Real-time Optimization in Aerospace Systems

Daniel Dueri

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2018

Reading Committee:

Behçet Açıkmeşe, Chair

Mehran Mesbahi

Lillian Ratliff

Program Authorized to Offer Degree:
Aeronautics & Astronautics

University of Washington

Abstract

Real-time Optimization in
Aerospace Systems

Daniel Dueri

Chair of the Supervisory Committee:
Behçet Açıkmeşe
William E. Boening Department Aeronautics & Aeronautics

Algorithm design for autonomous vehicles has been attracting immense research interest, and with the proliferation of self-driving cars, autonomous drone delivery systems, and the increasing availability of commercial off–the–shelf UAVs, interest will only continue growing. The goal of this dissertation is to leverage recent advances in optimal control theory and optimization to provide methods that aid in the design and safe operation of autonomous vehicles (i.e., operating vehicles without violating mission constraints or hardware limits). To this end, we leverage convex optimization and a recent result from optimal control theory called lossless convexification, whereby problems with non-convex control constraints are cast into equivalent convex ones. Convex optimization is then applied to all stages of a mission: from top level vehicle design, to high-level autonomy onboard the vehicle, to trajectory planning, and finally down to allocating desired controls to individual actuators.

This dissertation first presents an architecture for generating customized C solvers that were designed specifically for use onboard systems with limited computational resources. Solver customization exploits knowledge of the problem structure to generate solvers that are 2-3 orders of magnitude more efficient than generic solvers. One such customized solver was flight tested onboard Masten's Xombie rocket during the summers of 2012 and 2013. For mission planning and high-level autonomy, this document develops a convex optimization

based method for approximating constrained, finite-time-horizon reachable and controllable sets with inner and outer bounding polytopes. These sets can be used to quickly understand the impact of different design parameters during the design process, or to determine the feasibility of goal states once in flight. Then, the document extends the theory of lossless convexification to cover maximal divert trajectories with simultaneously active thrust pointing and velocity constraints, and develops a method for computing locally optimal trajectories that avoid obstacles. Finally, for a momentum control system, this document presents a convex optimization based real-time control allocation algorithm to optimally utilize multiple actuators without violating their physical constraints.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my loving wife for being my strength at difficult times and always encouraging me to be my best. To my parents and brother, I thank you for providing me moral and emotional support throughout my life. I would like to especially thank my very good friend Michael Szmuk for being like a brother to me, and Jahshan Bahtti for setting the bar high.

To my adviser, Behçet Açıkmeşe, I thank you for your brilliance, your patience, and your contribution to my research and career. To my committee members, Mehran Mesbahi, Lillian Ratliff, Shayan Oveis Gharan, and Linda Bushnell, I thank you for your continuous guidance. I'm also grateful for the cooperation and friendship of my fellow students, Michael Szmuk, Yuanqi Mao, and Utku Eren, who have inspired and motivated me in my research.

I would like to thank Lars Blackmore, whose work on the Falcon 9 rocket has brought my research area into the spotlight. Thank you Daniel Scharf for testing my customized solvers on JPL flight hardware. Fred Leve and Scott Erwin, thank you for the opportunity to work on a realistic attitude control test bed. And lastly, I would like to thank Jerry Ding, Andrew Sparks and Zohaib Mian for many insightful conversations and a great summer.

Finally, I would like to thank NASA's Flight Opportunities program, JPL, and Masten for the G-FOLD flight test, and grants from the Air Force Research Laboratory and Office of Naval Research for funding and facilitating my research.

DEDICATION

in loving memory of my mother,

Chapter 1

INTRODUCTION

This dissertation is focused on real-time convex optimization in aerospace control engineering, which includes both casting important aerospace control problems as convex optimization problems and solving them via the real-time algorithms detailed in this work. More specifically, we develop methods that are useful at multiple stages of an aerospace project, from mission design and high-level autonomy to onboard guidance and real-time control. The overarching goal throughout is to design and operate vehicles safely at the physical limits of their actuators' capabilities without violating mission constraints. There are a number of challenges involved with solving optimization problems onboard autonomous vehicles reliably in safety critical applications. We discuss methods for overcoming these challenges in Chapter 2.

The motivation for utilizing a framework based on convex optimization is threefold: i) casting engineering problems as optimization problems allows one to explicitly define mission and vehicle constraints that the solutions must satisfy; ii) solutions to optimization problems are optimal with respect to the selected objective function [2]; iii) by using numerical algorithms like the now well-understood Interior Point Method (IPM), solutions to convex optimization problems are found with polynomial time complexity [3–6]. For cases in which no solution exists, a certificate of infeasibility is returned in lieu of a solution (also with polynomial time complexity) [7–9]. In contrast, numerical methods for solving non-convex optimization problems require a good initial guess, cannot certify global optimality, and have no guarantees of converging to a solution in finite time [10].

Many if not most problems of practical concern in aerospace engineering, and controls in particular, are not naturally convex. To this end, this work partially deals with extending

a relatively recent addition to optimal control theory introduced by Açıkmeşe and Ploen in 2007 called lossless convexification [11]. Lossless convexification is a process in which a non-convex optimal control problem is *equivalently* cast as a convex optimization problem, and was originally developed for use in powered descent guidance (see Chapter 4 for more details). Naturally, such a procedure can only be carried out for a subset of optimal control problems, and many existing results are centered on variations of the powered descent guidance problem [12–14]. However, some more general results exist. In 2011, it was shown that lossless convexification holds for optimal control problems with convex objective, linear dynamics, and non-convex control constraints if the linear system is controllable [15]. Harris et al extended these results to general linear systems with convex objective, active polytopic state constraints, and non-convex control constraints [16]. Overall, a sizable class of non-convex optimal control problems can be cast as convex ones, with special interest in the literature on the powered descent guidance problem, which is important in aerospace engineering [17].

Since convex optimization has desirable computational properties and a large class of optimal control problems can be posed as convex optimization problems via lossless convexification, this dissertation espouses a tiered control hierarchy for autonomous agents (see Figure 1.1).

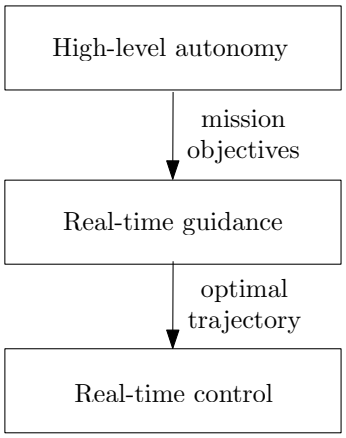


Figure 1.1: Proposed control hierarchy for autonomous agents

Above, the top tier of the control hierarchy is responsible for making decisions on the mission level, such as determining the desired final state in an upcoming trajectory. Making these determinations in a systematic manner that encompasses vehicle and actuator constraints is the subject of Chapter 3. The middle tier encapsulates trajectory optimization - once the desired final state is determined, what is the "best" (as defined by the objective function) manner of reaching the target? A systematic approach to real-time, embedded trajectory optimization is the subject of Chapter 4. The bottom tier is reserved for real-time control, and is responsible for specifying individual actuator commands. Chapter 5 deals with obtaining actuator commands in the context of optimal control mapping. While not within the scope of this document, we note that for problems that cannot be convexified losslessly, convex approximations over suitable sub-domains often represent a viable approach [2].

1.1 Background

For the remainder of the dissertation, we apply the design hierarchy in Figure 1.1 to the highly relevant planetary Powered Descent Guidance (PDG) problem. Powered descent refers to a stage in a landing vehicle's mission profile in which the vehicle utilizes its available thrust to land at or as close as possible to a designated landing site. Though initially motivated by interplanetary missions such as the Mars Science Laboratory (MSL), guidance techniques similar to those described in this document have also famously been used by SpaceX to recover their boosters on seafaring barges [17].

Early versions of the optimal PDG problem were known as the *soft-landing problem*, and a 1-D rocket landing problem was solved analytically for purely vertical motion [18]. This solution was important because it characterized the properties of fuel optimal landing trajectories, and their trademark bang-off-bang control input; that is, the control is either at its minimum or maximum value for the entirety of the optimal solution. The associated problems in practice, however, are at least three dimensional, and have state and control constraints. There have been several insightful real-time methods to analytically obtain suboptimal solutions for the PDG problem [19–21], but no known analytic solution exists

for the PDG problem in a realistic setting.

One recent example of the PDG problem is the final stage of the Mars Science Laboratory, whose success has garnered much research interest in this area. During the last stage of the Entry, Decent and Landing (EDL) phase, its thrusters ignite to correct some of the state error that resulted from the convolution of errors during the atmospheric entry and passive parachute descent stages of EDL [22–24]. Its onboard PDG algorithm was unable to fully correct for these errors and achieve a pinpoint landing. However, more recent PDG formulations consider a comprehensive set of mission and vehicle constraints to numerically solve the PDG problem to global optimality, for both the minimum fuel and minimum landing error problems [11–13]. These numerical methods make use of *lossless convexification* to cast the original, non-convex optimal control problem into a convex parameter optimization problem that can be solved via IPMs to global optimality with guaranteed convergence properties [12, 14].

The performance of the aforementioned numerical techniques depends on the efficiency of the specific IPM solvers used to solve the convexified PDG problem. To this end, customized IPM solvers were developed to ease the introduction of real-time optimization onboard embedded devices and vehicles. Customized solvers are capable of avoiding dynamic memory allocation, using less memory, and upper bounding the number of computations each solution will require [25, 26] - all properties that facilitate deployment onto safety critical missions, such as landers and rockets. These customized solvers for Guidance for Fuel Optimal Large Diverts (G-FOLD) algorithm were successfully flight tested under the NASA Flight Opportunities test program [27–29], clearing an important hurdle in technology maturation and flight readiness. Indeed, convex optimization has been used effectively for other aerospace applications as well [1, 30–35], where the rigorous convergence guarantees and global optimality of IPMs have been exploited [2, 5].

Throughout the dissertation, optimal control problems are solved via numerical solvers. Before doing so, the original infinite dimensional problem must be discretized into a finite-dimensional optimization problem. Unless otherwise stated, the simplest form of discretiza-

tion is employed; that is, equal time intervals with piecewise constant controls, and the constraints enforced at each time epoch. The resulting finite-dimensional problem is then parsed into solver inputs that in turn return either a solution or a certificate of infeasibility.

1.2 Thesis Outline

The work presented in this dissertation has been published in [1, 35–39]. More detailed information is provided at the beginning of each chapter.

1.2.1 Chapter 2

The second chapter describes a framework for generating real-time, customized C IPM solvers for use onboard embedded systems. In addition, the chapter discusses the implementation of the C++ solver that is the basis for generating the custom C solvers used during the G-FOLD flight tests.

1.2.2 Chapter 3

Chapter 3 focuses on two algorithms for approximating Constrained Reachable and Controllable (CRC) sets, which can replace expensive Monte Carlo analysis during rapid design iterations, advance high level autonomy, and quickly determine mission feasibility for existing vehicles. Both algorithms converge to the desired set by approximating it with an inner and outer polytopic approximation - as the algorithms progress, the inner and outer bounds become tighter, and in the case of the second algorithm are shown to converge to the desired set with a finite number of iterations.

1.2.3 Chapter 4

Chapter 4 describes the formulation of the optimal control problem for fuel-optimal PDG, and delves into the details of convexification. Further, the chapter provides an extension to previous lossless convexification theory that applies to the specific set of active constraints

that occurred during the G-FOLD flight tests. It also briefly treats the non-convex problem of obstacle avoidance via a relatively novel technique called *successive convexification*, where a method to guarantee obstacle avoidance between discrete time samples is developed.

1.2.4 Chapter 5

Chapter 5 applies the concept of real-time optimization to constrained control allocation problems, using control moment gyroscopes and reaction wheels as examples. Using the Moore-Penrose pseudo-inverse as a starting point, the chapter makes use of lexicographic optimization to develop a method for mapping torques to control moment gyroscopes in a manner that reduces both the mapping error and the power used to operate them.

Chapter 2

CUSTOM SOLVERS FOR REAL-TIME OPTIMIZATION

Solvers capable of solving optimization problems with realistic engineering constraints in real-time onboard embedded devices enable *safe* high-level autonomy, real-time guidance, and high performance control mappings [25, 36]. Customized IPM solvers were developed with these purposes in mind, and are key to moving many of the algorithms described in the following chapters into the real-time domain. In the context of aerospace systems, where path planning is a safety-critical aspect of each mission, onboard optimization has until *very* recently been vigorously avoided in practice. Even recent successes like the unprecedented landing of the Mars Science Laboratory made use of Apollo-era polynomial guidance [40]. Although fields such as Model Predictive Control (MPC) are mature in the controls literature and offer solutions to the the constrained path planning problem [11,30–34,41–43], the solvers they rely on are difficult to verify, were not designed for real-time use, and typically do not obtain solutions rapidly enough to enable their use on the usually outdated computational resources of aerospace systems. Other approaches [44–46] use offline optimization to obtain nearly optimal real-time performance without having to rely on solvers onboard autonomous agents.

This chapter develops a theoretical and algorithmic framework for generating customized C interior point method solvers that are designed *specifically* to be practical on platforms with limited computational resources. As these solvers operate on Second Order Cone Programs (SOCPs) rather than linear programs, this work can be thought of as an extension of previous developments that focused on linear programs [26]. This chapter first presents an effective IPM algorithm that has undergone nearly a decade of verification, and then introduces a customization methodology [25, 26, 47] that improves runtimes for onboard implementation.

Custom IPM solvers take full advantage of a specific problem structure to significantly reduce the number of mathematical operations and computational branches used by the solver's executable [25, 26, 48, 49]. It is worth mentioning that the customization process described here can be applied to other IPM algorithms as well. The convexified PDG problem has been successfully and repeatedly flight tested on a terrestrial vertical take-off and landing rocket [27–29] utilizing a customized IPM solver, with test flight data and verification methods presented in [50].

The following is adapted from [36] and full details of the work, including the specific implementation used onboard the Xombie rocket, are available in [36, 38].

2.1 A real-time IPM Algorithm to Solve SOCP Problems

This section reviews an IPM algorithm for SOCPs and introduces the notation necessary to describe the contributions detailed in Section 2.2 [25]. The specific IPM algorithm described here has successfully solved millions of PDG problems since 2007 [11] as part of extensive Monte Carlo studies for Mars landing [24]. In addition to its theoretical convergence guarantees [5, 6], these Monte-Carlo runs numerically demonstrated the IPM's efficacy and robustness. While there are a variety of IPM algorithms in the literature, the one presented here has been developed for the relevant aerospace application of planetary PDG, has been rigorously tested over the span of a decade, and has been demonstrated on terrestrial test flights [24, 28]. Thus, the IPM in this section has a very rich aerospace verification history that sets it apart from other solvers; especially since the verification effort for aerospace applications is very demanding due to severe safety and cost concerns.

