TRAJECTORY OPTIMIZATION USING COLLOCATION AND EVOLUTIONARY PROGRAMMING FOR CONSTRAINED NONLINEAR DYNAMICAL SYSTEMS

by

BRANDON MERLE SHIPPEY

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN AEROSPACE ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2008

To all the members of my family,

who have supported and encouraged me throughout my education.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Dr. Kamesh Subbarao, for providing me the opportunity to pursue graduate research. Dr. Subbarao demonstrated a contagious enthusiasm for the material he taught me in his astronautics classes when I was an undergraduate, and subsequently caused me to expand the limits of my understanding in his graduate classes.

Second, I would like to thank my committee members, Dr. Kent Lawrence and Dr. Atilla Dogan, for kindly agreeing to serve on my committee. Dr. Lawrence's finite element analysis class helped me to better understand my research topic. Dr. Dogan has proven to be a patient teacher, while still expecting full understanding of the advanced flight mechanics material.

I humbly acknowledge the Department of Mechanical and Aerospace Engineering for providing the financial support I received as a graduate teaching assistant and research assistant.

Lastly, I would like to express my appreciation for my family. My parents, Jimmy and Karen Shippey, have created numerous opportunities for me to pursue my goals and provided me with moral support and encouragement throughout my education. My brothers, Sheldon and Jason, reminded me to have fun every once in a while.

April 14, 2008

ABSTRACT

TRAJECTORY OPTIMIZATION USING COLLOCATION AND EVOLUTIONARY PROGRAMMING FOR CONSTRAINED NONLINEAR DYNAMICAL SYSTEMS

Brandon Merle Shippey, MS

The University of Texas at Arlington, 2008

Supervising Professor: Kamesh Subbarao

Trajectory design and optimization has a broad variety of applications in fields such as aerospace and electrical engineering. The solution of a trajectory that minimizes a cost function subject to nonlinear differential equations of motion and various types of constraints may be obtained by the methods of optimal control theory.

A framework is presented for numerical solution of the optimal control problem. The solution is converted to that of a constrained discrete parameter optimization problem. Direct collocation and nonlinear programming are used to perform a local gradient-based search for the optimal solution. A genetic algorithm combined with a shooting method conducts a global search of the solution space to provide a near-optimal, near-feasible initialization for the nonlinear program.

The framework is applied to three minimum-time case studies: *i*) a path planning problem for two mobile robots with obstacle avoidance; *ii*) an aircraft turning maneuver; *iii*) a low-thrust interplanetary transfer.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

A broad assortment of problems in industry and academia focus on the design of the time-histories of the states of dynamical systems. The need to design these trajectories in such a way that their solutions are feasible and realistic, while maintaining cost effectiveness, has lead to numerous developments in the methods available to solve the constrained trajectory optimization problem. This section provides a brief discussion of the aspects of the trajectory optimization problem, numerical formulations for its solution, and popular computational algorithms. The relationships of the ideas presented here are illustrated in the map of trajectory optimization methods shown in Figure 1.1. Later in the section, the method applied to problems in this thesis work is discussed and the organization of the document is provided.

1.1 Trajectory Design and Optimization

In its most analytical form, this problem and its variants are addressed in the optimal control literature, [1] for example. Given the equations of motion, boundary conditions, various types of constraints (equality, inequality; box constraints, general path constraints), and performance index or cost function, the solution of the optimal trajectory is obtained through the calculus of variations. First, a cost function is formed, augmented with Lagrange multipliers (or costates) associated with the constraints and state differential equations of the system. Defining a convenient Hamiltonian, the first variation of the cost function due to differential changes in the control inputs is written. Then, costate differential equations and boundary conditions are chosen to simplify this

expression. The process of writing a problem in terms of the original variables and Lagrange multipliers (or states and costates) is often referred to as "dualization" [2].

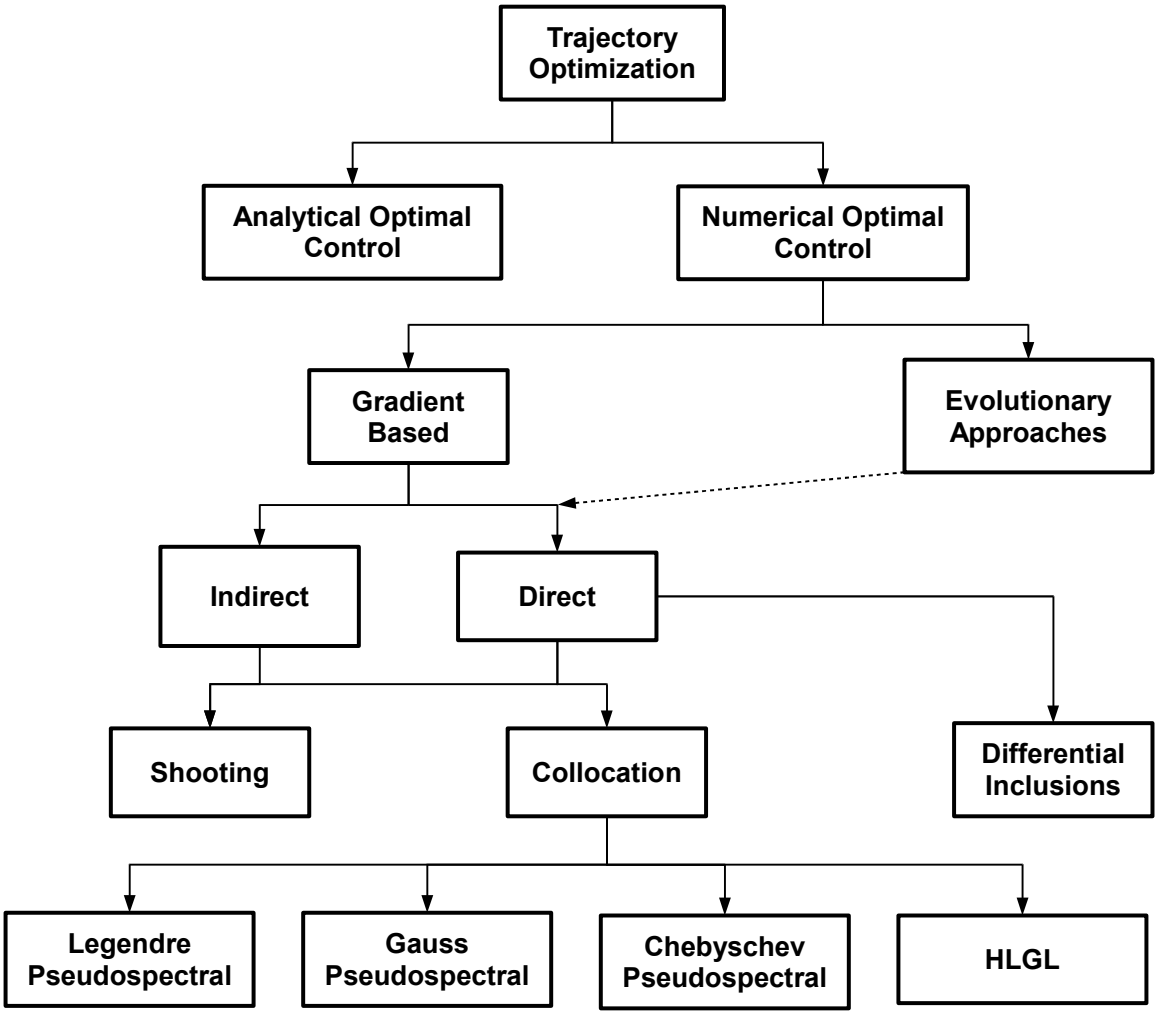


Figure 1.1: Map of trajectory optimization methods.

Certain classes of trajectory optimization problems are conveniently solved by analytical optimal control. When inequality constraints are applied to the control variables, then the solution of the problem comes from the application of Pontryagin's Minimum Principle [1]. If the system is entirely linear, including upper and lower bounds applied

only to the controls then a minimum-time control history has a predictable "bang-bang" behavior in which the control value is saturated with discontinuous jumps between the extremes [1]. However, if the problem to be addressed does not fit into such a special category, then the solution may not be so predictable. For example, with systems having complicated nonlinear dynamics of high dimension and many types of constraints, the application of Pontryagin's Minimum Principle (or analytical solutions in general) can become very difficult. For this reason, numerical solutions of the optimal control problem have rapidly developed.

1.2 Discussion of Numerical Optimal Control Methods

A few of the popular numerical optimal control formulations are discussed here, following the survey of [2]. Two distinct branches of numerical optimal control have arisen. Both branches attempt to minimize cost functions and constraint violations using discrete approximations of the parameters of the system. This is performed by some gradient-based or evolutionary algorithm, which are discussed shortly.

The first branch is that of indirect methods. An indirect method discretizes the system in its dualized form. That is, the states and costates are both solved. While indirect methods typically enjoy greater accuracy than direct methods, three major problems arise. Firstly, the analytical forms of the optimal control necessary conditions must be expressed, including the costate differential equations, the Hamiltonian, the optimality condition, and transversality conditions. This is the same difficulty restricting the use of an analytical approach. Numerically speaking, this also makes the problem size large due to discretization of the costates. Secondly, the analyst must guess certain aspects of the optimal control solution, such as the portions of the time domain containing constrained or unconstrained control arcs, when using a gradient-based method. Third, since the

gradient based approaches also require initial guesses for the costates, the domain of convergence can be very small.

The second branch is that of the direct methods. Direct methods are so called because the system is discretized in its original form without the need to express the optimal control necessary conditions and costate equations. Though direct methods are less accurate than indirect methods, the fact that they are easier to implement, have a larger domain of convergence, and have reduced problem size makes them very attractive. Unless otherwise stated, further discussion is restricted to direct methods.

Whether a direct or indirect method is chosen, the states must be integrated from some boundary condition or the equations of motion must be enforced through constraints.

One strictly direct method is that of differential inclusions. The method of differential inclusions enforces the equations of motion at each discrete node by applying inequality constraints on the state derivatives [3]. These inequality constraints are obtained by substituting the upper and lower bounds on the control vector into the equations of motion. When the inequality constraints are met, the states at one node are said to lie in the attainable set at that node given the state values at an adjacent node and the set of admissible controls. The advantage given by differential inclusions is that it effectively eliminates the explicit dependence on control values at each node [3]. However, methods such as this can become numerically unstable and the formulation can be problem dependent [2].

Shooting methods use marching integration to calculate the state histories given the control histories of the system. The gradient- or evolutionary-based algorithm then evaluates the objective function and constraint violations at each discrete node. Shooting methods are attractive because the equations of motion are enforced automatically by

the marching integration. This effectively reduces the size of the problem by reducing the number of constraints that must be applied as compared to collocation methods.

Collocation methods enforce the equations of motion through quadrature rules or interpolation [4]. An interpolating function (interpolant) is solved such that it passes through the state values and maintains the state derivatives at the nodes spanning one interval (or subinterval) of time. The interpolant is then evaluated at points between the nodes, called collocation points. At each collocation point, an equality constraint is formed, equating the interpolant derivative to the state derivative function, thus ensuring that the equations of motion hold (approximately) true across the entire interval of time [5].

The defining characteristics of a particular collocation method are, firstly, the selection of the interpolating function, and secondly, the selection of the nodes and collocation points within the time interval. One of the simplest methods of collocation is the Hermite-Simpson method [5][6][7]. This method is so called because a third-order Hermite interpolating polynomial is used locally within many intervals, each solved at the endpoints of an interval and collocated at the midpoint. When arranged appropriately, the expression for the collocation constraint is the same as the Simpson integration rule. A generalization of the method to use the n-th order Hermite interpolating polynomial, and choosing to take the nodes and collocation points from a set of Legendre-Gauss-Lobatto points defined within the local time intervals, gives rise to the Hermite-Legendre-Gauss-Lobatto (HLGL) method [5].

There are many other discretizations available for collocation. The pseudospectral methods use global orthogonal Lagrange polynomials as the interpolants while the nodes are selected as the roots of the derivative of the named polynomial, such as the Legendre-Gauss-Lobatto (Legendre pseudospectral) or the Chebyshev-Gauss-Lobatto (Chebyshev pseudospectral) points [8][9]. Since these methods use global interpolants defined over

the entire duration of the trajectory, the Gauss-Lobatto nodes are clustered near the endpoints. In such a situation, there may be difficulties characterizing fast dynamics near the midpoint of the trajectory. Refinement of the grid to accurately capture fast dynamics may increase the size of the discretized problem unnecessarily [10]. For this reason, a method with a more uniform grid, such as the HLGL method, may be desirable.

Needless to say, each method may be more appropriate under certain conditions. The accuracy of such discretizations has been compared in the literature, in [11] for example. However, the accuracies of individual methods are not rigorously examined in this thesis since the main focus is not upon the collocation method itself, but upon the combination of a genetic algorithm shooting method and a collocation technique with nonlinear programming (NLP), which is to be discussed in the next section. The overall approach is discussed later.

1.3 Computational Algorithms

1.3.1 Nonlinear Programming

Of the few types of computational methods commonly used to solve trajectory optimization problems, gradient based methods such as nonlinear programming seems to be the most popular. Nonlinear programming, also commonly referred to as a quasi-Newton algorithm (QNA), is the solution to a constrained discrete parameter optimization problem analogous to the optimal control solution. Considering that the states and controls take discrete values at the nodes in a direct or indirect method, the cost function of the dynamical system can be approximated by some function of the nodal state and control values, system parameters, and/or final time. Adjoining the cost function with any constraints (problem dependent or due to collocation, differential inclusions, etc) allows for derivation of the necessary and sufficient conditions to find a minimum in the cost

function with respect to the time histories of the states and controls. A quasi-Newton algorithm with line search or sequential quadratic programming is capable of solving the problem given a sufficiently near-optimal, near-feasible initial guess of all parameters of the system [2].

Since NLP uses gradient information, it is often capable of relatively quick convergence and very accurate results (within the accuracy of the discretization). They also have well-defined convergence criteria. This has lead to their popularity and the development of many individual software packages. These algorithms have been available for quite some time, such as NPSOL and the updated NZSOL used by the authors of [6]. The more recent implementation, SNOPT is used by the authors of [5]. Another available package is the function **fmincon()** of MATLAB's Optimization Toolbox, which is a general, multi-purpose constrained parameter optimizer for small, medium, and large problems [12].

The most noticeable problem with gradient based methods is that they require an initial guess of all parameters of the system. Since all nodal state and control values are parameterized, the analyst must have some a priori knowledge of the optimal trajectory. The consequences of poor initial guessing are failure to converge or convergence upon a non-global optimum in the cost function. A method to address this issue was proposed by the authors of [13], which leads to discussion of an alternative approach to solving the discrete direct or indirect trajectory optimization formulations: "evolutionary programming" or genetic algorithms.

1.3.2 Genetic Algorithms

A genetic algorithm (GA) simulates evolution upon an initial population of random candidate solutions [14]. Each candidate solution is described by an array of binary bits or real numbers (called a chromosome), which is decoded into a meaningful set of

parameters (called a phenotype). The phenotype belonging to a candidate is simply an array containing the values of the variables (e.g. nodal state or control values, final time, etc.) over which an objective function, also referred to as a cost function or penalty function, is to be minimized. Based on the objective function prescribed by the analyst, each chromosome is given a rank to describe the desirability of the candidate in relation to the others in the population. Pairs of chromosomes are randomly chosen by a selection function, giving those with better rank a higher probability of selection and, therefore, more opportunities to participate in crossover (analogous to mating). Crossover occurs, between the chromosomes of each pair, by exchanging binary bits (for binary-encoded chromosomes) or parameter values (for real-encoded chromosomes). This creates new individuals which may have more desirable (lower) objective values. It should be noted that there is a wide variety of crossover functions. Some of the chromosomes undergo mutation, randomly changing some of the binary bits or adjusting real parameter values, to ensure diversity and give the algorithm a non-zero chance of locating the global optimum of the objective function. The new candidates are reinserted into the population, replacing old ones. The entire process is then repeated for a pre-set number of generations. Performance of the GA is adjusted through population size, number of generations, and parameters including the probability that candidate chromosomes will be mutated or reinserted, etc.

The genetic algorithm has some features that make it more attractive than NLP solvers in some regards. Because the GA utilizes randomness in creating the candidate solutions, it has a greater probability of locating the global optimum than a gradient-based method when the cost function has multiple local extrema. Randomly generating the population also frees the analyst of providing an initial guess. Furthermore, the entire solution space of a problem can be searched within the precision of the encoding of parameters in the chromosomes. However, the GA has drawbacks which, to some

authors, make it unacceptable as a primary means of trajectory optimization [2]. Firstly, because the GA utilizes randomness, the population is not guaranteed to have sufficient diversity to find an optimal solution meeting all constraints. Second, there are no well-defined convergence criteria (i.e. no necessary and sufficient conditions), such as those used in NLP algorithms.

Depending on the type of problem formulation/discretization, the resulting solution may be nearly optimal and nearly feasible. While unsatisfactory as a final result, such a trajectory is likely to be better than the guess of an analyst who has no prior knowledge of the optimal trajectory. This suggests that the GA may provide an initialization good enough for the NLP to locate the desired globally optimal, feasible trajectory. This idea is the basis for the innovation of [13].

The original work, [13], was intended to be a solution of the multi-point boundary value problem (MPBVP) for systems with Mayer cost functions, with linearly appearing controls. The final numerical solution of the optimality conditions was initialized using guesses for the switching times, generated by examining the results of an NLP shooting technique initialized, in turn, by a GA shooting technique. The problem solved was the reorientation of an inertially symmetric spacecraft.

A different approach to the solution of a similar problem, combining the GA and NLP, was implemented by the authors of [15]. They performed a GA search of the switching structure and switch times (pre-supposing that the control should be bang-bang) without using a dense grid, as was used in [13]. The resulting bang-bang control and state histories (integrated via a 6-th order Runge-Kutta method) were then used to initialize the direct Legendre pseudospectral collocation method to be solved by the NLP. The method reduced the size of the original GA discretization as compared to [13], but still only applied to problems where the optimal control solution was bang-bang and the cost function was in Mayer form.

Other authors, such as those of [16], have opted to concentrate on modifying the functions of the GA to improve performance. Though the goal of that work was to improve the GA functions themselves, the framework is relevant because it allows a more general representation of the control history in a direct GA shooting technique while using the result to initialize the NLP solution of a direct collocation method. However, fidelity of the collocation result was not paramount and the authors required only the trapezoidal integration rule for collocation.

1.4 Contributions of the Thesis

The desired outcome of the research presented in this thesis is an autonomous, accurate means of solving trajectory optimization problems. For accuracy, an adjustable-order collocation technique is implemented, solved by nonlinear programming. This is the n-th order Hermite-Legendre-Gauss-Lobatto collocation technique. The NLP is initialized by interpolating the output of a genetic algorithm that attempts to solve a direct shooting formulation of the same trajectory optimization problem, without making limiting assumptions about the form of the controls. The reason for using a shooting technique in the GA phase is to avoid burdening the evolutionary process of the GA population with the task of meeting collocation constraints, though rigorous justification of this is a subject for future work.

The thesis is arranged as follows. Section 2.1 defines the optimal control problem as described in [5]. Section 2.2 provides an overview of the optimization method used in the thesis and describes certain aspects of the implementation. Section 2.3 formulates the Hermite-Legendre-Gauss-Lobatto collocation constraints used in the NLP phase of optimization, also described in [5]. The method is then extended to include 2nd-order and p-th-order dynamical information in the interpolating polynomial and the collocation constraints. Though the extensions are formulated, only the standard version is

implemented for this thesis. Section 2.4 briefly describes the shooting technique used in the GA (initial search) phase. A description of the handling of multiple continuous segments with inter-segment discontinuities or constraints is provided in Section 2.5.

Three application case studies are demonstrated in Chapter 3. The first is a minimum-time kinematic path planning problem for a multi-agent system of autonomous ground vehicles with obstacle avoidance. This problem demonstrates the ability of the method to sufficiently search the solution space of a cost function having many local minima. The second case study is a minimum-time turn-to-heading maneuver for an aircraft. That problem requires the satisfaction of linear inequality constraints on the states and nonlinear inequality constraints on the controls. The third case study is a minimum-time low-thrust interplanetary transfer from geosynchronous orbit to areosynchronous orbit (at Mars). This last case requires two changes of coordinates along the trajectory. Therefore, it is a multi-segment trajectory with inter-segment constraints. This problem provided the motivation for the development of the multi-segment algorithm. Finally, the Appendix contains detailed documentation of the MATLAB implementation of the method used in this thesis.

CHAPTER 2

METHOD OF TRAJECTORY OPTIMIZATION

2.1 Optimal Control Problem Formulation

The formulation of the optimal control problem follows that of [5].

2.1.1 Nonlinear Dyamics

The equations of motion are allowed take the form of general nonlinear differential equations,

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t) \tag{2.1}$$

where $\mathbf{x} \in \mathcal{R}^{n_x}$ is the state vector, $\mathbf{u} \in \mathcal{R}^{n_u}$ is the control input vector, and $\mathbf{p} \in \mathcal{R}^{n_p}$ is a vector of additional system parameters. The vectors $\mathbf{x}$, $\mathbf{u}$, and $\mathbf{p}$ belong to compact sets. The time is t and the final time shall be denoted as t_f. The state derivative function is $\mathbf{f} : \mathcal{R}^{(n_x+n_u+n_p+1)} \rightarrow \mathcal{R}^{n_x}$, a smooth vector field.

2.1.2 Cost Function

We will allow the scalar performance index to be a function of the final states, final time, and additional system parameters, as well as the integral of some function of the states, control inputs, time, and system parameters evaluated over the entire time domain,

$$J = \epsilon\left[\mathbf{x}(t_f), \mathbf{p}, t_f\right] + \int_{t_0}^{t_f} L\left[\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t\right] dt. \tag{2.2}$$

Equation (2.2) is the known as the Bolza cost function, where $\epsilon : \mathcal{R}^{(n_x+n_p+1)} \rightarrow \mathcal{R}$ and $L : \mathcal{R}^{(n_x+n_u+n_p+1)} \rightarrow \mathcal{R}$ are the Mayer and Lagrange components, respectively [2].

2.1.3 Endpoint Constraints

The states and time domain may be constrained at the initial and final conditions by providing upper and lower bounds to the general function $\mathbf{e}$ of arbitrary dimension,

$$\begin{aligned} \mathbf{e}_L^0 \leq \mathbf{e}[\mathbf{x}(t_0), t_0] \leq \mathbf{e}_U^0 \\ \mathbf{e}_L^f \leq \mathbf{e}[\mathbf{x}(t_f), t_f] \leq \mathbf{e}_U^f. \end{aligned} \tag{2.3}$$

Note that it is not necessary, for every problem, to specify all initial and final conditions.

2.1.4 Path Constraints

Conditions that are required along the intermediate trajectory may be enforced through upper and lower bounds on $\mathbf{g}$, a function of the states, controls, system parameters, and time,

$$\mathbf{g}_L \leq \mathbf{g}\left[\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t\right] \leq \mathbf{g}_U. \tag{2.4}$$

2.1.5 Box Constraints

Similarly to Equation (2.4), the states, controls, and system parameters may be constrained directly through box constraints,

$$\begin{aligned} \mathbf{x}_L \leq \quad \mathbf{x}(t) \quad \leq \mathbf{x}_U \\ \mathbf{u}_L \leq \quad \mathbf{u}(t) \quad \leq \mathbf{u}_U \\ \mathbf{p}_L \leq \quad \mathbf{p} \quad \leq \mathbf{p}_U. \end{aligned} \tag{2.5}$$

Note that specifying upper and lower bounds on all states, controls, and parameters is not always needed and could lead to an overly-constrained problem.

2.2 Overview of the Optimization Method

Since the goal of the thesis is a framework capable of autonomously (and accurately) solving for optimal trajectories in the category described by Section 2.1, a combination of methods is required. The first phase of the optimization consists of a direct shooting method solved by a genetic algorithm. High accuracy of the GA is not crucial to the results, however, because they are to be interpolated into a collocation discretization, which is then to be solved locally via a nonlinear program. If the GA produces a result which is nearly feasible and near the global optimum of the Bolza cost function, then the NLP should converge accurately onto the globally optimal trajectory, within the accuracy of the collocation method and the convergence criteria of the NLP.

The remainder of this section discusses the framework of optimization in this thesis. Since genetic algorithms and nonlinear programming were discussed in Chapter 1, the focus of this section will be on the specific choices of methods and solvers and the integration of components in the overall method.

In the GA/shooting phase, the parameters to be optimized are the nodal control input values and extra parameters including final time. The software package used to perform the GA search is the Genetic Algorithm Toolbox Version 1.2 for MATLAB, developed at the Department of Automatic Control and Systems Engineering, The University of Sheffield. This genetic algorithm package is well developed and documented, with a large selection of customizable functions. It is capable of handling binary- and real- encoded chromosomes, as well as multiple types of selection and crossover. For a detailed description of the GA functions, see the user manual [14]. The toolbox provides a template example of the simple genetic algorithm, which is used (heavily modified) for this framework. The toolbox, however, does not provide any trajectory optimization tools. Therefore, the shooting method used in this framework is provided in Section 2.4.

Once the user has entered the necessary equations of motion, cost, penalties due to constraint violations, integration time step, and other settings, a function is run which starts the genetic algorithm. When the GA has propagated the population through the specified number of generations, the parameters, final times, and histories of the states and controls are saved. The trajectory is displayed for inspection. The process of the genetic algorithm is illustrated in the flowchart in Figure 2.1.

After satisfactory convergence of the genetic algorithm, the state and control histories of the best candidate solution must be interpolated from the constant time-step discretization used in the GA/shooting phase to the nodes and collocation points selected for the NLP/collocation phase. When the user executes the second script, the MATLAB interpolation function is invoked, using a method chosen by the user (e.g. linear, cubic Hermite interpolating polynomial, spline interpolation, etc.). The nodes and collocation points to which the interpolated variables are mapped are chosen based on the collocation formulation, described in detail in Section 2.3.

The collocation method of choice for the local optimization is the Hermite-Legendre-Gauss-Lobatto technique mentioned in Section 1.3.1. Though there are more accurate collocation methods, the HLGL formulation is relatively easy to implement and still yields reasonable accuracy, as demonstrated in [5]. The parameters to be optimized by the NLP are the nodal state and control values and the control values at the collocation points. The final time may also be optimized.

