates any noteworthy improvement to performance. Nonetheless, if NLP solvers can be made to operate faster in the future,

such as through greater use of parallel computing in their operation as well, performance improvements of an order of magnitude appear feasible. Furthermore, for the most expensive trajectory optimization problems, the ones that are in greatest need of runtime reduction, these performance gains are already largely accessible.

Following this study into improved performance, some new applications are studied to test the operation of *Maverick*. In Chapter 5, transfers between halo-like orbits in the Earth-Moon system are studied, with specific application to the New Worlds Observer mission concept. A range of transfers is considered, and although *Maverick* reliably and effectively solves these sensitive transfers, it quickly becomes apparent that these are not suitable orbits for the New Worlds Observer. Chapter 6 then discusses a similar problem in a more complex dynamical regime, stabilizing a lunar Distant Retrograde Orbit (DRO) by effectively transferring from an unstable DRO to a stable DRO. It is shown that this can be done for a low cost and at a low time of flight. These transfers also highlight the unstable nature in the DRO orbital regime, and illustrate how *Maverick* can wander into local optima as a result of this, when the transfer time becomes sufficiently long.

Finally, a large number of transfers are computed that depart Low Earth Orbit (LEO) with a launch energy close to escape and transfer into a DRO. This is based off a 6 month ballistic transfer that uses gravity from the Sun, Earth, and Moon to accomplish this transfer, and combines at least three highly variable dynamical environments into a single trajectory optimization problem. Using a low-thrust propulsion system, transfers are computed that significantly reduce the time of flight. Using engine characteristics representative of modern electric propulsion systems, solutions are found that reduce the time of flight by 45% for a cost in propellant as low as 0.5% of the total spacecraft mass. Additionally, the flight time can be reduced by as much as 90% for a propellant cost of 4% of the total spacecraft mass, or by 83% for a propellant cost of less than 2%. These transfers specifically, and the knowledge that such cheap reductions in flight time exist, may be useful for a variety of future crewed and robotic missions. These transfers also show that *Maverick*

can seamlessly transition its solutions from full low-thrust, low-energy trajectories to the 'pure' low-thrust trajectories that define the shortest transfer trajectories. Perhaps more importantly, these cases highlight the complexity of the solutions that can be found with *Maverick*, and the improvements to collocation methods it represents. This is most clearly shown with the cases where *Maverick* autonomously introduces low-altitude gravity assists of the Moon and identifies these as optimal locations for thrust arcs. This capability has the potential to greatly increase the utility of these tools to a mission designer, and is typically not observed in low-thrust trajectory optimization tools. The author is not aware of earlier publication of similar capabilities.

While these LEO-DRO transfers are still susceptible to local optima due to their physical nature, this is an inevitable problem in many low-thrust trajectory optimization applications, with known solutions that come only at a cost in computational effort. In spite of these local optima, the ability of the trajectory designer to start an algorithm like *Maverick* and then pick up another task, knowing that the algorithm will consider a wide range of solutions, including those involving a complex sequence of gravity assists that the human operator in no way needs to define, is a capability that can be put to use effectively in many other applications. Altogether, the work in this thesis presents several significant steps forward for low-thrust trajectory optimization methods, draws some new and interesting conclusions for three practical and challenging applications, and in doing so illustrates the utility of these improved collocation methods to low-thrust trajectory optimization problems in general.

8.1 Significant contributions

This section briefly summarizes and describes the most significant contributions of the work presented in this thesis.

Parallel implementation of serial optimization process This thesis represents one of the early attempts in the astrodynamics community to reformulate an inherently serial process into one that can be handled effectively with multiple cores. It is already common to run processes that are inherently embarrassingly parallel on multiple cores, such as a Monte Carlo analysis, but this is only practical for a subset of all astrodynamics work. The work presented in this thesis takes a step towards opening these performance improvements for more inherently serial processes, such as trajectory optimization or numerical integration of individual trajectories. Significant performance gains are demonstrated for the affected functions, which underlines the utility of these steps. Nonetheless, for many applications there currently are significant obstacles that prevent the full realization of the achieved performance improvements. The source of these obstacles and the conditions under which they apply are also identified for the first time in this thesis, enabling future studies to address these.

It should be stressed that while parallel computing applications are no longer rare in astrodynamics, the work presented here distinguishes itself by focusing the parallelization at the level of individual steps in the numerical integration scheme. Most previous work has typically focused at parallelizing at a level either above or below (within) the individual integration step [2, 54, 37, 12, 11, 3, 43].

General improvements to collocation methods The implementation of collocation methods described in this thesis represents several improvements over earlier applications of these methods. To date, these methods have had their sequence of thrust and coast arcs predefined by the user, severely influencing the solutions that can be produced. By opening up the full throttle range of the control vector at all times, the problem is made significantly more complex, threatening a

successful convergence. Through a combination of carefully scaling all aspects of the problem and more involved reformulations of the thrust constraints and cost function, successful convergence is maintained while giving the optimizer full control over how many thrust arcs are introduced, and where. This is a necessary improvement for the optimizer to find non-intuitive solutions, which is of particular importance for long, complex transfers.

Demonstrated autonomous introductions of (powered) gravity assists Building on the improvements summarized above, the improved collocation algorithm is repeatedly shown to autonomously introduce powered gravity assists into transfers, without any direction from the user. These solutions are typically very competitive, with extremely low cost values associated with them. This capability is typically not available in low-thrust trajectory optimization tools, and enables a significant improvement in utility.

Study of trade space for various complex low-thrust, low-energy problems The most notable example of this are the LEO-DRO transfers that are designed and presented in this thesis. Such a trade study would have been difficult in the past, and the quality of the results depend directly on the improvements described above. Previously, it was difficult to even provide a single point design for these transfers, and the work in this thesis presents a trade space exploration that was largely automated. Besides being a credit to the achieved improvements to collocation methods, the results from these low-thrust LEO-DRO transfers show interesting characteristics that can be useful for a variety of future missions. Similarly, the transfers between halo-like orbits and the DRO stabiliation maneuvers, while relatively simple when compared to the LEO-DRO transfers, represent challenging trajectory optimization problems that few optimization tools can study. They are included here mainly as test cases for the optimization algorithm, but they nonetheless present some interesting conclusions that are relevant to a variety of missions.