In general, the objective of an SOCP solver is to find an optimal solution of the primal problem, given (in canonical form) by (2.1).

$$\begin{aligned} &\text{minimize} : c^{\mathrm{T}} x \\ &\text{subject to} : Ax = b, \\ &x \in \mathcal{K} \end{aligned} \tag{2.1}$$

where $x \in \mathcal{K} \subset \mathbb{R}^n$ is the solution variable, $c \in \mathbb{R}^n$ maps the solution variable to the cost function, $A \in \mathbb{R}^{p \times n}$ relates solution variables to constraint equations with $b \in \mathbb{R}^p$ on the right hand side, and $\mathcal{K}$ is given by:

$$\mathcal{K} = \mathcal{K}_L \times \mathcal{K}_{S_1} \times \ldots \times \mathcal{K}_{S_m} \tag{2.2}$$

where m second order cones are used to define the domain of the solution variable, which is comprised of $k = m + l$ total cones (l positive linear cones in addition to m second order cones). Casting Problem 4 in an equivalent canonical form is a straight-forward, but tedious process in which A, b, and c are ultimately defined for each problem instance. There are many parsers capable of taking the high-level definition given in Problem 4 and converting it into an equivalent standard form that can be used by solvers [51, 52]; however, these are generally not capable of parsing quickly enough for real-time applications so it may be necessary to parse the problem by hand in order to make full use of real-time solvers. The next important problem is the dual optimization problem, which is given by:

$$\begin{aligned} &\text{maximize} : b^T y \\ &\text{subject to} : A^T y + s = c, \\ &\qquad\qquad s \in \mathcal{K}^* \quad \text{where} \quad \mathcal{K}^* = \mathcal{K}, \end{aligned} \tag{2.3}$$

where the goal of the dual problem is to find $y \in \mathbb{R}^p$ and $s \in \mathcal{K} \subset \mathbb{R}^n$ that maximize $b^T y$, and the last constraint on s can be rewritten as $s \in \mathcal{K}$ since $\mathcal{K}$ is comprised of purely self-dual cones (i.e., positive linear cones and second order cones are self-dual). It is also useful to note that the duality gap, $x^T s = c^T x - b^T y \geq 0$ for any feasible primal-dual pair x, (y, s) [2]. Also, when a strictly feasible primal-dual pair exists (i.e., a pair that satisfies the equality constraints and lies strictly inside the cone $\mathcal{K}$), the duality gap is zero for any optimal primal-dual pair, i.e., $x_*^T s_* = c^T x_* - b^T y_* = 0$ [6]. Therefore, the duality gap can be used to quantify the current iterate's proximity to the optimal solution.

2.1.1 Interior Point Methods

We solve the primal and dual problems together by using a single primal-dual path-following IPM [3,4]. The remainder of this section describes the relevant components of the aforementioned IPM algorithm. For reference, one can find a detailed discussion of IPM algorithms in [3–6, 9, 53–56].

Homogenous Self-Embedding

This section presents a lifted version of the combined primal and dual problems that has two key benefits. The first is that there is a readily available feasible solution. The second benefit is that the solution of this lifted formulation fully describes the properties of the original primal problem. Given an initial guess, (x_0, s_0, y_0, τ_0, κ_0), the following theorem describes the relationship between the homogeneous self-embedded form and the original primal and dual problems [6, 57] ,

$$\begin{aligned} \text{minimize} : \; & \beta\nu \\ \text{subject to} : \; & Ax - b\tau - r_p\nu = 0, \\ & -A^T y + c\tau - s - r_d\nu = 0, \\ & b^T y - c^T x - \kappa - r_g\nu = 0, \\ & r_p^T y + r_d^T x + r_g\tau = -\beta, \\ & x, s \in \mathcal{K}, \;\; \tau, \kappa \geq 0, \end{aligned} \tag{2.4}$$

where $\nu \in \mathbb{R}_+$ and $y \in \mathbb{R}$ are free, and $r_p \in \mathbb{R}^p$, $r_d \in \mathbb{R}^n$, $r_g \in \mathbb{R}$, and $\beta \in \mathbb{R}$ are residuals [6, 9] defined by:

$$\begin{aligned} r_p &\triangleq \frac{Ax_0 - b\tau_0}{\nu_0}, & r_d &\triangleq \frac{-A^T y_0 + c\tau_0 - s_0}{\nu_0}, \\ r_g &\triangleq \frac{b^T y_0 - c^T x_0 - \kappa_0}{\nu_0}, & \beta &\triangleq -\left(r_p^T y_0 + r_d^T x_0 + r_g \tau_0\right). \end{aligned}$$

r_p, r_d, and r_g represent the infeasibility of the initial guess, and are zero if the initial guess happens to be feasible for the original primal and dual formulations in (2.1) and (2.3).

Remark 1. *Using this formulation, a path-following algorithm can be initialized trivially [6, 9]. Any x_0 and s_0 that are in the interior of $\mathcal{K}$ serve as a strictly feasible but not necessarily optimal solution to Equation* (2.4)*. Moreover, since y is free, y_0 can be taken to be a zero vector of appropriate dimensions; finally, τ_0, κ_0, and ν_0 must be non-negative, and are typically chosen to be 1.*

The following theorem summarizes the relationship between the original problem and the homogenous self-embedded formulation.

Thoerem 1. *Consider the primal and dual problem given by (2.1) and (2.3). Let (x, s, y, τ, κ) be an optimal solution of the homogenous self-embedded form in* (2.4)*, then one of the following conditions holds [6, 9, 48, 57]:*

1. *$\tau > 0$, $\kappa = 0$: An optimal solution of the self-embedded problem has been found. x/τ is the optimal solution of the primal problem.*

2. *$\tau = 0$, $\kappa > 0$, and $b^T y > 0$: Primal is infeasible.*

3. *$\tau = 0$, $\kappa > 0$, and $c^T x > 0$: Dual is infeasible.*

4. *$\tau = 0$, $\kappa = 0$: Problem is numerically ill-posed.*

Proof. Complete proofs of this theorem can be found in [6,9], so they are omitted from this chapter. □

Central Path

To solve the non-linear, homogenous, self-embedded problem in (2.4), a series of perturbed problems are solved. It can be shown that finding the optimal solution to (2.4) is equivalent to solving the following system of bilinear equations for $\mu = 0$ [6, 9, 57]:

$$\begin{aligned} Ax - b\tau - r_p\nu &= 0, \\ -A^T y + c\tau - s - r_d\nu &= 0, \\ b^T y - c^T x - \kappa - r_g\nu &= 0, \\ x^T s &= 0 + \mu, \\ \kappa\tau &= 0 + \mu, \\ \mu &= \mu_0\nu \geq 0, \end{aligned} \tag{2.5}$$

where μ is a perturbation on the system of equations formed by the constraints in (2.4), ν becomes a given parameter, and μ_0 is given by:

$$\mu_0 = \frac{x_0^T s_0 + \tau_0\kappa_0}{m + l + 1}.$$

The solution to the system of equations in (2.5) is unique for non-zero values of μ [6]; therefore for each positive μ, there exists a unique solution of the system that can be found by using numerical techniques (such as Newton's method). Moreover, as $\mu \to 0$ (or equivalently, $\nu \to 0$), the solution of the perturbed problem converges to the solution of the original problem (Equation 2.4). It is worth noting that the condition $\mu = 0$ is a necessary and sufficient condition for the optimality of convex programs, and that it is equivalent to the Karush-Kuhn-Tucker conditions for the lifted problem in Equation 2.4 [9,48]. The solutions, $(x_\mu, \tau_\mu, y_\mu, s_\mu, \kappa_\mu)$, for any non-negative μ define the *central path*. The difference between one IPM and another lies primarily in how one tracks the central path towards the unperturbed system solution.

Newton Search Directions

We now introduce terminology that will be used to solve the non-linear system of equations in (2.5). Given a vector, $v = [v_1, v_2, \ldots, v_n]^T \in \mathbb{R}^n$, we define the arrowhead matrix to be:

$$\text{arrow}(v) \triangleq \begin{pmatrix} v_1 & v_{2:n}^T \\ v_{2:n} & Iv_1 \end{pmatrix}.$$

Using this definition, we can replace the fourth equation in (2.5) with:

$$XSe = 0, \tag{2.6}$$

since it can be shown that $x^T s = 0 \iff XSe = 0$, where $e = \left[(e^{(1)})^T, \ldots, (e^{(k)})^T\right]^T$,

$$\begin{aligned} X &\triangleq \text{blkdiag}(\text{arrow}(x^{(1)}), \ldots, \text{arrow}(x^{(k)})) \\ S &\triangleq \text{blkdiag}(\text{arrow}(s^{(1)}), \ldots, \text{arrow}(s^{(k)})) \\ e_j^{(i)} &\triangleq \begin{cases} 0, & j \neq i, \\ 1, & j = i \end{cases}, \end{aligned} \tag{2.7}$$

and $x^{(i)}$ represents the i^{th} cone in the solution variable. That is, the block diagonal matrices are built from arrowhead matrices corresponding to each cone in the solution variables. Without loss of generality, the solution variables are arranged such that the linear cones are first, followed by the second order cones. Since the linear cones correspond to positive scalars, the first l elements of the block diagonal matrices reduce to the diagonal elements.

Suppose that the current estimate of the solution is given by (x, τ, y, s, κ). Then, the next iterate can be expressed as ($x + \Delta x$, $\tau + \Delta\tau$, $y + \Delta y$, $s + \Delta s$, $\kappa + \Delta\kappa$). Applying this

expression to system (2.5) and substituting (2.6) into the fourth equality, yields:

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_p\nu - (Ax - b\tau), \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_d\nu - (-A^T y + c\tau - s), \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_g\nu - (b^T y - c^T x - \kappa), \\ X\Delta Se + S\Delta Xe &= \nu\mu_0 e - E_{xs}, \\ \kappa\Delta\tau + \tau\Delta\kappa &= \nu\mu_0 - \kappa\tau - E_{\kappa\tau}, \end{aligned} \tag{2.8}$$

where $E_{xs} \in \mathbb{R}^n$ and $E_{\kappa\tau} \in \mathbb{R}$ are approximations (given subsequently) to the second order terms, $\Delta X\Delta Se$ and $\Delta\kappa\Delta\tau$, respectively. Also the ΔX and ΔS matrices are formed in the same fashion as X and S in (2.7). In its current form, the problem cannot be reliably and efficiently solved as shown in [9], so cone scalings are introduced that make it so.

Nesterov-Todd (NT) Scalings

A key property of scaling matrices is that they do not change the cone or the central path. Nesterov and Todd introduced one such set of symmetric scalings that are computationally inexpensive, and make the problem numerically well conditioned [58]. Consider the i^{th} linear or second order cone and define scaling variables $G = G^T$ and $\theta \geq 0$ as follows [58]:

$$\theta^2 = \sqrt{\frac{s^{(i)\mathrm{T}} Q s^{(i)}}{x^{(i)\mathrm{T}} Q x^{(i)}}},$$

where $Q = 1$ for linear cones and

$$Q = \mathrm{diag}(1, -1, \ldots, -1),$$

for second order cones. Similarly, $G = 1$ for linear cones and

$$G = -Q + \frac{(e+g)(e+g)^T}{1 + e^T g},$$

for second order cones, where:

$$g = \frac{s^{(i)}/\theta + \theta Q x^{(i)}}{\sqrt{2\left(x^{(i)\mathrm{T}} s^{(i)} + \sqrt{x^{(i)\mathrm{T}} Q x^{(i)} s^{(i)\mathrm{T}} Q s^{(i)}}\right)}}.$$

The scaled cones are then:

$$\bar{x}^{(i)} = \theta G x^{(i)}, \qquad \bar{s}^{(i)} = (\theta G)^{-1} s^{(i)}.$$

This operation can be repeated for every linear and quadratic cone in the variables x and s. The scaled solution variables result from concatenating all of the scaled cones to form $\bar{x}$ and $\bar{s}$, producing:

$$\bar{x} = \Theta \tilde{G} x, \qquad \bar{s} = \left(\Theta \tilde{G}\right)^{-1} s, \qquad \text{where}$$

$$\Theta = \mathrm{diag}(\theta^{(1)}, \ldots, \theta^{(k)}), \quad \tilde{G} = \mathrm{blkdiag}(G^{(1)}, \ldots, G^{(k)}).$$

$$x = (\Theta \tilde{G})^{-1} \bar{x}, \quad s = \Theta \tilde{G} \bar{s}. \tag{2.9}$$

Substituting Equation (2.9) into the fourth equality of the system in (2.8) leads to,

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_1, \\ -A^T \Delta y + c\Delta\tau - \Delta s &= r_2, \\ b^T \Delta y - c^T \Delta x - \Delta\kappa &= r_3, \\ \bar{X}\left(\Theta\tilde{G}\right)^{-1} \Delta\bar{s} + \bar{S}\Theta\tilde{G}\Delta\bar{x} &= r_4, \\ \kappa\Delta\tau + \tau\Delta\kappa &= r_5, \end{aligned} \tag{2.10}$$

where as in [9],

$$\begin{aligned} &r_1 = r_p\nu - (Ax - b\tau),\ r_2 = r_d\nu - (-A^T y + c\tau - s), \\ &r_3 = r_g\nu - (b^T y - c^T x - \kappa),\ r_4 = \nu\mu_0 e - E_{xs}, \\ &r_5 = \nu\mu_0 - \kappa\tau - E_{\kappa\tau}. \end{aligned}$$

Given the current solution (x, τ, y, s, κ), E_{xs}, and $E_{\kappa\tau}$, one can compute r_1 through r_5.

Rearranging the last equality in (2.10),

$$\Delta\kappa = \frac{r_5 - \kappa\Delta\tau}{\tau}. \tag{2.11}$$

The fourth equality in (2.10) implies that

$$\Delta s = \Theta\tilde{G}(\bar{X})^{-1}r_4 - (\Theta\tilde{G})^2\Delta x. \tag{2.12}$$

Let $D = \left(\Theta\tilde{G}\right)^{-1}$, then it can be shown that the following relations also hold [9]:

$$\Delta x = D^2\left(r_2' + A^T\Delta y - c\Delta\tau\right), \tag{2.13}$$

$$\Delta\tau = \frac{r_3' + a_1^T\Delta y}{a_2} \quad \text{where} \tag{2.14}$$

$$r_2' = r_2 + \Theta\tilde{G}(\bar{X})^{-1}r_4, \quad r_3' = r_3 + \frac{r_5}{\tau} + c^T D^2 r_2', \quad a_1 = -b + AD^2c, \; a_2 = \frac{\kappa}{\tau} + c^T D^2 c.$$

Given the above relationships, $(\Delta x, \Delta\tau, \Delta y, \Delta s, \Delta\kappa)$ are found by first computing Δy and substituting it into (2.11)-(2.14) as follows,

$$\left(AD^2A^T + \bar{a}a_1^T\right)\Delta y = \xi, \tag{2.15}$$

where

$$\bar{a} = \frac{-AD^2c - b}{a_2}, \quad \xi = r_1' + \frac{r_3'}{a_2}(AD^2c + b), \quad r_1' = r_1 - AD^2r_2'.$$

Equation (2.15) can be solved efficiently by making use of the Sherman-Morrison formula [9,59]. Let $\hat{P} = AD^2A^T$, then it can be shown that the solution to (2.15) is found by solving the following two linear systems for v_0 and v_1:

$$\hat{P}v_0 = \xi, \tag{2.16}$$

$$\hat{P}v_1 = \bar{a}. \tag{2.17}$$

Substituting back into (2.15),

$$\Delta y = v_0 - \frac{a_1^T v_0}{1 + a_1^T v_1} v_1.$$

Note that $\hat{P}$ is a symmetric, positive definite matrix due to the Nesterov-Todd scalings [58]; thus, a Cholesky factorization can be used to solve the pair of linear equations. Furthermore, since both Equation (2.16) and (2.17) have the same coefficient matrix, one Cholesky factorization of $\hat{P}$ is sufficient to solve both equations. This offers a significant advantage since the Cholesky factorization is the most computationally expensive part of solving (2.10). In practice, $\hat{P}$ is often a sparse matrix, and so sparse techniques further reduce solver runtimes, as discussed in Section 2.3. However, a method for obtaining E_{xs} and $E_{\kappa\tau}$ is still needed, and is presented next.

Mehrotra Predictor-Corrector

In order to estimate the second order terms that arise from the non-linear system of equations (2.10) and to update the centering term, ν, we use the Mehrotra predictor-corrector scheme [3, 4], which has exceptional convergence properties and is computationally inexpensive. During the prediction step, E_{xs} and $E_{\kappa\tau}$ are set to be zero; these values are then used to solve the system of equations in (2.10). At this point, a scaling term $0 < \alpha \leq 1$ is computed such that the next solution does not deviate far from the central path. We employ the maximum Newton step size method to select the scaling term. A value of α is computed for each cone such that the next iterate is in the interior of the cone:

$$\alpha = \min\{r\alpha_x,\, r\alpha_s,\, r\alpha_\tau,\, r\alpha_\kappa, 1\},$$

where $0 < r < 1$ close to one (e.g., $r = 0.995$), and α_x, α_s, α_τ, κ are the maximum values of α such that

$$x + \alpha_x \Delta x \in \text{int}\mathcal{K}, \quad s + \alpha_s \Delta s \in \text{int}\mathcal{K}, \alpha_\tau \tau + \Delta\tau > 0, \quad \kappa + \alpha_\kappa \Delta\kappa > 0.$$

The value of α can further be reduced to keep the next iterate in a prescribed neighborhood of the central path, for example by using self-regular proximity measures as in [6]. For the PDG problem, however, further limiting was not needed as convergence is reliable with the maximum Newton step size method.