The nonlinear program chosen to solve for the optimal state and control histories is the constrained nonlinear parameter optimizer, **fmincon()**, of MATLAB's Optimization Toolbox [12]. The function, **fmincon()**, is a general, multi-purpose optimizer designed to solve problems of low, medium, and high dimension. It considers the problems solved in this thesis to be of medium dimension and, therefore, uses a quasi-Newton algorithm with sequential quadratic programming to locate the optimum of the cost subject to the

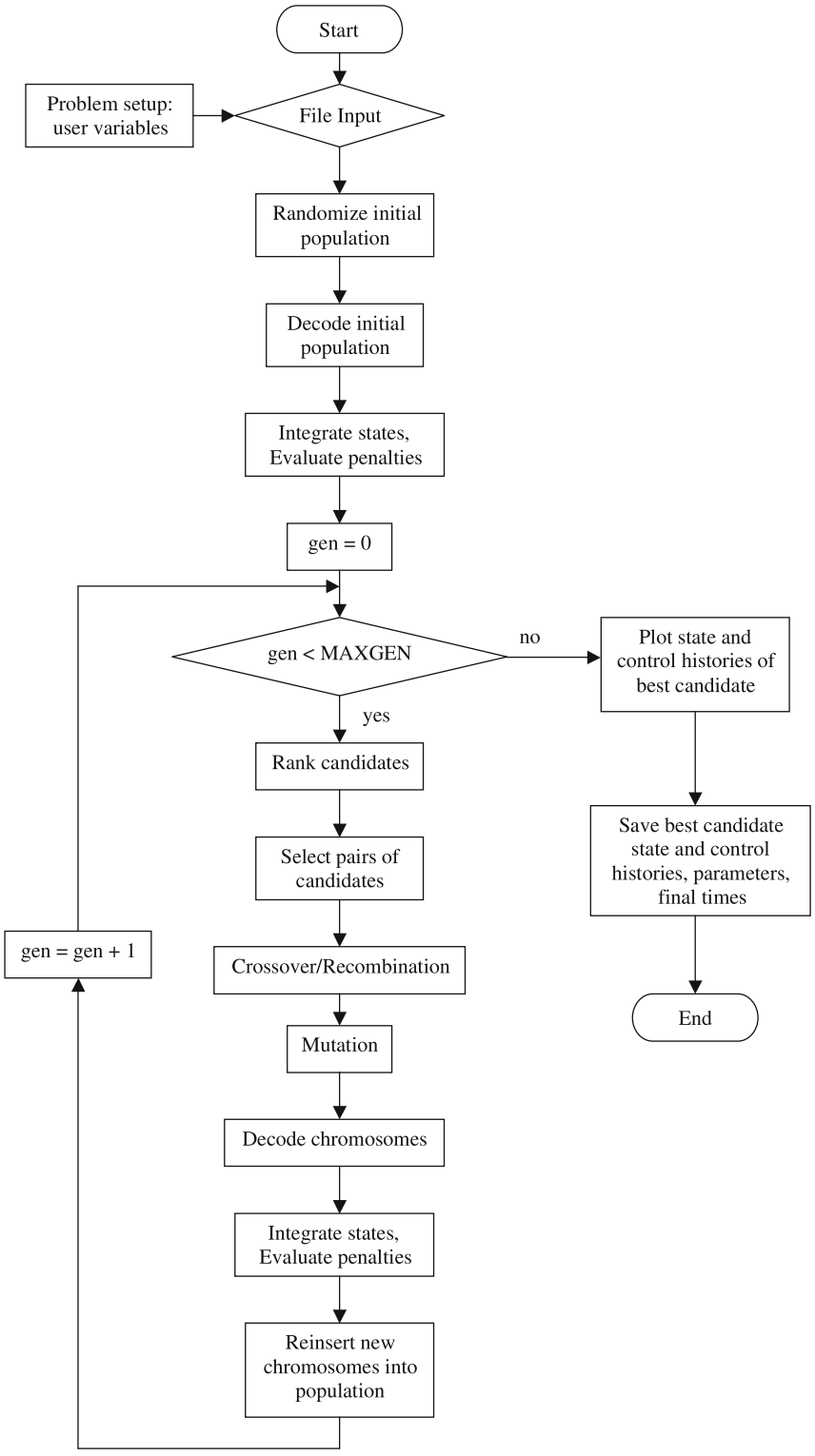


Figure 2.1: Flowchart of the genetic algorithm solution.

constraints. Since **fmincon()** is not specifically dedicated to solving the types of problems examined in this thesis, it may be reasonable to assume that other, more specialized NLP solvers will yield faster or more accurate results. However, it is expected that **fmincon()** will adequately solve the problems presented.

The script that starts the interpolation also begins the preparation and execution of the NLP. The Legendre-Gauss-Lobatto (LGL) nodes and collocation point locations are calculated by means to be explained in Section 2.3. Problem constraints of linear form, including box, path, and endpoint constraints are applied at the necessary nodes by a user defined function. The NLP is then started, calling user-defined functions which evaluate the Bolza cost and any path or endpoint constraints of nonlinear form, as well as an automatic collocation constraint-generating function. When **fmincon()** locates the stationary point of the cost subject to the constraints by achieving a low directional derivative (related to the jacobian of the cost) and low constraint violations, then the states, controls, and final times are extracted from the optimization vector and saved. The collocation interpolating polynomials are also calculated for plotting purposes. The process from interpolation of the GA/shooting results through the collocation solution is illustrated by the flowcharts in Figures 2.2 and 2.3.

2.3 Hermite-Legendre-Gauss-Lobatto Collocation for Local Optimization

A collocation technique is based upon minimizing the error between an interpolated parameter and an actual parameter. The interpolant is determined based upon the states and state derivatives at certain node points. Using this interpolant, the state derivatives are found at points between the nodes (called collocation points) via the derivative of the interpolant and the state derivative functions. The NLP is required to minimize the error between these derivatives by adjusting the states and controls at the nodes and collocation points. Methods of collocation vary, choosing different node or collocation

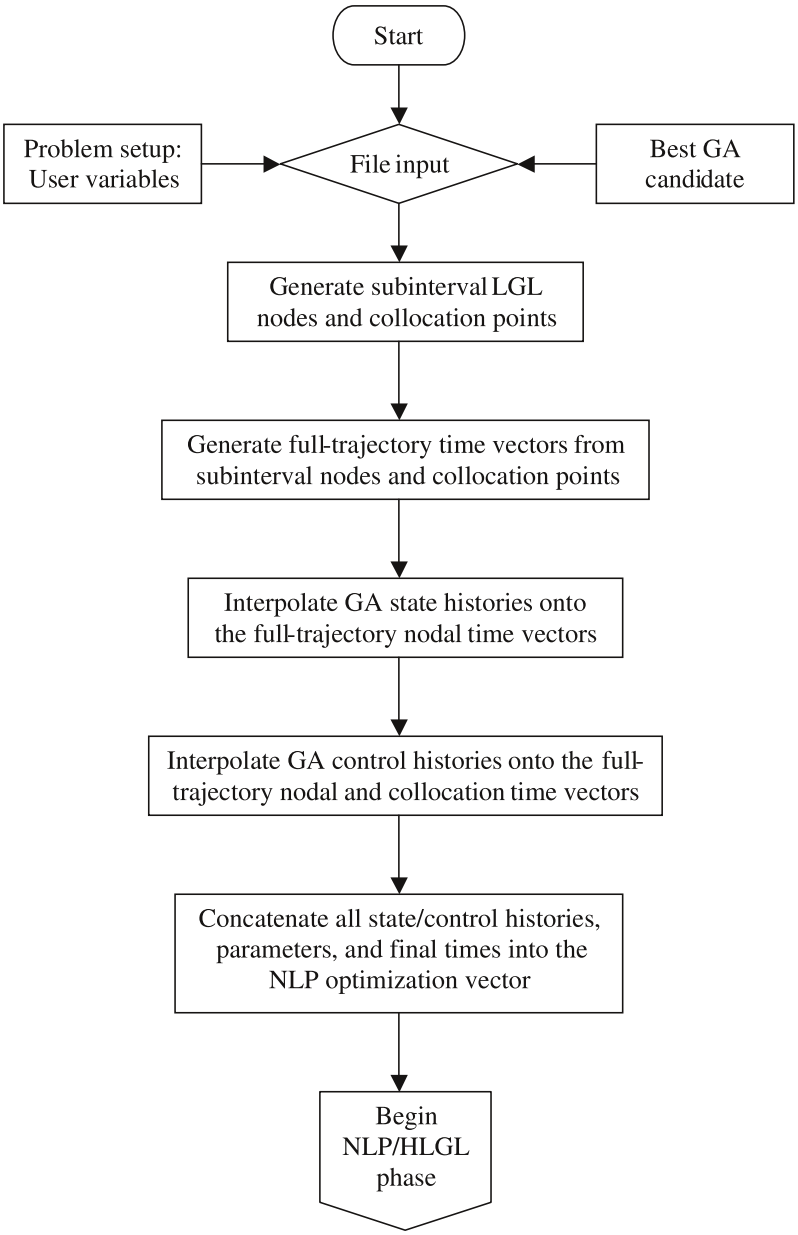


Figure 2.2: Flowchart of the interpolation from GA nodes to HLGL nodes and collocation points.

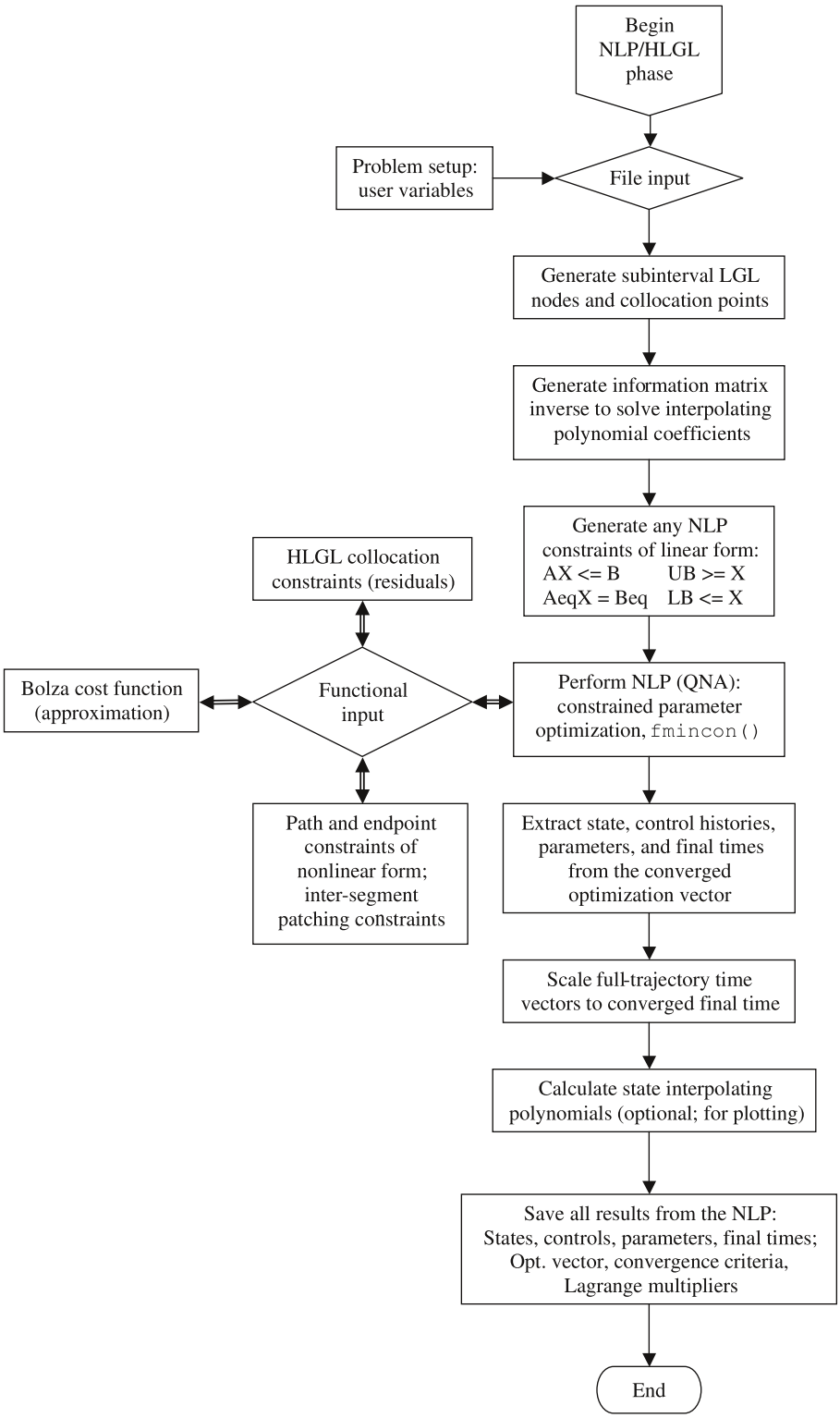


Figure 2.3: Flowchart of the NLP/collocation phase of the optimization.

points, as well as interpolating functions. The method used, in this work, to derive the collocation error residuals (based upon the original HLGL method [5]) is described below.

1. **Choose interpolant:** The standard HLGL method uses the n-th order Hermite interpolating polynomial, requiring the solution of $(n+1)$ unknown coefficients.
2. **Choose appropriate node points to solve the interpolant:** The standard method uses the states and their first derivatives, requiring $(n+1)/2$ nodes.
3. **Choose collocation points between the nodes:** $(n+1)/2$ nodes require $(n-1)/2$ collocation points.
4. **Form solution of the interpolant unknowns based on the node points.**
5. **Form error residuals:** The interpolated state derivatives must be equal to the state derivative functions evaluated at the collocation points.

One of the distinguishing characteristics of the HLGL approach to collocation is the use of the Hermite interpolating polynomial. Within an interval of time, the states of a system may be approximated by some other function. The HLGL method utilizes the Hermite interpolating polynomial for this purpose. The n-th order Hermite interpolating polynomial has the form

$$\mathbf{x}(\tau) \approx \mathbf{a}_0^i + \mathbf{a}_1^i\tau + \mathbf{a}_2^i\tau^2 + \mathbf{a}_3^i\tau^3 + \ldots + \mathbf{a}_n^i\tau^n, \quad \tau \in [-1, 1] \tag{2.6}$$

where $\mathbf{a}^i$ are column vectors of the same size as the state vector $\mathbf{x}$, containing coefficients describing the polynomials within the i-th interval. In Equation (2.6), the normalized time, τ, within the i-th interval is mapped to the global (dimensional) time by

$$t = t_i + \frac{h_i}{2}(\tau + 1) \tag{2.7}$$

where $h_i = t_{i+1} - t_i$.

To calculate the unknown coefficients, of which there are $(n+1)$ per state, we must provide $(n+1)$ equations that hold true within the interval of time. This is done through use of the discretization described above. The nodes within the interval provide $(n+1)/2$ values per state, and thus $(n+1)/2$ equations. To provide the remaining $(n+1)/2$ constraints, we differentiate the polynomial to obtain equations including the state derivatives at the nodes, which are also known in the discretization. The derivative of the polynomial is

$$\mathbf{x}'(\tau) \approx \mathbf{a}_1^i + 2\mathbf{a}_2^i\tau + 3\mathbf{a}_3^i\tau^2 + \ldots + n\mathbf{a}_n^i\tau^{(n-1)}. \tag{2.8}$$

Now, equating the polynomials to the states and state derivatives at the nodes, and using the relation between normalized time and global time, the following equations may be solved for the polynomial coefficients.

$$\mathbf{x}(\tau_l) = \mathbf{a}_0^i + \mathbf{a}_1^i\tau_l + \mathbf{a}_2^i\tau_l^2 + \ldots + \mathbf{a}_n^i\tau_l^n \tag{2.9}$$

and

$$\frac{h_i}{2}\mathbf{f}(\mathbf{x}(\tau_l), \mathbf{u}(\tau_l)) = \mathbf{a}_1^i + 2\mathbf{a}_2^i\tau_l + \ldots + n\mathbf{a}_n^i\tau_l^{n-1} \tag{2.10}$$

for $l \in [1, 2, \ldots, (n+1)/2]$. Expressing the above equations in convenient matrix form, the solution of the polynomial coefficients becomes

$$\begin{bmatrix} \mathbf{x}^T(\tau_1) \\ \vdots \\ \mathbf{x}^T(\tau_{\frac{n+1}{2}}) \\ \frac{h_i}{2}\mathbf{f}^T(\tau_1) \\ \vdots \\ \frac{h_i}{2}\mathbf{f}^T(\tau_{\frac{n+1}{2}}) \end{bmatrix} = \begin{bmatrix} 1 & \tau_1 & \tau_1^2 & \cdots & \tau_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \tau_1 & \tau_{\frac{n+1}{2}}^2 & \cdots & \tau_{\frac{n+1}{2}}^n \\ 0 & 1 & 2\tau_1 & \cdots & n\tau_1^{(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 2\tau_{\frac{n+1}{2}} & \cdots & n\tau_{\frac{n+1}{2}}^{(n-1)} \end{bmatrix} \begin{bmatrix} \mathbf{a}_0^{iT} \\ \mathbf{a}_1^{iT} \\ \vdots \\ \mathbf{a}_n^{iT} \end{bmatrix}. \tag{2.11}$$

The states and state derivatives may then be approximated at a normalized time, τ, within the interval using Equations (2.6) and (2.8). In the discretization, we choose

the collocation points, ζ_k, at which to evaluate and constrain the problem. Again, it is convenient to write these approximations in matrix form. Therefore, the interpolating polynomials and their derivatives evaluated at the collocation points are written as

$$\mathbf{x}^T(\zeta_k) = \begin{bmatrix} 1 & \zeta_k & \zeta_k^2 & \cdots & \zeta_k^n \end{bmatrix} \begin{bmatrix} \mathbf{a}_0^{iT} \\ \mathbf{a}_1^{iT} \\ \vdots \\ \mathbf{a}_n^{iT} \end{bmatrix} \tag{2.12}$$

and

$$\mathbf{x}'^T(\zeta_k) = \begin{bmatrix} 0 & 1 & 2\zeta_k & 3\zeta_k^2 & \cdots & n\zeta_k^{n-1} \end{bmatrix} \begin{bmatrix} \mathbf{a}_0^{iT} \\ \mathbf{a}_1^{iT} \\ \vdots \\ \mathbf{a}_n^{iT} \end{bmatrix}. \tag{2.13}$$

At this point, we can express the HLGL collocation residuals within a single time interval as

$$\boldsymbol{\Delta}_i = \begin{bmatrix} \mathbf{x}'^T(\zeta_1) - \frac{h_i}{2}\mathbf{f}^T\left(\mathbf{x}(\zeta_1)\right) \\ \mathbf{x}'^T(\zeta_2) - \frac{h_i}{2}\mathbf{f}^T\left(\mathbf{x}(\zeta_2)\right) \\ \vdots \\ \mathbf{x}'^T\left(\zeta_{\frac{n-1}{2}}\right) - \frac{h_i}{2}\mathbf{f}^T\left(\mathbf{x}(\zeta_{\frac{n-1}{2}})\right) \end{bmatrix}. \tag{2.14}$$

When the values of the states used to solve for the interpolating polynomial coefficients are true to the system dynamics, then the interpolated approximation of the state derivatives will have a minimum error with respect to the actual state derivatives evaluated at the interpolated states. Therefore, to ensure that the assumed values of the node-point states obey the equations of motion, we must drive the error residuals toward zero. That is, the NLP solver must minimize the error residuals given by Equation (2.14) at all collocation points over all of the time intervals.

Previously mentioned, the solution of the n-th order Hermite interpolating polynomial requires $(n+1)/2$ points within the interval at which we can evaluate the states and state derivatives. This calls for the choice of the locations of these point, called nodes. According to [5], the Legendre-Gauss-Lobatto (LGL) points provide maximum accuracy when using quadrature approximations. The LGL points are defined as the roots of the derivative of the $(n-1)$-th order Legendre polynomial [17],

$$P_{(n-1)}(\xi) = \frac{1}{2^{(n-1)}(n-1)!}\frac{d^{(n-1)}}{d\xi^{(n-1)}}\left[(\xi^2-1)^{(n-1)}\right], \quad \xi \in [-1,1]. \tag{2.15}$$

The LGL points also include the endpoints -1 and 1. Figure 2.4 illustrates the Legendre polynomial, its derivative, and the LGL points as they appear within the normalized interval of time.

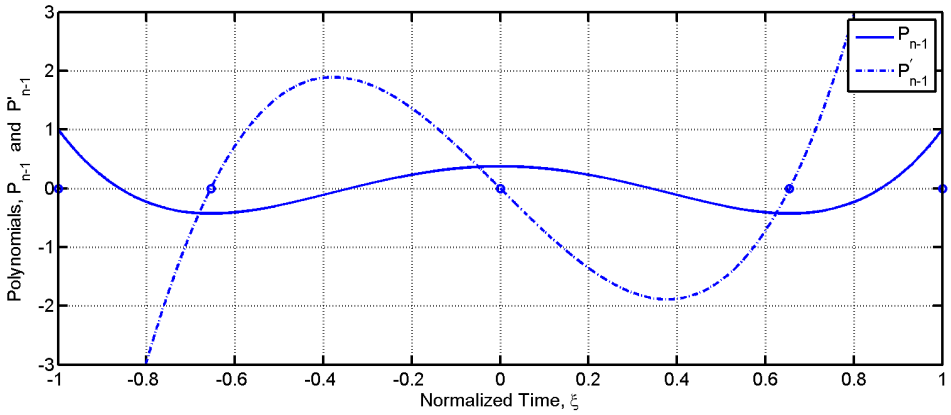


Figure 2.4: Legendre polynomial, its derivative, and nodes and collocation points for a 5-th order Hermite interpolating polynomial.

Since we choose to fit the n-th order polynomial through $(n-1)/2$ of the LGL points (the nodes), we choose every other LGL point in the interval starting with the

first endpoint. This leaves the LGL points in between to act as the required collocation points. Denoting the LGL points as ξ_j, the nodes are

$$\tau_l = \xi_{2l-1}, \quad l \in [1, 2, \ldots, (n+1)/2] \tag{2.16}$$

and the collocation points are

$$\zeta_k = \xi_{2k}, \quad k \in [1, 2, \ldots, (n-1)/2]. \tag{2.17}$$

2.3.1 Augmentation to Higher Order Dynamics

While the standard HLGL formulation can provide accurate results, it may be beneficial to extend the same collocation technique to account for higher-order dynamical information. This section reformulates the HLGL collocation technique to include second-order state dynamical information. Then the extension is generalized to the case where p-th order state derivatives are available. This may be particularly useful for minimum-jerk trajectory design, such as the problem of coordination of prostheses. In such applications, safety is a significant concern when designing smooth controllers for human-interfacing machines. The previous solutions tho the minimum-jerk arm coordination problem have been performed via the analytical dual optimal control methods of [1] to obtain polynomial trajectories [18][19]. Extension of the collocation method to include p-th order state derivatives will ensure that the accuracy and continuity of the higher-order derivatives across intervals (up to p-th order) are enforced, making the approximation of the jerk of an arm prosthesis, for example, more accurate.

The use of higher-order derivatives in direct collocation has been addressed by [20], but the method presented there requires that the dynamics can be expressed in the desired order of derivative prior to conversion to state space representation, and does not seek to include control derivatives in the collocation. That method may be inappropriate to solve problems with systems with inherently first-order equations of motion, such as

the kinematic path planning problem of Section 3.1. The benefit of that method is that the number of the NLP constraints is reduced, increasing the speed of convergence of the NLP.

The framework presented here may use the state space dynamical model used in the original HLGL formulation, under the assumption that the state derivative functions and the control inputs are differentiable. The control input derivatives may be approximated via interpolating functions. However, this section instead assumes that control rates are known, effectively augmenting the state vector with actuator dynamics.

By increasing the order of the HLGL collocation in this fashion, it is expected that a few benefits may be realized. One benefit that becomes clear in the derivation is that the number of nodes required for the discretization is reduced, while the interpolant is still exactly specified. This should reduce the computational burden placed upon the quasi-Newton solver. Another advantage to accounting for second-order information is that the final solution may be more dynamically accurate. Furthermore, because the control rates will be available, the problem-specific definition of path and box constraints may include bounds on control derivatives.

The higher-order formulation follows the process outlined in the previous section. Firstly, the n-th order Hermite interpolating polynomial is chosen as the interpolant, requiring $(n+1)$ pieces of information to solve the coefficients. The interpolant and its derivative are expressed by Equations (2.6) and (2.8), while its second derivative is given by

$$\mathbf{x}''(\tau) \approx 2\mathbf{a}_2^i + 6\mathbf{a}_3^i\tau + 12\mathbf{a}_4^i\tau^2 + \ldots + n(n-1)\mathbf{a}_n^i\tau^{(n-1)}. \tag{2.18}$$

To solve for the coefficients, the nodal equations for the states and state derivatives (Equations (2.9) and (2.10)) are used, as well as an the equation for the nodal state accelerations,

$$\left.\frac{d^2\mathbf{x}}{d\tau^2}\right|_{\tau_l} = 2\mathbf{a}_2^i + 6\mathbf{a}_3^i\tau_l + 12\mathbf{a}_4^i\tau_l^2 + \ldots + n(n-1)\mathbf{a}_n^i\tau_l^{(n-1)} \tag{2.19}$$

where

$$\frac{d^2\mathbf{x}}{d\tau^2} = \frac{h_i^2}{4}\left[\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{f} + \frac{\partial\mathbf{f}}{\partial\mathbf{u}}\dot{\mathbf{u}}\right] \tag{2.20}$$

and $l \in \{1, 2, \ldots, (n+1)/3\}$.

The previous statement implies that there must be $(n+1)/3$ nodes. This is true because there are three pieces of information contributed to the solution of the interpolant by each node (states, state derivatives, and state accelerations). This is incompatible with the previous discretization which does not take the state accelerations into account, requiring $(n+1)/2$ nodes. To create the new discretization, consider the set of LGL points resulting from the roots of the derivative of the m-th order Legendre Polynomial. Including the endpoints of the interval $[-1, 1]$, there are $(m+1)$ resulting LGL points, of which $(m+2)/2$ will become nodes and $m/2$ will become collocation points. Therefore, the LGL points are the interval endpoints and the roots of the derivative of the Legendre polynomial of order

$$m = \frac{2}{3}(n-2). \tag{2.21}$$

This yields the appropriate number of nodes for the new discretization.

In a fashion similar to the original HLGL formulation, we denote the LGL points as ξ_j where $j \in \{1, 2, \ldots, (2n-1)/3\}$. Taking the odd-indexed LGL points as nodes and the even-indexed points as collocation points, we have the nodes

$$\tau_l = \xi_{2l-1}, \quad l \in \{1, 2, \ldots, (n+1)/3\} \tag{2.22}$$

and the collocation points

$$\zeta_k = \xi_{2k}, \qquad k \in \{1, 2, \ldots, (n-2)/3\}. \tag{2.23}$$

Note that for a meaningful number of LGL points, $(n+1)$ must be divisible by three. Furthermore, since collocation requires at least two nodes and one collocation point, $n \geq 5$. This discretization rules out the third-order interpolant used in the well known Hermite-Simpson method [5].