8.2 Future Work

This section summarizes a number of recommendations for future work to effectively build on the work presented here.

Writing a better NLP solver As remarked in Chapter 4, the NLP solver remains a large obstacle to using parallel computing effectively. While many very effective NLP solvers exist, few use any degree of parallelization, and even those that do are very limited in the extent they use it. If an NLP solver could be written that reduces the computational cost required, through the use of parallel computing or through other means, that would enable significant performance improvements for the algorithms presented here, as well as many others.

Combining NLP solvers As described in References [6] and [67], interior point solvers like IPOPT do not benefit from a good initial guess as much as SQP solvers like SNOPT do. On the other hand, interior point algorithms perform very well when a poor initial guess exists. On top of that, the current state of IPOPT is simply more modern than SNOPT, giving it a variety of performance and ease-of-use benefits. This is one reason IPOPT was used for this research, but in reality it is not ideal for the later iterations of the mesh refinement process. More thorough comparison of IPOPT and SNOPT, as well as other NLP solvers could be valuable. The result would likely show that using an interior point solver like IPOPT for the first iteration(s) of the mesh refinement process would be ideal, but for best convergence it would then be ideal to switch to an SQP solver like SNOPT. Combining these algorithms within a single mesh refinement process could speed up convergence.

Using the mesh refinement process for additional formulation adjustments In the work presented here, the mesh refinement process is only used to adjust the spacing of grid points in time, and possibly adding additional points. However, these moments are great opportunities to adjust other aspects of the formulation. One example of this is to adjust the central body of each

input state to match the one it is most affected by. In the work presented here, all states are given with respect to a single central body. It would not be a large task to actually check for each state what central body is most appropriate. In the context of the results in Chapter 7: the states early in the trajectory could have Earth as their central body, those on the very high altitude arc either the Earth or the Sun, and as the trajectory approaches the Moon, the Moon could be used as the central body, especially in the case of lunar gravity assists. It would not be difficult to automate the assignment of the central body, which could be updated in every mesh refinement iteration, to ensure the problem formulation is updated to best suit the trajectory as it currently is. Other aspects that could be updated similarly are the tolerances set on different constraints, a variety of scaling factors, the 'fudge factor' values, or the weights/constraints placed to discourage subsurface gravity assists. Many of these would be relatively specific to the problem that is currently studied, but some are more general.

Improvements to provided derivatives Some straightforward though not trivial improvements would be the implementation of analytical derivatives for the defect constraints. This would certainly speed up convergence, and potentially improve the produced solutions. It would also make it much more laborious to switch force models or reference frames. As a result it may only be interesting when a single, very specific problem is of extreme interest, justifying the effort spent on setting up the applicable derivatives. Similarly, the use of second order derivatives is supported by many NLP solvers, such as IPOPT. Providing these, either through finite differencing or analytically, could similarly improve convergence.

Defining control vector within segments The work presented here stuck with the traditional approach in Gauss-Lobatto collocation of defining the control vector either as a linear interpolation between the end points of a segment, or as a constant defined at the beginning of the segment. However, without any significant change to the current formulation, the control vector could easily be defined at the defect points and/or the internal points as well. For the GL-4

method this provides one additional point per segment, but for the GL-12 method this provides up to 5 additional points where the control vector could be defined. Considering the high numerical integration order of GL-12, there are occasions where the mesh refinement process will converge to a low number of segments, but there are indications that the control history could do with more refinement of the pointing of the control vector. In those occasions having additional points where the control vector is defined as an input parameter could improve the convergence without dramatically increasing the problem size. For future implementations of GL-12 methods, it is recommended that the control at every node point, internal point, and defect point, is defined as an input parameter.

Investigating impact of a 'fudge factor' in collocation methods This work was the first to introduce hyperbolic thrust constraints and 'fudge factors' into collocation methods, but no thorough study is done of their impact on convergence as a function of their magnitude. Studying this could lead to useful performance gains. Additionally, this could benefit the development of schemes that adaptively adjust the magnitude as the solution evolves through the mesh refinement process.

Adding a global search outer loop Similar to the work done with EMTG, which places a monotonic basin hopping outer loop on top of a Sims-Flanagan inner loop, it would not be difficult to implement a global search strategy on top of a collocation-based local search strategy. While this is significantly more computationally demanding than something that builds around a Sims-Flanagan method, it is potentially a very powerful combination when studying complex problems such as low-thrust, low-energy transfers. It would further automate the search for useful solutions, leaving the user to focus on other tasks.

Investigate alternative collocation schemes Instead of using a 12^{th} order Gauss-Lobatto scheme, a comparable order Gauss-Legendre or BLC-IRK scheme could be used, which should in theory allow for a small decrease in the number of segments required to accurately model the problem. Neither of those schemes naturally include the end points of the polynomial, though, which means the overall formulation would need to be adjusted with a few additional constraints to ensure continuity from one segment to the next. Similarly, higher order polynomial degrees could be investigated, although it is not certain this would lead to any performance gains.

Investigate relative performance of multiple shooting schemes The work presented here illustrates the strength of collocation schemes for solving a variety of problems, in particular low-thrust, low-energy transfers, but it makes no attempt to quantify how multiple shooters (or single shooters) would compare. Some work has been done to this end, as discussed in Section3.1. Based on this, as well as general experience with a variety of optimization schemes, it is expected that multiple shooters, and especially single shooters, would struggle with the problems presented in this work, especially the problems from Chapter 7. Nonetheless, it may be interesting to further study how collocation methods compare to single or multiple shooting schemes for a range of problems, including the low-thrust, low-energy transfers presented here, but also much more simple problems.