After the predicted search direction is obtained, the predicted complementarity gap is:

$$g_p = (x + \alpha \Delta x_p)^T (s + \alpha \Delta s_p) + (\kappa + \Delta\kappa_p)(\tau + \Delta\tau_p),$$

where subscript "p" indicates the solution with E_{xs} and $E_{\kappa\tau}$ zero. The new centering parameter, ν, is then taken to be [9]:

$$\nu = \left(\frac{g_p}{x^T s + \kappa\tau} \right)^2 \frac{g_p}{k+1}. \tag{2.18}$$

The predicted Newton directions are then used to approximate the second order terms:

$$E_{xs} = \Delta\tilde{X}_p \Delta\tilde{S}_p e, \tag{2.19}$$

$$E_{\kappa\tau} = \Delta\kappa_p \Delta\tau_p, \quad \text{where} \tag{2.20}$$

$$\Delta\tilde{X}_p = \text{blkdiag}(\text{arrow}(\Delta\tilde{x}_p^{(1)}), \ldots, \text{arrow}(\Delta\tilde{x}_p^{(k)})),$$
$$\Delta\tilde{S}_p = \text{blkdiag}(\text{arrow}(\Delta\tilde{s}_p^{(1)}), \ldots, \text{arrow}(\Delta\tilde{s}_p^{(k)})),$$
$$\Delta\tilde{x}_p = \Theta\tilde{G}\Delta x_p, \quad \Delta\tilde{s}_p = (\Theta\tilde{G})^{-1}\Delta s_p.$$

The updated second order approximations and the new centering parameter are then used to

solve system (2.10) again [3,4]. Since the coefficient matrix, $\hat{P}$, is unchanged, the factorized matrix from the prediction step is reused.

2.1.2 Algorithm Overview

The entire IPM algorithm that is being customized is summarized below.

Data: a tolerance, ϵ, an initial point (x_0, s_0, y_0, τ_0, κ_0) as described above, and the problem structure given as: A, b, and c.

Result: an optimal solution, (x^*, s^*, y^*, τ^*, κ^*)

begin

- $x = x_0$, $\tau = \tau_0$, $y = y_0$, $s = s_0$, $\kappa = \kappa_0$;
- **while** $x^T s + \kappa\tau > \epsilon$ **do**
  - predict: solve (2.10) with $\nu = 0$, $E_{xs} = 0$, $E_{\kappa\tau} = 0$;
  - calculate Newton step size, α;
  - update ν, Eq. (2.18);
  - correct: solve (2.10) with E_{xs}, $E_{\kappa\tau}$ in Eqs. (2.19, 2.20);
  - calculate Newton step size, α;
  - $x = x + \alpha\Delta x$, $\tau = \tau + \alpha\Delta\tau$;
  - $y = y + \alpha\Delta y$, $s = s + \alpha\Delta s$;
  - $\kappa = \kappa + \alpha\Delta\kappa$;
- **if** $\tau > 0$ **then**
  - $x^* = \frac{x}{\tau}$, $s^* = \frac{s}{\tau}$;
  - $y^* = \frac{y}{\tau}$, $\tau^* = \tau$;
  - $\kappa^* = \kappa$;
- **else if** $\kappa = 0$ **then** Problem is ill posed;
- **else if** $c^T x < 0$ **then** Dual is infeasible;
- **else if** $b^T y > 0$ **then** Primal is infeasible;

2.2 Solver Customization

In this section, we present the methods developed to generate efficient C code for specific SOCP problem classes (formally defined below), and the customization techniques that enabled an onboard implementation of the fully constrained PDG [25, 28]. Most embedded systems are constrained by limited memory and strict real-time requirements. To conform ourselves to these constraints, we employ the Approximate Minimum Degree (AMD) [60] algorithm and explicit coding [25, 26, 56] to reduce memory usage and reduce computation time.

Many problems that naturally occur in engineering tend to translate into sparse optimization problems, such as the discretized control problem found in [61]. We leverage sparsity in the problem structure to minimize both memory usage and the number of numerical operations required to obtain a solution. Moreover, autonomous systems rely on the results of onboard optimization to safely operate, so it is also important to minimize logical branches for ease of verification, a reduction in the number of possible failure cases, and for increased computational speed.

In general, sparse linear algebra operations (like sparse matrix-matrix multiplication) reduce the total number of elementary operations, but increase the cost of each operation; in other words, the number of operations is reduced by avoiding the addition and multiplication of zeros, but it is now necessary to determine how the remaining non-zero elements interact. By using explicit coding, the computationally expensive task of determining the interactions between non-zero elements can be determined once beforehand and hard coded to generate software that avoids the logic altogether.

Both a generalized SOCP solver and a capability for automatically generating customized solvers in ANSI-C were implemented in performing this work.

2.3 Sparsity

Solving a SOCP reduces to solving a pair of related linear system of equations $\hat{P}v_0 = \xi$ and $\hat{P}v_1 = \bar{a}$, where $\hat{P} = \hat{P}^T \succ 0$ by construction [36]. Since $\hat{P}$ is a symmetric, positive definite matrix, a Cholesky factorization is performed to obtain:

$$\hat{P} = \hat{L}\hat{L}^T, \tag{2.21}$$

where $\hat{L}$ is a lower triangular matrix. To speed computation, the goal is to reduce the number of non-zero elements present in $\hat{L}$, which depends on the sparsity pattern - that is, the location of non-zero elements - of $\hat{P}$. Therefore, a permutation matrix, R, is sought that reduces the number of non-zero elements in the Cholesky factorization by a similarity transform of $\hat{P}$, giving the new factorization matrix, L:

$$R\hat{P}R^T = LL^T, \tag{2.22}$$

where L is a lower triangular matrix with a reduced number of non-zero entries. The Approximate Minimum Degree method [60] is used to obtain an R that is both inexpensive to compute and effective at reducing the number of non-zero elements in L. Substituting (2.21) and (2.22) into the linear equality constraint gives:

$$LL^T Rv_0 = R\xi. \tag{2.23}$$

where $RR^T = I$, and the Cholesky factorization and corresponding forward/back substitutions have a reduced number of non-zero entries to process. A similar transformation can be done to the second linear equation to obtain:

$$LL^T Rv_1 = R\bar{a}, \tag{2.24}$$

where L and R are the same as in (2.23).

2.4 Memory Allocation and Explicit Coding

Much of the computational effort associated with sparse linear algebra stems from not knowing the problem structure beforehand. Each time an operation is carried out, the interaction between non-zero element is determined on the fly. Fortunately, a given embedded system typically solves problems with a uniform structure; that is, the locations of most non-zero elements in A, b, and c do not change. For example, changing the initial conditions in the PDG problem does not affect the structure of A, b, or c. To take advantage of this fixed structure, a problem class is formally defined as follows.

Definition 1. *Given $A_0 \in \mathbb{R}^{p\times n}$, $b_0 \in \mathbb{R}^p$, $c_0 \in \mathbb{R}^n$, and $\mathcal{K}_0 \subset \mathbb{R}^n$, a problem class, $\mathcal{P}$, is defined as:*

$$\mathcal{P} = \{A \in \mathbb{R}^{p\times n}, b \in \mathbb{R}^p, c \in \mathbb{R}^n, \mathcal{K} \subset \mathbb{R}^n : \mathrm{str}(A) \leq \mathrm{str}(A_0), \mathrm{str}(b) \leq \mathrm{str}(b_0), \\ \mathrm{str}(c) \leq \mathrm{str}(c_0), \mathcal{K} = \mathcal{K}_0\},$$

where the $\leq$ operator denotes element-wise inequality, and str maps any non-zero element to a 1 and leaves 0 elements undisturbed, thereby forming the sparsity structure of its input. Thus, any A that is more sparse than A_0 is also in $\mathcal{P}$ if any zero element in A_0 is also a zero element in A (similarly for b and c).

Without loss of generality, assume that a given embedded system solves a set of problems $\mathcal{P}_0 \subseteq \mathcal{P}$. Then, the problem class, and therefore an upper bound on the sparsity structure, is known and implementing a generic solver on an embedded device is wasteful. Since the problem size is known, the exact amount of memory that is necessary to solve $\mathcal{P}$ is statically allocated, removing the need for dynamic memory allocation altogether. Removing the need to dynamically allocate memory is essential to any application with a strict timing requirement. Furthermore, this practice eliminates memory leaks/fragmentation, overflows, and more importantly reduces the complexity of flight software verification.

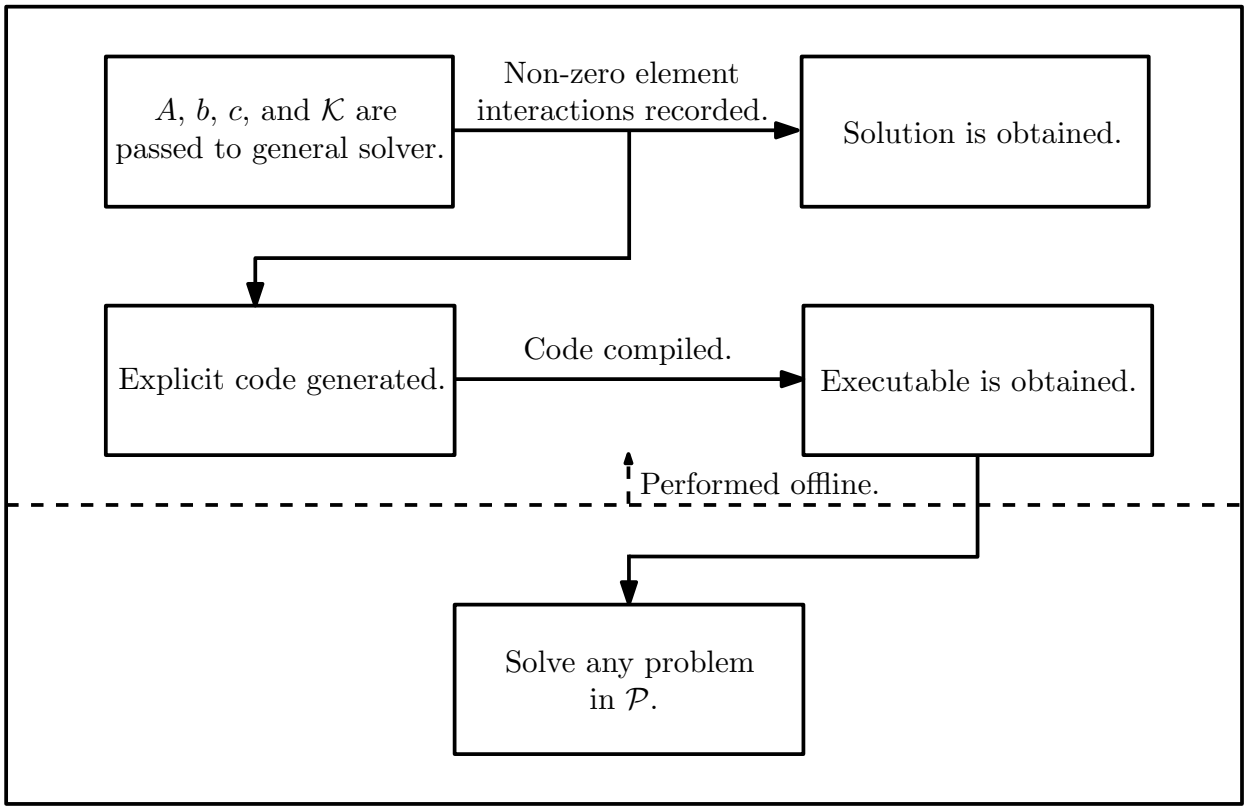


Figure 2.1: Flow diagram for explicit code generation.

Furthermore, the generated code is free of the logical operations introduced by sparse algorithms. Once the interaction between two non-zero elements has been determined, one line of C code is generated to handle the interaction directly instead of using logical comparisons or conditions. Figure 2.1 illustrates the procedure for generating a customized solver for a given problem class. This customized sparse framework supports the IPM algorithm described in Section III of [36] , which is also included in the generated code.

Explicit coding is also used to avoid certain expensive operations, such as finding a suitable R via AMD and some matrix multiplications. Note that R in (2.22) above depends only on the sparsity structure of $\hat{P}$ [60]. Further, recall that $\hat{P} = AD^2A^T$, str(A) is bounded from above by some A_0, and str(D) relies solely on $\mathcal{K} = \mathcal{K}_0$. Thus, there is an upper bound on

$\text{str}(\hat{P})$ for each problem class. The permutation matrix corresponding to this upper bound is found when the custom solver is generated, and is then hard coded into the solver. Furthermore, since R represents a similarity transformation, the matrix multiplication, $R\hat{P}R^T$ in (2.22), and all of the matrix-vector multiplications involving R in (2.23) and (2.24) are treated as known element-wise mappings. For example, given a similarity mapping,

$$R = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

and an arbitrary $v \in \mathbb{R}^2$ such that $v = [v_1 \ v_2]^T$. A $u = [u_1 \ u_2]^T = Rv$ can be computed without any explicit multiplications by treating the operation as a mapping:

$$u_1 = v_2, \quad u_2 = v_1.$$

These mappings are also hard-coded into the generated C code, and help reduce the complexity of the runtime operations.

As problem size increases, the amount of C code that is generated grows rapidly. Therefore, the sheer amount of machine instructions causes instruction cache misses which eventually outweigh the algorithmic advantages of avoiding branching and conditional operations. For this reason, customization is best suited for small-to-medium problem sizes (1-1000 solution variables), as observed empirically in Section 2.5. Therefore, the customization described herein is beneficial for real-world applications with around 1,000 solution variables or less. Increasing the effectiveness of this type of code generation for larger problems is the subject of on-going research; however, it is noted that the authors have successfully and repeatedly landed a rocket using approximately 750 variables.

2.5 Solver Performance

In this subsection, we compare the runtime performance of the solver developed in [36] (called Bsocp) with several other well-known solvers. Every problem class is run 500 times, and the

average runtime on a workstation with an Intel Core i7 (3.4 GHz) processor and 15 GB of RAM is presented.

SDPT3 [62], SeDuMi [63], ECOS [8], Bsocp, and customized solvers are used over a range of problem sizes to find optimal solutions of the planetary soft landing problem described in [36]. The problem size is varied by increasing the number of points in the discretization of the dynamics [11]. Table 2.1 shows the types of cones used in the performance comparisons of Figures 2.2 and 2.3.

n	# $\mathcal{K}_L$	# $\mathcal{K}_s$
320	153	49
567	270	88
757	360	118
1,137	540	178
1,327	630	208
1,612	765	253
1,897	900	298
3,797	1,800	598
9,497	4,500	1,498
15,197	7,200	2,398

Table 2.1: Benchmark Details

where n represents the solution variable size, # $\mathcal{K}_L$ the number of linear cones in the solution variable, and the # $\mathcal{K}_s$ column represents the number of second order cones in the solution variable. Furthermore, all second order cones are 4-dimensional.

For small problems (1-1,000 solution variables), the customized solvers solved the problems about two times faster than the Bsocp solver (Figure 2.2). Problems with about 2,000 solution variables gain almost no advantage by customization, as expected. ECOS and Bsocp performed similarly for small-to-medium problems, with each taking turns outperforming the other over different regions.