The interpolant polynomial coefficients are then solved by

$$\mathbf{A} = \mathbf{\Phi}^{-1}\mathbf{X} \tag{2.24}$$

where the square matrix is given by

$$\mathbf{\Phi} = \begin{bmatrix} 1 & \tau_1 & \tau_1^2 & \tau_1^3 & \tau_1^4 & \cdots & \tau_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \tau_{\frac{n+1}{3}} & \tau_{\frac{n+1}{3}}^2 & \tau_{\frac{n+1}{3}}^3 & \tau_{\frac{n+1}{3}}^4 & \cdots & \tau_{\frac{n+1}{3}}^n \\ 0 & 1 & 2\tau_1 & 3\tau_1^2 & 4\tau_1^3 & \cdots & n\tau_1^{(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 2\tau_{\frac{n+1}{3}} & 3\tau_{\frac{n+1}{3}}^2 & 4\tau_{\frac{n+1}{3}}^3 & \cdots & n\tau_{\frac{n+1}{3}}^{(n-1)} \\ 0 & 0 & 2 & 6\tau_1 & 12\tau_1^2 & \cdots & n(n-1)\tau_1^{(n-2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 2 & 6\tau_{\frac{n+1}{3}} & 12\tau_{\frac{n+1}{3}}^2 & \cdots & n(n-1)\tau_{\frac{n+1}{3}}^{(n-2)} \end{bmatrix}, \tag{2.25}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T(\tau_1) \\ \vdots \\ \mathbf{x}^T(\tau_{\frac{n+1}{3}}) \\ \frac{h_i}{2}\mathbf{f}^T(\tau_1) \\ \vdots \\ \frac{h_i}{2}\mathbf{f}^T(\tau_{\frac{n+1}{3}}) \\ \frac{d^2\mathbf{x}}{d\tau^2}\Big|_{\tau_1}^T \\ \vdots \\ \frac{d^2\mathbf{x}}{d\tau^2}\Big|_{\tau_{\frac{n+1}{3}}}^T \end{bmatrix}, \tag{2.26}$$

and the matrix containing the coefficients is

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_0^{iT} \\ \mathbf{a}_1^{iT} \\ \vdots \\ \mathbf{a}_n^{iT} \end{bmatrix}. \tag{2.27}$$

Now, the error residuals to be minimized by the NLP include the first- and second-order state derivatives evaluated at the interpolated collocation points. The error residual vector is constructed as

$$\boldsymbol{\Delta}_i = \begin{bmatrix} \mathbf{x}'^T(\zeta_1) - \frac{h_i}{2}\mathbf{f}^T\left(\mathbf{x}(\zeta_1)\right) \\ \mathbf{x}'^T(\zeta_2) - \frac{h_i}{2}\mathbf{f}^T\left(\mathbf{x}(\zeta_2)\right) \\ \vdots \\ \mathbf{x}'^T\left(\zeta_{\frac{n-1}{2}}\right) - \frac{h_i}{2}\mathbf{f}^T\left(\mathbf{x}(\zeta_{\frac{n-1}{2}})\right) \\ \mathbf{x}''^T(\zeta_1) - \frac{d^2\mathbf{x}}{d\tau^2}\Big|_{\zeta_1}^T \\ \vdots \\ \mathbf{x}''^T(\zeta_{\frac{n-2}{3}}) - \frac{d^2\mathbf{x}}{d\tau^2}\Big|_{\zeta_{\frac{n-2}{3}}}^T \end{bmatrix} \tag{2.28}$$

where the states, state derivatives, and state accelerations are calculated by Equations (2.12), (2.14), and

$$\mathbf{x}''^{T}(\zeta_k) = \begin{bmatrix} 0 & 0 & 2 & 6\zeta_k & 12\zeta_k^2 & \cdots & n(n-1)\zeta_k^{n-2} \end{bmatrix} \begin{bmatrix} \mathbf{a}_0^{iT} \\ \mathbf{a}_1^{iT} \\ \vdots \\ \mathbf{a}_n^{iT} \end{bmatrix} \tag{2.29}$$

respectively.

Next, this section generalizes the method to include the p-th order dynamics. The expression for the p-th derivative of the interpolant, Equation (2.6), is

$$\mathbf{x}^{(p)} = \sum_{j=p}^{n} \left[\mathbf{a}_j \tau^{(j-p)} \prod_{k=0}^{p-1} (j-k) \right]. \tag{2.30}$$

To include the p-th order state derivatives such that the interpolant is exactly specified, the number of nodes required is $(n+1)/(p+1)$. This means that only interpolants which satisfy the condition that $(n+1)$ is divisible by $(p+1)$ may be chosen. Furthermore, since there must be at least 2 nodes, then

$$n \geq 2p+1. \tag{2.31}$$

The nodes and collocation points are taken from the set of LGL points defined as the roots of the derivative of the m-th Legendre polynomial and the endpoints of the interval $[-1, 1]$, where the appropriate number of LGL points is obtained by

$$m = 2\left(\frac{n+1}{p+1} - 1\right). \tag{2.32}$$

The polynomial coefficients are then solved via Equation (2.24) where the the terms $\mathbf{\Phi}$ and $\mathbf{X}$ include derivatives of the states and interpolating polynomial up to p-th order. Similarly, derivative error residuals up to p-th order are appended to the expression in Equation (2.28).

2.4 Shooting Method for Initial GA Search

As was stated in the Introduction, the genetic algorithm phase of the optimization attempts to search the solution space of the optimal control problem using a direct shooting method. Since a shooting method uses a marching integration to solve the nodal state values, the states need not be parameterized in the optimization vector. For the GA/shooting phase, only the control histories, system parameters, and final times are included in the optimization vector. For problems where some of the boundary conditions (B.C.'s) necessary to initialize the marching integration are not known initially, those B.C.'s may be treated as part of the system parameter vector included in the optimization vector.

The marching integration uses a fixed-step method, such as a 4th-order Runge-Kutta or Euler method. The analyst must choose a boundary of each continuous segment to initialize the integrator. If the initial conditions (I.C.'s) are known, then the segment is propagated forward to find the final conditions (F.C.'s). If the F.C.'s are known, then the integration is performed backwards toward the I.C.'s. For problems with multiple continuous segments where the initializing B.C.'s of one segment must be calculated based on those of an adjacent segment, then a coordinate transformation may be performed. The low-thrust interplanetary transfer problem solved in Section 3.3 requires this type of transformation at the junction between geocentric and heliocentric segments. In that problem, the final geocentric states (position and velocity of the spacecraft) are transformed into the initial conditions for the heliocentric segment. The heliocentric segment is then integrated until the spacecraft has reached the junction between the heliocentric and areocentric segments.

The objective (penalty) function of a candidate solution is the sum of components including the cost function described in the optimal control problem and constraint violations. The optimal control cost of Equation (2.2) is calculated based on the integrated

states. If the cost has a Lagrange component, then it is integrated in parallel with the states or approximated later by a user defined function that may be problem specific.

Note that the GA is not designed to solve constraint enforcement problems, but only penalty minimization problems. The framework for meeting constraints is not included in the framework of a GA, as it is in the dualized constrained parameter optimization algorithm of the NLP. Therefore, the analyst must perform the task of adding constraint violation to the objective function of the GA. After integration, any endpoint, path, and box constraints must be tested for violations at all nodes to which they apply. For nodes where constraints are violated, a weighted function (either the norm or norm-squared) of the violation is added to the penalty of the candidate solution. Note that, since the genetic algorithm does not depend on the gradient of the penalty function, the penalty is not required to be smooth over the solution space of the problem. This is not the case for the NLP solver used for the collocation phase, however.

2.5 Multiple Continuous Segments and Inter-Segment Constraints

One drawback to collocation methods in general is that discontinuities in states or controls can be difficult to characterize because interpolating polynomials are smooth within their intervals. This necessitates a way to handle such discontinuities. Such a method exists in the direct trajectory optimization literature.

Originally, the idea, as applied to trajectory optimization, came from the concept of spectral patching [10]. Spectral patching was intended to allow node redistribution to better characterize relatively fast dynamical behavior in regions of the time domain where the LGL nodes were sparse. This need was especially great because the spectral and pseudospectral methods used by the authors utilized global interpolants, so increasing the node density at a particular point in the trajectory dramatically affected the node distribution elsewhere and greatly increased the computational burden. Patching was

achieved by using multiple segments (with interpolants that were global within each segment) with constraints enforcing continuity at the point of the patch. The idea was later generalized to the concept of knotting, where the inter-segment constraints were allowed to take functional equality or inequality forms [21]. This proved to be useful for two situations. The first includes specific problems that require discontinuities at points along the trajectory, such as a change of mass of a spacecraft due to rocket stage separation. This is referred to as a hard knot. A soft knot is more similar to a patch in that it is not a part of the specific problem formulation or scenario, but the solution may benefit from having multiple segments with continuity constrained across the knot point. This is particularly helpful in characterizing switch points in bang-bang controls by freeing the control values across the knot point, but maintaining state continuity.

The concept of knotting is utilized in this thesis to accommodate the change of coordinates between geocentric, heliocentric, and areocentric segments of the low-thrust interplanetary transfer case study. The knotting method can also be used to accommodate a change in system dynamics or state vector dimension. The software tool developed for this thesis readily integrates this feature in both the GA/shooting and NLP/collocation phases. A formal expression of the inter-segment (knotting) constraints is provided below, similar to that in [21].

Consider two continuous segments discretized in the HLGL method, each having its own interpolating polynomial order and node distribution. This is illustrated in Figure 2.5. Each segment may be treated as a separate problem, given a unique definition as expressed by Equations (2.1) through (2.6). Though the problem formulation may be different for each segment, the segments are to be solved in parallel in the same optimization vector used in the GA and NLP. Let the time domain of the i-th segment

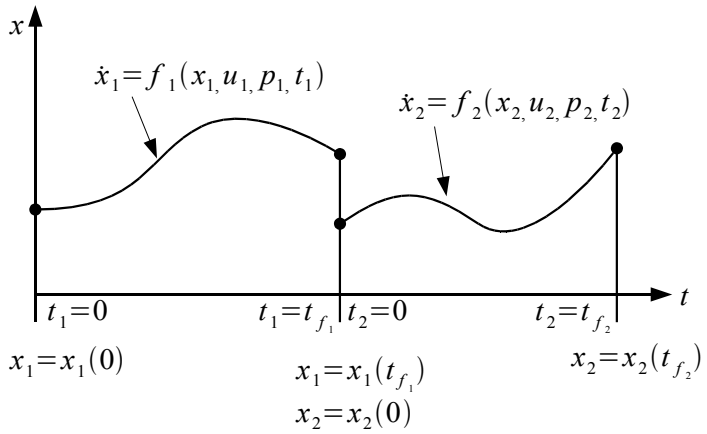


Figure 2.5: Illustration of multiple segments with inter-segment constraints.

be $t_i \in [0, t_{f_i}]$. The inter-segment constraints relating, for example, segment 1 to segment 2 are generally expressed as

$$\mathbf{c}_{1/2}^L \leq \mathbf{c}_{1/2}(\mathbf{x}_1(t_{f_1}), \mathbf{u}_1(t_{f_1}), \mathbf{p}_1, t_{f_1}, \mathbf{x}_2(0), \mathbf{u}_2(0), \mathbf{p}_2) \leq \mathbf{c}_{1/2}^U. \tag{2.33}$$

Specialization to equality constraints is achieved by setting the lower and upper bounds equal.

CHAPTER 3

APPLICATION CASE STUDIES

3.1 Mobile Robot Path Planning

The first problem to which the method is applied is a kinematic path planning problem for two autonomous ground vehicles, or mobile robots. The problem statement and implementation of the optimal control problem follow in this section. Differences between the GA implementation and the HLGL implementation are briefly discussed. Then results from both the GA and HLGL optimizations are presented after initializing the HLGL collocation method with the GA result. Observations are made regarding the convergence characteristics of the genetic algorithm in comparison to the collocation method.

3.1.1 Problem Statement

Given the state dynamical equations below, the solution to this problem must meet certain specific criteria. Those are: *i*) both vehicles must arrive at or within the perimeter of the goal; *ii*) the trajectory should be completed in minimum time given that both vehicles arrive at the goal simultaneously (i.e. at least one vehicle will have a minimum-time trajectory and its partner will arrive at the same time); *iii*) the paths should not cross obstacle perimeters; *iv*) the vehicles should take different paths. The last requirement is implemented as if the vehicles should not collide, though the motivation is to make the vehicles "search" different paths. Figure 3.1 illustrates the layout of the two-dimensional field in which the vehicles travel. The following paragraphs discuss the

details of the problem as applied to the generalized formulation expressed by Equations (2.2) through (2.6).

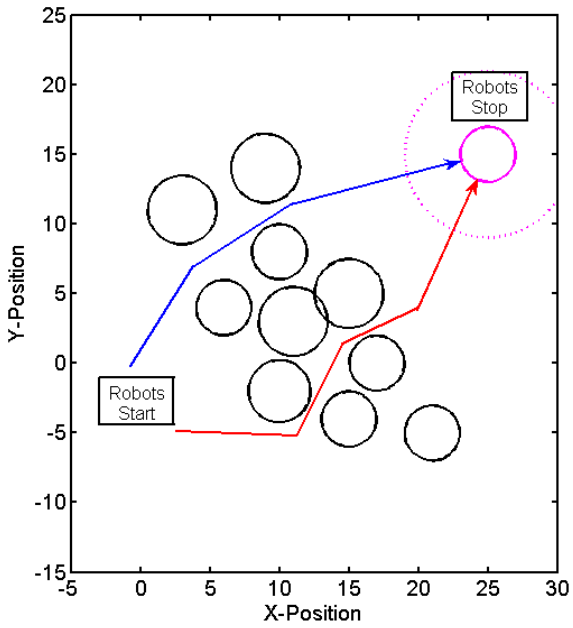


Figure 3.1: Obstacle course for the vehicles showing obstacles, goal, and the "safe zone" near the goal.

Each vehicle is assumed to have the configuration of a tricycle with two independent wheels in the rear and one steering wheel in the front. This is illustrated in Figure 3.2. The wheels are assumed to travel perpendicular to their axes without slipping. The instantaneous center of curvature (rotation) of a wheel's path lies on its axis. Since the body rigidly connects all of the wheels, the body and wheels share their rotational velocity and instantaneous center of rotation in the plane of motion. The center of rotation lies at the intersection of the wheel axes. Therefore, the wheels instantaneously trace concentric circles about the center of rotation with an angular velocity, $\dot{\theta}$, given an instantaneous

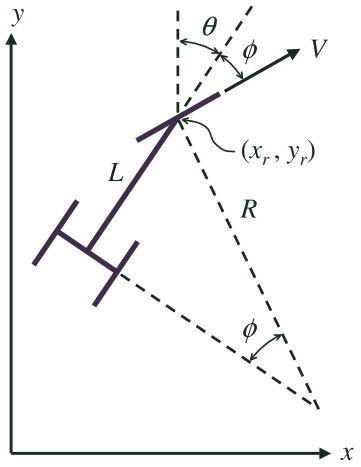


Figure 3.2: Configuration of the mobile robot

value of the steering angle, ϕ. The steering wheel travels at a constant speed, V, with a turning radius, R. Its angular velocity is given by

$$\dot{\theta} = V/R.$$

Since $L = R\sin\phi$, then the rotational velocity of the vehicle is

$$\dot{\theta} = (V/L)\sin\phi \tag{3.1}$$

where L is the wheelbase, measured along the centerline of the vehicle from the rear axle to the steering wheel. The remaining equations of motion for the in-plane coordinates are

$$\dot{x}_r = V\sin(\phi+\theta) \tag{3.2}$$

$$\dot{y}_r = V\cos(\phi+\theta) \tag{3.3}$$

where θ is the heading angle and (x_r, y_r) is the position of the steering wheel in the plane of motion.

The cost function for this problem is

$$J = t_f \tag{3.4}$$

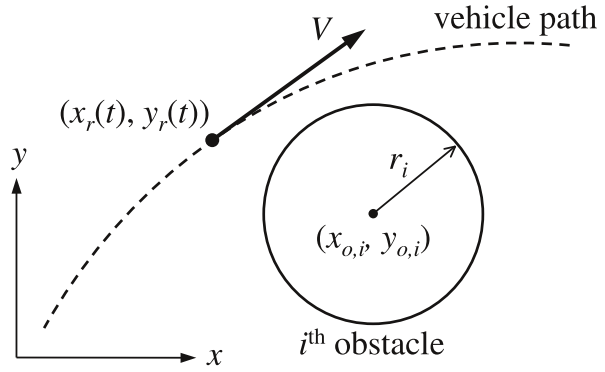


Figure 3.3: Path of the mobile robot and the circular boundary of the i-th obstacle.

where t_f is the final time.

The endpoint constraints include the arbitrary initial position and orientation of each vehicle, and the final position of each vehicle at the center of the goal.

The path constraints require that the distance from a vehicle to the center of the i-th obstacle is greater than the radius of that obstacle. This is illustrated in Figure 3.3 and enforced by the nonlinear path constraint,

$$\sqrt{(x_r(t) - x_{o,i})^2 + (y_r(t) - y_{o,i})^2} \geq r_i. \tag{3.5}$$

The required minimum vehicle proximity is enforced by treating the location of one vehicle as the center of an obstacle. This is enforced at each point in time along the trajectory except within a "safe zone" around the goal. This is acceptable because the individual vehicles need not search different paths through the obstacle field once they have come close to the goal. The steering saturation limits are enforced through box (inequality) constraints in the form

$$\phi_{\min} \leq \phi \leq \phi_{\max}. \tag{3.6}$$

These constraints are applied to the collocation phase of the optimization directly. However, certain changes must be made to the cost function and constraint violation

penalties for implementation in the GA search phase due to the differing characteristics of the shooting method and convergence properties of the GA.

3.1.2 GA Considerations

Since the framework presented here utilizes differing optimization methods for the initial guess phase and the collocation phase, the convergence characteristics of the GA initialization are different from the HLGL collocation phase. One specific example of this is the weighting of penalties in the GA. Since the quasi-Newton algorithm used for the HLGL optimization seeks to meet all constraints to within a small tolerance, the magnitude or weighting of those constraints has a much weaker effect on the final result of that phase by comparison to the GA. In the GA itself, the magnitude of a single penalty can cause the algorithm to converge upon a minimum in the cost function which is local but not global. Some authors have created methods which set up the GA in such a way that convergence is more robust under the same constraints, costs, or penalties used in the collocation algorithm, such as in [16]. However, under the scenario presented here, early attempts at creating good penalties for the GA seem to work well without modification to the GA itself.

Since the shooting method used in the GA does not locate the zero of a constraint function through gradient information, any constraint violations are added to the cost function used for ranking. In this case, the cost function, J, has three components. Their sum is

$$J = J_{t_f} + J_G + J_c. \tag{3.7}$$

The first component is the weighted final time given by

$$J_{tf} = \frac{1}{10} t_f. \tag{3.8}$$

The second component is the combined squares of the distances from the goal center to each vehicle.

$$J_G = (x_1 - x_{Goal})^2 + (y_1 - y_{Goal})^2 + (x_2 - x_{Goal})^2 + (y_2 - y_{Goal})^2. \tag{3.9}$$

The third component represents penalties imposed for nodes which lie inside obstacle perimeters or instances where the vehicles are too close. If

$$r_{r|o}(i,j) = \sqrt{(x_{r,j} - x_{o,i})^2 + (y_{r,j} - y_{o,i})^2} \tag{3.10}$$

is the distance of the vehicle from the center of the i-th obstacle evaluated at the j-th node in time, then

$$J_c = \sum_{i=1}^{N_{obst}} \sum_{j=1}^{N} \begin{cases} \frac{100}{r_{r|o}(i,j)^2}, & r_{r|o}(i,j) < r_i \\ 0, & r_{r|o}(i,j) \geq r_i \end{cases} \tag{3.11}$$

where N_{obst} is the number of obstacles and N is the number of discrete time nodes in the shooting integration. Vehicle proximity violations are treated identically to the obstacle penalties. That is, one vehicle is treated as a moving obstacle by the other vehicle. The penalty added for time nodes having the vehicles too close is the same as the penalty for nodes with vehicles falling inside obstacle radii.

Notice that the penalties added for obstacle or proximity violations in Equation (3.11) follow an inverse-square law. Though the penalty is discontinuous across obstacle boundaries, the inverse-square penalties serve to soften the penalties near the boundaries while the penalties are severe near the center of the obstacle.

The weighting coefficients present in Equations (3.8) and (3.11) were chosen by trial and error. The obstacle penalty weight, in particular, is observed to be important to the convergence of the genetic algorithm to a feasible, near-globally-optimal solution. When those penalties are too low, evolved vehicle paths tend to pass between the centers of overlapping obstacles. When the penalties are too high, the evolved paths tend to

go around the group of obstacles rather than going between them. Though the genetic algorithm is sensitive to these parameters, convergence is very fast. The trial and error process in this case was not difficult or lengthy.

One additional change is made to the problem for implementation in the GA dealing with how the vehicle trajectories are treated when a vehicle reaches the goal. According to the problem statement, both vehicles will arrive at the goal at the same time. This is easy to enforce in the collocation phase of the solution through endpoint constraints. However, we do not wish to exclude, from the GA search, those families of candidate solutions in which one vehicle has arrived at the goal earlier than its partner (because the faster trajectory may be near the global optimum for that vehicle). Therefore, when the location of either vehicle falls inside the goal perimeter, that vehicle stops. The equations of motion are modified with the velocity setting

$$V = \begin{cases} 0.2, & r_{r|G} > \text{goal radius} \\ 0, & r_{r|G} \leq \text{goal radius} \end{cases} \tag{3.12}$$

where $r_{r|G}$ is the distance from the goal center to the vehicle. This modification to speed is made only in the GA phase of optimization.

3.1.3 Results

Before discussing the results, a few parameters used to generate them are provided. For the genetic algorithm, parameters which provide acceptable or reasonable results over a low number of iterations (and converge in a low amount of time) are selected. The control and final time encoding used in the GA chromosome is binary with precision of 7 digits. This allows 128 possible values for each parameter in the chromosome. For the steering angle, this yields a precision of 0.0164 rad. The population size is held at 400 individuals, allowing enough diversity to find an acceptable result. The number of generations is 140, allowing the most drastic "evolutionary" changes to cease prior to

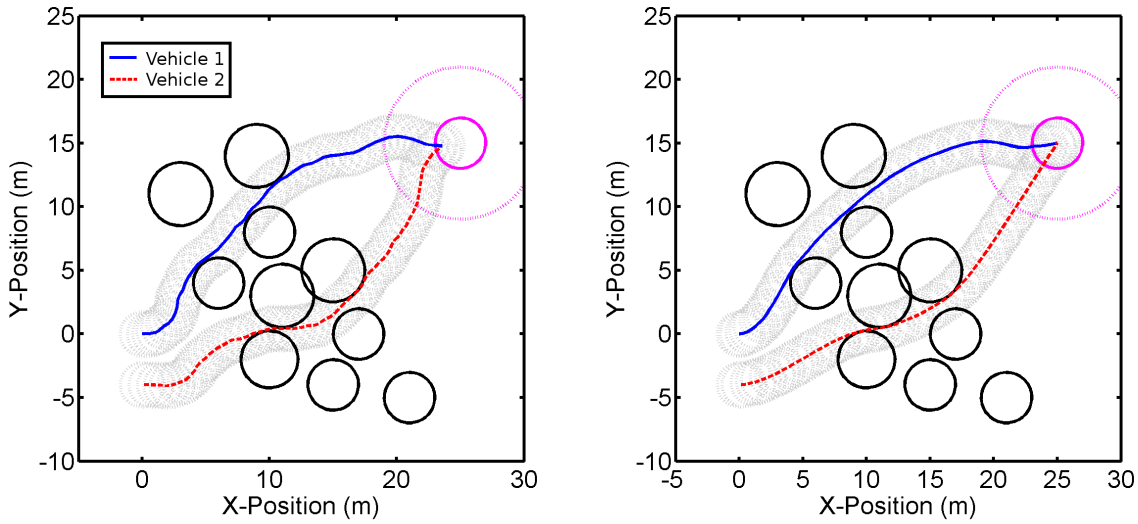
ending the GA. The upper bound on final time is chosen such that there is enough time for the lower vehicle to travel around the lowest obstacle and still reach the goal. The lower and upper bounds on final time are 165 and 250 seconds, respectively. The time step for the Euler integration is $t_f/60$, generating the same number of node points as used in the HLGL discretization later, but spaced equidistantly.

In configuring the HLGL collocation phase of the optimization, a Hermite interpolating polynomial of order $n = 7$ is chosen with 20 subintervals. This allows for higher-order accuracy without using a costly number of subintervals.

The first vehicle is placed at the origin, and the second is placed directly below it at $y = -4$. Both vehicles are oriented parallel to the x-axis. The minimum allowable distance between the vehicles is 3.5 m. This is illustrated on the figures by gray circles of radius 3.5/2 m drawn around each node of the trajectories. The vehicle speed is a constant 0.2 m/s, and the steering wheel may be deflected up to $\pm\pi/3$ radians from the axis of the vehicle. Each vehicle has a wheelbase of 2 m. The radius of each obstacle is considered sufficient to prevent physical collision, though this is not critical to the demonstration of the method.

The goal is placed at $(x_{Goal}, y_{Goal}) = (25, 15)$ with a radius of 2 m. The "safe zone" around the goal has a radius of 6 m. Inside this region, no penalties for vehicle proximity are assessed.

Figure 3.4a shows the best candidate solution produced by the genetic algorithm. Best candidates from early generations were observed to cross obstacle boundaries, as well as allow vehicle collisions. As the generations progressed, the population tended to favor solutions which met the obstacle boundary constraints, but did not necessarily arrive at the goal. Furthermore, some candidate solutions allowed the lower vehicle to follow a similar path to the upper vehicle after a short detour. This resulted from the setup of the penalties and may be avoided through other penalty functions. However,



(a) (b)

Figure 3.4: Mobile robot paths of (a) the best GA candidate solution after 140 generations and (b) the converged result of the HLGL collocation. The robot "collision radii" are shown as gray dotted circles around each node of the trajectory.

those candidate solutions were generally of higher final time, and the GA ended upon the solution shown in the figure with a final time of 205.16 seconds.

The paths resulting from the HLGL collocation optimization are shown in Figure 3.4b. The noteworthy characteristics of this solution are, first, that all of the path and boundary constraints are met, and second, that the solution meets the requirements of the problem statement with a final time of 166.09 seconds. In particular, notice that, while the lower vehicle appears to travel in a straight line when not in the vicinity of an obstacle, the upper vehicle seems to take a non-minimum-time path. Actually, the problem setup allows this due to the vehicles sharing a final time. Both vehicles arrive at the goal at the same time, so the path lengths of the two vehicles are equal. This can be changed by parameterizing separate final times for each vehicle.