Investigate autonomously introduced gravity assists An unexpected outcome from this work is that *Maverick* demonstrates the ability to introduce gravity assists of the Moon autonomously. This is most effectively seen in the results of Chapter 7, but it is also discussed in Chapter 6. Some additional insight in the strength of this behavior can be gained by the discussion in Section 7.1.2 of additional constraints that need to be used to temper this behavior into physical reality. While *Maverick*'s tendency to introduce gravity assists is taken advantage of in this work, and discussed to some extent, no attempt is made to study the extent or the limitations of this behavior. It would be interesting to perform a follow-on study to detail these abilities more. Some

questions that come to mind are how sensitive the introduction of these gravity assists are to the initial guess, as well as the dynamics relevant to the problem. For example; could gravity assists be effectively introduced in interplanetary transfer problems as well? And how close must the initial guess be to a transfer that incorporates a gravity assist for *Maverick* to identify the opportunity? Similar to the previous suggestion for future work, a comparison could be done to single or multiple shooters, and see to what extent they could autonomously introduce gravity assists, and whether or not collocation methods present a significant advantage in this regard.

Investigate applicability of parallel formulation to initial value problem If the same formulation as presented in this work is used without any control vectors or cost function, the problem reduces to an initial value problem. The exact same approach to parallelization could be followed as presented here, but there would be no need for a complex NLP solver like IPOPT or SNOPT, since only a single feasible solution exists, rather than an infinite amount. This problem could be solved with a much computationally cheaper least squares algorithm, which would remove the chief obstacle to performance gains identified in Chapter 4, the computationally expensive NLP solver. In that case, this could be a very straightforward path to improving performance by up to an order of magnitude for the initial value problem, as well as allowing GPUs to be used effectively for the numerical integration of individual trajectories. This would be of great utility for a variety of applications, but most significantly for the space situational awareness community, which faces an enormous computational challenge in the propagation of all known objects in Earth orbit.

Investigate ideal level of parallelization efforts Between the work presented in References [2, 54, 37, 12, 11, 3, 43] and that presented here, parallelization is implemented at a variety of levels. The work presented here parallelizes individual numerical integration steps, while other work has introduced parallelization above that level (for example at the level of a Monte Carlo analysis), or below that level (for example at the level of parallelized force model calls). All of these approaches compete for using the parallel computing hardware available, and it is unclear which

approach, or some combination of these approaches, is optimal for a given application. It may be interesting to study the various levels of introduced parallelization for a series of applications to see if performance differences exist, or if they all converge to similar performance improvements that are dictated by the available hardware.

Additional applications, such as missions to GEO Some of the results presented in Chapter 7 show the existence of extremely low cost, low time of flight transfers from LEO to DROs. It would be interesting to study what other destinations in the Earth-Moon system can be reached at a similar low cost, enabled by a combination of the Sun, Earth, Moon, and an electric propulsion system. One exciting candidate for this is geostationary Earth orbit, a region targeted for many spacecraft transfers. The spacecraft traveling here are usually already equipped with an electric propulsion system for station-keeping, or even to enable part of the transfer. It is not unlikely that tools like *Maverick* could produce attractive transfers for missions targeting GEO altitudes that use the electric propulsion system along with the gravitational attraction of the Sun, Earth, and Moon. Similarly, it may be that interesting escape trajectories exist that allow attractive departures/arrivals into the Earth-Moon system. This would be especially relevant for very high mass vehicles, which can benefit dramatically even from small savings in propellant.

Bibliography

[1] A.E. Bryson and Y.C. Ho. Applied Optimal Control. Hemisphere Publishing Corporation, 1975.

[2] N. Arora, V. Vittaldev, and R.P. Russell. Parallel computation of multiple space trajectories using gpus and interpolated gravity models. Journal of Guidance, Control, and Dynamics, 38(8):1345–1355, 2015.

[3] X. Bai and J.L. Junkins. Modified chebyshev-picard iteration methods for solution of initial value problems. The Journal of the Astronautical Sciences, 59(1-2):327–351, 2012.

[4] J. T. Betts. Survey of Numerical Methods for Trajectory Optimization. AIAA Journal of Guidance, Control and Dynamics, 21:193–207, 1998.

[5] J. T. Betts and S. O. Erb. Optimal low thrust trajectories to the moon. SIAM Journal on Applied Dynamical Systems, 2(2):144–170, 2003.

[6] J.T. Betts. Practical Methods for Optimal Control Using Nonlinear Programming. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1st edition, 2001.

[7] J.T. Betts and W. P. Huffman. Mesh refinement in direct transcription methods for optimal control. Optimal Control Applications and Methods, 19(1), 1998.

[8] C.J. Bezrouk and J.S. Parker. Ballistic Capture Into Distant Retrograde Orbits From Interplanetary Space. AAS 15-302, AIAA/AAS Spaceflight Mechanics Meeting, Williamsburg, Virginia, January 11-15, 2015.

[9] F. Biscani, D. Izzo, and C.H. Yam. A Global Optimisation Toolbox for Massively Parallel Engineering Optimisation. International Conference on Astrodynamics Tools and Techniques, 2010.

[10] R.A. Broucke. Periodic Orbits in the Restricted Three-Body Problem with Earth-Moon Masses. Tech. Rep. 32-1168, Jet Propulsion Laboratory, California Institute of Technology, 1968.

[11] A. Brown, J. Tichy, and M. Demoret. GPU Accelerated Conjunction Assessment: Parallel Construction of Ordered Binary Radix Trees for Collision Detection. AIAA/AAS Space Flight Mechanics Meeting, Santa Fe, New Mexico, 2014.

[12] A. Brown, J. Tichy, M. Demoret, and D. Rand. GPU Accelerated Conjunction Assessment with Applications to Formation Flight and Space Debris Tracking. AIAA/AAS Astrodynamics Specialist Conference, Hilton Head, South Carolina, 2013.