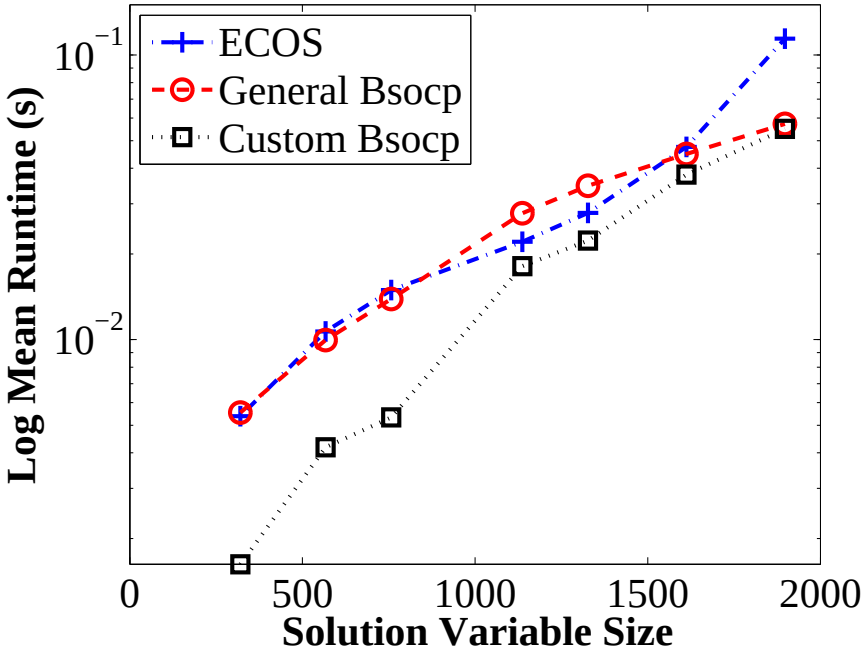


Figure 2.2: Planetary Landing Benchmark for real-time solvers and problems of small to medium size

For problems with more solution variables than that of the simulations shown in Figure 2.2, instruction cache misses dominate any gains from customization. For this reason, benchmarking results are shown for larger problems without customized solvers. As one can see from Figure 2.2, ECOS scales better than Bsocp for problems with more than about 10,000 solution variables. This trend is due to the linear algebra libraries in Bsocp not being intended for larger problems. Nonetheless, Bsocp outperforms SDPT3 and SeDuMi for all problem sizes in the benchmark.

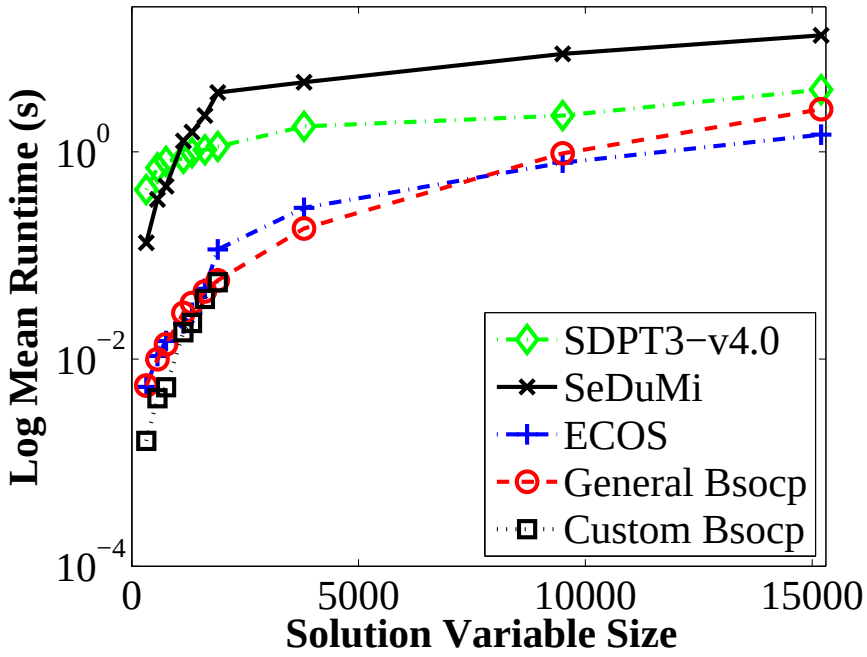


Figure 2.3: Planetary Landing Benchmark with broader range of solvers and problem sizes.

2.6 Performance on a Flight Processor

As shown in Section 2.5, solution times on the order of milliseconds to tens-of-milliseconds are possible for smaller problem sizes when run on a 3.4-GHz desktop processor. For planetary landing, however, PDG would be expected to run on a radiation-hardened flight processor. As an example of a flight processor, the Mars Science Laboratory used a BAE RAD750 PowerPC for its entry, descent, and landing. This flight processor currently has a clock speed on the order of 200 MHz.

Therefore, for eventual landing applications on bodies other than the Earth, it is important to characterize the runtime of a representative PDG problem on a flight processor. Due primarily to differences in instruction sets (x86 processors are CISC or hybrid, whereas the PowerPC is RISC) and memory access speeds, a PDG runtime on a desktop processor cannot simply be scaled by clock speed to a flight processor (i.e., increasing the runtime on a desktop by a factor of 3000 MHz / 200 MHz).

Hence, we ran a convexified PDG algorithm in C on a 200-MHz RAD750 that is part of a flight software testbed at NASA JPL. The experimental setup of the runtime characterization is first discussed and then the results.

Setup of Runtime Experiment

For this experiment, the real-time operation system used is VxWorks. Besides the PDG, an Ethernet driver is the only other process running on the processor, so PDG has close to 100% of the processor's computing resources.

In addition to the solver described in 2.3, the PDG algorithm also consists of a parser that translates the guidance problem and data into an SOCP. This parser is written in C and includes approximations to the standard Problem 3 that reduce runtime. For example, at each timestep some second order cone constraints are replaced with several tangent planes. Runtime data was collected for both the parser and the solver.

The resulting SOCP problem is then passed to the *solver* as in Eqn. 2.1. The parser is also written in C and includes approximations to the standard Problem found in 3 that reduce runtime. Runtime data has been collected for both the parser and the solver.

Furthermore, similar parameters for this PDG were chosen to better compare against the results in 2.5. The problem's size parameters are $n = 403$, $\#\mathcal{K}_L = 263$, and $\#\mathcal{K}_S = 35$. All second order cones are 4-dimensional. In addition, there are 286 equality constrains, which is the number of rows in A in (2.1). This problem size lies approximately between the first two points of Figure 2.2. However, as noted, there are fewer second order cones than in the benchmark problem.

For the results in this section, a stressing instantiation of the PDG problem is desired. "Stressing" in this context does not mean size: given the speed of flight processors, a smaller-sized problem would likely be flown. Instead, the goal is to have as many inequality constraints active as possible. An active constraint has at least one time step in which the associated slack variable is zero. Colloquially, the solution "rides" each inequality constraint at least once.

The stressing problem is a speed-limited divert as discussed in the companion paper. The maximum-thrust-angle, maximum-speed, and maximum-thrust constraints are all active. Due to the desire to activate the maximum-speed constraint—it is the sole second order cone state constraint—the resulting divert trajectory approximates a straight line from initial position to final position. This straight-line behavior, in turn, necessarily deactivates the glideslope constraint. The behavior also deactivates the minimum-thrust constraint as follows. The thrust profile is essentially "bang-off-bang" in magnitude. During the "off" portion, the vehicle is coasting at maximum speed with thrust that cancels weight. Since the minimum thrust is less than the vehicle weight at all points in the trajectory—allowing the vehicle to "fall"—minimum-thrust is never needed.

Finally, since the stressing problem rides the maximum-speed constraint, the minimum-time solution is empirically determined to be the minimum-fuel solution. That is, considering the propellant mass needed to divert as a function of time-of-flight, if the time-of-flight is less than some t^*, the resulting PDG problem is infeasible. Above t^*, the propellant mass increases monotonically to the physical limits of problem (see Figure 2.4). The relevance of this behavior to runtime characterization is explained subsequently.

The variation of the runtime depends on the feasibility of the given time-of-flight, as well as the distance between initial position and the feasibility boundary. If the time-of-flight is infeasible, the runtime to certify the infeasiblity also varies. To fully analyze the performance, we characterized the runtime across both the feasible and infeasible ranges.

For this problem, a value of $t^* = 42.1748\,s$ was empirically determined. Note that this value will vary with the number of points in the time discretization. The times-of-flight tested are centered on this t^* and go from $29.7998\,s$ to $54.5498\,s$ in $0.25\,s$ steps, producing 100 times-of-flight. These times-of-flight stretch sufficiently far into the feasible range for the runtime to exhibit a steady-state behavior. Similarly, they stretch far enough into the infeasible range that the apparent variations in runtime are captured. Also, these times give both an infeasible and a feasible time-of-flight within $0.125\,s$ of t^*.

To average out any runtime variations due to operating system schedule, the PDG cal-

culation is repeated 25 times for each of the 100 times-of-flight.

Finally, the default optimization level `-O0` was used to ensure reliable performance. Nonetheless we deployed other individual optimization flags and the `-march` option.

Runtime Results

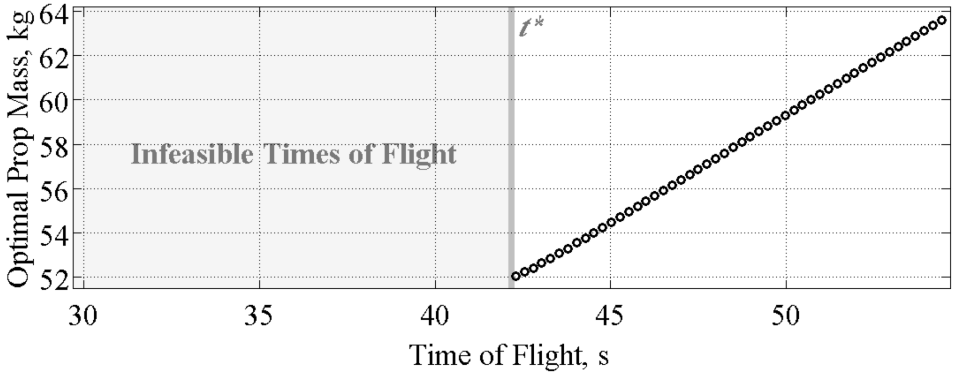


Figure 2.4: Optimal Propellant as a Function of Time-of-Flight for a Small, Stressing PDG Problem.

The main result is shown in Figure 2.5a, which shows mean runtimes for the customized solver for the stressing PDG problem as a function of time-of-flight. Mean solver runtimes vary between approximately $0.4\,s$ and $1.3\,s$ for solutions with duality gaps that are less than 1e-9. Observe that simply scaling a $3\,ms$ runtime from Figure 2.3 by 3.4 GHz/200 MHz gives $0.051\,s$, which underestimates the runtimes by an order of magnitude.

Before discussing these results further, consider Figure 2.5c, which shows the variations from the mean runtime for each of the times-of-flight. Observe that the variations are clustered around $0\,ms$ and are uncorrelated with the mean runtimes in Figure 2.5a and with feasibility. These independent, small variations from the mean show that no intermittent tasks were meaningfully interrupting the solver. Also, since the magnitudes of the variations from the mean are $< 1\%$, the mean runtimes can accurately be referred to simply as "runtimes."

In addition, over all of the 2500 PDG trajectory optimizations, the parser runtime averaged $2.1\,ms$ per trajectory with variations of only several microseconds. This repeatable runtime for the parser is expected because it only encodes data: its computations are not affected by feasibility or infeasibility.

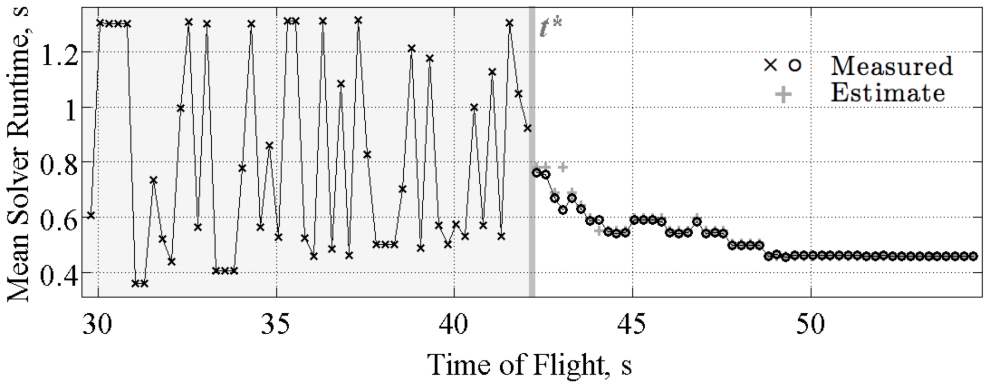
Comparing Figures 2.5a and 2.5b, it is clear—and expected—that for feasible trajectories, runtime is effectively the number of iterations multiplied by the average time for an iteration. It is also expected that more iterations are required to converge as the time-of-flight approaches the feasibility boundary from the feasible side. To obtain an average time per iteration, note that for times-of-fight larger than $50\,s$, the number of iterations is 11 and the runtime is almost constant (there is a slight downward trend). Computing an average runtime-per-iteration from these times-of-flight and multiplying by the number of iterations produces the estimates shown in Figure 2.5a. The two outliers in the estimates are an area for future investigation.

Considering the left-hand side of Figure 2.5a, the time to determine infeasibility is uncorrelated with number of iterations, which is not unexpected since each iteration can move to a significantly different area of the solution domain. Moreover, it can take 50% longer to determine infeasibility than to calculate a barely-feasible trajectory.

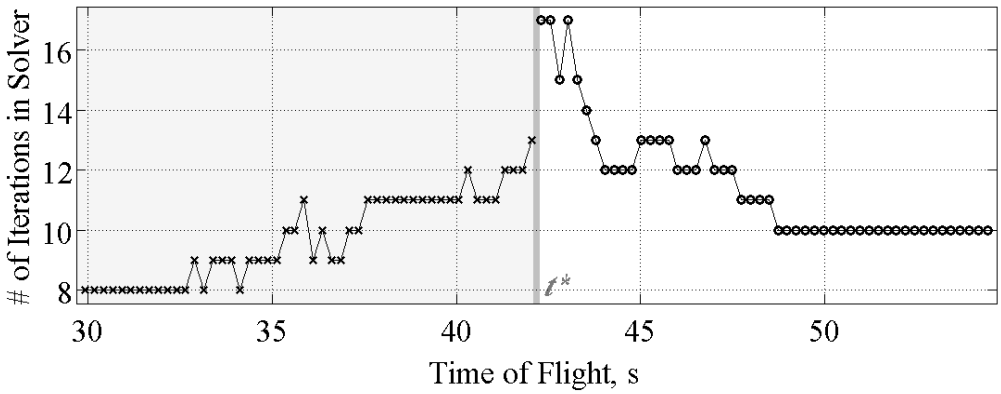
The non-customized, general solver was also tested. All the characteristics displayed in Figure 2.5 are identical for the general solver on a flight processor, except that the time to determine infeasibility is relatively constant at $1.55\,s$. More importantly, since the `-O3` compile flag could not be enforced in the current implementation of the customized solver, customization gives only a 16% speed improvement over the general solver on this flight processor. A 50% improvement was obtained on a desktop processor.

Nonetheless, it has been shown that per time-of-flight, infeasibility or an optimal trajectory can be calculated in approximately $0.7\,s$—the average of all 2500 solver runtimes—on a state-of-the-art radiation-hardened flight processor. In practice, PDG would not have the whole processor to itself, but would be running in the background with, for example, 25%-to-30% processor availability. In this case, the effective runtime per time-of-flight increases

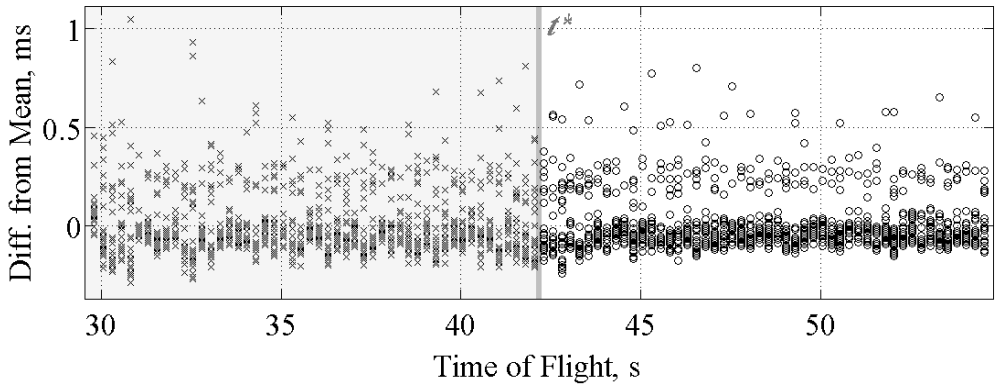
to approximately 2.5 s, which is still suitable for on-board implementation [64].