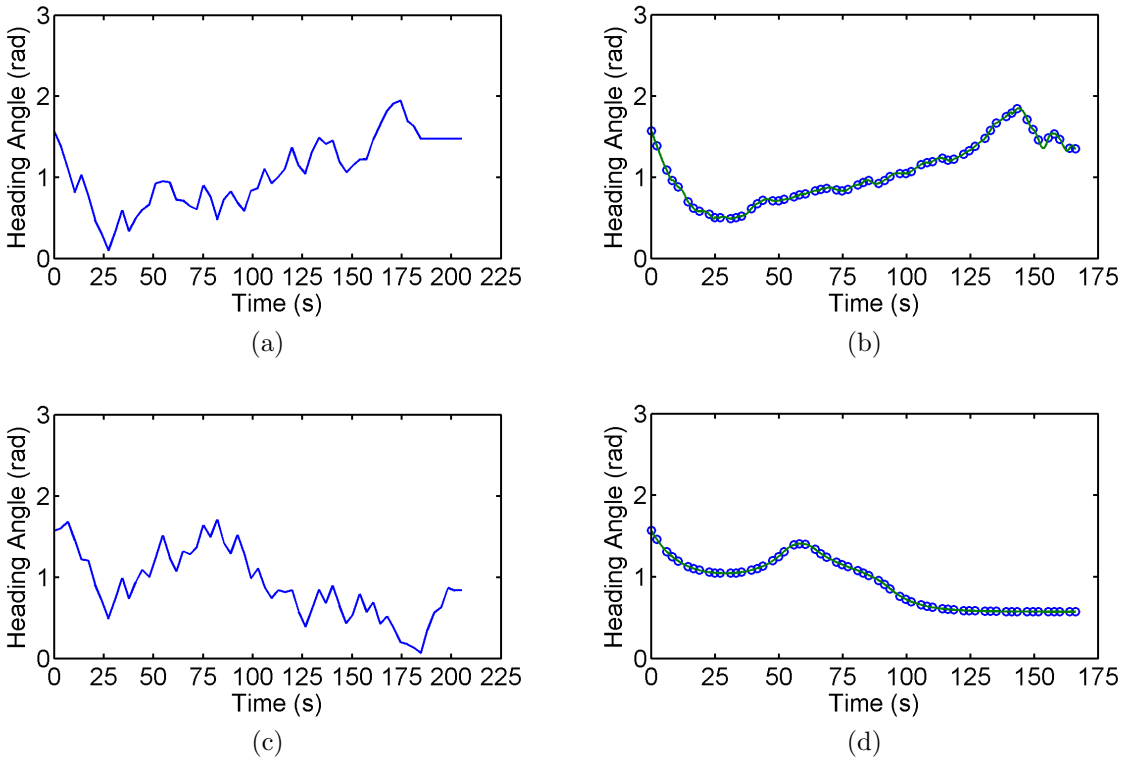


Figure 3.5: Heading angle histories for vehicle 1 from (a) the GA result and (b) the HLGL result; for vehicle 2 from (c) the GA result and (d) the HLGL result.

Notice that the heading and steering angles shown in Figures 3.5 and 3.6 resulting from the GA search are not smooth. This is due to the poor performance of the GA and shooting technique as a primary optimization scheme, though the GA search phase has performed adequately in locating a near-optimal, feasible path for the total system.

From the initialization provided by the GA, the HLGL collocation phase produces the minimum-time trajectory as defined by the problem statement. As required by the problem statement, vehicle 2 has a minimum-time path to the goal, while vehicle 1 takes a non-minimum-time path of equal length. This is reflected by the smooth behavior of the steering angle of vehicle 2 in contrast to the chattering steering angle of vehicle 1.

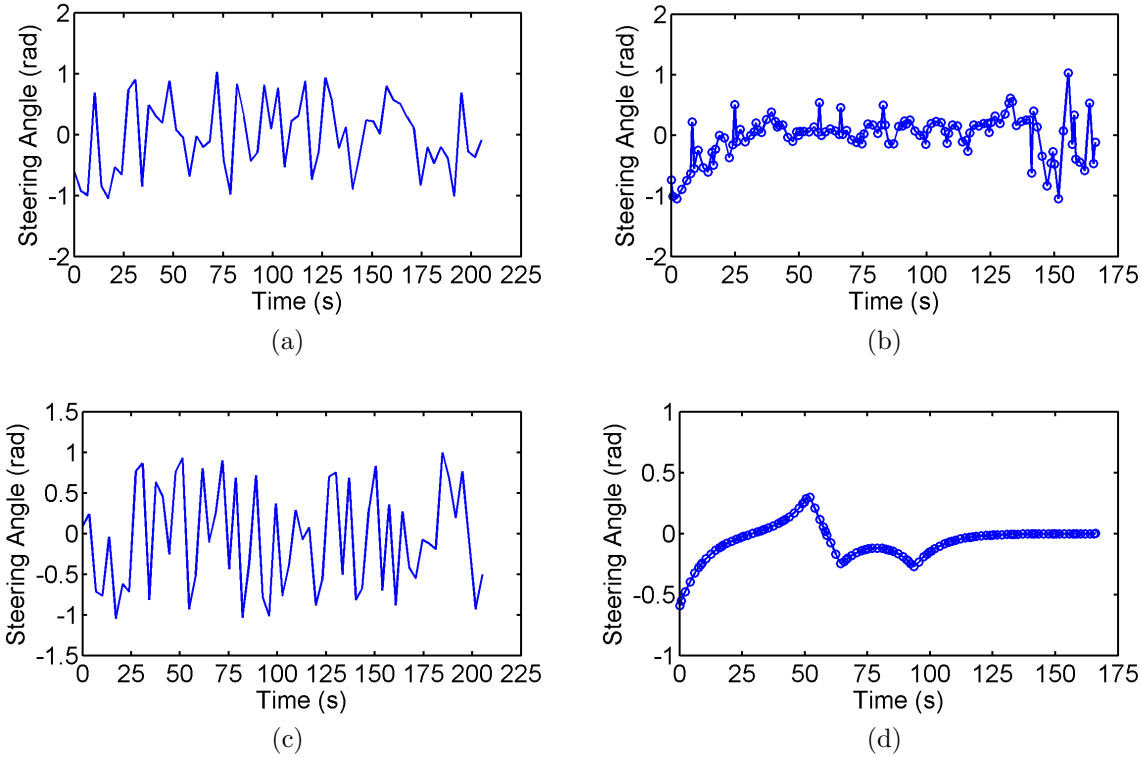


Figure 3.6: Steering angle histories for vehicle 1 from (a) the GA result and (b) the HLGL result; for vehicle 2 from (c) the GA result and (d) the HLGL result.

Unless the final time for each vehicle is parameterized separately, one vehicle will exhibit this chattering.

Another remedy to the discontinuous control history could be made by implementing the p-th order HLGL collocation discussed in Section 2.3. If the second-order dynamical information is available, then continuity of control rates could be ensured across intervals. The control rates could also be bounded directly through path or box constraints.

3.2 Aircraft Turn To Heading Maneuver

The second problem to which the method is applied is the design of a minimum-time $\frac{\pi}{2}$-rad turning maneuver for an aircraft. The problem statement and implementation of the optimal control problem follow in this section. Differences between the GA implementation and the HLGL implementation are briefly discussed. Then results from both the GA and HLGL optimizations are presented after initializing the HLGL collocation method with the GA result. Observations are made regarding the optimality of the genetic algorithm and collocation results. The benefits of augmenting the aircraft model with actuator dynamics are briefly discussed, as well as the expected effects of the higher-order collocation technique outlined in Section 2.3.1.

3.2.1 Problem Statement

Given the equations of motion below, the solution trajectory of the aircraft should complete a heading change in minimum time. The formal statement of the requirements follows: *i*) the aircraft should begin and end in level flight (zero flight path angle) at zero altitude; *ii*) the aircraft should begin flying parallel to the X-axis and should end flying parallel to the Y-axis, requiring a $\frac{\pi}{2}$-rad heading change; *iii*) the final airspeed should be the same as the initial airspeed; *iv*) the aircraft should not lose airspeed; *v*) the load factor should not exceed 5g; *vi*) the maneuver should be completed in minimum time.

In the context of the constrained optimal control problem described above, the cost function to be minimized is given by

$$J = t_f, \tag{3.13}$$

where t_f is the final time. The nonlinear state dynamics are similar to a horizontal planar-motion aircraft model used in [22], augmented with equations of motion for altitude, specific energy, flight path angle, and throttle.

$$\dot{X} = V\cos\gamma\cos\chi \tag{3.14}$$

$$\dot{Y} = V\cos\gamma\sin\chi \tag{3.15}$$

$$\dot{H} = V\sin\gamma \tag{3.16}$$

$$\dot{E} = \frac{V}{W}(T-D) \tag{3.17}$$

$$\dot{\gamma} = \frac{1}{V}(a_p - g\cos\gamma) \tag{3.18}$$

$$\dot{\chi} = \frac{a_y}{V\cos\gamma} \tag{3.19}$$

$$\dot{\eta} = -\lambda_\eta(\eta - \eta_c). \tag{3.20}$$

Additionally, the model includes linear first-order lateral and longitudinal actuator dynamics.

$$\dot{a_p} = -\lambda_p(a_p - a_{pc}) \tag{3.21}$$

$$\dot{a_y} = -\lambda_y(a_y - a_{yc}). \tag{3.22}$$

In these dynamics, X and Y can be thought of as the inertial north/south and east/west coordinates of the aircraft, respectively, while $H = -Z$ is the altitude. The direction of flight with respect to the inertial frame is illustrated in Figure 3.7. The specific energy, E, is energy per unit weight of the aircraft. The flight path angle is γ and χ is the heading angle measured positive clockwise from the X axis. Of the three control inputs, η_c commands the throttle, a_{pc} commands pitch, and a_{yc} commands yaw. These equations

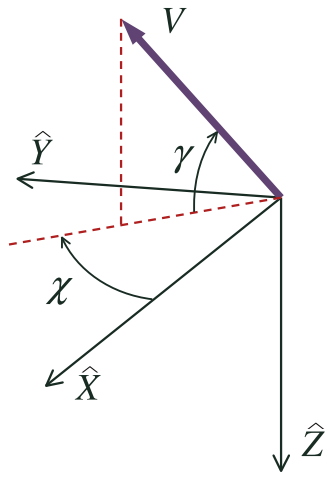


Figure 3.7: Velocity of the aircraft oriented with respect to an inertial $\hat{X}$-$\hat{Y}$-$\hat{Z}$ unit vectors attached to the aircraft center of mass.

of motion assume that the aircraft has a constant weight and that the thrust is in the direction of the velocity. The parasite, induced, and total drag are calculated by

$$D_0 = qSC_{D0} \tag{3.23}$$

$$D_i = k\left(\frac{W}{qS}\right)^2 \tag{3.24}$$

$$D = D_0 + n^2 D_i \tag{3.25}$$

where $q = \frac{1}{2}\rho V^2$ is the dynamic pressure, $\rho = 1.23$ kg/m^2 is the density of air, $S = 27.87$ m^2 is the wing reference area, $C_{D0} = 0.015$ is the zero-lift drag coefficient, $k = 0.02$ is the drag polar constant, and $W = 111,210$ N is the constant weight of the aircraft. These values are not presumed to accurately model a particular aircraft. The velocity is calculated from the specific energy and altitude as

$$V = \sqrt{2g(E - H)} \tag{3.26}$$

where $g = 9.81$ m/s^2 is the acceleration due to gravity. Thrust is calculated directly from the throttle position and constant maximum thrust, $T_{\max} = 63,765$ N (the weight of 6,500 kg), as

$$T = \eta T_{\max}. \tag{3.27}$$

The endpoint constraints applied to the trajectory include the initial and final conditions for some of the states. The initial X, Y, and H locations are at the origin of the X-Y-Z frame. The initial flight path angle and heading are both 0 radians. The initial specific energy (and, therefore, airspeed) is also required. The initial velocity is set to 90 m/s and the specific energy is calculated from Equation (3.26). The final conditions specified are the altitude, flight path angle, heading, and airspeed as a nonlinear equality constraint.

The path constraints require that the airspeed, calculated via Equation (3.26), not fall below its initial value. So the inequality constraint given by

$$V \geq 90\text{m/s} \tag{3.28}$$

is enforced in the optimization. We also require the loadfactor to be less than $5g$. This is enforced by

$$n,\ n_c \ \leq \ 5, \tag{3.29}$$

where the loadfactor is

$$n = \frac{1}{g}\sqrt{a_p^2 + a_y^2} \tag{3.30}$$

and the commanded loadfactor is

$$n_c = \frac{1}{g}\sqrt{a_{pc}^2 + a_{yc}^2}. \tag{3.31}$$

Box constraints are applied to the control inputs as

$$0 \leq \ (a_{pc},\ a_{yc}) \ \leq 5g \tag{3.32}$$

$$0 \leq \qquad \eta_c \qquad \leq 1, \tag{3.33}$$

and similarly to their counterparts a_p, a_y, and η.

These constraints are applied to the collocation phase of the optimization directly. However, for the GA search phase, certain changes must be made to the cost function and constraint violation penalties.

3.2.2 GA Considerations

Since the shooting method used in the GA does not locate the zero of a constraint functions through gradient information, any constraint violations are added to the cost function used for ranking. In this case, the cost function, J, has three components. Their sum is

$$J = J_{tf} + J_{\text{FC}} + J_{\text{path}}. \tag{3.34}$$

The first component is the weighted final time given by

$$J_{tf} = 10^4 t_f. \tag{3.35}$$

The second component accounts for the final condition constraint violations.

$$\begin{aligned} J_{\text{FC}} = (H(t_f) - 0 \text{ m})^2 + 10(\gamma(t_f) - 0 \text{ rad})^2 \\ + 10^5(\chi(t_f) - \frac{\pi}{2} \text{ rad})^2 + 10(V(t_f) - 90 \text{ m/s})^2 \end{aligned} \tag{3.36}$$

The third component is the integral of all path inequality constraint violations over then entire trajectory:

$$\begin{aligned} J_{\text{path}} = \int_{t_0}^{t_f} \Big[w_V(90 \text{ m/s} - V) + w_n(n - 5) \\ + w_{nc}(n_c - 5) + w_{\eta u}(\eta - 1) + w_{\eta l}(0 - \eta) \Big] dt, \end{aligned} \tag{3.37}$$

where

$$w_V = \begin{cases} 10^3, & V < 90 \text{ m/s} \\ 0, & V \geq 90 \text{ m/s} \end{cases} \tag{3.38}$$

$$w_n = \begin{cases} 10^3, & n > 5 \\ 0, & n \leq 5 \end{cases} \tag{3.39}$$

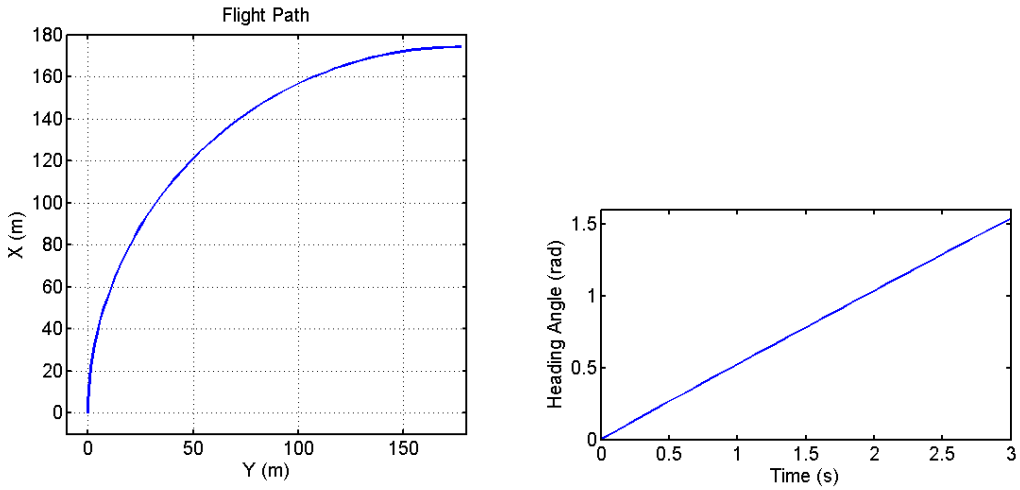
$$w_{nc} = w_{\eta u} = w_{\eta l} = 0. \tag{3.40}$$

The coefficients (weights) present in Equations (3.35) through (3.37) are necessary due to the convergence properties of the genetic algorithm. Inappropriate weights may cause the GA to select trajectories that violate constraints in favor of minimizing the final time. Conversely, the GA may meet the constraints very well, but become insensitive to the final time. Though the weights used here were chosen through trial and error, they seem to work well.

3.2.3 Results

Before discussing the results, a few parameters used to generate them are provided. For the genetic algorithm, parameters which provide acceptable or reasonable results over a low number of iterations (and converge in a short period of time) are selected. The binary precision of the control values and final time in the chromosome is 5 bits. The population size is held at 250 individuals. For this problem, a population of this size has enough diversity to find an acceptable result. The number of generations propagated is 160, allowing the most drastic "evolutionary" changes to cease prior to ending the GA. Because the final time is free, it is parameterized within the GA chromosome. Its lower and upper search bounds are 1 s and 5 s, respectively. The time step for the fourth-order Runge Kutta integration is one fiftieth of the largest final time searchable by the GA, making the largest possible step equal to 0.1 s.

In configuring the HLGL collocation portion of the optimization, a Hermite interpolating polynomial of order $n = 5$ is chosen with 11 subintervals. The termination criterion depends upon a low maximum constraint violation and a low directional derivative.



(a) (b)

Figure 3.8: Genetic algorithm results for (a) the flight path in the horizontal plane and (b) heading angle.

The aircraft is initially placed at the origin of the (X, Y, Z) frame with zero flight path angle and heading. The initial airspeed is set to 90 m/s. The initial throttle position is set such that the thrust equals the drag.

We see from Figures 3.8a through 3.11b that the best GA candidate meets the final conditions (for altitude, flight path angle, airspeed, and heading angle) and path constraints (for airspeed and loadfactor) of the problem statement very well, though not exactly. This is acceptable, however, since the remaining constraint errors are to be remedied by the NLP in the HLGL collocation phase of the optimization.

The resulting HLGL solution after initialization by the GA is presented in Figures 3.12a through 3.15b. The endpoint and constraint violations of this final solution are improved from those of the GA initialization. Though the final time of 2.9687 seconds given by the HLGL optimization is lower than the final time of 3.0645 seconds given by the GA, the final solution is still only near-optimal. This may be improved by adjusting the convergence criterion of the NLP [12] within the accuracy of the HLGL collocation formulation.

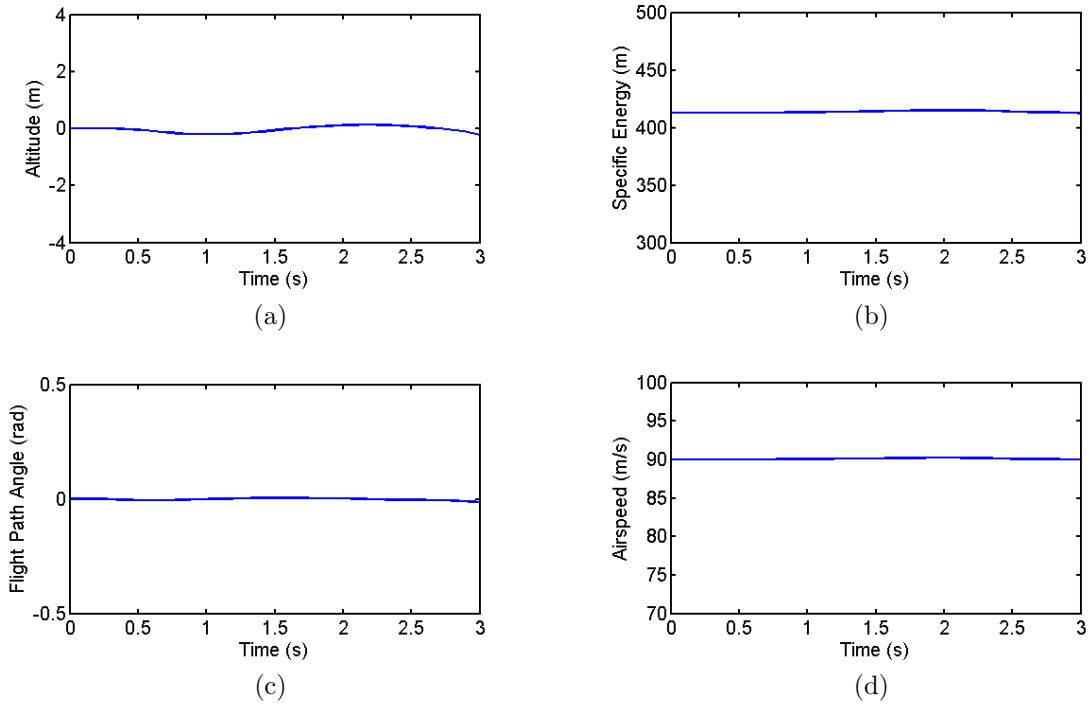


Figure 3.9: Genetic algorithm results for (a) the altitude, (b) specific energy, (c) flight path angle, and (d) airspeed.

Returning to the discussion of the use of higher order dynamical information in the optimization solution, the addition of actuator dynamics has a noticeable impact on the results of the aircraft turning problem, and further improvements may be expected from the extension to second-order collocation. In the GA phase of the solution, the actuator dynamics filter the commanded control such that the initialization is closer to the expected optimal solution. This is especially noticeable in the altitude histories. If the aircraft maintains a constant flight path angle (no altitude change) with a turn rate that produces a constant 5g loadfactor, then the time required to turn is 2.9416 seconds (shorter than the HLGL result using actuator dynamics). The solution without actuator dynamics produces an initialization similar to the HLGL result in Figure 3.16a, in which the aircraft changes its altitude more dramatically than the original results with actuator dynamics in Figures 3.9a and 3.13a. This additional change in altitude causes the path

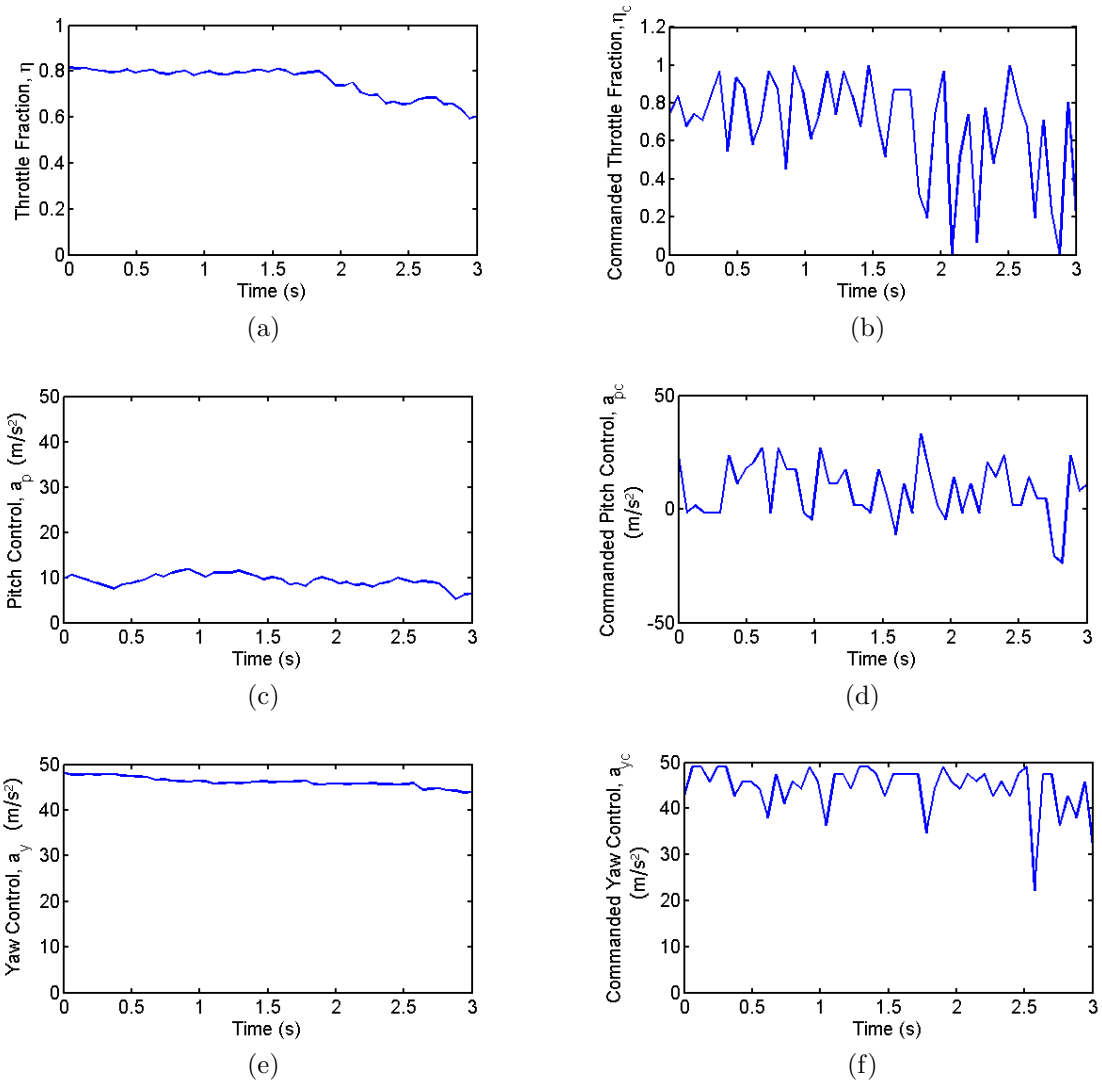


Figure 3.10: Genetic algorithm results for (a) throttle, (b) commanded throttle, (c) pitch control, (d) commanded pitch control, (e) yaw control, and (f) commanded yaw control.