[13] W. Cash, S. Kendrick, C. Noecker, J. Bally, J. Demarines, J. Green, P. Oakley, A. Shipley, S. Benson, S. Oleson, D. Content, D. Folta, S. Garrison, K. Gendreau, K. Hartman, J. Howard, T. Hyde, D. Lakins, J. Leitner, D. Leviton, R. Luquette, B. Oegerley, K. Richon, A. Roberge, S. Tompkins, J. Tveekrem, B. Woodgate, M. Turnbull, D. Dailey, K. Decker, R. Dehmohseni, B. Gaugh, T. Glassman, M. Haney, R. Hejal, C. Lillie, A. Lo, D. O'Conner, G. Oleas, R. Polidan, R. Samuele, S. Shields, J. Shirvanian, D. Soohoo, G. Tinetti, B. Dorland, R. Dudik, R. Gaume, and B. Mason. The New Worlds Observer: the astrophysics strategic mission concept study. In Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, volume 7436, page 6, August 2009.

[14] C. A. Coello Coello, D. A. "Van Veldhuizen", and G. B. Lamont. Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic / Plenum Publishers, New York, 2002.

[15] C. de Boor. Good Approximation by Splines with Variable Knots, II. Conference on the Numerical Solution of Differential Equations, Lecture Notes in Mathematics, 363, 1973.

[16] S. Elvik. Optimization of a low-energy transfer to Mars using Dynamical Systems Theory and Low-Thrust Propulsion. Master's thesis, Faculty of Aerospace Engineering, Delft University of Technology, 2004.

[17] J.A. Englander, B.A. Conway, and T. Williams. Automated mission planning via evolutionary algorithms. Journal of Guidance, Control, and Dynamics, 35(6):1878–1887, 2012.

[18] A.S. Feistel and C.L. Ranieri. Modeling perturbation and operational considerations when using indirect optimization with equinotical elements. 2009.

[19] W. M. Folkner, J. G. Williams, D. H. Boggs, R. S. Park, and P. Kuchynka. The planetary and lunar ephemerides DE430 and DE431. Interplanetary Network Progress Report, 196, February 2014.

[20] A. Ghosh, R. Beeson, D. Ellison, L. Richardson, D. Carroll, and V. Coverstone. A Non-Linear Parallel Optimization Tool (NLPAROPT) For Solving Spacecraft Trajectory Problems. In AAS/AIAA Astrodynamics Specialist Conference, Vail, Colorado, August 9-13, 2015.

[21] P.E. Gill, W. Murray, and M.A. Saunders. SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization. SIAM Review, 47(1):99–131, 2005.

[22] D.M. Goebel and I. Katz. Fundamentals of Electric Propulsion: Ion and Hall Thrusters. JPL Space Science and Technology Series. Wiley, 2008.

[23] R.G. Gottlieb. Fast gravity, gravity partials, normalized gravity, gravity gradient torque and magnetic field: Derivation, code and data. NASA Technical Report, 1993.

[24] NASA GRC. Engine List. http://spaceflightsystems.grc.nasa.gov/SSPO/ISPTProg/LTTT/documents/engine_list.m, last accessed March 18, 2014.

[25] D.J. Grebow. Trajectory Design in the Earth-Moon System and Lunar South Pole Coverage. PhD thesis, Purdue University, 2010.

[26] D.J. Grebow, M.T. Ozimek, and K.C. Howell. Advanced modeling of optimal low-thrust lunar pole-sitter trajectories. Acta Astronautica, 67(7-8):991–1001, October-November 2010.

[27] A.L. Herman. Improved Collocation Methods with Application to Direct Trajectory Optimization. PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[28] A.L. Herman and B.A. Conway. Direct optimization using collocation based on high-order gauss-lobatto rules. Journal of Guidance, Control, and Dynamics, 19(3):592–599, May-June 1996.

[29] J.F.C. Herman. Preliminary mission analysis for a far-side, highly inclined solar observatory using electric propulsion. Master's thesis, Delft University of Technology, 2012.

[30] J.F.C. Herman, B.A. Jones, G.H. Born, and J.S. Parker. A Comparison of Implicit Integration Methods for Astrodynamics. AIAA/AAS Astrodynamics Specialist Conference, Hilton Head, South Carolina, 2013.

[31] J.F.C. Herman and J.S. Parker. Low-Energy, Low-Thrust Transfers Between Earth and Distant Retrograde Orbits About the Moon. 38^{th} Annual Guidance and Control Conference, Breckenridge, CO, January 30 - February 4, 2015.

[32] J.F.C. Herman, J.S. Parker, B.A. Jones, and G.H. Born. High-Speed, High-Fidelity Low-Thrust Trajectory Optimization through Parallel Computing & Collocation Methods. AAS 15-298, AIAA/AAS Spaceflight Mechanics Meeting, Williamsburg, Virginia, January 11-15, 2015.

[33] K. Hill and B.A. Jones. TurboProp Version 4.0. Colorado Center for Astrodynamics Research, May 2009.

[34] M.R. Ilgen. A hybrid method for computing optimal low thrust OTV trajectories. AAS/AIAA Spaceflight mechanics meeting, Cocoa Beach, FL, 1994.

[35] W.S. Koon, M.W. Lo, J.E. Marsden, and S.D. Ross. Dynamical Systems, the Three-Body Problem and Space Mission Design. California Institute of Technology, 2006. http://www.cds.caltech.edu/ koon/book/KoLoMaRo_DMissionBk.pdf.

[36] C.R. Koppel, F. Marchandise, and M. Prioul. The SMART-1 electric propulsion subsystem. Regulation, 33(July):1–10, 2005.

[37] S. Lee, R.P. Russell, W. Fink, R.J. Terrile, A.E. Petropoulos, and P. von Allmen. Low-thrust mission trade studies with parallel, evolutionary computing. In IEEE Aerospace Conference, Big Sky, Montana,, March 2006.

[38] T.T. McConaghy. GALLOP Version 4.5 User's Guide. Purdue University, 2005.

[39] T.T. McConaghy and J.M. Longuski. Parameterization Effects on Convergence when Optimizing a Low-Thrust Trajectory with Gravity Assists. AIAA/AAS Astrodynamics Specialist Conference, Providence, Rhode Island, 2004.