(a) Mean runtime of customized solver. Estimate is number of iterations multiplied by mean time for an iteration for times-of-flight $\geq 50\,s$.



(b) Number of iterations in solver.



(c) Per time-of-flight, difference between 25 runtimes and mean.

Figure 2.5: Performance of the Customized Solver on a Flight Processor for a Small, Stressing PDG Problem as a Function of Time-of-Flight.

Chapter 3

REAL-TIME OPTIMIZATION IN MISSION PLANNING AND HIGH–LEVEL AUTONOMY

Controllability and reachability in the presence of control and state constraints is a topic of great importance, and has attracted research interest since the onset of optimal control [65–71]. Although widely unused in industry, finding Constrained Reachable and Controllable (CRC) sets has the potential to replace costly Monte Carlo analysis early in the mission design process when design parameters are changing rapidly [1]. Indeed, controllability and reachability analyses provide a systematic approach to determining the feasibility of mission goals given nominal design parameters, and can be extended to uncertain systems by incorporating a chance constrained framework [72–75]. Moreover, thanks to recent computational and algorithmic advancements (such as the customized, embedded solvers described in Chapter 2), the methods described in this chapter can be used to rapidly compute polytopic approximations of a broad class of compact, convex CRC sets in as little as 300 ms using MATLAB on a single threaded desktop workstation for three dimensional CRC sets.

CRC sets are also pivotal for higher-level decision making in the synthesis of hierarchical control strategies, and are useful for systems abstractions used in the validation and verification of autonomous systems [76]. Spurred by a recent demand for autonomy from industry and consumers, the field has begun to see interest in characterizing CRC sets via optimization–based algorithms [65, 66, 68, 69]. These methods are designed to make use of reliable real–time convex optimization solvers [7, 36, 48, 77] – typically either second order cone programming or linear programming solvers for real–time applications. The reliability and speed of these solvers are key for the development of online applications with a far greater degree of resilience and robustness to sudden, unforeseen, or unpredictable variations

in the vehicle dynamics and environment. For example, after not having precise control during its atmospheric entry and parachute stages, a Mars lander equipped with thrusters may autonomously select a point of interest within its reachable set as a landing site, or it may disqualify other landing sites based on each landing site's controllable set (Figure 3.1). It is the goal of this chapter to show that such high–level, real–time decision making is within our grasp.

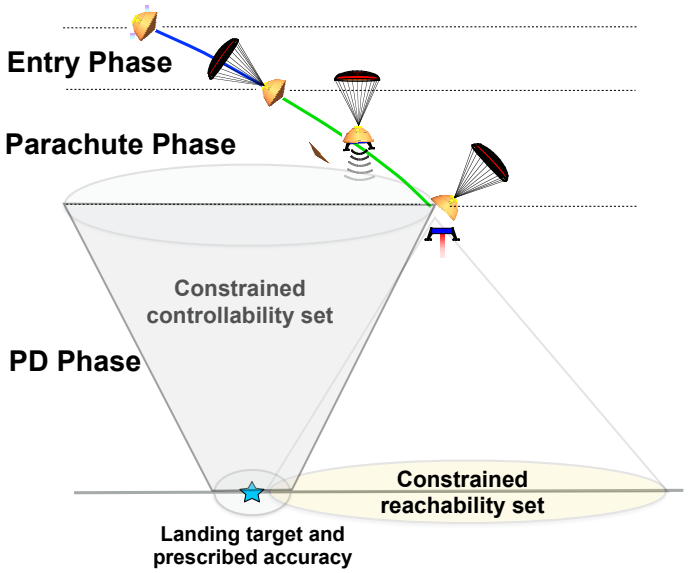


Figure 3.1: Illustration of CRC sets for planetary landing. Shaded areas indicate the CRC sets [1].

In this chapter, we describe different initialization schemes, and give two algorithms for computing inner and outer polytopic approximations of compact, convex CRC sets: (1) a method that is driven by a monotonously growing inner approximation (an outer approximation is obtained as a by-product) [1,37,44,75]; (2) a method that is driven by a monotonously shrinking outer approximation and has convergence guarantees (an inner approximation is obtained as a by-product). We begin by defining the problem and introducing some useful definitions.

3.1 Problem Formulation

As in [37], we consider a time snapshot of all constraints associated with a discrete–time, finite–horizon, control problem (for linear, time–varying dynamics and convex constraints) specified as follows:

$$\forall j \in \mathcal{N}_0^{N-1}, x_{j+1} = \ A_j x_j + B_j u_j, \tag{3.1}$$

$$\forall j \in \mathcal{N}_0^{N-1}, (x_j, u_j) \in \mathcal{Y}_j, \tag{3.2}$$

$$x_0 \in \mathcal{X}_0, \tag{3.3}$$

$$x_N \in \mathcal{X}_f. \tag{3.4}$$

Here, j represents an index of future times relative to the current time, $t = t_0$. Also, N is the number of discrete time steps, each $x_j \in \mathbb{R}^{n_x}$ is the state at time t_j, each $u_j \in \mathbb{R}^{n_u}$ is the control at time t_j, and the state–control constraint sets $\mathcal{Y}_j \in \mathbb{R}^{n_x+n_u}$ are specified as the intersections of **finitely** many second order cone and affine constraints. Similarly, the sets $\mathcal{X}_0$ and $\mathcal{X}_f$ are specified as the intersections of finitely many second order cone and affine constraints in $\mathbb{R}^{n_x}$.

3.1.1 N–step Controllability under constraints

The set of admissible control sequences for N–step controllability to a target set, $\mathcal{X}_f$, is specified for any $x_0 \in \mathbb{R}^{n_x}$ as follows:

$$\mathcal{U}_N(x_0) = \left\{ \{u_j\}_{j=0}^{N-1} \ : \ (3.1), (3.2) \text{ and } (3.4) \text{ hold} \right\}. \tag{3.5}$$

The associated N–step controllable set is defined as the set of all initial states, x_0, for which there exists at least one admissible control sequence (i.e., a control sequence belonging to $\mathcal{U}_N(x_0)$), as formally specified below.

Definition 2. *The N–step controllable set, $\mathcal{X}_N^-$, to a target set, $\mathcal{X}_f$, is given by:*

$$\mathcal{X}_N^- = \{x_0 \in \mathbb{R}^{n_x} \ : \ \mathcal{U}_N(x_0) \neq \emptyset\} \tag{3.6}$$

In what follows, for typographical convenience, we make use of the term "controllable set/s" instead of the full expression "N–step controllable set/s to a target set."

3.1.2 N–step Reachability under constraints

The set of admissible initial states and control sequence pairs for N–step reachability from an initial set, $\mathcal{X}_0$, is specified for any $x_N \in \mathbb{R}^{n_x}$ as follows:

$$\mathcal{W}_N(x_N) = \left\{ \left(x_0, \{u_j\}_{j=0}^{N-1}\right) \ : \ (3.1), (3.2) \text{ and } (3.3) \text{ hold} \right\}. \tag{3.7}$$

The associated N–step reachable set is defined as the set of all terminal states, x_N, for which there exists at least one admissible initial state and control sequence pair (i.e., an initial state and control sequence pair belonging to $\mathcal{W}_N(x_N)$), as formally specified below.

Definition 3. *The N–step reachable set, $\mathcal{X}_N^+$, from an initial set, $\mathcal{X}_0$ is given by:*

$$\mathcal{X}_N^+ = \{x_N \in \mathbb{R}^{n_x} \ : \ \mathcal{W}_N(x_N) \neq \emptyset\} \tag{3.8}$$

Note that the above notion can be seen as *strongly weak* reachability, in that we are interested in all possible states obtained through N–step iteration of dynamics by utilizing *all admissible initial state and control sequence pairs*. In a similar manner as above, we make use of the term "reachable set/s" instead of the full expression "N–step reachable set/s from an initial set."

3.1.3 Generalized Problem and Regular Setting

The CRC sets, as defined in (3.6) and (3.8), are typically known only implicitly, so the main computational objective is to find their explicit representations, or a pair of inner and outer

polytopic approximations to an arbitrarily small, prescribed precision. In particular, the implicit representations of the CRC sets are modeled as:

$$\mathcal{X} = \{x \ : \ \exists \ y \text{ s.t. } (x, y) \in \mathcal{Z}\}\,. \tag{3.9}$$

Above, in the case of controllable sets, x should be identified with initial state x_0, y should be identified with a control sequence $\{u_j\}_{j=0}^{N-1}$, and $\mathcal{Z}$ should be identified with the graph of the set of admissible control sequences $\mathcal{U}_N(x_0)$, i.e.,

$$\mathcal{Z} = \{(x_0, \{u_j\}_{j=0}^{N-1}) \ : \ (3.1), (3.2) \text{ and } (3.4) \text{ hold}\}.$$

Likewise, in the case of reachable sets, x takes on the role of the terminal state x_N, y is the initial state and control sequence pair $(x_0, \{u_j\}_{j=0}^{N-1})$, and $\mathcal{Z}$ is the graph of the set of admissible initial state and control sequence pairs $\mathcal{W}_N(x_N)$, i.e.,

$$\mathcal{Z} = \{(x_N, x_0, \{u_j\}_{j=0}^{N-1}) \ : \ (3.1), (3.2) \text{ and } (3.3) \text{ hold}\}.$$

The above generalized problem is considered in a regular compact setting, namely in the case when the underlying set $\mathcal{Z}$ has a non–empty interior and is a compact set. An immediate implication of this regular compact setting is t that non–emptiness of the interior and compactness of $\mathcal{X}$ are inherited from the corresponding properties of $\mathcal{Z}$. Within this setting, the set $\mathcal{Z}$ is the intersection of finitely many second order cone and affine constraints, and its compactness is guaranteed by the compactness of the sets $\mathcal{Y}_j$ and $\mathcal{X}_0$ or $\mathcal{X}_f$. Note that the case when $\mathcal{X}$ is not a full dimensional set in $\mathbb{R}^n$ is addressed by applying an appropriate change of variables which transforms the original problem to a full–dimensional problem relative to a suitable subspace of $\mathbb{R}^n$. For the above reasons and conceptual simplicity, this chapter focuses on the setting in which the set $\mathcal{Z}$ has a non–empty interior, is compact, and its explicit representation is given via *finitely* many second order cone and affine constraints. The remainder of the chapter focuses on approximating such a set $\mathcal{X}$.

3.2 Preliminaries

In this section, we provide definitions that are necessary for the development of the two iterative approximation algorithms that follow. Both algorithms extensively utilize the well-known Hausdorff distance metric as a topologically suitable metric for convergence. In both cases, changes in consecutive iterations are driven by the minimization of a Hausdorff distance (or an upper bound of the Hausdorff distance), similar to work done in model predictive control [44]. This minimization induces a monotonically non–increasing sequence of Hausdorff distances that play a crucial role in their convergence properties. Moreover, as we will see in the following sections, both algorithms make use of a particular structure to minimize the computations required to evaluate the Hausdorff distance at each iteration.

3.2.1 Hausdorff Distance

Given compact sets, $\mathcal{A}$ and $\mathcal{B}$, their Hausdorff distance is:

$$H(\mathcal{A},\mathcal{B}) := \min_{\alpha\in\mathbb{R}_+} \{\alpha : \mathcal{A} \subseteq \mathcal{B} \oplus \alpha\mathbb{B}, \mathcal{B} \subseteq \mathcal{A} \oplus \alpha\mathbb{B}\}, \tag{3.10}$$

where $\mathbb{B}$ denotes the closed unit norm ball with respect to the Euclidean distance. In the case of polytopic sets, the Hausdorff distance can be computed efficiently as discussed below.

Point–to–set Distance: The distance from a point r to the set $\mathcal{S} = \text{conv}\{s_1, s_2, \ldots, s_m\}$, is computable via the following optimization:

$$\begin{aligned} &\text{minimize } \|r - \sum_{i=1}^{m} \lambda_i s_i\| \\ &\text{w.r.t. } \lambda_i,\ i \in \mathcal{N}_1^m, \\ &\text{subject to } \sum_{i=1}^{m} \lambda_i = 1 \text{ and } \ \forall i \in \mathcal{N}_1^m,\ \lambda_i \geq 0. \end{aligned} \tag{3.11}$$

Note that the optimization in (3.11) evaluates the value of the distance function $d(r, \mathcal{S}) := \inf_s\{\|s - r\| \ : \ s \in \mathcal{S}\}$ at the point r with respect to to the set $\mathcal{S}$.

Semi–Hausdorff Distance: The semi–Hausdorff distance between the sets $\mathcal{A} = \text{conv}\{a_1, a_2, \ldots, a_p\}$ and $\mathcal{B} = \text{conv}\{b_1, b_2, \ldots, b_q\}$ is the maximum of the point–to–set distances $d(\mathcal{B}, a_i)$, $i \in \mathcal{N}_1^p$. In particular, the semi–Hausdorff distance is given by:

$$\begin{aligned} h(\mathcal{A}, \mathcal{B}) &= \min_{\alpha \in \mathbb{R}_+} \{\alpha : \mathcal{A} \subseteq \mathcal{B} \oplus \alpha\mathbb{B}\} \\ &= \max\{d(a_i, \mathcal{B}) \ : \ i \in \mathcal{N}_1^p\}. \end{aligned} \tag{3.12}$$

Since the semi–Hausdorff distance is not symmetric, we also define:

$$\begin{aligned} h(\mathcal{B}, \mathcal{A}) &= \min_{\alpha \in \mathbb{R}_+} \{\alpha : \mathcal{B} \subseteq \mathcal{A} \oplus \alpha\mathbb{B}\} \\ &= \max\{d(b_j, \mathcal{A}) \ : \ j \in \mathcal{N}_1^q\}. \end{aligned} \tag{3.13}$$

Note that each of the point–to–set distances $d(\mathcal{B}, a_i),\ i \in \mathcal{N}_1^p$ and $d(\mathcal{A}, b_j),\ j \in \mathcal{N}_1^q$ can be evaluated directly using (3.11). Finally, the Hausdorff distance definition in (3.10) can equivalently be expressed as the maximum of the related semi–Hausdorff distances:

$$H(\mathcal{A}, \mathcal{B}) = \max\{h(\mathcal{A}, \mathcal{B}), h(\mathcal{B}, \mathcal{A})\}. \tag{3.14}$$

The expression for the Hausdorff distance given in (3.14) is particularly useful for the algorithms given in the following sections due to its computational tractability when one set is a subset of the other. Suppose $\mathcal{A} \subseteq \mathcal{B}$, then $h(\mathcal{A}, \mathcal{B}) = 0$ and one need only evaluate the semi–Hausdorff distance $h(\mathcal{B}, \mathcal{A})$ to obtain the Hausdorff distance. This observation drastically reduces the amount of point–to–set distances that must be computed at each iteration.

3.2.2 Support Function

For a compact, convex set $\mathcal{A} \subset \mathbb{R}^n$, the support function, $s_{\mathcal{A}} : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$s_{\mathcal{A}}(x) := \sup_a \{\langle x, a\rangle : a \in \mathcal{A}\}.$$

Owing to the convexity and compactness of $\mathcal{A}$, the support for an arbitrary $x \in \mathbb{R}^n$ can be computed efficiently via an IPM solver.

3.2.3 Polytope Definitions

The following terms are extensively used throughout the chapter, and are therefore formally defined here.

Definition 4. ***A face*** *is a lower dimensional polyhedron on the boundary of a higher dimensional polyhedron.*

Definition 5. ***A k-face*** *is a face that has non-zero volume in* $\mathbb{R}^k$.