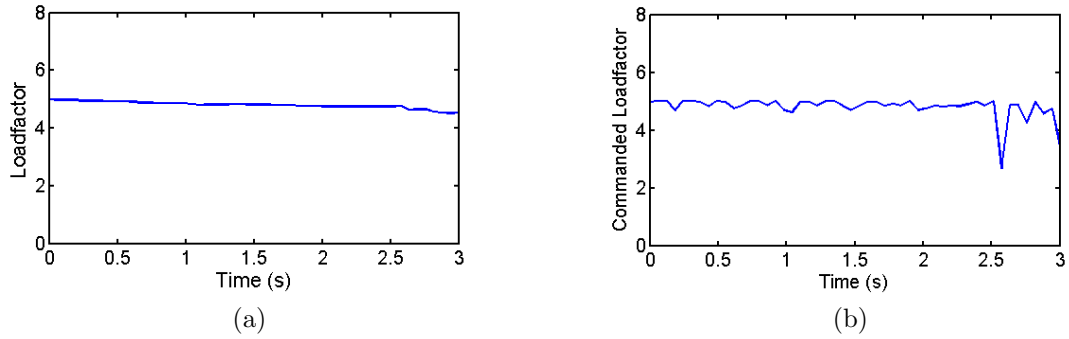
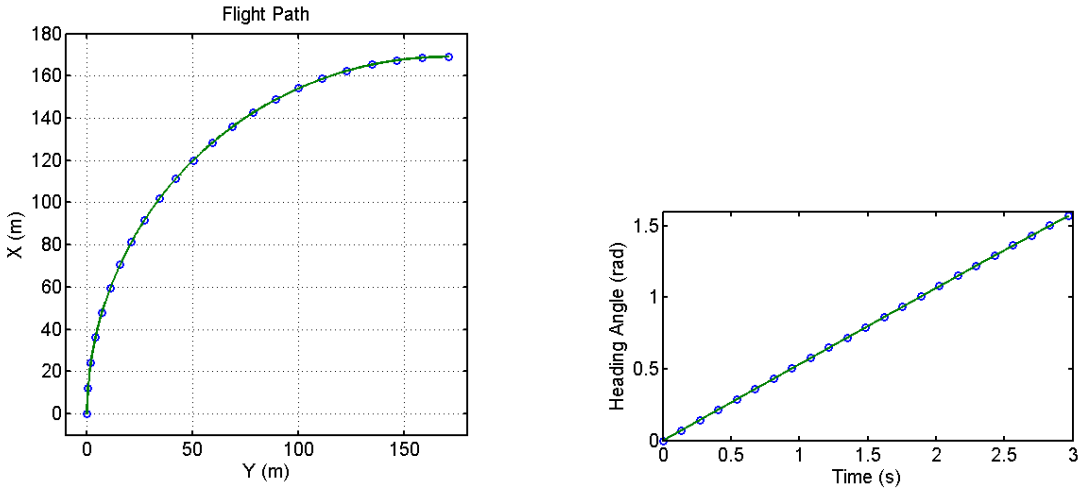
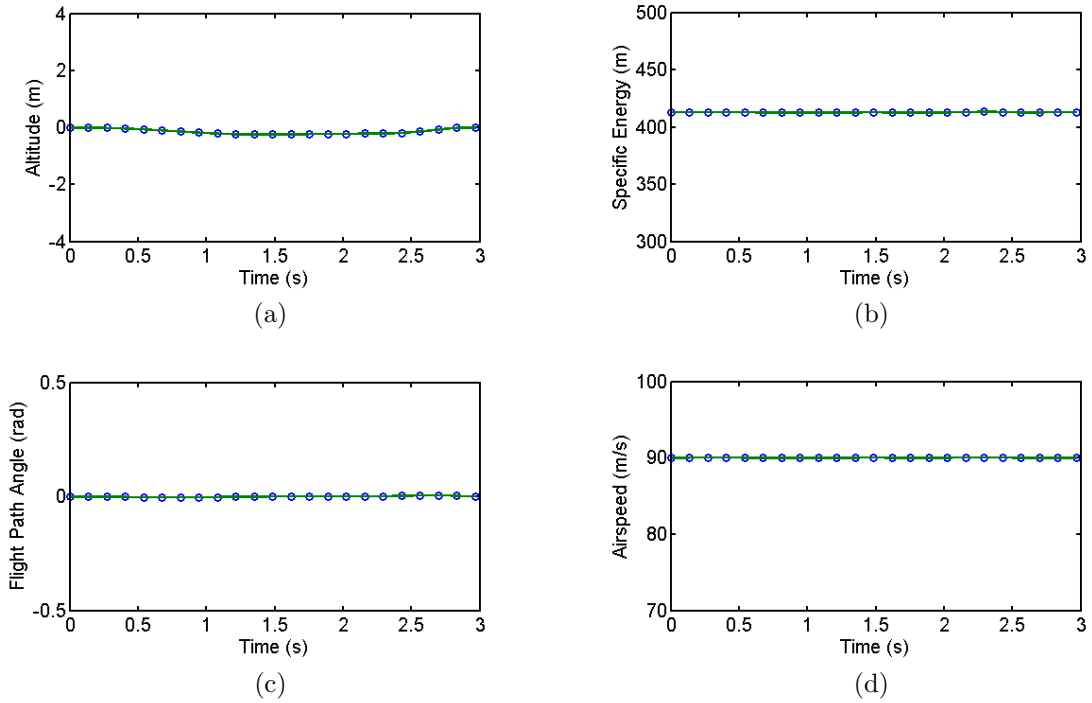


Figure 3.11: Genetic algorithm results for (a) loadfactor and (b) commanded loadfactor.



(a) (b)

Figure 3.12: HLGL collocation results for (a) the flight path in the horizontal plane and (b) heading angle.



(a) (b)

(c) (d)

Figure 3.13: HLGL collocation results for (a) the altitude, (b) specific energy, (c) flight path angle, and (d) airspeed.

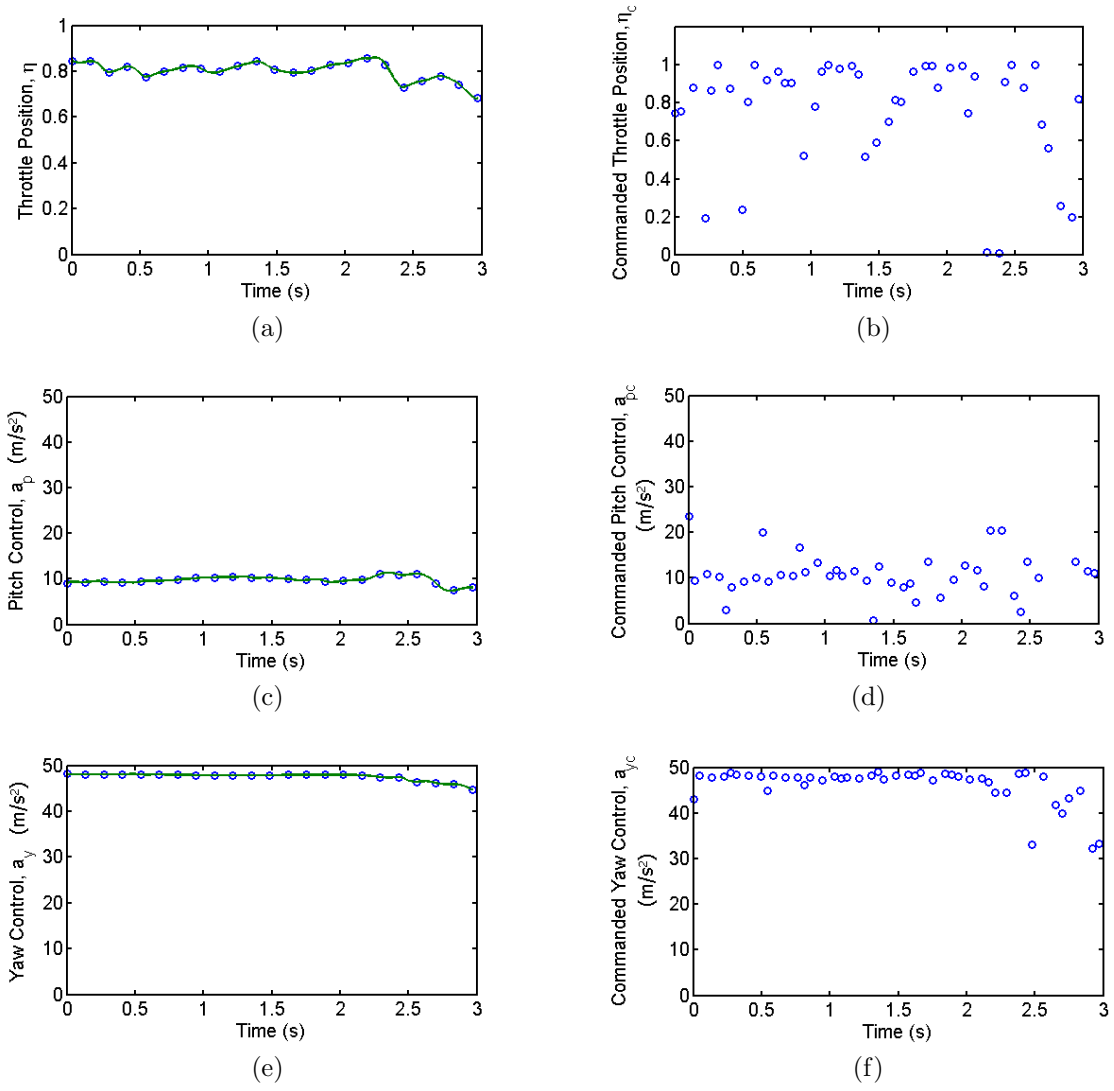


Figure 3.14: HLGL collocation results for (a) throttle, (b) commanded throttle, (c) pitch control, (d) commanded pitch control, (e) yaw control, and (f) commanded yaw control.

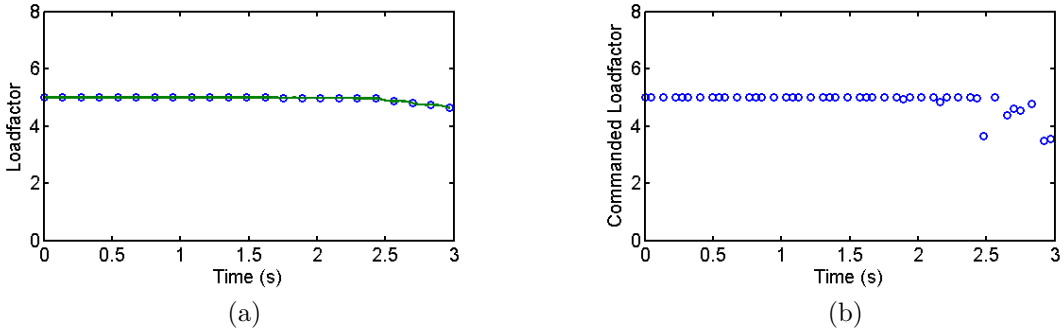
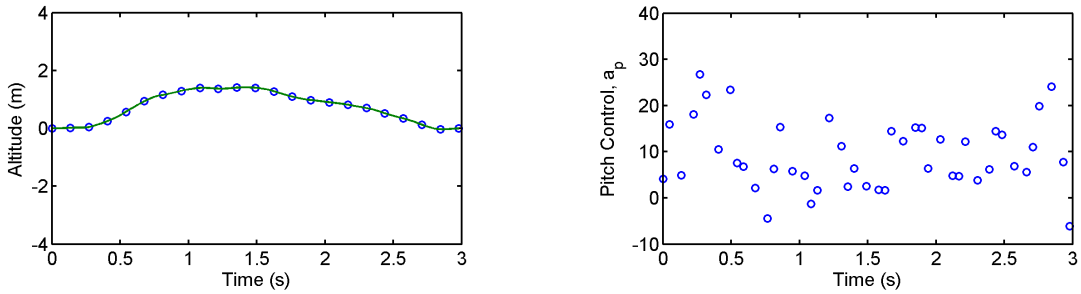


Figure 3.15: HLGL collocation results for (a) loadfactor and (b) commanded loadfactor.

of the trajectory to be slightly longer than the solution with actuator dynamics. The differences in the results are presumed to be numerically caused by the addition of extra NLP variables for the actuator dynamics, not the dynamical characteristics of the system.



(a) (b)

Figure 3.16: HLGL collocation results without the use of lateral and longitudinal actuator dynamics for (a) altitude and (b) pitch control.

Augmentation with actuator dynamics also benefits the solution by smoothening the control values a_p and a_y. Though the commanded signals a_{pc} and a_{yc} still exhibit unsmooth behavior, the solution may still be considered improved since the open-loop trajectory design does not necessarily require the commanded control histories. These can be designed in post-trajectory-design closed-loop control laws.

Finally, the addition of actuator dynamics allows for the extension of the collocation technique to include second-order constraints. Since the second-order constraints include the control rates and state accelerations, the extension is expected to increase the dynamical accuracy of the states and controls with respect to each other. Therefore, we would expect to see less fluctuation in both the controls and their command inputs unless those fluctuations are more accurate and produce a more optimal trajectory.

3.3 Low-Thrust Interplanetary Transfer

Many trajectory optimization or planning problems encounter points in the trajectory where discontinuities occur in the states, their derivatives, or the controls. Some may require a transformation of coordinates. One such problem is the minimum-time low-thrust interplanetary trajectory [6]. This problem consists of three continuous segments with two intermediate changes of coordinates and dynamical constants, as will be shown. These aspects of the trajectory design provide motivation for development of the multi-segment trajectory solution framework. Modifications to the trajectory optimization framework developed for this problem were made to accommodate such multi-segment trajectories. As such, the considerations for this problem include additional considerations necessary for the interfacing of segments.

3.3.1 Problem Statement

Given the dynamical models below, the solution to this problem should yield a minimum-time low-thrust transfer between Earth and Mars. The problem definition is taken directly from the literature [6] and its formal statement follows: *i*) the spacecraft should begin in a geosynchronous orbit and end in an areosynchronous orbit (about Mars); *ii*) the transfer should occur in minimum time.

In the context of the optimal control problem described above, the cost function to be minimized is given by

$$J = t_{f_1} + t_{f_2} + t_{f_3}$$

where t_{f_1}, t_{f_2}, and t_{f_3} are the lengths (in time) of the geocentric, heliocentric, and areocentric trajectory segments, respectively.

The nonlinear state dynamics describe the two-body motion of a point-mass about the sun, Earth, or Mars. The simplification of the model to two dimensions throughout

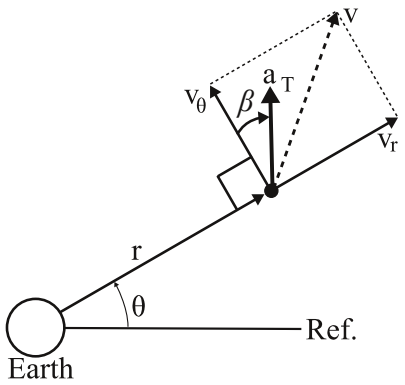


Figure 3.17: Coordinates and control of the spacecraft.

the entire trajectory relies upon the assumption that the orbits of Earth and Mars are coplanar. It should be noted that the size of the problem would increase dramatically if three-dimensional trajectories were considered. The problem setup is further simplified by the assumption that the orbits of Earth and Mars are circular. Though this last assumption is not necessary even to limit the size of the problem, it is acceptable because the primary motivation to solve this problem is the implementation of the multi-segment trajectory optimization framework, not a high-fidelity trajectory design.

The equations of motion are

$$\begin{aligned} \dot{r} &= v_r \\ \dot{\theta} &= (v_\theta / r) \\ \dot{v}_r &= (v_\theta^2 / r) - (\mu / r^2) + a_T \sin\beta \\ \dot{v}_\theta &= -(v_r v_\theta / r) + a_T \cos\beta \end{aligned} \tag{3.41}$$

where r and θ are the polar coordinates of the spacecraft. The states v_r and v_θ are the perpendicular radial and tangential components of the velocity vector. The parameter a_T is the control acceleration magnitude, which is held constant at 0.0001g. The control input, β, is the thrust angle measured positive upward from the local horizon. The geocentric coordinates and thrust angle of the spacecraft are shown in Figure 3.17. The

Table 3.1: Conversion factors for normalized geocentric, heliocentric, and areocentric units.

Normalized Unit	Conventional Units [23]
Geocentric (R_E = 1 Earth Radius)	
1 DU_E (R_E)	6,378.137 km
1 TU_E	0.00933809 solar days
Heliocentric (AU = 1 Astronomical Unit)	
1 DU_S (AU)	149,597,870 km
1 TU_S	58.132 solar days
Areocentric (R_M = 1 Mars Radius)	
1 DU_M (R_M)	3397.2 km
1 TU_M	0.011045 solar days

gravitational parameter, μ, is normalized to 1 DU^3/TU^2 for all three segments. The units DU and TU are the normalized astronomical "distance unit" and "time unit", respectively. They are defined differently for each segment. Some conversion factors relating the normalized astronomical units to conventional units is provided in Table 3.1. Note that the state names used in Equations (3.41) are appropriate for the geocentric segment of the trajectory only. Though the form of the dynamics is the same for each segment, description of the inter-segment constraints requires that different state names be applied for each segment. The geocentric states $(r, \theta, v_r, v_\theta)$ are replaced by (R, η, v_R, v_η) during heliocentric flight and $(\rho, \delta, v_\rho, v_\delta)$ during areocentric flight.

Since each segment of the trajectory has its own endpoint, path, and box constraints, they are treated separately. Constraints involving more than one segment are discussed later in this section.

The first segment consists of a geocentric trajectory (the spacecraft is influenced only by the gravity of Earth) beginning from a circular geosynchronous orbit to a given final radius. The endpoint constraints include the initial and final radius and the initial

radial and tangential velocities. The initial radius is calculated as that of a circular orbit with a period of $P = 1$ day by [24]

$$r = \left(\frac{\mu}{(2\pi/P)^2} \right)^{1/3} = 6.6107 \ R_E. \tag{3.42}$$

The final radius is chosen to be the radius of the sphere of influence for Earth, 145 R_E, in keeping with the example in [6] for comparison. The initial radial velocity is zero for a circular orbit and the initial tangential velocity is calculated by

$$v_\theta = \sqrt{\frac{\mu}{r}} = 0.38893 \ R_E/\mathrm{TU_E} \tag{3.43}$$

for circular orbits [23].

Path and box constraints are added for stability of the NLP. The primary reason to include such constraints is that early executions of the HLGL/NLP algorithm on this problem were numerically unstable due to the fast dynamical behavior of the system when the orbital radius is smaller than the desired planetosynchronous orbit. The addition of box constraints seemed to improve NLP performance when the GA was not used for initialization, but a satisfactory solution was ultimately achieved only after initializing the NLP with the GA results.

Bounds are applied to the radius, given by the geosynchronous orbit radius and the sphere of influence radius,

$$6.6107 \ R_E \leq r \leq 145 \ R_E. \tag{3.44}$$

An upper bound on the tangential velocity equal to the tangential velocity of the geosynchronous orbit,

$$v_\theta \leq 0.38893 \ R_E/\mathrm{TU_E} \tag{3.45}$$

is enforced.

The second (heliocentric) segment of the trajectory has lower and upper bounds on the orbital radius.

$$0.9 \text{ AU} \leq R \leq 1.62 \text{ AU} \tag{3.46}$$

The third (areosynchronous) segment requires similar constraints to the geosynchronous segment. The endpoint constraints include initial radius, final radius, final radial velocity, and final tangential velocity. The initial radius is the radius of the sphere of influence for Mars, 170 R_M, in keeping with the example in [6]. The final radius of $\rho = 6.0236\ R_M$ is calculated by Equation (3.42) using areocentric coordinates and a period equal to the rotational period of Mars, 1.0260 days [23]. The final radial velocity is zero for a circular orbit, and the final tangential velocity of $v_\delta = 0.40745\ R_M/\text{TU}_\text{M}$ is calculated by Equation (3.43) using the areocentric orbit radius found previously.

Similar to the geocentric segment, path and box constraints are used here to stabilize the NLP. These are

$$6.0236\ R_M \leq \rho \leq 170\ R_M \tag{3.47}$$

using the areosynchronous period and

$$v_\delta \leq 0.40745\ R_M/\text{TU}_\text{M} \tag{3.48}$$

using the areosynchronous radius, and

$$v_\rho \leq 0\ R_M/\text{TU}_\text{M}. \tag{3.49}$$

To address the constraints which must be applied between the phases to ensure proper transition, expressions may be composed using the geometry in Figures 3.18a and 3.18b. Between the geocentric and heliocentric segments, the states are related by the

constraints of Equations (3.50), and between the heliocentric and areocentric segments by those of Equations (3.51).

$$\begin{aligned} R\cos\alpha - r - a_E\cos(\theta-\phi) &= 0 \\ R\sin\alpha - a_E\sin(\theta-\phi) &= 0 \\ v_\eta\cos\alpha(1+\tan^2\alpha) - a_E\dot{\phi}\cos(\theta-\phi) - v_\theta & \\ - a_E\dot{\phi}\sin(\theta-\phi)\tan\alpha - v_r\tan\alpha &= 0 \\ v_R\cos\alpha - a_E\dot{\phi}\sin(\theta-\phi) - v_r + v_\eta\sin\alpha &= 0 \end{aligned} \tag{3.50}$$

$$\begin{aligned} R\sin\omega - a_M\sin(\delta-\psi) &= 0 \\ R\cos\omega - a_M\cos(\delta-\psi) - \rho &= 0 \\ v_\eta\cos\omega - v_R\sin\omega - a_M\dot{\psi}\cos(\delta-\psi) - v_\delta &= 0 \\ v_R\cos\omega + v_\eta\sin\omega - a_M\dot{\psi}\sin(\delta-\psi) - v_\rho &= 0. \end{aligned} \tag{3.51}$$

In the above expressions, $\alpha = \theta - \eta$ and $\omega = \delta - \eta$. The orbit angles of Earth and Mars at the points of transition between segments are

$$\phi = \dot{\phi}t_{f_1} \tag{3.52}$$

for Earth at time t_{f_1} and

$$\psi = MLA + \dot{\psi}(t_{f_1} + t_{f_2}) \tag{3.53}$$

for Mars at time $t_{f_1} + t_{f_2}$. In Equations (3.52) and (3.53), the mean motions of the planetary orbits are

$$\dot{\phi} = \sqrt{\frac{\mu}{a_E^2}} = 1 \text{ rad/TU}_\text{S} \tag{3.54}$$

$$\dot{\psi} = \sqrt{\frac{\mu}{a_M^2}} = 0.53169 \text{ rad/TU}_\text{S} \tag{3.55}$$

where $a_E = 1$ AU and $a_M = 1.5237$ AU are the circular orbital radii of Earth and Mars about the sun, respectively [23]. The parameter MLA is the angle by which Mars leads

Earth at the beginning of the geocentric segment. The specific choice of $MLA = 0.9666$ rad conforms to the example in [6].

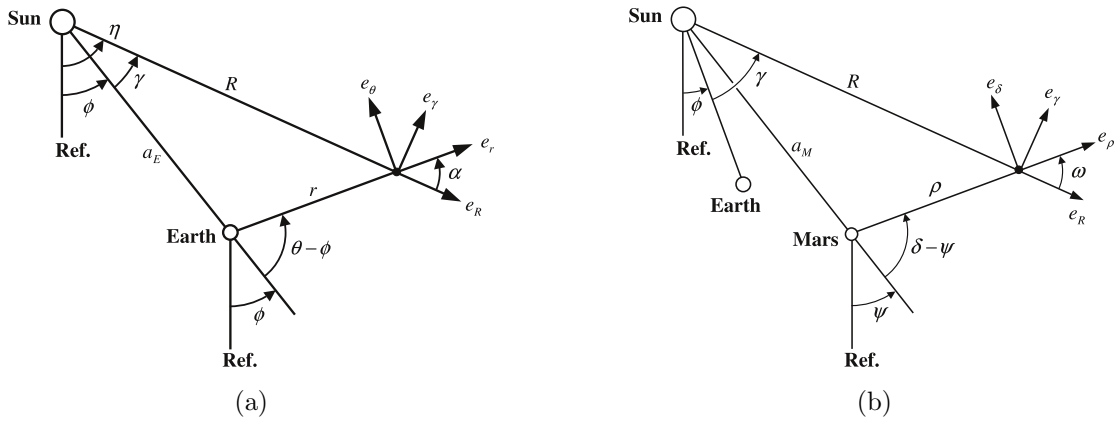


Figure 3.18: Geometry for transformations between (a) geocentric and heliocentric coordinates and (b) heliocentric and areocentric coordinates.

3.3.2 GA Considerations

Like the other problems solved in this work, special considerations must be addressed regarding the setup of this problem in the framework of the GA search. This problem, however, presents some additional complications due to the multiple segments of the trajectory, each having its own set of dynamics and boundary conditions. The considerations discussed here will include, first, an augmentation of the equations of motion (with a control filter) which aides the convergence of the genetic algorithm. Then the known boundary conditions are discussed considering the benefit of integrating the areocentric segment backwards in time. This leads discussion to the dependency of the B.C.'s of the heliocentric segment upon those of the geocentric segment and the transformation of coordinates between them. Finally, the costs and penalties used for ranking are provided.

This problem provides one example of a dynamical system with significant nonlinear effects. Examination of the equations of motion reveals that, when the spacecraft comes close to the attracting body, the natural frequency of the system is increased. That is, the period of the orbits decreases dramatically as the spacecraft flies lower to Earth or Mars. In order to sufficiently capture the behavior of the spacecraft, this requires a high density of nodes in both the GA and HLGL discretizations. However, the high density of nodes adversely affects the evolution and convergence of the GA, since the control value of each node has less effect on the trajectory overall. One possible way to address this issue is to filter the signal of the control input, making trends in the control history more prevalent and smoothing the actual thrust angle history. If the commanded thrust angle is then discarded in favor of the actual thrust angle when initializing the HLGL phase of optimization, then the NLP will be initialized with less collocation constraint error.

The filter is applied in each segment of the trajectory. The thrust angle is appended to the state vector. Its time derivative is

$$\dot{\beta} = -\lambda(\beta - \beta_c) \tag{3.56}$$

where β_c is the commanded thrust angle (control variable). The gain, λ, is 10^{-2} for the geocentric segment, 1 for the heliocentric segment, and 5×10^{-4} for the areocentric segment.

Rather than integrating all three segments of the trajectory forward in time, integrating the final (areocentric) segment backwards is beneficial to the convergence of the GA. In justifying the choice of integration direction, we first consider what boundary conditions are known for each segment. Any B.C.'s that are not inherently enforced due to the starting point of the integration must be enforced through evolution in the GA by a penalty function.

The geocentric segment has three known initial conditions (r, v_r, and v_θ) and only one known final condition (r). Since it is better to ensure that the three I.C.'s are met while letting the GA search only for the F.C., the geocentric phase is integrated forward, initialized by the I.C.'s. The initial orbit angle (θ) is included as an optimization parameter in the GA search.

In contrast, three of the F.C.'s of the areocentric segment are known (ρ, v_ρ, and v_δ) while only one I.C. is known (ρ). In this case, the GA will be more effective if only the initial radius is searched for. Therefore, the choice is to integrate the areocentric segment backwards.

For the heliocentric segment, none of the B.C.'s are known. The marching integration must be initialized by transforming the coordinates at the boundary of an adjacent segment. The coordinate transformation at the other boundary may be enforced through constraint violation penalties in the GA cost function. The choice is to integrate the heliocentric segment forward, initializing the states by transforming the final geocentric states (r, θ, v_r, and v_θ) into heliocentric coordinates (R, η, v_R, and v_η). The transformation is given by

$$\begin{aligned}
R &= \sqrt{(r + a_E \cos(\theta - \phi))^2 + (a_E \sin(\theta - \phi))^2} \\
\alpha &= \sin^{-1}\left(\frac{a_E}{R}\sin(\theta - \phi)\right) \\
\eta &= \theta - \alpha \\
v_\eta &= \frac{a_E\dot{\phi}\cos(\theta - \phi) + v_\theta + a_E\dot{\phi}\sin(\theta - \phi)\tan\alpha + v_r \tan\alpha}{\cos\alpha(1 + \tan^2\alpha)} \\
v_R &= \frac{a_E\dot{\phi}\sin(\theta - \phi) + v_r - v_\eta \sin\alpha}{\cos\alpha}
\end{aligned} \tag{3.57}$$

evaluated at time t_{f_1}. The penalties enforcing the constraints between the heliocentric F.C.'s and the areocentric I.C.'s are derived from Equations (3.51) and are provided later in this section.