[40] G. Mingotti, F. Topputo, and F. Bernelli-Zazzera. Low-energy, low-thrust transfers to the moon. Celestial Mechanics and Dynamical Astronomy, 105(1-3):61–74, 2009.

[41] G. Mingotti, F. Topputo, and F. Bernelli-Zazzera. Numerical methods to design low-energy, low-thrust sun-perturbed transfers to the moon. Proceedings of the 49$^{\text{th}}$ Israel Annual Conference on Aerospace Sciences, pages 1–14, 2009.

[42] G. Mingotti, F. Topputo, and F. Bernelli-Zazzera. Efficient invariant-manifold, low-thrust planar trajectories to the moon. Communications in Nonlinear Science and Numerical Simulation, 17(2):817 – 831, 2012.

[43] N. Parrish. Accelerating Lamberts Problem on the GPU in MATLAB. Senior project report, California Polytechnic State University, Aerospace Engineering Department, San Luis Obispo, 2012.

[44] NVIDIA. CUDA Zone. https://developer.nvidia.com/cuda-zone, Last accessed March 26, 2014.

[45] Office of the Chief Technologist. TA02 - In-Space Propulsion Systems. NASA, 2012.

[46] M.T. Ozimek. Low-Thrust Trajectory Design and Optimization of Lunar South Pole Coverage Missions. PhD thesis, Purdue University, 2010.

[47] M.T. Ozimek, D.J. Grebow, and K. C. Howell. A Collocation Approach for Computing Solar Sail Lunar Pole-Sitter Orbits. AIAA/AAS Astrodynamics Specialist Conference, Pittsburgh, Pennsylvania, 2009.

[48] M.T. Ozimek, D.J. Grebow, and K.C. Howell. Solar sails and lunar south pole coverage. In AIAA/AAS Astrodynamics Specialist Conference, Honolulu, Hawaii, August 18-21 2008.

[49] J.S. Parker and C.J. Bezrouk. Low-Energy Transfers to Distant Retrograde Orbits. AAS 15-311, AIAA/AAS Spaceflight Mechanics Meeting, Williamsburg, Virginia, January 11-15, 2015.

[50] Nano Patents and Innovations. New Electric Propulsion System for Cube Satellites Developed at MIT-System Relies on Electrospraying. http://nanopatentsandinnovations.blogspot.com/2010/02/new-electric-propulsion-system-for-cube.html, last accessed February 23, 2012.

[51] R. B. Roncoli and K. K. Fujii. Mission design overview for the gravity recovery and interior laboratory (GRAIL) mission. In AIAA/AAS Astrodynamics Specialist Conference, number AIAA 2010-8383, Toronto, Ontario, Canada, August 2–5, 2010. AIAA/AAS.

[52] C. Russell and C. Raymond. The Dawn Mission to Vesta and Ceres. Space Science Reviews, 163:3–23, 2011.

[53] R. Russell and J. Christiansen. Adaptive Mesh Selection Strategies for Solving Boundary Value Problems. SIAM Journal on Numerical Analysis, 15(1):59–80, 1978.

[54] R.P. Russell and N. Arora. Global point mascon models for simple, accurate and parallel geopotential computation. Journal of Guidance, Control, and Dynamics, 35(5):1568–1581, 2012.

[55] S. Naffziger. Technology Impacts from the New Wave of Architectures for Media-rich Workloads. http://www.monolithic3d.com/uploads/6/0/5/5/6055488/sam_naffziger_vlsi_keynote.pptx, Last accessed March 22, 2014.

[56] B. Sanders, L. Van Vliet, and F. T. Nardini. Development of MEMS based Electric Propulsion. 2010.

[57] J. Sanders and E. Kandrot. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, 1st edition, 2010.

[58] J. Schoenmaekers, J. Pulido, and J.L. Cano. SMART-1 - Moon mission: Trajectory design using the Moon gravity. 1999.

[59] S. Seager et al. The Exo-S Probe Class Starshade Mission. In Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, in press, 2015.

[60] J.A. Sims, P.A. Finlayson, E.A. Rinderle, M.A. Vavrina, and T.D. Kowalkowski. Implementation of a low-thrust trajectory optimization algorithm for preliminary design. In AIAA/AAS Astrodynamics Specialist Conference Proceedings, Keystone, Colorado, volume 3, pages 1872–1881, August 20-25, 2006.

[61] J.A. Sims and S.N. Flanagan. Preliminary Design of Low-Thrust Interplanetary Missions. In AAS/AIAA Astrodynamics Specialist Conference, Girdwood, Alaska, August 16-18, 1999.

[62] S. De Smet. Preliminary mission analysis of a crewed mars flyby solar electric propulsion mission. Master's thesis, Delft University of Technology, 2014.

[63] E. M. Standish. JPL Planetary and Lunar Ephemerides, DE405/LE405. Technical Report IOM 312.F–98–048, Jet Propulsion Laboratory, California Institute of Technology, August 1998.

[64] N. Strange, D. Landau, T. McElrath, G. Lantoine, T. Lam, M. McGuire, L. Burke, M. Martini, and J. Dankanich. Overview of Mission Design for NASA Asteroid Redirect Robotic Mission Concept. 33rd International Electric Propulsion Conference (IEPC2013), Washington, D.C., October 6-10, 2013.

[65] V. Szebehely. Theory of Orbits: The Restricted Problem of Three Bodies. Academic Press, New York, 1967.

[66] B. Tapley, J. Ries, S. Bettadpur, D. Chambers, M. Cheng, F. Condi, B. Gunter, Z. Kang, P. Nagel, R. Pastor, T. Pekker, S. Poole, and F. Wang. GGM02 - An improved Earth gravity field model from GRACE. Journal of Geodesy, 79:467–478, November 2005.

[67] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming, 106(1):25–57, 2006.

[68] K.F. Wakker. Lecture Notes Astrodynamics - II. Delft University of Technology, 2007.