Definition 6. ***A facet*** $\mathcal{F}$ *is an* $(n-1)$*-face.*

Definition 7. *Two vertices of a polytope* $\mathcal{P} \subset \mathbb{R}^n$ *are* ***adjacent*** *if they share* $n-1$ *linearly independent facets of* $\mathcal{P}$.

3.3 Initialization of the Bounding Polytopes

The goal here is to obtain an inner polytopic approximation $\underline{\mathcal{X}}$ whose vertices lie on the boundary of the CRC set $\mathcal{X} \in \mathbb{R}^n$ (for some $n \leq n_x$), and an outer approximation $\overline{\mathcal{X}}$ whose hyperplanes support $\mathcal{X}$ such that $\underline{\mathcal{X}} \subseteq \mathcal{X} \subseteq \overline{\mathcal{X}}$. As in [1, 37, 75], this is the first step to approximating $\mathcal{X}$, and two approaches can be utilized to accomplish this task. One relies on finding the symmetric simplex with the largest volume contained in the set $\mathcal{X}$ via the solution of a Semi-Definite Program (SDP), and the other solves a series of SOCPs to bound the polytope. While both are useful and presented here, the latter is substantially more computationally tractable.

3.3.1 SDP Initialization

The first step is to characterize a symmetric simplex of maximal volume contained in $\mathcal{X}$ by formulating the problem as a linear matrix inequality [2]. We begin by making a simple, but very useful observation.

Remark 2. *The volume of a simplex in* $\mathbb{R}^n$ *with* $n+1$ *vertices* $\{\underline{x}_0, \underline{x}_1, \ldots, \underline{x}_n\}$ *can be computed via the associated matrix* $Q = [\underline{x}_1 - \underline{x}_0, \ldots, \underline{x}_n - \underline{x}_0] \in \mathbb{R}^{n\times n}$ *as:*

$$\mathrm{vol}_n\{\mathrm{conv}\,\{\underline{x}_0, \underline{x}_1, \ldots, \underline{x}_n\}\} = \frac{1}{n!}|\det Q|. \tag{3.15}$$

A direct consequence of (3.15) is that maximizing the determinant of the matrix Q also maximizes the volume of the associated simplex. Furthermore, if Q is symmetric, it can be obtained via the solution of the following SDP by noting that the logdet() function is a concave function [2].

$$\begin{aligned} &\text{maximize } \mathrm{logdet}\, Q \\ &\text{w.r.t. } Q \in \mathbb{R}^{n\times n}, \underline{x}_j \in \mathcal{X}, j \in \mathcal{N}_0^n \\ &\text{subject to } Q = Q^T = [\underline{x}_1 - \underline{x}_0, \ldots, \underline{x}_n - \underline{x}_0]. \end{aligned} \tag{3.16}$$

Note that the monotonicity and continuity of the logarithm function over $\mathbb{R}_+$ ensure that the maximizer of (3.16) is also a maximizer of (3.15). Moreover, $\underline{x}_j,\ j \in \mathcal{N}_0^n$ lie on the boundary of $\mathcal{X}$, are distinct, and are suitable vertices with which to initialize the simplex $\underline{\mathcal{X}}_0$. The associated facets of $\underline{\mathcal{X}}_0$ are trivial to construct from the vertices since $\underline{\mathcal{X}}_0$ is a regular simplex. Initializing the outer approximation requires the solution of $n+1$ additional SOCPs, one for each facet of $\underline{\mathcal{X}}_0$. More specifically, let $\ell \in \mathbb{R}^n$ be an outward-pointing facet normal of unit length for a facet of $\underline{\mathcal{X}}_0$, then the SOCP problem takes the following form:

$$\begin{aligned} &\text{maximize } \langle \ell, \underline{x} \rangle \\ &\text{subject to } \underline{x} \in \mathcal{X}. \end{aligned} \tag{3.17}$$

Owing to the optimality of the maximizer, each ℓ is taken as a facet of $\overline{\mathcal{X}}_0$ since the related facet supports $\mathcal{X}$ at $\underline{x}$. It is trivial to convert the facets of $\overline{\mathcal{X}}_0$ into the corresponding set of vertices of $\overline{\mathcal{X}}_0$ since $\overline{\mathcal{X}}_0$ is also a regular simplex.

3.3.2 SOCP Initialization

In general, solving a Linear Matrix Inequality (LMI) is a computationally expensive operation that is not well suited to real-time performance. In this section, a method for instantiating suitable inner and outer polytopic approximations of $\mathcal{X}$ without the need for an LMI is introduced. The reduction in initialization complexity is achieved by relaxing the optimality of $\underline{\mathcal{X}}_0$ with respect to $\mathcal{X}$ - that is, $\underline{\mathcal{X}}_0$ is no longer the largest symmetric simplex contained in $\mathcal{X}$. However, due to the rapid convergence of the methods presented in this chapter, the sub-optimal initialization given herein does not significantly affect runtimes. As before, $\underline{\mathcal{X}}_0$ and $\overline{\mathcal{X}}_0$ are treated independently.

Outer Approximation

Let $\ell_j,\ j \in \mathcal{N}_1^{n+1}$, be $n+1$ distinct unit vectors that represent the facet normals of a regular simplex in $\mathbb{R}^n$ (so that any combination of n ℓ_j's span $\mathbb{R}^n$). Then, $n+1$ points on the boundary of $\mathcal{X}$ can be computed by evaluating the values of the corresponding support functions $s_{\mathcal{X}}(\ell_j),\ j \in \mathcal{N}_1^{n+1}$, i.e., by solving $n+1$ SOCPs

$$\begin{aligned} &\text{maximize } \ell_j^T x \\ &\text{w.r.t. } x \in \mathbb{R}^n, \\ &\text{subject to } \ x \in \mathcal{X}, \end{aligned} \tag{3.18}$$

where, as before, the implicit form of the constraint $x \in \mathcal{X}$ is used. Note, that the associated arg–maximizers $\underline{x}_j,\ j \in \mathcal{N}_1^{n+1}$ become candidates for the vertices of $\underline{\mathcal{X}}_0$, and are neither necessarily distinct nor unique. In the latter case, their selection is induced by the numerical solver. The facial representation of $\overline{\mathcal{X}}_0$ is obtained directly from $n+1$ linear inequalities,

each of which takes the form:

$$\langle \ell_j, x \rangle \leq s_{\mathcal{X}}(\ell_j).$$

The set $\mathcal{V}(\overline{\mathcal{X}}_0)$ is also easy to construct. Namely, each vertex $\overline{x}_i,\ i \in \mathcal{N}_1^{n+1}$ is the solution to a set of n linearly independent linear equations specified by:

$$\langle \ell_j, \overline{x}_i \rangle = s_{\mathcal{X}}(\ell_j),\ j \in \mathcal{N}_1^{n+1} \setminus \{i\}$$

The set $\overline{\mathcal{X}}_0 = \operatorname{conv}\{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_{n+1}\}$ is guaranteed to have non–zero volume due to the appropriate choice of ℓ_j's. We note that a convenient selection of ℓ_j's makes use of the Euclidean basis (unit) vectors for the first n ℓ_j's while the last ℓ_j is $-1_n/\sqrt{n}$, where $1_n \in \mathbb{R}^n$ is a vector of ones. Naturally, any non–degenerate affine transformation of this collection of ℓ_j's can also be employed.

Inner Approximation

If the determinant of the related matrix $Q = \left[(\underline{x}_2 - \underline{x}_1), \ldots, (\underline{x}_{n+1} - \underline{x}_1)\right]$ is not equal to 0, then the points $\underline{x}_j,\ j \in \mathcal{N}_1^{n+1}$ define the vertices of a regular simplex. In this case, $\mathcal{V}(\underline{\mathcal{X}}_0) := \{\underline{x}_1, \ldots, \underline{x}_{n+1}\}$ and the facial representation of $\underline{\mathcal{X}}_0$ can be constructed in a similar fashion as that of $\overline{\mathcal{X}}_0$. If, however, $\det(Q) = 0$, then a suitable $\underline{\mathcal{X}}_0$ can be constructed from the solution of $n+2$ additional SOCPs. First, solve one optimization to obtain a point, say $\underline{x}_c$, in the interior of $\mathcal{X}$ (e.g., the analytic center). Then, the following optimization is solved for all $j \in \mathcal{N}_1^{n+1}$:

$$\begin{aligned} &\text{maximize } \alpha \\ &\text{w.r.t. } \alpha \in \mathbb{R}_+, \\ &\text{subject to } \ \underline{x}_c + \alpha(\overline{x}_j - \underline{x}_c) \in \mathcal{X}. \end{aligned} \tag{3.19}$$

The optimal values $\alpha_j,\ j \in \mathcal{N}_1^{n+1}$ are finite because $\mathcal{X}$ is compact, and are also strictly positive because $\underline{x}_c$ is in the interior of $\mathcal{X}$. Thus, the points,

$$\underline{x}_j := \underline{x}_c + \alpha_j(\overline{x}_j - \underline{x}_c),\ j \in \mathcal{N}_1^{n+1}$$

form an appropriate set of $n+1$ vertices for $\underline{\mathcal{X}}_0$ and can also be used to construct its facial representation.

It is worth pointing out that the $2n+3$ SOCP optimizations used in the worst case to construct $\underline{\mathcal{X}}_0$ and $\overline{\mathcal{X}}_0$ are far more computationally efficient than a single max logdet LMI optimization. In addition, initial inner and outer approximate sets are regular simplices whose construction neither depends on nor requires additional evaluations of the support functions specified in (3.18). All in all, the described initialization, though arbitrary, provides computationally relevant benefits over the previously utilized one.

3.3.3 Runtime comparison

The two initialization schemes discussed above are utilized to generate suitable $\underline{\mathcal{X}}_0$ and $\overline{\mathcal{X}}_0$ for a Mars Planetary Descent Guidance problem with $n = 3$.

	Section 3.3.1	Section 3.3.2
Runtime (ms)	11793	11

Table 3.1: Initialization runtimes for $n = 3$

3.4 Iterations Driven by Inner Bounding Polytope

With an inner and outer polytopic approximation of $\mathcal{X}$ initialized, this section focuses on an algorithm driven by the growth of the inner polytopic approximation, $\underline{\mathcal{X}}_0$. The inner polytope is grown via the solution of SOCPs, whose goal is to increase the number of vertices of $\underline{\mathcal{X}}$ on the boundary of $\mathcal{X}$.

3.4.1 Haudorff Distance Computation

In order to take advantage of (3.14), the non-zero Hausdorff semi-distance between $\underline{\mathcal{X}}_0$ and $\overline{\mathcal{X}}_0$ must be computed. This is accomplished by computing the point-to-set distance between each vertex of $\overline{\mathcal{X}}_0$ and the set $\underline{\mathcal{X}}_0$ via $n+1$ solutions of (3.11). For computational efficiency, the resulting point-to-set distances are stored in the set, $\mathcal{D}_0 := \{d(\overline{x}_1, \underline{\mathcal{X}}_0), \ldots, d(\overline{x}_{n+1}, \underline{\mathcal{X}}_0)\}$. Then, the Hausdorff distance $H(\underline{\mathcal{X}}_0, \overline{\mathcal{X}}_0) = \max(\mathcal{D}_0)$. In the following algorithm, vertices will be introduced to and removed from the verticial representation of $\overline{\mathcal{X}}$, and as each vertex is added or removed, the associated point-to-set distance is also added to or removed from $\mathcal{D}$. This procedure, in addition to updating stale distances, ensures $H(\underline{\mathcal{X}}, \overline{\mathcal{X}})$ remains easily computable via $\max(\mathcal{D})$.

3.4.2 Hausdorff–based Facet Prioritization

The objective is to add vertices to $\underline{\mathcal{X}}$ in a manner that decreases the Hausdorff distance between the inner and outer polytopic approximations of $\mathcal{X}$. Naturally, the information gleaned from adding a vertex to $\underline{\mathcal{X}}$ is utilized to remove some volume from the outer approximation, $\overline{\mathcal{X}}$. At the k^{th} iteration of this algorithm, $\mathcal{V}(\underline{\mathcal{X}}_k) = \{\underline{x}_1, \ldots, \underline{x}_{n+1+k}\}$ and $\underline{\mathcal{X}}_k$ admits a facial representation containing outward facing facet normals $\mathcal{L}_k = \{\ell_1, \ldots, \ell_s\}$. In addition, suppose the set of upper bounds of the point–to–set distances from $\mathcal{V}(\overline{\mathcal{X}}_k)$ to the set $\underline{\mathcal{X}}_k$ is known. Thus, the related Hausdorff distance satisfies $H(\underline{\mathcal{X}}_k, \overline{\mathcal{X}}_k) = \max\{\mathcal{D}_k\}$. The progression from the pair $(\underline{\mathcal{X}}_k,\ \overline{\mathcal{X}}_k)$ to the pair $(\underline{\mathcal{X}}_{k+1}, \overline{\mathcal{X}}_{k+1})$ such that

$$\underline{\mathcal{X}}_k \subseteq \underline{\mathcal{X}}_{k+1} \subseteq \mathcal{X} \subseteq \overline{\mathcal{X}}_{k+1} \subseteq \overline{\mathcal{X}}_k \tag{3.20}$$

is accomplished by minimizing an upper bound of the Hausdorff distance between the pair $(\underline{\mathcal{X}}_{k+1}, \overline{\mathcal{X}}_{k+1})$. The inner approximate polytopic set $\underline{\mathcal{X}}_{k+1}$ is obtained from the set $\underline{\mathcal{X}}_k$ by adding a suitably determined external point $\underline{x}_{n+1+k+1}$ of the set $\mathcal{X}$. Likewise, the facial representation of $\overline{\mathcal{X}}_{k+1}$ is updated by adding a linear inequality describing a hyperplane that supports the set $\mathcal{X}$ at the point $\underline{x}_{n+1+k+1}$. The determination of the point $\underline{x}_{n+1+k+1}$ and

corresponding supporting hyperplane is performed by detecting the furthest vertex of the set $\overline{\mathcal{X}}_k$ from the set $\underline{\mathcal{X}}_k$ and related facet normal from the set $\mathcal{L}_k$. To accomplish this, first identify:

1. $\overline{x}^* \in \mathcal{V}(\overline{\mathcal{X}}_k)$ that also belongs to the set $\arg\max\{\mathcal{D}_k\}$, and

2. a facet normal of $\underline{\mathcal{X}}_k$, ℓ^* that belongs to the set $\arg\max\{\langle \ell, \overline{x}^* \rangle - s_{\underline{\mathcal{X}}_k}(\ell) \ : \ \ell \in \mathcal{L}_k\}$.

Then, the detected facet normal ℓ^* is utilized to find the value of the support function $s_{\mathcal{X}}(\ell^*)$. In turn the argmax of this maximization, or its selection induced by the numerical solver, yields the desired vertex of $\underline{\mathcal{X}}_{k+1}$, namely the point $\underline{x}_{n+1+k+1}$ on the boundary of $\mathcal{X}$. Furthermore, this also yields a linear inequality $\langle \ell^*, \underline{x}_{n+1+k+1} \rangle \leq s_{\mathcal{X}}(\ell^*)$ that updates the facial representation of $\overline{\mathcal{X}}_k$ in order to obtain the facial representation of $\overline{\mathcal{X}}_{k+1}$. The facial representation of $\underline{\mathcal{X}}_{k+1}$ is obtained by removing the facet corresponding to the normal of ℓ^* and adding the minimal number of facets passing through the newly discovered vertex $\underline{x}_{n+1+k+1}$ and supporting the set $\underline{\mathcal{X}}_k$. (The normals belonging to the set $\mathcal{L}_k \setminus \{\ell^*\}$ and these new facet normals constitute the convex hull of $\underline{\mathcal{X}}_{k+1}$.) Likewise, the vertices of $\overline{\mathcal{X}}_{k+1}$ are obtained by finding the vertices on the newly discovered facet of $\overline{\mathcal{X}}_{k+1}$ induced by the linear equality $\langle \ell^*, x \rangle = s_{\mathcal{X}}(\ell^*)$, and retaining those vertices of $\overline{\mathcal{X}}_k$ which satisfy $\langle \ell^*, x \rangle \leq s_{\mathcal{X}}(\ell^*)$.