The cost function used for ranking penalizes the cost function of the minimum-time problem definition and any constraint violations that apply. The total cost function used for the GA is

$$J = J_{seg1} + J_{seg2} + J_{seg3} + J_{2/3} \tag{3.58}$$

where the components J_{seg} correspond to penalties applying to the segment indicated by the subscript, and $J_{2/3}$ is the penalty for violation of the constraints that enforce the transformation of coordinates at the boundary between the heliocentric and areocentric segments. For the geocentric segment, the penalties include the final time, final orbit radius error, and the following constraint violations evaluated for all nodes: maximum orbit radius, minimum orbit radius, and maximum tangential velocity. The penalties are expressed as

$$\begin{aligned} J_{seg1} = t_{f_1} + 10^4 \left| r_{N_1} - 145\ R_E \right| + \sum_{i=1}^{N_1} w_{ru} \left| r_i - 145\ R_E \right| \\ + \sum_{i=1}^{N_1} w_{rl} \left| r_i - 6.6107\ R_E \right| + \sum_{i=1}^{N_1} w_{v\theta} \left| v_{\theta,i} - 0.38893\ R_E/\mathrm{TU_E} \right| \end{aligned} \tag{3.59}$$

where N_1 is the number of discrete nodes in the Runge-Kutta integration in the geocentric segment, and the weights are

$$w_{ru} = \begin{cases} 1, & r_i > 145\ R_E \\ 0, & r_i \leq 145\ R_E \end{cases} \tag{3.60}$$

$$w_{rl} = \begin{cases} 1, & r_i < 6.6107\ R_E \\ 0, & r_i \geq 6.6107\ R_E \end{cases} \tag{3.61}$$

$$w_{v\theta} = \begin{cases} 1, & v_\theta > 0.38893\ R_E/\mathrm{TU_E} \\ 0, & v_\theta \leq 0.38893\ R_E/\mathrm{TU_E}. \end{cases} \tag{3.62}$$

For the heliocentric segment, the penalties include the final time (t_{f_2}) and penalties evaluated at all nodes for violation of maximum and minimum orbit radii. The penalties are

$$J_{seg2} = 10^{-2} t_{f_2} + \sum_{i=1}^{N_2} w_{Ru} \left| R_i - 1.62 \text{ AU} \right| + \sum_{i=1}^{N_2} w_{Rl} \left| R_i - 0.9 \text{ AU} \right| \tag{3.63}$$

where N_2 is the number of discrete nodes used in the Runge-Kutta integration for the heliocentric segment, and the weights are

$$w_{Ru} = \begin{cases} 1, & R_i > 1.62 \text{ AU} \\ 0, & R_i \leq 1.62 \text{ AU} \end{cases} \tag{3.64}$$

$$w_{Rl} = \begin{cases} 1, & R_i < 0.9 \text{ AU} \\ 0, & R_i \geq 0.9 \text{ AU}. \end{cases} \tag{3.65}$$

For the areocentric segment, the penalties include the final time (t_{f_2}), the error of the initial radius, and the following constraint violations evaluated at all nodes: maximum radial velocity, maximum tangential velocity, and maximum and minimum orbit radius. The penalties are expressed as

$$\begin{aligned} J_{seg3} = t_{f_3} + 10^2 \left| \rho_{N_3} - 170 \ R_M \right| + \sum_{i=1}^{N_3} w_{\rho u} \left| \rho_i - 170 \ R_M \right| + \sum_{i=1}^{N_3} w_{v\rho} \left| v_{\rho,i} \right| \\ + \sum_{i=1}^{N_3} w_{\rho l} \left| \rho_i - 6.0236 \ R_M \right| + \sum_{i=1}^{N_3} w_{v\delta} \left| v_{\delta,i} - 0.40745 \ R_M/\text{TU}_\text{M} \right| \end{aligned} \tag{3.66}$$

where N_3 is the number of discrete nodes in the Runge-Kutta integration for the areocentric segment, and the weights are

$$w_{\rho u} = \begin{cases} 1, & \rho_i > 170 \ R_M \\ 0, & \rho_i \leq 170 \ R_M \end{cases} \tag{3.67}$$

$$w_{\rho l} = \begin{cases} 1, & \rho_i < 6.0236 \ R_M \\ 0, & \rho_i \geq 6.0236 \ R_M \end{cases} \tag{3.68}$$

$$w_{v\rho} = \begin{cases} 1, & v_{\rho,i} > 0 \ R_M/\mathrm{TU_M} \\ 0, & v_{\rho,i} \leq 0 \ R_M/\mathrm{TU_M} \end{cases} \tag{3.69}$$

$$w_{v\delta} = \begin{cases} 1, & v_{\delta,i} > 0.40745 \ R_M/\mathrm{TU_M} \\ 0, & v_{\delta,i} \leq 0.40745 \ R_M/\mathrm{TU_M}. \end{cases} \tag{3.70}$$

Finally, the component of the cost penalizing constraint violations at the boundary connecting the heliocentric and areocentric segments is formulated from the constraints of Equations (3.51) and is expressed as

$$J_{2/3} = |c_1| + |c_2| + |c_3| + |c_4| \tag{3.71}$$

where

$$c_1 = R\sin\omega - a_M \sin(\delta - \psi) \tag{3.72}$$

$$c_2 = R\cos\omega - a_M \cos(\delta - \psi) - \rho \tag{3.73}$$

$$c_3 = v_\eta \cos\omega - v_R \sin\omega - a_M\dot{\psi}\cos(\delta - \psi) - v_\delta \tag{3.74}$$

$$c_4 = v_R \cos\omega + v_\eta \sin\omega - a_M\dot{\psi}\sin(\delta - \psi) - v_\rho. \tag{3.75}$$

3.3.3 Results

Before discussing the results, a few parameters used to generate them are provided. For the genetic algorithm, the chromosome uses real encoding for the parameters rather than binary encoding. This reduces the size of the chromosome, since the array used to store the digits is smaller. For a problem of this size and complexity, this may relieve some of the computational burden due to storage and base conversion of the chromosomes. The GA chromosome operators may also evolve the candidate solutions more effectively (quickly) than when a binary encoded chromosome to be used. Though the real-coded GA is successful in generating an initialization for the collocation phase of optimization for this problem, the previous statement is not asserted here.

The genetic algorithm is parameterized as follows. The population is held at 150 chromosomes and is propagated for 75 generations. The final times of the three trajectory segments, as well as the initial orbit angle of the geocentric segment and the final orbit angle of the areocentric segment, are treated as optimization parameters in the chromosomes. The search bounds for each parameter are provided in Table 3.2. The maximum time step for each segment is chosen separately. For the geocentric segment, the maximum time step is 10 $\mathrm{TU_E}$, making the length of time between nodes equal to $t_{f_1}/370$ $\mathrm{TU_E}$ as the final time changes. For the heliocentric segment, the maximum time step is 0.1 $\mathrm{TU_S}$, making the time between nodes $t_{f_2}/35$ $\mathrm{TU_S}$. For the areocentric segment, the maximum time step is 1 $\mathrm{TU_M}$, making the time between nodes $t_{f_3}/2700$ $\mathrm{TU_M}$. The time steps are chosen by trial and error such that the Runge-Kutta integrations do not diverge due to numerical error, and are not intended to yield high-fidelity results.

Table 3.2: Search bounds for parameters in the genetic algorithm.

Parameter	Lower Bound	Upper Bound
Seg. 1 Final Time, t_{f_1}	2500 $\mathrm{TU_E}$	3700 $\mathrm{TU_E}$
Seg. 2 Final Time, t_{f_2}	2.5 $\mathrm{TU_S}$	3.5 $\mathrm{TU_S}$
Seg. 3 Final Time, t_{f_3}	1500 $\mathrm{TU_M}$	2700 $\mathrm{TU_M}$
Seg. 1 Initial Orbit Angle, θ	0 rad	2π rad
Seg. 3 Final Orbit Angle, δ	0 rad	2π rad

The HLGL collocation phase of the optimization is parameterized such that the nodes and intervals are the same as in [6]. All three segments use the third-order Hermite interpolating polynomial, making the collocation equivalent to the Hermite-Simpson method. The geocentric and areocentric segments have 80 time intervals each, while the heliocentric segment has 20 intervals. This is intended to provide a sufficient density of nodes to capture the behavior of the dynamics. The convergence criteria for the NLP is a

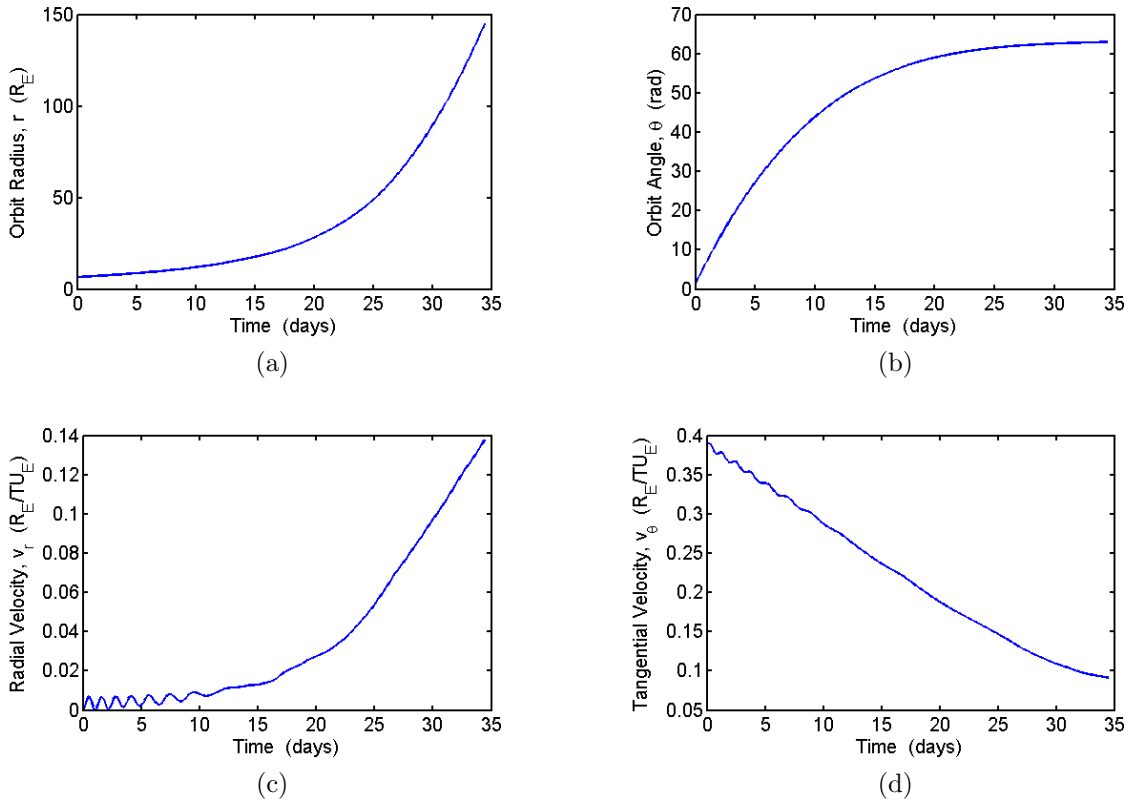
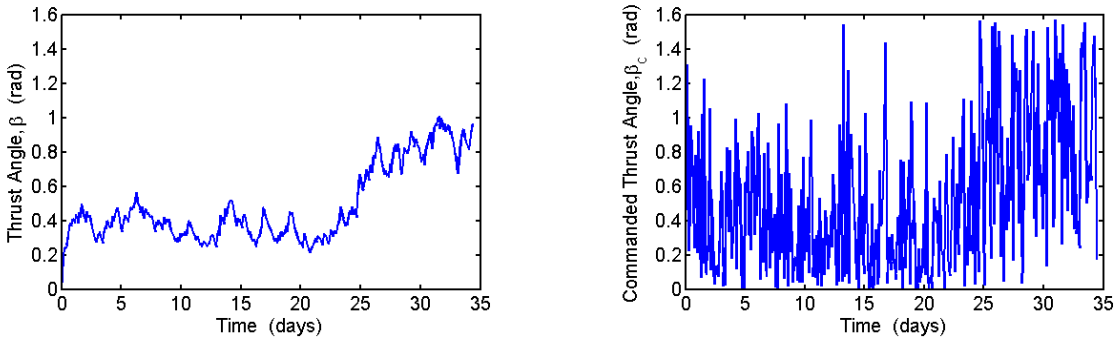


Figure 3.19: GA results for the geocentric segment states: (a) orbit radius, (b) orbit angle, (c) radial velocity, and (d) tangential velocity.

maximum constraint violation of 1×10^{-12} (`TolCon`) and a reduction in the cost function of less than 2×10^{-6} (`TolFun`) for one iteration.

The final resulting candidate solution by the genetic algorithm is shown in Figures 3.19 through 3.27. A few of the problem requirements are met very well. For instance, the final orbit radius of the geocentric segment (resulting in 145.000 R_E) and the initial orbit radius of the areocentric segment (170.004 R_M) are satisfied very well. It is also apparent from Figure 3.20a that the thrust angle follows the upward trend shown later in the HLGL solution, as well as in [6].

The poorly met requirements that are most noticeable are the final conditions of the heliocentric trajectory. In this segment, the spacecraft overshoots the sphere of influence



(a) (b)

Figure 3.20: GA results for the geocentric segment (a) thrust angle and (b) commanded thrust angle.

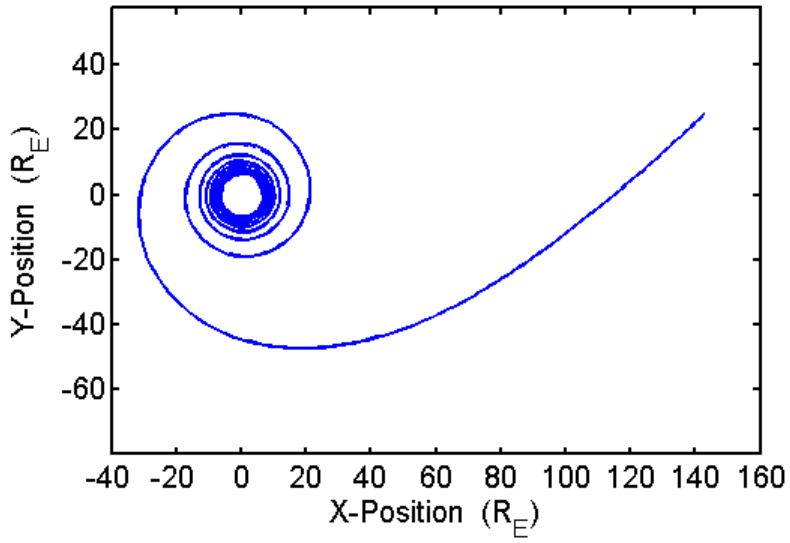


Figure 3.21: GA result for the Earth departure spiral.

of Mars by a significant distance even though error penalties are applied at the boundary between the heliocentric and areocentric segments to ensure proper continuity of position and velocity. This is also revealed by comparing the heliocentric radial velocity in Figure 3.22c to that of the HLGL result, shown later in Figure 3.31c. The more accurate solution

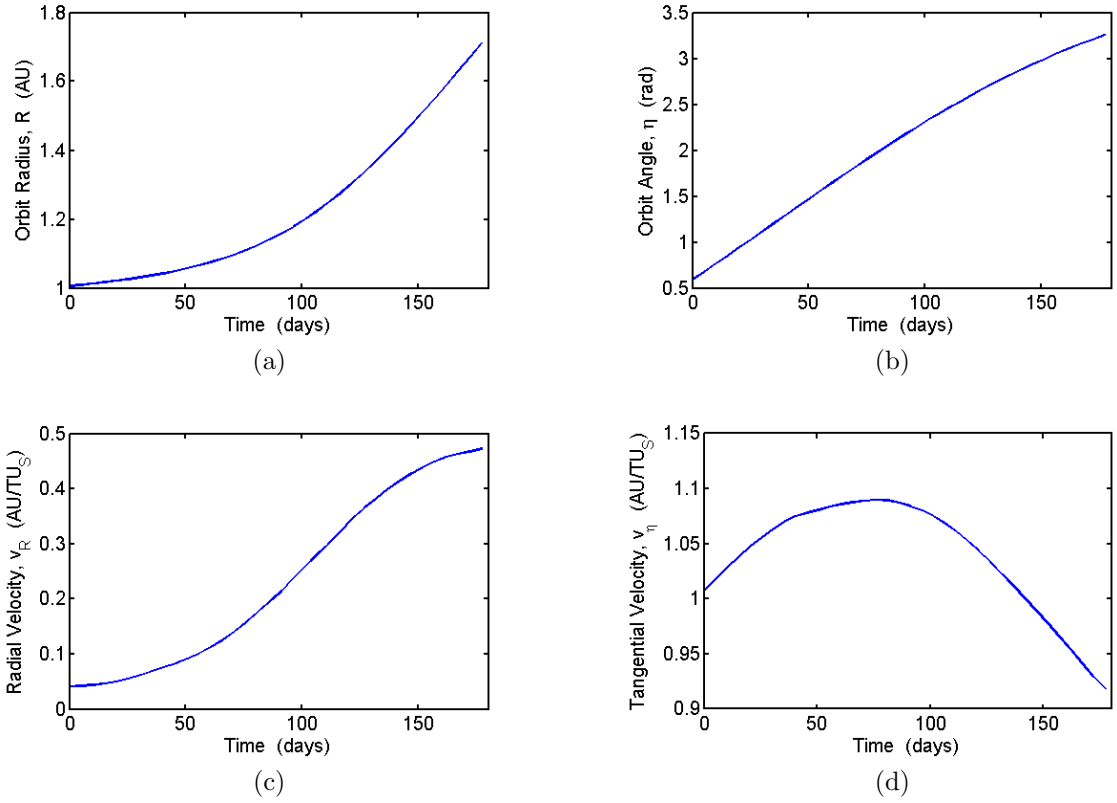


Figure 3.22: GA results for the heliocentric segment states: (a) orbit radius, (b) orbit angle, (c) radial velocity, and (d) tangential velocity.

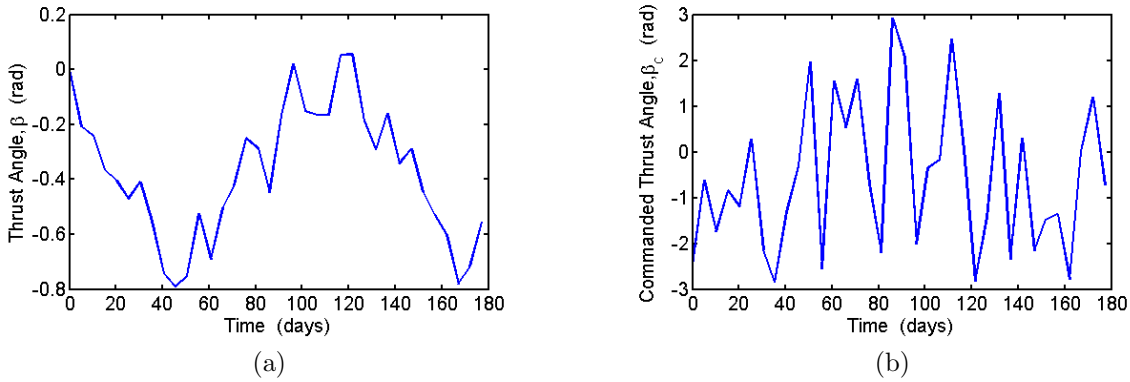


Figure 3.23: GA results for the heliocentric segment (a) thrust angle and (b) commanded thrust angle.

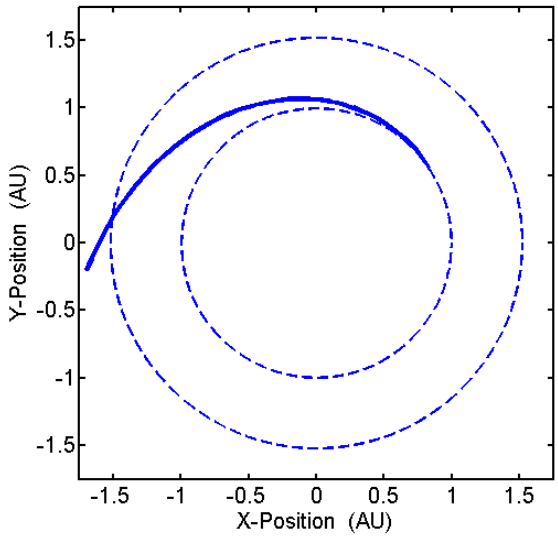


Figure 3.24: GA result for the heliocentric transfer orbit.

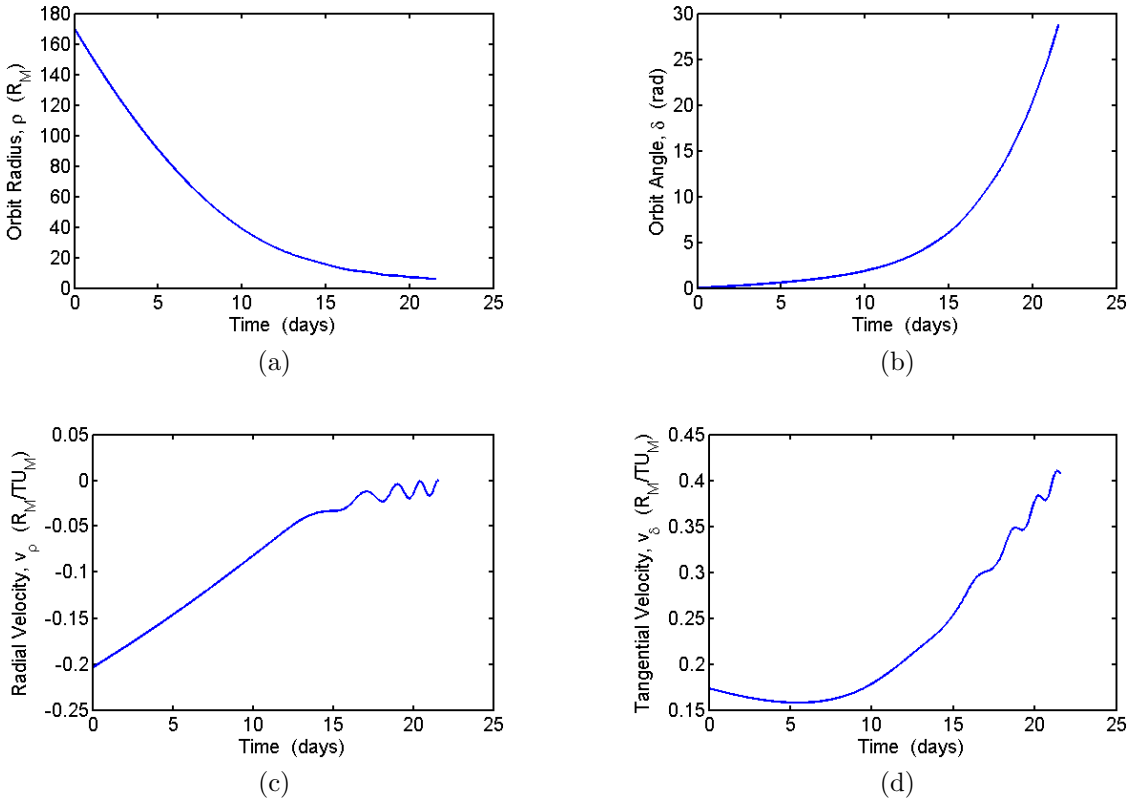
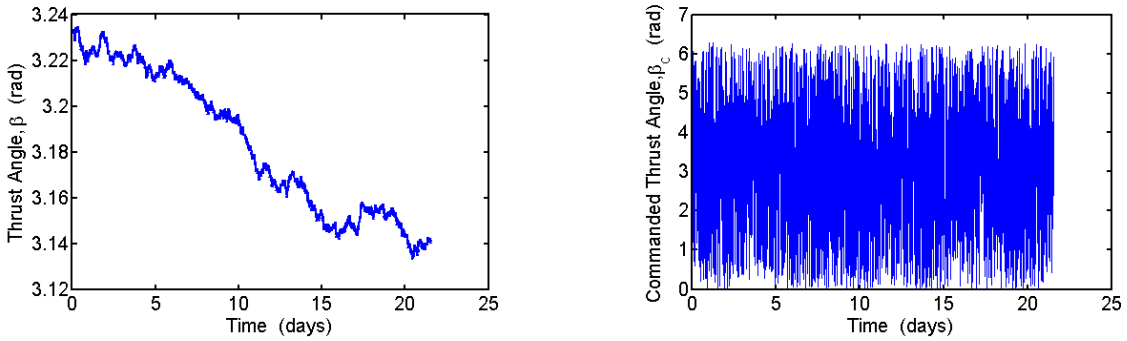


Figure 3.25: GA results for the areocentric segment states: (a) orbit radius, (b) orbit angle, (c) radial velocity, and (d) tangential velocity.



(a) (b)

Figure 3.26: GA results for the areocentric segment (a) thrust angle and (b) commanded thrust angle.

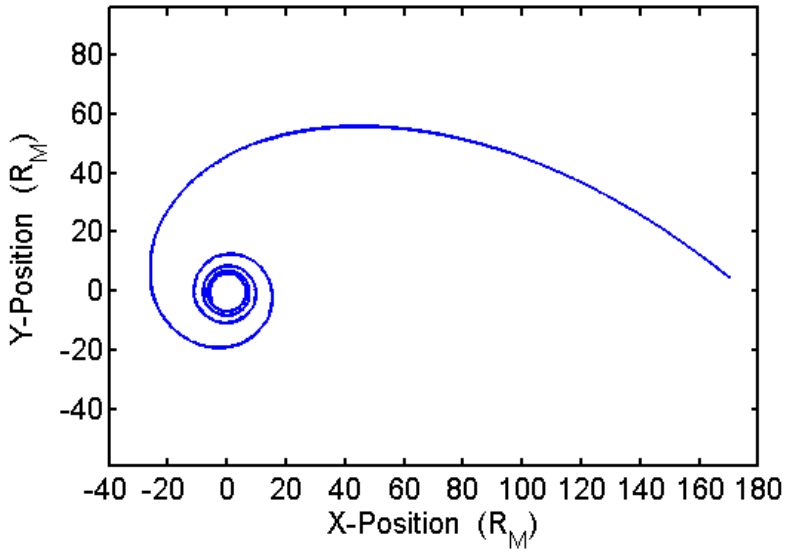


Figure 3.27: GA result for the Mars capture spiral.

by the HLGL collocation phase shows that the radial velocity of the spacecraft should decrease near the end of the heliocentric segment as its position approaches Mars.