[69] G.J. Whiffen. Mystic: implementation of the static dynamic optimal control algorithm for high-fidelity, low-thrust trajectory design. In AIAA/AAS Astrodynamics Specialists Conference, Keystone, Colorado, August 21-24, 2006.

[70] M. T. Zuber, D. E. Smith, S. W. Asmar, A. S. Konopliv, F. G. Lemoine, H. J. Melosh, G. A. Neumann, R. J. Phillips, S. C. Solomon, M. M. Watkins, M. A. Wieczorek, J. G. Williams, J. C. Andrews-Hanna, J. W. Head, W. S. Kiefer, I. Matsuyama, P. J. McGovern, F. Nimmo, G. J. Taylor, R. C. Weber, S. J. Goossens, G. Kruizinga, E. Mazarico, R. S. Park, and D.-N. Yuan. Gravity Recovery and Interior Laboratory (GRAIL): Extended Mission and Endgame Status. In Lunar and Planetary Institute Science Conference Abstracts, volume 44 of Lunar and Planetary Institute Science Conference Abstracts, page 1777, mar 2013.

Appendix A

Code Architecture

This appendix goes into specific detail of the code that is written for this research, describing the structure and flow of the various elements, and discussing some specifics on inputs and outputs. Figure A.1 presents the general architecture of *Maverick*. This figure assumes that only the serial CPU version of the 12^{th} order Gauss-Lobatto scheme is used, and this is indeed the only version that was fully maintained until the conclusion of this research. Figure A.2 shows the structure if the 4^{th} order method and GPU versions of both methods are also included. The red blocks in this figure represent the Python front- and back-end, which sets up the variables that define the problem at hand, whether that is a simple spiral problem in GEO, or an elaborate LEO-DRO transfer. This is then handed down to be declared as a problem in C, where it then enters the NLP solver through the appropriate interface. The NLP solver then iterates until convergence (or some other termination condition), during which it calls the user functions many times. Once the NLP solver hands the problem back, a mesh refinement iteration is performed, and then the Python back-end decides how to proceed. All blocks are discussed in more detail below. The exact inputs and outputs of functions are not discussed here for the sake of brevity, for which the actual code should be consulted. The general functions that are used are all called out, so the reader knows how to navigate the different files of source code.

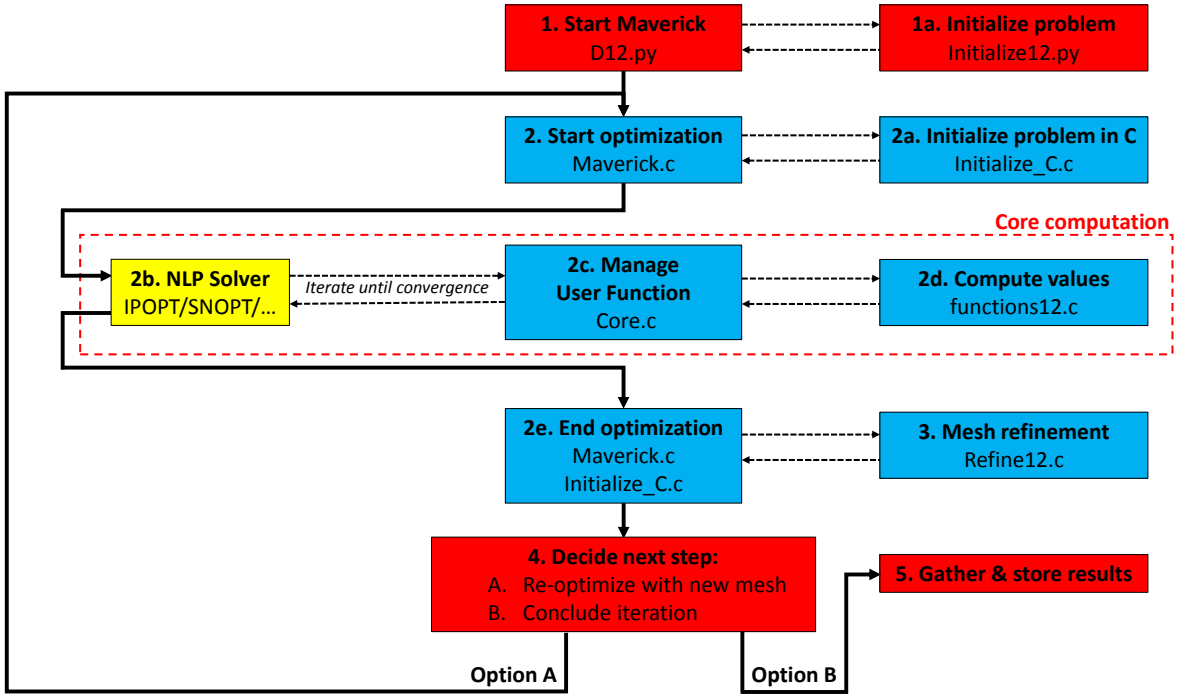


Figure A.1: The structure and flow of the code that defines *Maverick* when set to use the serial CPU version of the 12^{th} order Gauss-Lobatto scheme. Bold text within blocks indicates the general function, whereas plain text within blocks shows the filename of the actual file(s) used for this. Red blocks represent Python code, blue blocks represent C code, and the yellow block is the NLP solver, in whatever language the selected NLP solver uses. The dashed arrows indicate blocks that are essentially subroutines of the blocks that the dashed arrows originate from. The solid arrows indicate significant transitions from one phase in the program to another. The red dashed box outlines where more than 99% of the computational effort occurs.

Block 1: Start Maverick This is an initialization of the problem in Python that can be very simple or very elaborate. The function R_init() in D12.py is called, which calls the appropriate initialization function (each of which correspond to a different problem) in Initialize12.py (Block 1a). Initialize12.py will somehow get an initial guess, which can be as simple as taking a random number of points, or as complicated as a numerical propagation of an estimated control history. This has to be defined by the user somehow for every problem at hand. It could also be loaded, for example from a previous mesh refinement iteration. This is then fed into the Transcribe() function, where this initial guess is formulated to match the very specific transcription described in Chapter 3. This transcription is fed back to D12.py, which then stores all the relevant variables

to disk at full precision, to be accessed later by the C source code. An actual Python-C interface could be written to accomplish the same goal, but this is not done at this time. After all necessary variables are written to disk, the C code is called from Python.