The rationale of the procedure described above is that the minimization of the worst case point–to–set distance from the vertices of $\overline{\mathcal{X}}_k$ to the set $\underline{\mathcal{X}}_k$ is employed for facet prioritization. This aims to reduce the value of the Hausdorff distance $H(\underline{\mathcal{X}}_k, \overline{\mathcal{X}}_k)$ as a guaranteed upper bound for the Hausdorff distance $H(\underline{\mathcal{X}}_{k+1}, \overline{\mathcal{X}}_{k+1})$. Since, by construction, $\underline{\mathcal{X}}_k \subseteq \underline{\mathcal{X}}_{k+1} \subseteq \mathcal{X}$ and $\mathcal{X} \subseteq \overline{\mathcal{X}}_{k+1} \subseteq \overline{\mathcal{X}}_k$, this indirectly minimizes the Hausdorff distance $H(\underline{\mathcal{X}}_{k+1}, \overline{\mathcal{X}}_{k+1})$. Once $\underline{\mathcal{X}}_{k+1}$ and $\overline{\mathcal{X}}_{k+1}$ are obtained, their actual Hausdorff distance is recomputed. This re-computation is carried out by retaining the valid point-to-set distances from the previous set of point-to-set distances $\mathcal{D}_k$ as well as by quickly evaluating the point-to-set distances related to the newly discovered vertices lying on the facet $\{x \ : \ \langle \ell^*, x \rangle = s_{\mathcal{X}}(\ell^*)\} \cap \overline{\mathcal{X}}_k$ to $\underline{\mathcal{X}}_{k+1}$. This leverages the fact that the majority of the point-to-set distances remain the same

as the algorithm advances from iteration k to $k+1$. Furthermore, note that the number of variables appearing in (3.11) is equal to $(n+1+k)$, thus the complexity of the optimizations used to solve the point-to-set distances is easily manageable. In any case, the Hausdorff distance satisfies $H(\underline{\mathcal{X}}_{k+1}, \overline{\mathcal{X}}_{k+1}) = \max\{\mathcal{D}_{k+1}\}$, where $\mathcal{D}_{k+1}$ denotes the set of point–to–set distances from the vertices of $\overline{\mathcal{X}}_{k+1}$ to $\underline{\mathcal{X}}_{k+1}$.

In view of the guaranteed relation (3.20), the above procedure enjoys the desired monotonicity properties summarized by the following technical observation:

$$\forall k \geq 0, \quad H(\underline{\mathcal{X}}_{k+1}, \overline{\mathcal{X}}_{k+1}) \leq H(\underline{\mathcal{X}}_{k}, \overline{\mathcal{X}}_{k}).$$

The procedure described above forms our main iteration step, and is formally summarized next.

3.4.3 Main Iteration Step

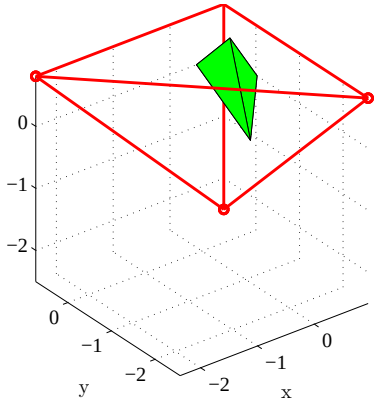
Once a facet has been selected, the following procedure is applied to advance to $\underline{\mathcal{X}}_{k+1}$ and $\overline{\mathcal{X}}_{k+1}$:

1. Compute $s_{\mathcal{X}}(\ell^*)$ to find a new vertex of $\underline{\mathcal{X}}_{k+1}$: $\underline{x}_{n+1+k+1}$.
2. Append facets as necessary to $(\mathcal{L}_k \backslash \{\ell^*\})$ in order to obtain the facial representation of $\underline{\mathcal{X}}_{k+1}$.
3. Append ℓ^* to the facial representation of $\overline{\mathcal{X}}_k$. Remove vertices of $\overline{\mathcal{X}}_k$ that satisfy $\langle \ell^*, x \rangle \geq s_{\mathcal{X}}(\ell^*)$, and add the new vertices that lie on the facet $\{x : \langle \ell^*, x \rangle = s_{\mathcal{X}}(\ell^*)\} \cap \overline{\mathcal{X}}_k$, thus transitioning to $\overline{\mathcal{X}}_{k+1}$.
4. Update the set of point-to-set distances $\mathcal{D}_k$ to obtain $\mathcal{D}_{k+1}$.

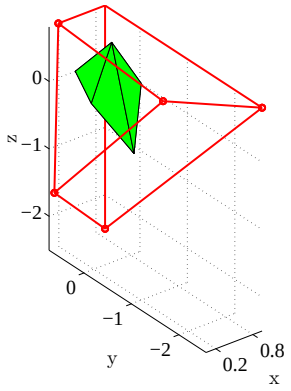
The above procedure terminates when $H(\underline{\mathcal{X}}_k, \overline{\mathcal{X}}_k)$ becomes smaller than a prescribed value. It is worth pointing out that the procedure can also be utilized to generate inner and outer approximate polytopic sets of desired complexity - this allows for a meaningful trade-off between allowed complexity of the representation and the accuracy of the approximation.

3.4.4 3–Dimensional Illustrative Example

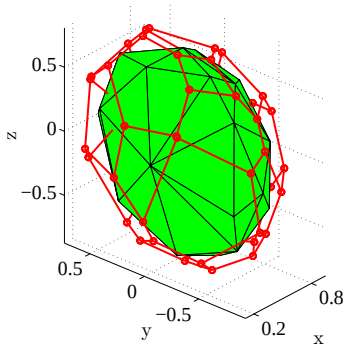
The controllability set for a 3–dimensional Planetary Descent Guidance problem is computed here to illustrate the proposed algorithm.



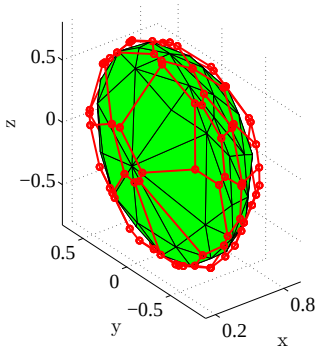
(a) $k = 0$: Simplex initialization as in Section 3.3.2.



(b) $k = 1$: Support has been computed once. A significant amount of volume is removed from the outer approximation as the procedure in Section 3.4.2 is followed.



(c) $k = 25$: Support has been found 25 times as the procedure in Section 3.4.2 is followed.



(d) $k = 50$: The converged inner and outer approximations after computing support 50 times.

Figure 3.2: 3–D Controllability Set (position) for Constrained MPL at different stages of the algorithm. $\underline{\mathcal{X}}_k$ (green) and $\overline{\mathcal{X}}_k$ (red) are plotted for different values of k, illustrating the algorithm's progression. Runtime: 1.15 s in MATLAB. The algorithm is considered to be converged when $H(\underline{\mathcal{X}}_k, \overline{\mathcal{X}}_k) \leq \epsilon = 0.0915$.

Both $\underline{\mathcal{X}}_0$ and $\overline{\mathcal{X}}_0$ are initialized without the need to solve an LMI (Figure 3.2a). 6.14

ms later, the algorithm iterates to $k = 1$ by following the procedure in Section 3.4.2. Since $H(\underline{\mathcal{X}}_k, \underline{\mathcal{X}}_{k+1})$ is small (the facet growth was not large), a significant amount of volume is removed from $\overline{\mathcal{X}}_0$ when iterating to $\overline{\mathcal{X}}_1$ (Figure 3.2b). As the algorithm iterates, the Hausdorff distance between the inner and outer approximations decreases monotonically (Figure 3.3). Once 25 iterations have been evaluated, the polytopic approximations begin to resemble the controllability set (Figure 3.2c), and the rate of decrease in $H(\underline{\mathcal{X}}_k, \overline{\mathcal{X}}_k)$ begins to settle (Figure 3.3) – this occurs when many vertices of $\overline{\mathcal{X}}_k$ have similar point-to-set distances to the set $\underline{\mathcal{X}}_k$, and signifies that the two sets are converging somewhat uniformly. At the 50^{th} iteration, the two sets are considered converged since their Hausdorff distance is below the predefined threshold (Figure 3.2d).

The same problem was solved using the simplex initialization and heuristic–based facet prioritization from previous work [1, 75]. The first five data points in Figure 3.3 suggest that utilizing an arbitrary initial inner and outer simplex pair does not significantly affect convergence rate. The evolution of $H(\underline{\mathcal{X}}_k, \overline{\mathcal{X}}_k)$ for the previous method is characteristic of non–uniform convergence, as evidenced by the series of discontinuities in Figure 3.3. Uniform convergence is an important property to have in a real-time CRC approximation algorithm, as lack of development in a any region of the CRC set can mislead automated decision–making systems and compromise higher–level goals. The runtimes for both the proposed and previous algorithms at $n = 3$ are given in Table 3.2 for comparison.

Computation Type	Proposed [37] (ms)	Previous [1, 75] (ms)
Initialization ×2	22	23585
Computing Support×50	307	307
Hausdorff ×243	38	N/A
Volume ×243	N/A	29

Table 3.2: Total optimization runtimes for $n = 3$

Overall, 367 ms (in total) are spent computing the optimization-related portions of the proposed approach. The proposed algorithm takes an average of 1.15 seconds to compute the 54-vertex approximation of $\underline{\mathcal{X}}_k$ above, compared to 30.77 seconds spent with the previous

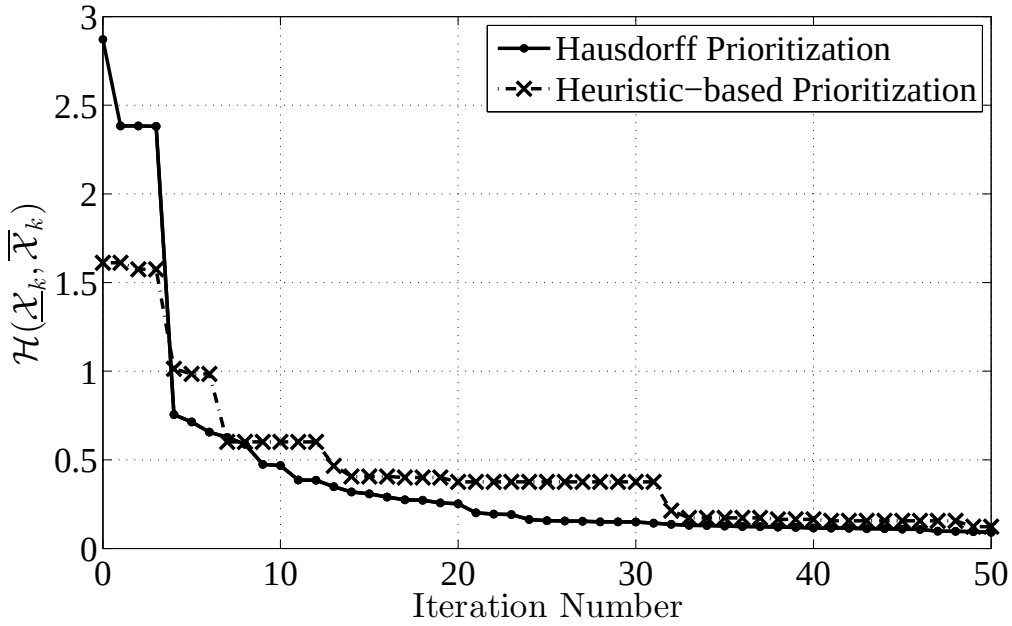


Figure 3.3: $H(\underline{\mathcal{X}}_k, \overline{\mathcal{X}}_k)$ as the algorithms converge. Note that $\underline{\mathcal{X}}_k$ and $\overline{\mathcal{X}}_k$ approach each other monotonically by construction.

approach. This represents a strong push towards the real–time computation of polytopic approximation for CRC sets.

3.5 Iterations Driven by Outer Bounding Polytope

The previous section discusses an approach that is driven by the growth of the inner bounding polytope. In this section, the proposed algorithm is driven by shrinking the outer bounding polytope, and importantly, *is proven to converge*. To the best of our knowledge, this is the first algorithm for the polytopic approximation of an arbitrary, compact, bounded, convex set in $\mathbb{R}^n$ for which a convergence proof is given.

3.5.1 Regular Geometry

We first introduce a regular geometry to facilitate the convergence proof in Section 3.5.3. The ultimate objective of this section is to approximate a compact, convex, and full-dimensional

body $\mathcal{X} \subset \mathbb{R}^n$ with a sequence of regular outer-bounding polytopes, $\mathcal{X} \subseteq \{\overline{\mathcal{X}}_k\}_{k=0}^{\infty} \subset \mathbb{R}^n$. To this end, we introduce the first regularity definition.

Definition 8. *A polytope $\overline{\mathcal{X}} \subset \mathbb{R}^n$ is a **regular outer-bounding polytope** (ROBP) of a compact, convex, and full-dimensional set $\mathcal{X} \subset \mathbb{R}^n$ if it satisfies the following properties:*

1. $\mathcal{X} \subseteq \overline{\mathcal{X}}$.

2. *For each facet $\mathcal{F} \subset \overline{\mathcal{X}}$, $\exists\ x \in \mathcal{F}$ such that $x \in \mathcal{X}$.*

Suppose $\overline{\mathcal{X}}$ is a ROBP of $\mathcal{X}$, and the objective is to remove the vertex $\overline{x}_{cut} \in \mathcal{V}(\overline{\mathcal{X}})$ from $\overline{\mathcal{X}}$ by intersecting $\overline{\mathcal{X}}$ with a cutting half-space. This leads to the introduction of a *regular* cut.

Definition 9. *A halfspace $\mathcal{H}_{cut}^{-}(\hat{n})$ uniquely defined by the direction $\hat{n}$ ($\|\hat{n}\|_2 = 1$), $\mathcal{H}_{cut}^{-}(\hat{n}) = \{r \in \mathbb{R}^n : \langle \hat{n}, r \rangle \leq s(\mathcal{X}, \hat{n})\}$ is a **regular cut** for $\overline{x}_{cut} \in \mathcal{V}(\overline{\mathcal{X}})$ and $\mathcal{X}$ if it satisfies the following properties:*

1. *(Separation): $\overline{x}_{cut} \notin \mathcal{H}_{cut}^{-}(\hat{n})$.*

2. *(Support): $\exists\ x \in \mathcal{X}$ such that $x \in \mathcal{H}_{cut}(\hat{n})$.*

By the Separation Theorem (or Separating Hyperplane Theorem), a regular cut always exists for $\overline{x}_{cut}$ and $\mathcal{X}$ since $\mathcal{X}$ and $\{\overline{x}_{cut}\}$ are both compact and convex [2, 78]. Further, the *Support* property of a regular cut, along with the convexity of $\mathcal{X}$ ensure that the intersection of a regular cut and a ROBP is a ROBP.