Neither the thrust angle in the heliocentric nor areocentric segments follow the trends given by the HLGL solution. In the heliocentric phase, the initially downward-pointing thrust angle may be due to the need to initially compensate for any residual

radial velocity of the spacecraft after leaving the sphere of influence of Earth. This residual radial velocity is noticeable in Figure 3.22c.

The results given by the genetic algorithm may be improved through some adjustment of parameters in the cost function and the population. Firstly, the overshoot experienced by the spacecraft near Mars during the heliocentric segment is a product of the de-emphasis of the penalty applied to the states at the boundary of the heliocentric and areocentric segments. Increasing the magnitude of the penalty given by Equation (3.71) would, likewise, increase the importance of those constraints in the GA search. Other constraint violations, or sub-optimal behaviors in general, may be mitigated by increasing the number of individuals in the population to provide greater diversity. The final GA solution may also be improved by propagating the population for more generations, allowing better candidate solutions to evolve. Although improvements to the GA solution may be possible, the inaccurate result is shown to provide a reasonable initialization for the HLGL collocation phase.

Upon examination of the final result of the HLGL collocation phase (Figures 3.28 through 3.36), all of the problem boundary condition constraints are very well met, including the constraints relating the states of the heliocentric segment to the planetocentric segments at their boundaries. The final times of the three segments are given in Table 3.3 along with the final times published in the literature [6]. Although the final times are in good agreement with the published values, the results can still only be considered suboptimal. The reason for this becomes apparent in the geocentric and areocentric thrust angle histories shown in Figures 3.29 and 3.35. In the geocentric segment, the thrust angle history follows the published trend of near tangential thrusting early in the trajectory with a dramatic increase in the angle near the end. However, the HLGL results here show spurious (or suboptimal) un-smooth behaviors that are not present in the results of [6].

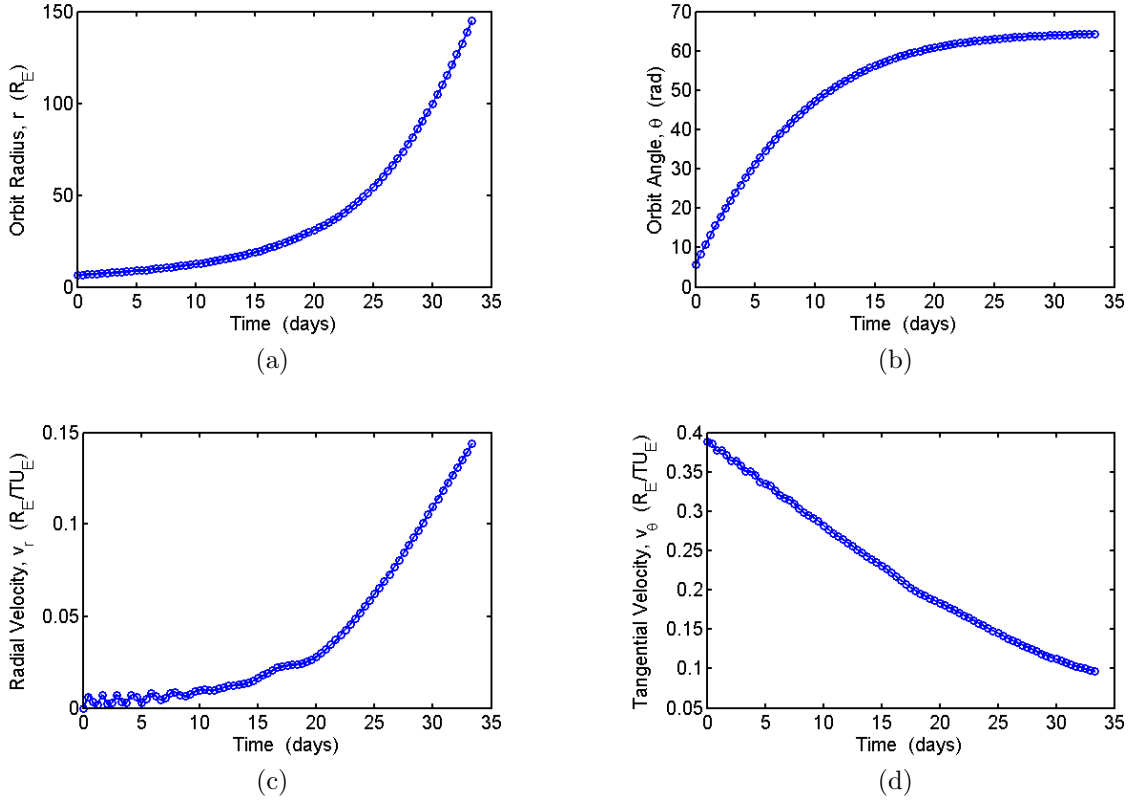


Figure 3.28: HLGL results for the geocentric segment states: (a) orbit radius, (b) orbit angle, (c) radial velocity, and (d) tangential velocity.

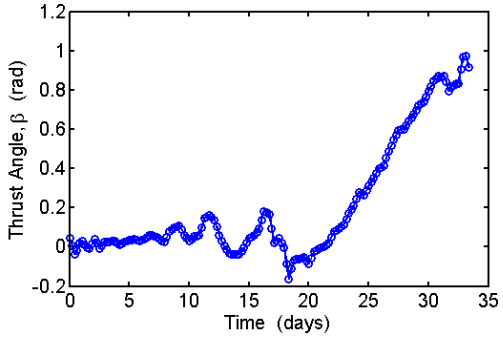
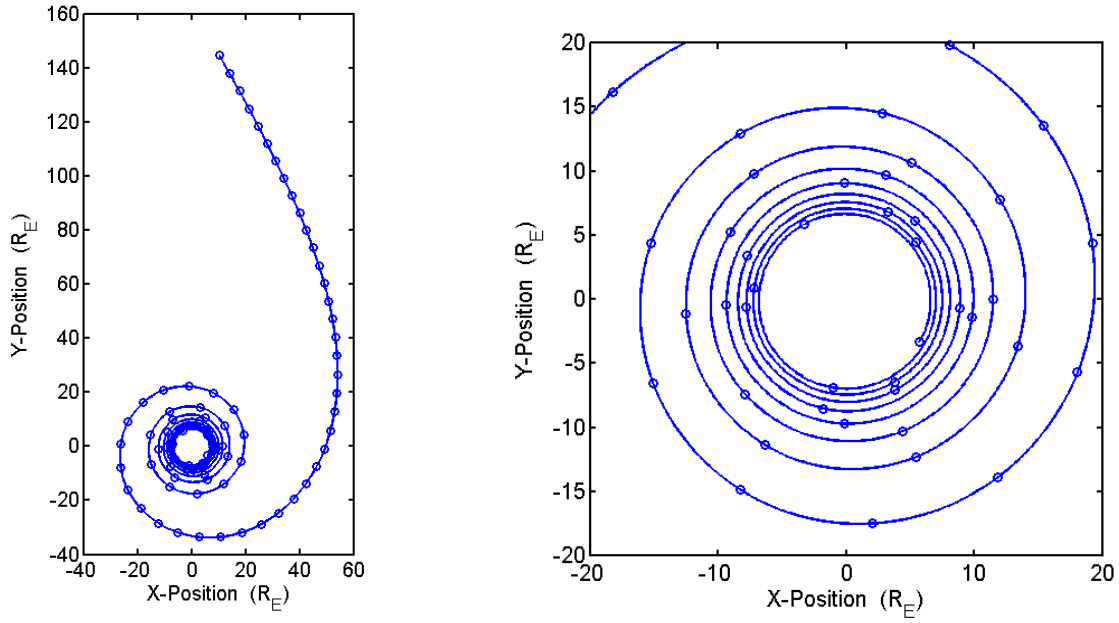


Figure 3.29: HLGL results for the geocentric segment thrust angle.



(a) (b)

Figure 3.30: HLGL result for the Earth departure spiral. (a) shows the entire spiral. (b) shows a close-up of the early part of the spiral.

The areocentric control history also exhibits some behavior that is suboptimal by comparison to the that of [6]. The published results show the thrust angle climbing from a nearly radial thrusting direction to oscillate about π rad in the latter portion of the trajectory.

Though the NLP did exit due to satisfaction of the convergence criteria (low directional derivative of the cost and low constraint violations) and Figure 3.35 reflects a

Table 3.3: Final times for the geocentric, heliocentric, and areocentric segments.

Segment	Final Time (days)	Published (days) [6]
Seg. 1 (geocentric)	33.34	33.08
Seg. 2 (heliocentric)	169.90	169.78
Seg. 3 (areocentric)	19.90	19.28
Total	223.14	222.14

result similar to what is published, the solution may still benefit from further NLP iterations with a stricter convergence criterion on the directional derivative. However, for a problem of this size and complexity, execution of the NLP (**fmincon()**) is prohibitively long. In this work, as the iterations of the NLP converged, the convergence criteria were adjusted to be more strict. Using the new convergence criteria, the NLP was re-initialized and allowed to converge again. This was repeated several times to improve the results and obtain the figures presented in this section. Convergence after each re-initialization was observed to take several hours.

Though the second-order augmentation discussed in Section 2.3.1 may more effectively influence the constraint violations with the control values, the size of this problem may be prohibitive to the addition of the necessary actuator dynamics or control filters in the NLP/collocation phase. However, if the interpolants in the collocation formulation are more accurate due to the increased dynamical information included, then fewer intervals may be required.

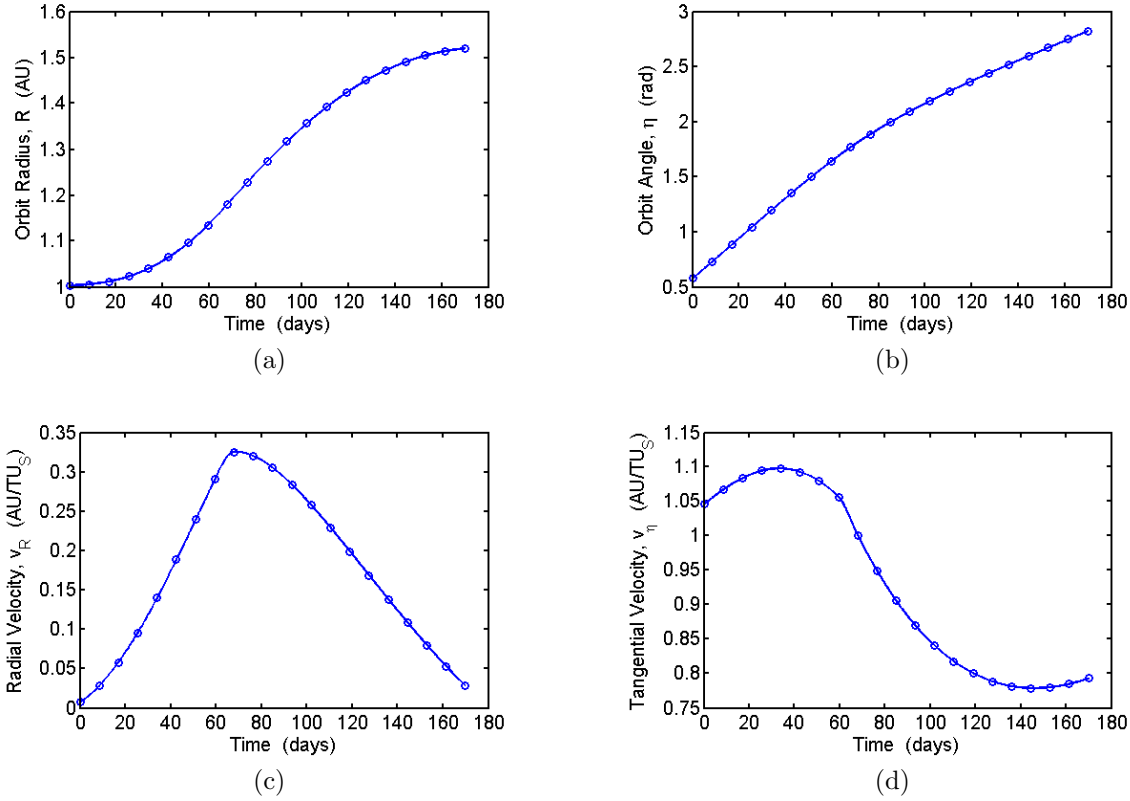


Figure 3.31: HLGL results for the heliocentric segment states: (a) orbit radius, (b) orbit angle, (c) radial velocity, and (d) tangential velocity.

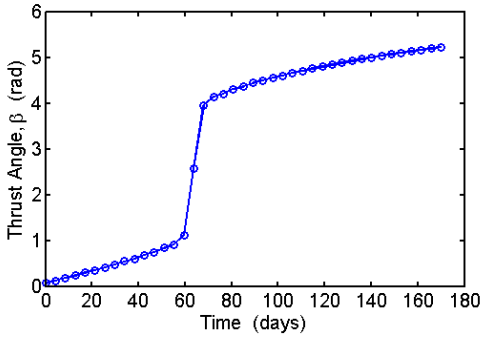


Figure 3.32: HLGL results for the heliocentric segment thrust angle.

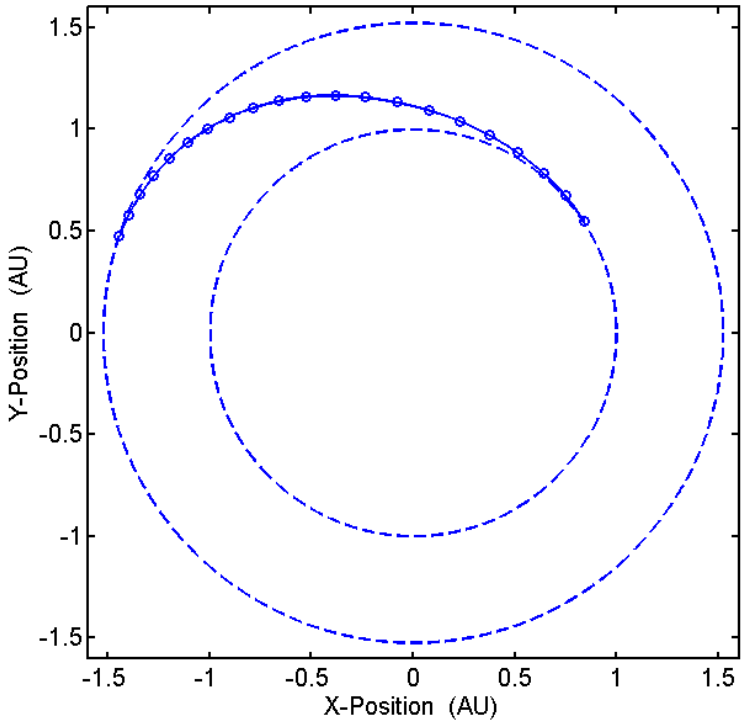


Figure 3.33: HLGL result for the heliocentric transfer orbit.

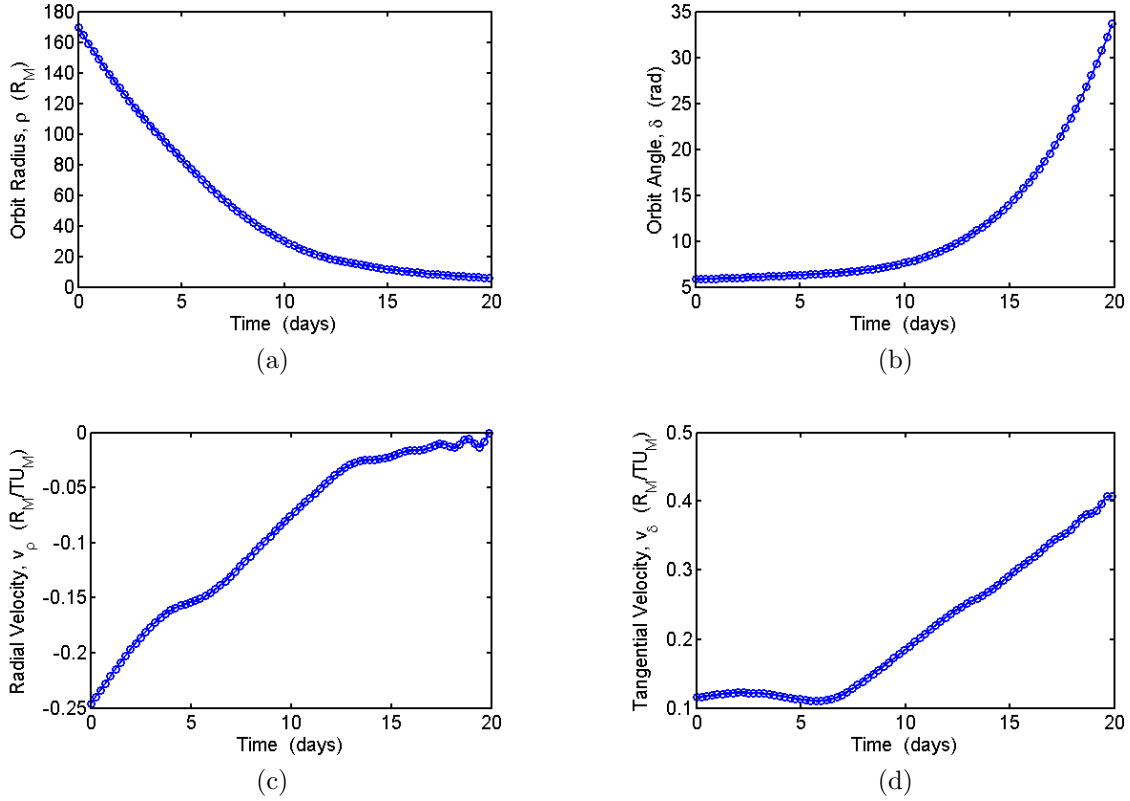


Figure 3.34: HLGL results for the areocentric segment states: (a) orbit radius, (b) orbit angle, (c) radial velocity, and (d) tangential velocity.

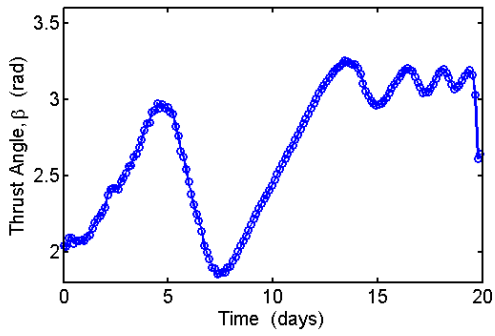
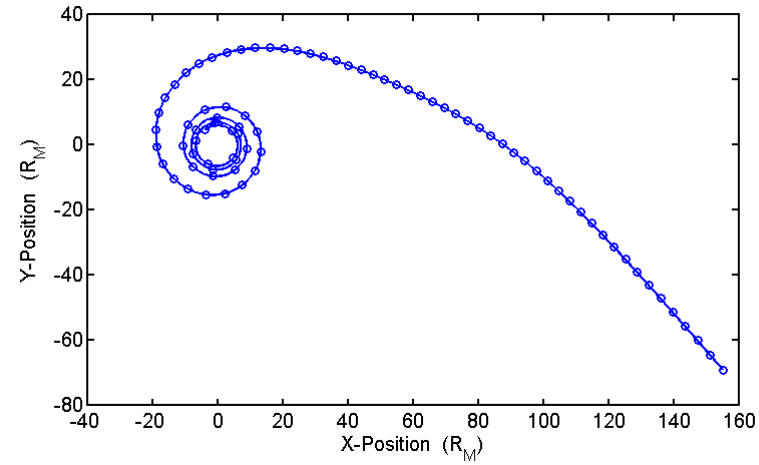
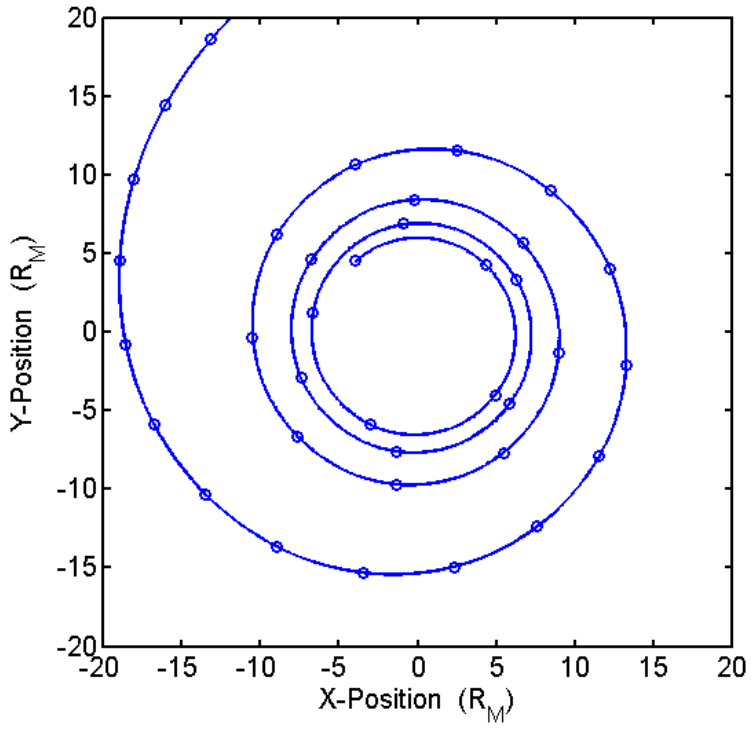


Figure 3.35: HLGL results for the areocentric segment thrust angle.



(a)



(b)

Figure 3.36: HLGL result for the Mars capture spiral. (a) shows the entire spiral. (b) shows a close-up of the latter part of the spiral.

CHAPTER 4

CONCLUSION AND RECOMMENDATIONS

In this thesis, a framework for autonomously (to an extent) generating approximate globally optimal trajectories was presented for a general class of trajectory optimization problems. The method involves numerically solving the optimal control problem of Section 2.1 via the direct Hermite-Legendre-Gauss-Lobatto collocation method, solved with a local gradient-based NLP algorithm. A level of autonomy is achieved by first approximately solving the problem via direct shooting and a globally-searching evolutionary program (genetic algorithm), and interpolating the resulting trajectory to provide an initial guess for the NLP/collocation phase.

The method has been conceptualized in the trajectory optimization literature, but implementation was usually either limited to special cases where the controls were simplified to bang-bang form in the description of the parameters used in the GA initialization phase, or limited to the use of low-accuracy collocation methods in the NLP phase.

The method presented here does not assume a specific form of the optimal control input history in either the GA/shooting phase or the NLP/collocation phase. The accuracy of the NLP/collocation phase was made adjustable by using the HLGL collocation technique, which allows the user to choose the interpolating polynomial order and number of polynomial subintervals spanning each continuous segment of the trajectory.

An extension to the HLGL collocation formulation was presented to include interpolation and collocation of arbitrarily higher-order state derivatives. Implementation and evaluation of the higher-order method is a subject for further research.

A set of MATLAB functions was created to solve any trajectory optimization problem described by the formulation in Section 2.1. The versatility of the method was shown by application to three types of problems. The first was a minimum-time kinematic path planning problem for a multi-agent system of autonomous ground vehicles (mobile robots) with obstacle avoidance, demonstrating the ability of the GA/shooting method to search the solution space of a problem with multiple local minima in the cost function. The second was a minimum-time turn-to-heading maneuver for an aircraft with nonlinear path constraints in the form of a velocity lower bound and an upper bound on loadfactor. The third case was a minimum-time low-thrust interplanetary orbit transfer from a geosynchronous orbit to an areosynchronous orbit (at Mars), which requires the solution of three continuous segments of the trajectory having inter-segment constraints required to change between geocentric, heliocentric, and areocentric coordinates.

Although some spurious behavior is present in the solutions of the aircraft and spacecraft problems, it is expected that these issues may be resolved within the framework of this research through tuning of the GA and NLP programs. In the GA, additional candidate solutions (larger populations) would increase the diversity of the population, thereby increasing the probability that the global optimum would be found. Allowing the population to evolve for more generations would allow more opportunities for the candidate solutions to be refined. This may have noticeable effects on the smoothness (and optimality for minimum-time problems) of the control histories. Further improvements to the GA results may be achieved on a problem-specific basis by adjusting the way in which the constraint violations are added to the cost function. If certain constraints are not met, then the results may be improved by increasing the penalties due to those violations. This could have a noticeable effect on the inter-segment constraints of the spacecraft problem, which were not well met in the GA solution. However, it is important to remember that the result of the GA need not meet all constraints as well

as is required of the final solution. That said, the GA/shooting phase is considered to have performed adequately for the problems in this thesis. It is recommended that any changes to the GA/shooting phase should seek to formalize a means of adding properly weighted constraint violation penalties to the GA penalty function such that the algorithm converges near the global optimum in a robust fashion, as was the goal of [16] through different means.

Regarding the NLP solution, the final results may be improved by tuning of the convergence criteria or other parameters. For instance, lowering the tolerance on constraint violations increases the accuracy of the converged solution, while other parameters may be changed to improve calculation of the constraint and cost Jacobians.

Addition of the higher-order HLGL collocation method presented is expected to improve the results and versatility of the method as well. Addressing any spurious behavior in the collocation results of the spacecraft and aircraft problems, the higher-order collocation constraints would help to ensure continuity of state accelerations (and control rates) across subintervals. It is expected that this will also help maintain feasibility of the trajectory during NLP iterations by better maintaining dynamical accuracy of the state histories. Furthermore, the higher-order collocation constraints may make the method applicable to problems requiring accuracy of higher-order dynamics, such as minimum-jerk problems.

One extension to this work, applicable to the field of guidance, is its application to the generation of trajectories in real-time. A well known framework for conducting trajectory optimization in real-time is the two degree of freedom controller, where an inner loop performs feedback control for the plant to track a reference trajectory generated by an outer loop based on the current states of the system [25].

The trajectory optimization framework, as it is implemented in this thesis, is too slow to generate trajectories in real-time (requiring many hours of run time for the space-

craft problem). There may, however, be developments in the literature that could aide in the modification of this framework for real-time generation of trajectories. One known method to increase the speed of trajectory generation was proposed by the authors of [25]. In that method, the size of the NLP optimization vector is reduced by approximating the trajectory with B-splines. This relies upon reducing the output space of the equations of motion, which is related to the application of differential flatness [26]. Reduction of the trajectory parameterization by the B-spline approach sacrifices accuracy, however, as noted in [27]. In that work, the trajectory is solved via pseudospectral collocation and the computation time is reduced by exploiting the sparsity of the constraint jacobian with respect to the optimization parameters. The choice of optimizer, in this case MATLAB's Optimization Toolbox function **fmincon()**, also plays an important role in the speed of convergence of the NLP. Such examples as [27] demonstrate improvements in computation time compared to the framework in this thesis through the use of optimization software dedicated to the solution of problems having a large number of variables and constraints, rather than general multi-purpose tools like **fmincon()**. Significant changes to the method and implementation used here (including the genetic algorithm) are necessary prior to its application to real-time optimal control problems.