Block 2: Start optimization Maverick.c is essentially just an interface for the chosen NLP solver, in this case IPOPT. The main() function in Maverick.c calls the Initialize() function from Initialize_C.c (Block 2a), which knows where to find all the relevant variables and how to store them as the appropriate C datatypes. This includes not only the variables for the problem formulation, but also a variety of settings for the NLP solver. Once this is done, Maverick.c feeds this information to the NLP solver in the correct format and starts it. In order to tie into a different NLP solver, the Initialize_C.c file could essentially be used unchanged (except perhaps the input of NLP solver options), and only Maverick.c would have to be (partially) rewritten.

Block 2b: NLP solver The NLP solver now takes control and runs through as many iterations as necessary/appropriate based on the different tolerances and termination conditions that are set. In order to acquire the values it needs, it needs to communicate with the 'user functions', Block 2c and 2d. Once the conditions for termination are met, the NLP solver will hand the results back to Maverick.c, as represented by Block 2e.

Block 2c & 2d: User functions The files Core.c and functions12.c could be combined into a single file. However, the NLP solver only communicates directly with Core.c. Core.c could then decide if it should send the request for updated values to the 4^{th} or 12^{th} order scheme, or the CPU or GPU versions of these schemes. By keeping Core.c as a separate file, such switches are easily accommodated. In the case discussed here, only the CPU version of the 12^{th} order scheme is applicable. Core.c does compute the value of the cost function and the cost derivatives, since this is a very similar process regardless of what method is used. The values of the constraint functions and their derivatives are computed by functions12.c (or the corresponding file for an alternate method,

as shown in Figure A.2). Block 2d is also where a variety of different force models are called, either user-written or existing ones from TurboProp.

Block 2e: End optimization Maverick.c and Initialize.c perform the proper memory deallocations and store the appropriate values to disk. Maverick.c also calls the main() function of Refine12.c (Block 3), which uses the stored information to perform the Mesh refinement process. Maverick.c then hands the problem to the Python back-end.

Block 3: Mesh refinement Based on the outcome of the optimization, as well as tolerances set in the Python front-end, Refine12.c determines if an equidistribution iteration or an error reduction iteration is required, or if the error has sufficiently converged. It stores this information, along with the redistributed mesh, for the Python back-end. Refine12.c calls the same force models as functions12.c, but uses the information in a completely different manner, as described in Chapter 3.

Block 4: Decide next step The Python back-end considers the outcome of the optimization as well as the mesh refinement process, in combination with user-specified tolerances, to determine what happens next. It may be that the problem is handed back to Block 2, for an optimization with a new mesh. Alternatively, if the problem meets all tolerances, it is concluded and stored to disk (Block 5). Additionally, some plots may be provided, or some other form of output.

Outer loops It may be that the user wants to run a similar problem many times, for example for different flight times, as is the case for many of the results presented in this thesis. In that scenario, Block 1 would be repeatedly called, with slightly different inputs to account for the different flight times. Block 1a could be adapted so that the initial guess builds on the previous optimal solution, which is indeed what was done for the results presented in Chapter 7.

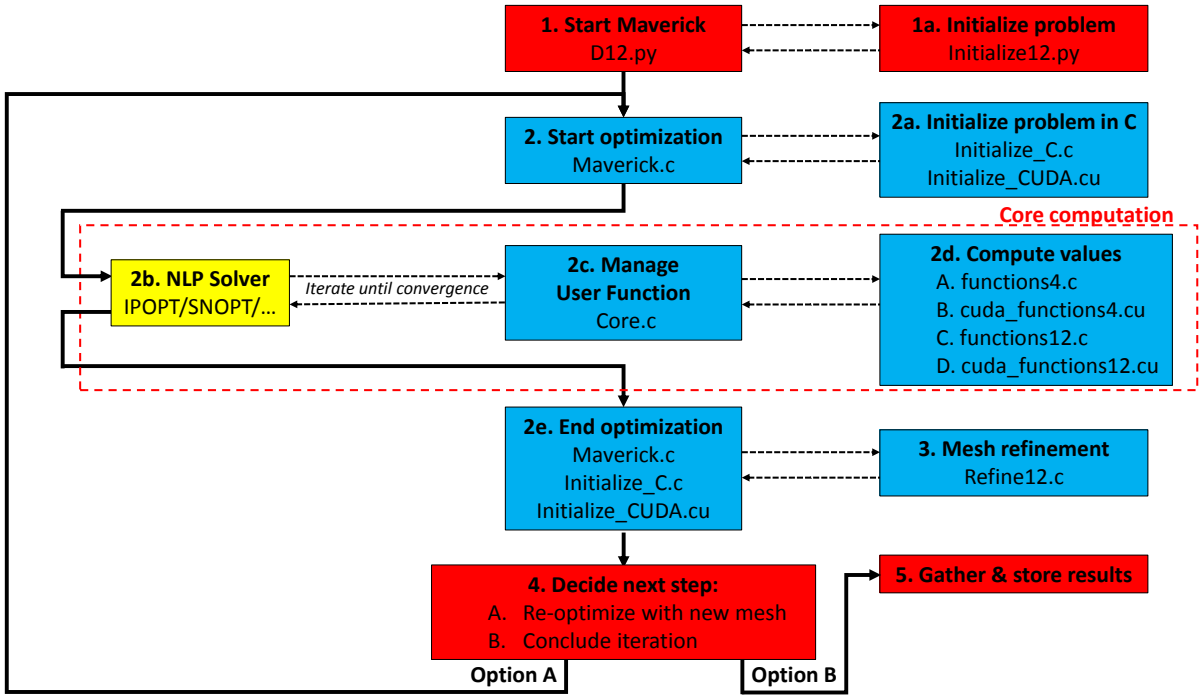


Figure A.2: The structure and flow of the code that defines *Maverick*, with all different methods included. Bold text within blocks indicates the general function, whereas plain text within blocks shows the filename of the actual file(s) used for this. Red blocks represent Python code, blue blocks represent C code, and the yellow block is the NLP solver, in whatever language the selected NLP solver uses. The dashed arrows indicate blocks that are essentially subroutines of the blocks that the dashed arrows originate from. The solid arrows indicate significant transitions from one phase in the program to another. The red dashed box outlines where more than 99% of the computational effort occurs.