APPENDIX A

DOCUMENTATION OF THE MATLAB IMPLEMENTATION

A.1 Shared File: uservariables.m

The purpose of the script **uservariables.m** is to initialize necessary global variables, some of which may be problem dependent, and set variables and parameters that are critical to the NLP/collocation and GA/shooting phases of the optimization. The script is run automatically by any function that requires the data defined inside. The user should not need to run **uservariables.m** manually. At least the following variables should be declared as global: *nseg*, *seg*, *intOrder*, *bcDependency*. The following variables must be defined before the end of the script.

Shared Variable

nseg

Set this to the number of continuous segments of the trajectory (an integer).

HLGL-Specific Variables

seg

This is an *nseg*-by-1 array of data structures used in the NLP/collocation phase. Its members, described below, are the HLGL discretization parameters and other problem parameters. The parameters must be defined for all segments, and can be different in each segment. In the script, replace # with the number of the segment. The results of the NLP/collocation phase are added to the structure once completed.

seg(#).*n*

The Hermite interpolating polynomial order.

$seg(\#).nint$

The number of subintervals in the segment, each spanned by one n-th order Hermite interpolating polynomial.

$seg(\#).nst$

The number of states.

$seg(\#).NCONT$

The number of control inputs.

$seg(\#).UMIN$

This is an $seg(\#).NCONT$-by-1 vector containing the lower bounds on the control inputs, to be applied through linear inequality constraints.

$seg(\#).UMAX$

This is an $seg(\#).NCONT$-by-1 vector containing the upper bounds on the control inputs, to be applied through linear inequality constraints.

GA-Specific Variables

$NIND$

Size of the population (number of individuals).

$MAXGEN$

The number of generations through which the GA will loop.

$PRECI$

The number of binary bits used to describe variables in a candidate chromosome when using binary encoding.

NP

The number of extra parameters to be optimized in the GA. The final time of each

segment is handled as a parameter. Therefore, the number of parameters should be at least as great as the number of segments. (Final time can be fixed by setting equal upper and lower search bounds. See below.)

$PMIN$

This is an NP-by-1 vector of lower search bounds for the extra parameters (including final time).

$PMAX$

This is an NP-by-1 vector of upper search bounds for the extra parameters (including final time).

bcDependency

This is a vector whose constituent elements are the indexes of the segments whose initializing boundary conditions must be calculated from the prior solution of the boundary conditions of an adjacent segment. For example, consider a trajectory with four continuous segments, where segments 1 and 2 are integrated forward while segments 3 and 4 are integrated backwards. If the initial conditions of segment 2 must be calculated by transforming the final conditions of segment 1, and the final conditions of segment 3 are calculated from the initial conditions of segment 4, then an appropriate statement is $bcDependency = [2\ 3]$. Note: see *intOrder* below.

intOrder

This is a vector of length *nseg* containing the sequence of segment indexes describing the order in which the segments must be integrated. Consider the example above (*bcDependency*). In the example, segment 1 must be solved prior to segment 2. Likewise, segment 4 must be solved prior to segment 3. Two acceptable statements are $intOrder = [1\ 4\ 2\ 3]$ or $intOrder = [1\ 2\ 4\ 3]$.

gaSeg

This is an *nseg*-by-1 array of data structures used in the GA/shooting phase. Its

member variables, described below, are the shooting discretization, GA parameters, and problem parameters. The parameters must be defined for all segments, and can be different in each segment. In the script, replace # with the number of the segment. The results of the GA/shooting phase are added to the structure once completed.

$gaSeg(\#).nst$

The number of states.

$gaSeg(\#).NCONT$

The number of control inputs.

$gaSeg(\#).initialStates$

This is the state vector boundary condition used to initialize the marching integration for the segment. If any of the states need to be transformed from an adjacent segment, then they will be overwritten by the user-editable function **segStateTransform.m**. If any of the initializing B.C.'s are parameterized in the optimization vector (chromosome), then they must be manually set in the function **initialStates.m**. Otherwise, **initialStates.m** should be edited to avoid changing any B.C.'s.

$gaSeg(\#).dt$

The time-step used to create the scalable time vector. Since the time vector (below) is calculated here in **uservariables.m** and the marching integrator uses the time vector to find the time-step, then setting $gaSeg(\#).dt$ is strictly optional.

$gaSeg(\#).timeSGA$

The scalable time vector used by the marching integrator used for shooting. It will be linearly scaled to meet the final time taken from the parameter vector, P. Take care to avoid allowing the time-step to be scaled so high that the integrator loses accuracy. For instance, choose $gaSeg(\#).dt$ to be the largest allowable time-step.

Then ensure that the time-step will not go above $gaSeg(\#).dt$ by setting the final node of the time vector as the maximum searchable final time, where the final time is chosen to be the first parameter in P:

$$gaSeg(\#).timeSGA = 0 : gaSeg(\#).dt : PMAX(1);$$

$gaSeg(\#).UMIN$

An $NCONT$-by-1 vector of control lower bounds.

$gaSeg(\#).UMAX$

An $NCONT$-by-1 vector of control upper bounds.

$gaSeg(\#).iDirect$

The direction of integration for the segment. Set as 1 if the segment is to be integrated forward, or -1 if it is to be integrated backwards.

A.2 Genetic Algorithm/Shooting Phase Files

A.2.1 runsga.m

This script calls performs four tasks. First, it calls the function **mysga.m**, which performs the simple genetic algorithm and returns the results in the $gaSeg$ data structure and parameter vector, P. The state histories are stored in $gaSeg(\#).x$ and the control histories are stored in $gaSeg(\#).u$. Second, it scales the time vectors of each segment, $gaSeg(\#).timeSGA$, using the final times stored in the parameter vector. **Since the user decides the order of the parameters in P, this file should open file to ensure that the appropriate elements are treated as the final times.** Third, it plots the state and control histories from the GA solution. The user may change the plotting commands. Fourth, the MATLAB memory environment is saved, along with the results of the GA, into the finle **sgaResults.mat**.

A.2.2 mysga.m

This function executes the simple genetic algorithm loop using various functions from the Genetic Algorithm Toolbox [14]. It also calls the function **evaluateIndividual.m** to integrate the states of a candidate solution using the function **eomsga.m**, and to evaluate cost via two functions, **costIntegrand.m** and **penaltyFunction.m**. The user should only modify **mysga.m** if it is desired to change type of marching integrator. This is done by changing the variable *integrator* to a string other than 'rk4'. Note that, in the final version of the code, only fourth-order Runge-Kutta is implemented.

Two versions of **mysga.m** are available. One uses binary-encoded chromosomes and the other uses real-encoding.

A.2.3 eomsga.m

This file returns the state derivative vector given the current time, state vector, control vector, system parameters, and which segment of the trajectory is being integrated.

Output:	**f**	The state derivative vector of dimension $nst \times 1$.
Input:	t	Current time.
	$tfold$	The final value of the time vector prior to scaling. See $gaSeg(\#).timeSGA$ in Section A.1.
	$STATE$	The current state vector.
	U	The current control vector.
	P	The vector containing parameters that are variable in the GA, including final times of the segments.
	s	The index of the current segment being integrated.

If there are multiple segments with different equations of motion, then this function should provide a different **f** given the segment index indicated by the input s. For example:

```
switch s
     case 1
          x1dot = ...     % for segment 1
          ...
     case 2
          x1dot = ...     % for segment 2
          ...
end
```

Note that the final state derivative vector should be scaled using the appropriate final time from the parameter vector. For example, if the final time for the current segment is the s-th element of P, then

```
f = [ x1dot; x2dot; ... ] * P(s) / tfold;
```

should appropriately scale the state derivatives.

A.2.4 costIntegrand.m

This function should return the integrand of the Lagrange component of the Bolza cost function, Equation (2.2). It will be integrated in parallel with the states in **evaluateIndividual.m**.

Output:	J	The integrand of the Lagrange component of the Bolza cost.
Input:	t	Current time.
	$tfold$	The final value of the time vector prior to scaling. See $gaSeg(\#).timeSGA$ in Section A.1.
	x	The current state vector.
	u	The current control vector.
	P	The vector containing parameters that are variable in the GA, including final times of the segments.
	s	The index of the current segment being integrated.

Note that the derivative of the cost should be scaled, similarly to the state derivatives in **eomsga.m** (see Section A.2.3).

A.2.5 penaltyFunction.m

This function should return the sum of all non-integrated penalties, including the Mayer component of the cost and constraint violation penalties associated with the shooting problem, for all segments.

Output:	J	The total of all non-integrated penalties of the trajectory.
Input:	$gaSeg$	The data structure containing the state and control histories for all segments after the marching integration.
	P	The vector containing parameters that are variable in the GA, including final times of the segments.

The state histories are stored in $gaSeg(\#).x$ and the control histories are stored in $gaSeg(\#).u$.

A.2.6 initialStates.m

This function should return the *gaSeg* data structure after modifying the initializing states that are parameterized in the GA.

Output:	*gaSeg*	The data structure containing the problem parameters (*initialStates* specifically), prior to the marching integration.
Input:	*gaSeg*	The data structure containing the problem parameters (*initialStates* specifically), prior to the marching integration.
	P	The vector containing parameters that are variable in the GA, including final times of the segments.

As an example, if the initial condition of the second (2) state of the first (1) segment of the trajectory is variable, parameterized by the fourth (4) element of the parameter vector, then the following statement would be appropriate:

```
gaSeg(1).initalStates(2) = P(4);
```

This file should be used to initialize F.C.'s for segments being integrated backwards as well as forward. The member *initialStates* is the initializing state vector for both forward and backward integrated segments. If no initializing B.C.'s are parameterized, then all commands should be removed from this function.

A.2.7 segStateTransform.m

This function transforms the B.C. of one segment of the trajectory to the coordinates of another trajectory. This may be useful for calculating inter-segment constraint violations.

Output:	Xto	The transformed state vector, corresponding to a boundary condition of the segment indicated by *sto*.
Input:	$Xfrom$	The state vector corresponding to the boundary condition of the boundary condition of the segment indicated by $sfrom$.
	$sfrom$	Indicates the segment from which the B.C. is to be transformed.
	sto	Indicates the segment to which the B.C. is to be transformed.
	P	The system parameter vector.

This file should check which inter-segment transformation is to take place and perform it accordingly. For example:

```
if sto == 2 & sfrom == 1
   ...                      % statements converting F.C. of
   Xto = [ ...              % segment 1 to I.C. of segment 2
end
if sto == 3 & sfrom == 4
   ...                      % statements converting I.C. of
   Xto = [ ...              % segment 4 to F.C. of segment 3
end
```

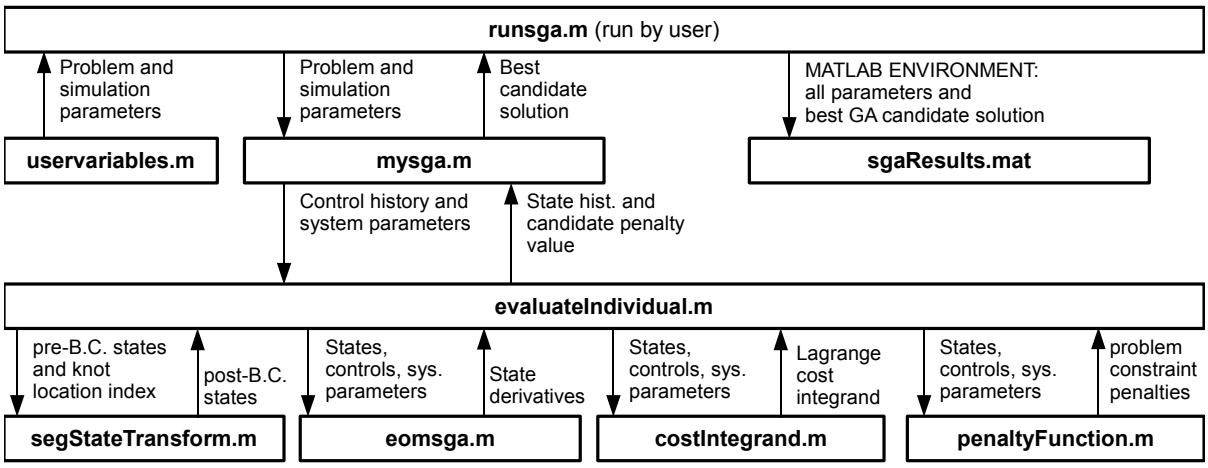


Figure A.1: Input/output dependency illustration for the GA/shooting phase of the program.

A.3 Nonlinear Programming/Collocation Phase Files

A.3.1 HLGLafterGA.m

This script creates the time vectors for the HLGL discretization and interpolates the state and control histories from the results of the gA. It does this by looping through each of the segments and creates the LGL time vectors for each, using the functions written for generating LGL points, then scaling the time vector based on the system parameter vector of the GA output. If the final time of segment s is *not* located at $P(s)$, then the user should manually specify which element of P to use as the final time of each segment.

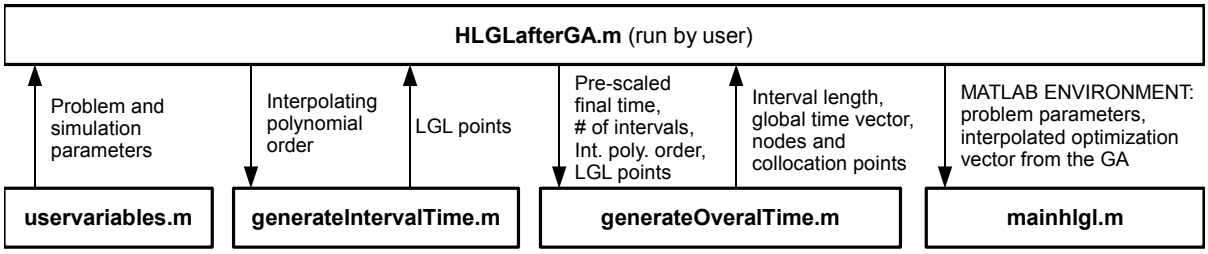


Figure A.2: Input/output dependency illustration for the interpolation script between GA/shooting and NLP/collocation phases of the program.

A.3.2 cost.m

This function should return the sum of the Bolza cost. The method of calculation is left up to the user.

Output:	J	The total Bolza cost over all segments of the trajectory.
Input:	X	The optimization vector used by **fmincon()**.

The i-th node of the j-th state of the s-th segment is located at the element of the optimization vector given by $succeeding + 1$ where

$$succeeding = \sum_{k=1}^{s-1} seg(k).LSEG + (j-1)seg(s).LSTATE \tag{A.1}$$

For the i-th node (or collocation point) of the j-th control variable of the s-th segment,

$$succeeding = \sum_{k=1}^{s-1} seg(k).LSEG + (seg(k).nst)(seg(k).LSTATE) \tag{A.2}$$

The final time of the s-th segment is at the index given by $\sum_{i=1}^{nseg} seg(i).LSEG + s$.

A.3.3 customConstraints.m

This function should return any constraint violations that are nonlinear in nature. This may include any constraints that are not handled elsewhere in the program.

Output:	C	A column vector of inequality constraint violations to be passed to **fmincon()** as nonlinear constraint violations.
	Ceq	A column vector of equality constraint violations to be passed to **fmincon()** as nonlinear constraint violations.
Input:	$STATE$	A $seg(s).nst$-by-$seg(s).LSTATE$ matrix containing all nodal state vector histories for segment s.
	U	A $seg(s).NCONT$-by-$seg(s).LCONT$ matrix containing all nodal control vector histories for segment s. This can be manually changed to include collocation points in the control histories by editing the line calling **customConstraints.m** in **nlcon.m**.
	tf	The final time of segment s.
	s	The segment to be evaluated.

The function evaluates constraints for only one segment per call, but it should be able to distinguish which segment is to be evaluated based on s and return the appropriate constraint violations.

A.3.4 EOM.m

This file returns the state derivative vector given the current state vector, control vector, and which segment of the trajectory is being evaluated.

Output:	$xdot$	The state derivative vector of dimension $nst \times 1$.
Input:	t	Current time.
	$STATE$	The current state vector.
	U	The current control vector.
	s	The index of the current segment being evaluated.

If there are multiple segments with different equations of motion, then this function should provide a different $xdot$ given the segment index indicated by the input s. For example:

```
switch s
     case 1
          ...
          xdot = [...     % for segment 1
     case 2
          ...
          xdot = [...     % for segment 2
end
```

A.3.5 generateLinearConstraints.m

This script should return any constraints of linear form relating the elements of the optimization vector. This is usually used for constant boundary conditions and box constraints. Three types of linear constraints are allowed:

- Equality constraints of the form $Aeq\ X = B$
- Inequality constraints of the form $A\ X \leq B$
- Box constraints of the form $LB \leq X \leq UB$

Each segment is treated separately in this file, storing the coefficient matrices for each segment in $seg(\#).Aeq$, $seg(\#).Beq$, etc. The overall matrices Aeq, Beq, etc. are assembled

automatically in **mainhlgl.m**. For example, the Aeq applied to the entire optimization vector is assembled as

$$Aeq = \begin{bmatrix} seg(1).Aeq & 0 \\ 0 & seg(2).Aeq \end{bmatrix}$$

for a two-segment trajectory. The corresponding Beq is

$$Beq = \begin{bmatrix} seg(1).Beq \\ seg(2).Beq \end{bmatrix}$$

Please define the members $seg(\#).Aeq$, etc. of every segment in this script. The columns of Aeq, A, UB, and LB corresponding to the i-th state of the s-th segment are

$$(i-1) * seg(s).LSTATE + 1 \quad \text{through} \quad i * seg(s).LSTATE$$

For the i-th control,

$$seg(s).nst * seg(s).LSTATE + (i-1) * seg(s).LCONT + 1$$

through

$$seg(s).nst * seg(s).LSTATE + i * seg(s).LCONT$$

and the final time is located at column

$$seg(s).nst) * seg(s).LSTATE + seg(s).NCONT * seg(s).LCONT + 1$$

A.3.6 transSeg.m

This function should return any constraint violations relating the boundary conditions of one segment to those of an adjacent segment.

Output:	$transC$	A column vector of inequality constraint violations to be appended to the nonlinear inequality vector C from **customConstraints.m**.
	$transCeq$	A column vector of equality constraint violations to be appended to the nonlinear equality vector Ceq from **customConstraints.m**.
Input:	i	The index of the "knot", indicating that the boundary conditions of segments i and $i+1$ are to be treated.
	$STATEnew$	The state vector of the I.C. of segment $i+1$.
	$STATEold$	The state vector of the F.C. of segment i.
	$Unew$	The control vector at the I.C. of segment $i+1$.
	$Uold$	The control vector at the F.C. of segment$i$.
	$tfold$	An array containing the final times of segments 1 through i. This was particularly useful when calculating the orbit angles of Earth and Mars for Equations (3.50) and (3.51).

A.3.7 mainhlgl.m

This script runs the nonlinear program **fmincon()**. First, it must construct the constraint matrices A, Aeq, B, Beq, UB, and LB. It then runs **fmincon()**, extracts the state and control histories from the optimization vector to $seg(\#).STATE$ and $seg(\#).U$, respectively. It stores the final times in $seg(\#).tf$. It then scales the time vectors and stores them to $seg(s).tnode$ and $seg(s).time$, where $tnode$ contains the nodal time values and $time$ contains nodes and collocation points. The vector $tnode$ is appropriate for plotting the state histories, whereas $time$ is appropriate to plot control histories.

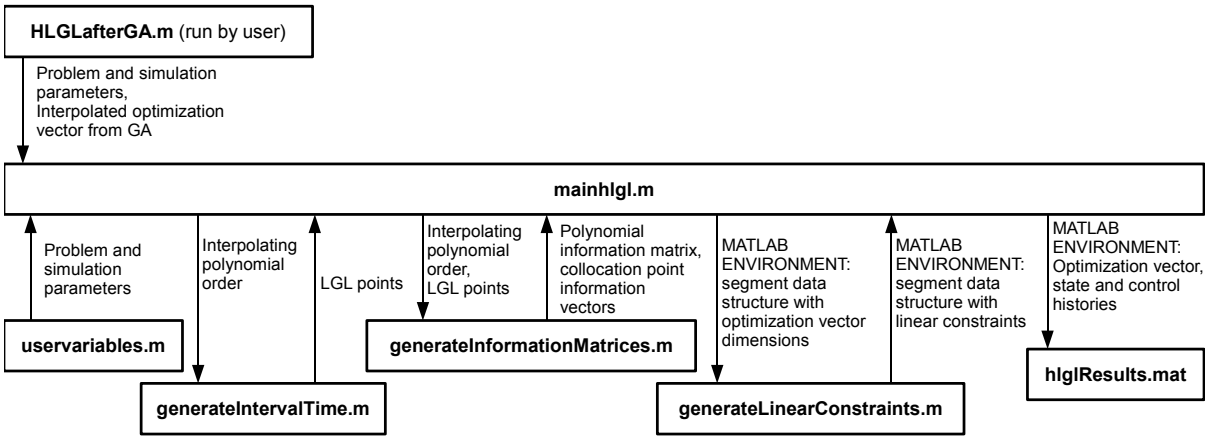


Figure A.3: Input/output dependency map for the NLP collocation phase of the program.

REFERENCES

[1] Arthur E. Bryson, J. and Ho, Y.-C., *Applied Optimal Control*, Blaisdell Publishing Company, London, 1969.

[2] Betts, J. T., "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, March-April 1998, pp. 193–207.

[3] Seywald, H., "Trajectory Optimization Based on Differential Inclusion," Tech. rep., NASA, February 1993, Contractor Report 4501.

[4] Hargraves, C. R. and Paris, S. W., "Direct Trajectory Optimization Using Nonlinear Programming and Collocation," *Journal of Guidance, Control, and Dynamics*, Vol. 10, 1987, pp. 338.

[5] Williams, P., "Hermite-Legendre-Gauss-Lobatto Direct Transcription Methods in Trajectory Optimization," *AAS/AIAA Space Flight Mechanics Conference*, Paper AAS 05-131, Copper Mountain, CO, Jan 23-27 2005.

[6] Tang, S. and Conway, B. A., "Optimization of Low-Thrust Interplanetary Trajectories Using Collocation and Nonlinear Programming," *Journal of Guidance, Control, and Dynamics*, Vol. 18, No. 3, May-June 1995, pp. 599–604.

[7] Conway, B. A. and Larson, K. M., "Collocation Versus Differential Inclusion in Direct Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 5, September-October 1998, pp. 780–785.

[8] Ross, I. M. and Fahroo, F., "Legendre Pseudospectral Approximations of Optimal Control Problems," *Lecture Notes in Control and Information Sciences*, Vol. 295, 2003, pp. 327–342.

[9] Fahroo, F. and Ross, I. M., "Direct Trajectory Optimization by a Chebyshev Pseudospectral Method," *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 1, January-February 2002, pp. 160–166.

[10] Fahroo, F. and Ross, I. M., "A Spectral Patching Method for Direct Trajectory Optimization," *The Journal of the Astronautical Sciences*, Vol. 48, No. 2 and 3, April-September 2000, pp. 269–286.

[11] Williams, P., "A Comparison of Differentiation and Integration Based Direct Transcription Methods," *Advances in the Astronautical Sciences*, Vol. 120, No. 1, 2005, pp. 389–408, AAS 05-128.

[12] The Mathworks, Inc., Natick, MA, *Optimization Toolbox 3 User's Guide*, 2007.

[13] Seywald, H., Kumar, R. R., and Deshpande, S. M., "Genetic Algorithm Approach for Optimal Control Problems with Linearly Appearing Controls," *Journal of Guidance, Control, and Dynamics*, Vol. 18, No. 1, January-February 1995, pp. 177–182.

[14] Chipperfield, A., Flemming, P., Pohlheim, H., and Fonseca, C., *Genetic Algorithm Toolbox Version 1.2 User's Guide*, Department of Automatic Control and Systems Engineering, University of Sheffield.

[15] Proulx, R. J. and Ross, I. M., "Time-Optimal Reorientation of Asymmetric Rigid Bodies," *Advances in the Astronautical Sciences*, Vol. 109, No. 2, 2002, pp. 1207–1228.

[16] Yokoyama, N. and Suzuki, S., "Modified Genetic Algorithm for Constrained Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 1, January-February 2005, pp. 139–144.

[17] Spiegel, M. R. and Liu, J., *Mathematical Handbook of Formulas and Tables*, McGraw-Hill, New York, 2nd ed., 1999.

[18] Flash, T. and Hogan, N., "The Coordination of Arm Movements: An Experimentally Confirmed Mathematical Model," *The Journal of Neuroscience*, Vol. 5, No. 7, July 1985, pp. 1688–1703.

[19] Amirabdollahian, F., Loureiro, R., and Harwin, W., "Minimum Jerk Trajetory Control for Rehabilitation and Haptic Applications," *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, Washington, DC, May 2002, pp. 3380–3385.

[20] Ross, I. M., Rea, J., and Fahroo, F., "Exploiting Higher-Order Derivatives in Computational Optimal Control," *Proceedings of the 10th Mediterranean Conference on Control and Automation*, July 2002.

[21] Ross, I. M. and Fahroo, F., "Pseudospectral Knotting Methods for Solving Optimal Control Problems," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 3, May-June 2004, pp. 397–405.

[22] Segal, S., Ben-Asher, J. Z., and Weiss, H., "Derivation of Formation-Flight Guidance Laws for Unmanned Air Vehicles," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 4, July-August 2005, pp. 733–742.

[23] Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, Space Technology Library, London, 2nd ed., 2001.

[24] Battin, R. H., *An Introduction to the Mathematics and Methods of Astrodynamics*, AIAA, Reston, VA, revised ed., 1999.

[25] Milam, M. B., Mushambi, K., and Murray, R. M., "A New Computational Approach to Real-Time Trajectory Generation for Constrained Mechanical Systems," *Proceedings ofthe 39th IEEE Conference on Decision and Control*, Sydney, Australia, December 2000, pp. 845–851.

[26] Faiz, N., Agrawal, K., and Murray, R. M., "Trajectory Planning of Differentially Flat Systems," *Journal of Guidance, Control, and Dynamics*, Vol. 24, No. 2, March-April 2001, pp. 219–227.

[27] Strizzi, J., Ross, I. M., and Fahroo, F., "Towards Real-Time Comutation of Optimal Controls for Nonlinear Systems," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Monterey, California, August 2002, AIAA Paper 2002-4945.

BIOGRAPHICAL STATEMENT

Brandon graduated from MacArthur High School in Irving, Texas in May of 2000. He intended to pursue studies in Aerospace Engineering, motivated by a fascination with physics and space exploration. He earned his Bachelor of Science degree in Aerospace Engineering from the University of Texas at Arlington in 2005. Interested in controls and astronautics, he continued into the graduate program and will earn his Master of Science degree in Aerospace Engineering in May of 2008. Afterward, Brandon intends to work in the space industry for a few years. During that time, he will consider topics for Ph.D. studies that will better enable him to contribute to space exploration.