Appendix B

Performance Variations

This Appendix summarizes the most significant variations that were attempted to improve the division of runtime between the NLP solver and the 'user code', as described in Chapter 4. This list is not completely exhaustive, but lists the majority of attempted modifications and includes all of the major attempts. None of these improved performance with respect to what is presented in Chapter 4, and indeed in many cases it was worse, or at the very least resulted in worse convergence or values for the resulting cost function. It is recorded here to help others avoid repeating these efforts.

Changing the piecewise continuous polynomial formulation The formulation as presented in Chapter 3 uses the exact same input states at the node points to define the segment to the 'left' as well as to the 'right' of that node point. This automatically makes two adjacent segments continuous, but it also means that changing one of these input states affects two segments, even if only one segment is in need of change. This is not required of course. In one of the more dramatic modifications, the entire formulation was rewritten to define every node point twice, once as the end of one segment and once as the beginning of the next segment. This increases the number of input states, and it also means constraints must be added to ensure the segments are continuous, but it results in a significantly more sparse formulation. It also means a single segment can be adjusted at a time by changing a node point. Unfortunately, this did not significantly alter performance, and typically led to a poorer convergence, resulting in reverting back to the original formulation.

Manually introducing slack variables In an attempt to more directly control the constraint values, all constraints were rewritten as equality constraints with a slack variable in the optimization state. This is similar to what the NLP solver is doing under the hood, but it takes that effort into the user code. It also allows more control over the scaling of these slack variables. While this in some cases led to faster convergence, it did not noticeably alter the relative runtime between NLP solver and user code. It remains an option to use this within *Maverick*.

Changing control vector formulation Two significant modifications were tried for the formulation of the control vector. One related to using a constant thrust vector per segment, or one that is linearly interpolated between node points, as briefly discussed in Chapter 3. This did not dramatically affect the relative runtime, but in general the constant thrust vector did lead to better overall convergence, and it became the baseline for *Maverick*, although it is easily modified. Additionally, the option was introduced to switch to a VNC reference frame, which is described in Chapter 3. This slightly improved the runtime balance, as well as the overall convergence, when the trajectory orbited a single central body, such as the GEO test cases from Chapter 4. For the LEO-DRO transfers, where the definition of the VNC frame is less relevant, these benefits were not observed. It remains an option within *Maverick* to formulate the control vector in a VNC frame or simply in an inertial frame with respect to the chosen central body.

Complex derivatives finite differencing In an attempt to give the NLP solver better derivative information, all numbers in *Maverick* were redeclared as complex variables, which enables the use of complex finite differencing. This deals much better with the numerical precision issues that finite differencing can encounter. Short of writing out all the analytical derivatives, this results in the best possible derivative information. While a small improvement in overall convergence was observed, the relative runtime between the NLP solver and user code did not improve. Furthermore, complex numbers require twice the memory as a real number, which is not appealing when GPUs are used due to their operations being sensitive to frequent memory access. As a result, the complex

finite differencing modifications were abandoned.

Adapting problem size In a more generic attempt, the same problem was solved for a variety of problem sizes, ranging from very few segments to an abundance. No mentionable differences in relative runtime were detected.

Scaling factors & Tolerances Similarly, an extensive trade study was performed in modifying the scaling for all aspects of the problem, as well as the tolerances set for feasibility or convergence. This refers to constraints, input states, and cost function. Even within those distinctions, different scalings would be applied to position elements, velocity elements, mass elements, and so on. It was attempted to weight the impact of the control vector, to avoid the NLP solver trying to focus on finding a way to cheat the dynamics, and instead focus on the input states that actually relate to something the optimizer can control. Many of these attempts resulted in terrible convergence. None of them produced a better runtime balance combined with an acceptable convergence.

Changing NLP solver & Linear solvers For a time, *Maverick* was written to use either SNOPT or IPOPT. Eventually, it became apparent that IPOPT consistently outperformed SNOPT, and the SNOPT interface was no longer updated. This was one of the most successful changes to the runtime balance of the NLP solver and the user code. IPOPT allows the use of a variety of linear solvers. While some of these did have a better relative runtime than discussed in Chapter 4, it inevitably came with worse overall performance, making ma86 the solver of choice.

Adapting NLP solver settings Both SNOPT and IPOPT have a variety of settings and tolerances that let the user control a wide range of elements, such as the scaling that happens within the solver, a variety of termination conditions, or the method used to select the next step in the optimization, and many more other aspects. These were widely experimented with, but typically the best performance resulted from the default settings.

Using optimized BLAS libraries When installing IPOPT, it can be compiled with BLAS libraries of the user's preference. Some comparison was done between the standard BLAS routines that come with IPOPT and OpenBLAS, but there were no detectable differences. It is likely that the super-sparse nature of this formulation results in a large number of small blocks of data being fed to the BLAS functions, limiting the benefit of optimized functions, with most cost in the overhead of accessing these functions.

Using parallelized elements in the NLP solver Related to the previous points, IPOPT supports the use of parallelized BLAS libraries, as well as some parallelized linear solvers. Using this, again no detectable performance difference was found. The explanation for this is similar to that of using optimized BLAS libraries; while super-sparse formulations have a lot of positive aspects, it also results in many operations happening on small blocks of data, many times, instead of these occuring a handful of times on enormous blocks of data. For denser formulations, the gains would likely be more noticeable.