3.5.2 Preparatory Computations

In order to present the algorithm and its convergence analysis, we extend the notation for the point-to-set distance used above to include an arbitrary convex, compact set. Consider

a compact, convex set $\mathcal{X}$ and a point $\overline{x} \notin \mathcal{X}$, then the closest point to $\overline{x}$ in $\mathcal{X}$ is given by

$$\underline{x} = \arg\min_{x \in \mathcal{X}} \|x - \overline{x}\|_2, \tag{3.21}$$

with corresponding point-to-set distance $d(\overline{x}, \underline{x})$, where it is clear that

$$d(\overline{x}, \underline{x}) = \min_{x \in \mathcal{X}} \|x - \overline{x}\|_2.$$

Note that since $\mathbb{R}^n$ is a finite dimensional Euclidean vector space and $\mathcal{X}$ is a compact, convex set, the point $\underline{x}$ is uniquely defined for any $\overline{x} \in \mathbb{R}^n$ [2, 78]. Thus, the Hausdorff distance between a convex, compact set $\mathcal{X}$ and a ROBP of $\mathcal{X}$, $\overline{\mathcal{X}}$ can be obtained by solving (3.21) finitely many times, i.e.,

$$d(\overline{\mathcal{X}}, \mathcal{X}) = \max_{\overline{x} \in \mathcal{V}(\overline{\mathcal{X}})} d(\overline{x}, \mathcal{X}).$$

3.5.3 Approximation Algorithm

In this section, we present an algorithm for determining the direction of each cut in such a way that the sequence of ROPBs is guaranteed to converge to the compact, convex body, $\mathcal{X}$. Suppose $\overline{\mathcal{X}}_k$ is a ROBP of $\mathcal{X}$, and let $\overline{x}_{cut,k} \in \mathcal{V}(\overline{\mathcal{X}}_k)$ be any vertex that satisfies

$$\overline{x}_{cut,k} \in \arg\max_{\overline{x} \in \overline{\mathcal{X}}_k} d(\overline{x}, \mathcal{X}). \tag{3.22}$$

Consider an algorithm that transitions from $\overline{\mathcal{X}}_k$ to $\overline{\mathcal{X}}_{k+1}$ by intersecting a regular cut $\mathcal{H}^-_{cut,k}(\hat{n}_k)$ with $\overline{\mathcal{X}}_k$, where

$$\hat{n}_k = \frac{\overline{x}_{cut,k} - \underline{x}_{cut,k}}{\|\overline{x}_{cut,k} - \underline{x}_{cut,k}\|_2}, \tag{3.23}$$

where $\underline{x}_{cut,k} \in \mathcal{H}^-_{cut,k}(\hat{n}_k)$ is found via (3.21). As a result of (3.23), $\overline{x}_{cut,k} \notin \overline{\mathcal{X}}_{k+1} \subset \overline{\mathcal{X}}_k$ for all $\overline{x}_{cut,k}$ such that $d(\overline{x}_{cut,k}, \mathcal{X}) > 0$. Clearly, the sequence $\{d(\overline{\mathcal{X}}_k, \mathcal{X})\}_{k=0}^{\infty}$ is monotonically non-increasing. Further, each time an ROBP $\overline{\mathcal{X}}_k$ is intersected with a regular cut, new vertices

are introduced, forming the set

$$\overline{\mathcal{X}}_{new,k} := \mathcal{V}(\overline{\mathcal{X}}_{k+1}) \setminus \mathcal{V}(\overline{\mathcal{X}}_k).$$

Data: Implicit representation of $\mathcal{X}$ and an $\epsilon \geq 0$
Result: ROBP $\overline{\mathcal{X}}_k$ and inner polytopic approximation $\underline{\mathcal{X}}_k$
begin
  Initialize $\overline{\mathcal{X}}_0$ as in [37], $k = 0$;
  Compute and store $d(\overline{x}, \mathcal{X}),\ \forall\, \overline{x} \in \mathcal{V}(\overline{\mathcal{X}}_0)$;
  while $d(\overline{\mathcal{X}}_k, \mathcal{X}) > \epsilon$ **do**
    Identify $\overline{x}_{cut,k}$ as in (3.22);
    Compute $\hat{n}_k$ using (3.23);
    $\overline{\mathcal{X}}_{k+1} = \overline{\mathcal{X}}_k \cap \mathcal{H}^-_{cut,k}(\hat{n}_k)$;
    Compute and store $d(\overline{x}, \mathcal{X}),\ \forall\, \overline{x} \in \overline{\mathcal{X}}_{new,k}$;
    $k = k + 1$;
  return $\overline{\mathcal{X}}_k$ and $\underline{\mathcal{X}}_k = \text{conv}\left\{\underline{x}_{cut,j}\right\},\ j \in \mathcal{N}_0^k$;

Note that the "compute and store" operations in the algorithm store both the point-to-set distance and the point in $\mathcal{X}$ which minimizes the distance from $\overline{x}$, i.e. $\underline{x}_{cut,k}$. Also, note that the iterative portion of the algorithm only solves *one* type of optimization problem: the point-to-set distance problem in (3.11). Further, the time required to solve each optimization remains constant as the iterations progress since the implicit representation of $\mathcal{X}$ remains unchanged by the algorithm.

3.5.4 Convergence Analysis

In this section, we show that the the proposed algorithm converges to $\mathcal{X}$, that is,

$$\lim_{k \to \infty} d(\overline{\mathcal{X}}_k, \mathcal{X}) = 0.$$

For notational simplicity, we henceforth refer to 1-dimensional hyperplanes as *lines*.

Proposition 2. *Each* $\overline{x}_{new,k} \in \overline{\mathcal{X}}_{new,k}$ *can be expressed as* $\overline{x}_{new,k} = \gamma\,\overline{x}_i + (1-\gamma)\,\overline{x}_j$, $\gamma \in (0,1)$ *for adjacent vertices* $\overline{x}_i, \overline{x}_j \in \mathcal{V}(\overline{\mathcal{X}}_k)$.

Proof. $\overline{\mathcal{X}}_{new,k}$ is formed by the intersection of $\mathcal{H}^-_{cut,k}(\hat{n}_k)$ with $\overline{\mathcal{X}}_k$. Since $\mathcal{H}_{cut,k}(\hat{n}_k)$ is a hyperplane, new vertices form on existing 1-faces of $\overline{\mathcal{X}}_k$. All 1-faces lie on the convex combination of two adjacent vertices by Definition 4. Thus, $\overline{x}_{new,k} = \gamma\,\overline{x}_i + (1-\gamma)\,\overline{x}_j$, $\gamma \in [0,1]$ for two adjacent vertices $\overline{x}_i, \overline{x}_j \in \mathcal{V}(\overline{\mathcal{X}}_k)$. Further $\gamma \neq 0$ and $\gamma \neq 1$, since those values represent existing vertices of $\overline{\mathcal{X}}_k$. □

Proposition 3. *Every* $\overline{x}_{new,k} \in \overline{\mathcal{X}}_{new,k}$ *satisfies* $d(\overline{x}_{new,k}, \mathcal{X}) \leq d(\overline{x}_{cut,k}, \mathcal{X})$.

Proof. Note that the point-to-set distance function is a convex function for compact, convex sets. Thus, from Proposition 2,

$$\begin{aligned}
\overline{x}_{new,k} &= \gamma\,\overline{x}_i + (1-\gamma)\,\overline{x}_j, \\
d(\overline{x}_{new,k}, \mathcal{X}) &\leq \gamma\, d(\overline{x}_i, \mathcal{X}) + (1-\gamma)\, d(\overline{x}_j, \mathcal{X}), \\
&\leq \gamma\, d(\overline{x}_{cut,k}, \mathcal{X}) + (1-\gamma)\, d(\overline{x}_{cut,k}, \mathcal{X}), \\
&\leq d(\overline{x}_{cut,k}, \mathcal{X}),
\end{aligned}$$

since $\overline{x}_{cut,k}$ is chosen via (3.22). □

Corollary 4. *Suppose* $\overline{x} = \gamma\,\overline{x}_i + (1-\gamma)\,\overline{x}_j$, $\gamma \in (0,1)$ *for adjacent vertices* $\overline{x}_i, \overline{x}_j \in \mathcal{V}(\overline{\mathcal{X}}_k)$. *Then,* $d(\overline{x}, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X}) \implies d(\overline{x}_i, \mathcal{X}) = d(\overline{x}_j, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X})$.

Our objective is therefore to show that there exists a convergent subsequence of Hausdorff distances within the sequence $\{d(\overline{\mathcal{X}}_k, \mathcal{X})\}_{k=0}^{\infty}$. To this end, we more carefully consider the conditions under which $d(\overline{x}_{new,k}, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X})$.

Lemma 5. *Suppose* $\overline{x}_{cut,k}$ *is chosen as in* (3.22)*, then there are only finitely many instances for which* $d(\overline{x}_{new,k}, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X})$ *as* $k \to \infty$.

Proof. Let $d(\overline{x}_{new,k}, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X})$ for some $\overline{x}_{new,k} \in \overline{\mathcal{X}}_{new,k}$ such that $\overline{x}_{new,k} = \gamma\,\overline{x}_i + (1-\gamma)\,\overline{x}_j,\ \gamma \in (0,1)$, with adjacent $\overline{x}_i,\ \overline{x}_j \in \mathcal{V}(\overline{\mathcal{X}}_k)$. Then $d(\overline{x}_i, \mathcal{X}) = d(\overline{x}_j, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X})$ from Corollary 4. Further, suppose that $\underline{x}_i, \underline{x}_j \in \mathcal{X}$ are the solutions to (3.21) for $\overline{x}_i$ and $\overline{x}_j$ respectively. Then, it is easy to show that $\underline{x}_{new,k} = \gamma\,\underline{x}_i + (1-\gamma)\,\underline{x}_j$ is the solution to (3.21) for $\overline{x}_{new,k}$ since $d(\overline{x}_{new,k}, \underline{x}_{new,k}) = d(\overline{x}_{new,k}, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X})$ and (3.21) has a unique solution for compact, convex sets. Thus, $\underline{x}_i, \underline{x}_j$, and $\underline{x}_{new,k}$ are three distinct, co-linear points on the boundary of $\mathcal{X}$. Due to the convexity of $\mathcal{X}$, the line that passes through them is also on the boundary of $\mathcal{X}$ and contained on a polytopic k-face of $\mathcal{X}$ (see Figure 3.4). Since there are finitely many linear constraints on $\mathcal{X}$, there are finitely many k-faces on the boundary of $\mathcal{X}$, and $d(\overline{x}_{new,k}, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X})$ can only occur finitely many times. □

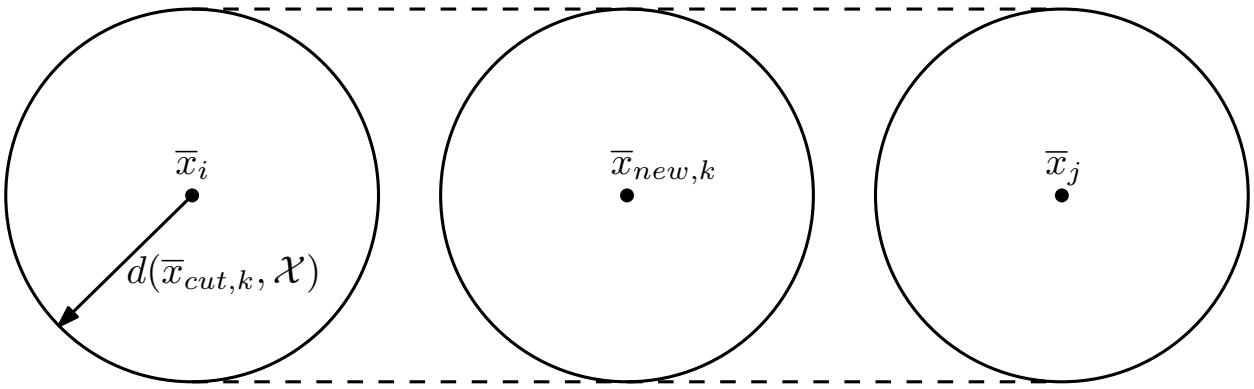


Figure 3.4: Illustration of the case when $d(\overline{x}_i, \mathcal{X}) = d(\overline{x}_j, \mathcal{X}) = d(\overline{x}_{new,k}, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X})$. One of the dashed lines must be contained in the boundary of $\mathcal{X}$.

Corollary 6. *There exists a subsequence that satisfies* $d(\overline{\mathcal{X}}_{k+1}, \mathcal{X}) < d(\overline{\mathcal{X}}_k, \mathcal{X})$ *in the sequence* $\{d(\overline{\mathcal{X}}_k, \mathcal{X})\}_{k=0}^{\infty}$.

Proof. Initially, there can be at most $n+1$ $\overline{x}_i \in \mathcal{V}(\overline{\mathcal{X}}_0),\ i \in \mathcal{N}_0^n$ that satisfy $d(\overline{x}_i, \mathcal{X}) = d(\overline{x}_{cut,0}, \mathcal{X})$, and by Lemma 5, only finitely many additional vertices with the property $d(\overline{x}_{new,k}, \mathcal{X}) = d(\overline{x}_{cut,k}, \mathcal{X})$ can be introduced. Thus, by removing these finitely many instances from $\{d(\overline{\mathcal{X}}_k, \mathcal{X})\}_{k=0}^{\infty}$, we arrive at the desired result. □

Note that the case considered in Lemma 5 only occurs when a k-face of $\overline{\mathcal{X}}_k$ is parallel to a k-face of $\mathcal{X}$. Also note that the regular cut hyperplane for $\overline{x}_{new,k}$ and $\mathcal{X}$ contains a k-face of $\mathcal{X}$ in that case.

Proposition 7. *$d(\overline{x}_{new,k}, \overline{x}_{cut,k}) \geq d(\overline{x}_{cut,k}, \mathcal{X})$, $\forall\, \overline{x}_{new,k} \in \overline{\mathcal{X}}_{new,k}$ in the subsequence that satisfies $d(\overline{\mathcal{X}}_{k+1}, \mathcal{X}) < d(\overline{\mathcal{X}}_k, \mathcal{X})$ due to Corollary 6.*

Proof. Each $\overline{x}_{new,k} \in \overline{\mathcal{X}}_{new,k}$ also lies on $\mathcal{H}_{cut,k}(\hat{n}_k)$. Further, $d(\overline{x}_{new,k}, \overline{x}_{cut,k})$ can be expressed as (see Figure 3.5),

$$\begin{aligned} d(\overline{x}_{cut,k}, \overline{x}_{new,k}) &= \sqrt{d^2(\overline{x}_{cut,k}, \mathcal{X}) + d^2(\underline{x}_{cut,k}, \overline{x}_{new,k})}, \\ &\geq d(\overline{x}_{cut,k}, \mathcal{X}) \end{aligned}$$

□

Lemma 8. *For every $\overline{x}_{cut,k}$ in the subsequence from Corollary 6, there exists a semi-hypersphere,*

$$\mathcal{S}_k := \{x \in \mathcal{B}_{2,k} :\ \langle \hat{n}_k, x \rangle \geq \langle \hat{n}_k, \overline{x}_{cut,k} \rangle\}, \tag{3.24}$$

that satisfies $\mathcal{S}_k \cap \mathcal{S}_j = \emptyset$, $\forall\ j \neq k$, where

$$\mathcal{B}_{2,k} := \mathcal{B}_2(\overline{x}_{cut,k}, d(\overline{x}_{cut,k}, \mathcal{X})).$$

Proof. From Proposition 7, the hypersphere $\mathcal{B}_{2,k}$ exists and represents the minimum distance between $\overline{x}_{cut,k}$ and any $\overline{x}_{new,k} \in \overline{\mathcal{X}}_{new,k}$ (see Figure 3.5). Note that $\left\{\mathcal{B}_{2,k} \setminus \{\underline{x}_{cut,k}\}\right\} \cap \mathcal{H}^-_{cut,k}(\hat{n}_k) = \emptyset$, so all elements of the hypersphere are removed from $\overline{\mathcal{X}}_k$ except for $\underline{x}_{cut,k}$. Thus, the distance between $\overline{\mathcal{X}}_{k+1}$ and $\mathcal{S}_k$ is $d(\overline{x}_{cut,k}, \mathcal{X})$, and since $d(\overline{\mathcal{X}}_{k+1}, \mathcal{X}) < d(\overline{\mathcal{X}}_k, \mathcal{X})$, $\mathcal{S}_k \cap \mathcal{B}_{2,j} = \emptyset$, $\forall\ j \neq k$. Therefore, $\mathcal{S}_k \cap \mathcal{S}_j = \emptyset$ since $\mathcal{S}_j \subset \mathcal{B}_{2,j}$ □

We now introduce the set valued function $\overline{\mathcal{Y}}(\cdot)$ over the domain $\mathbb{R}_+$,

$$\overline{\mathcal{Y}}(z) := \operatorname{conv}\{y \in \mathbb{R}^n :\ d(y, \mathcal{X}) = z\},$$

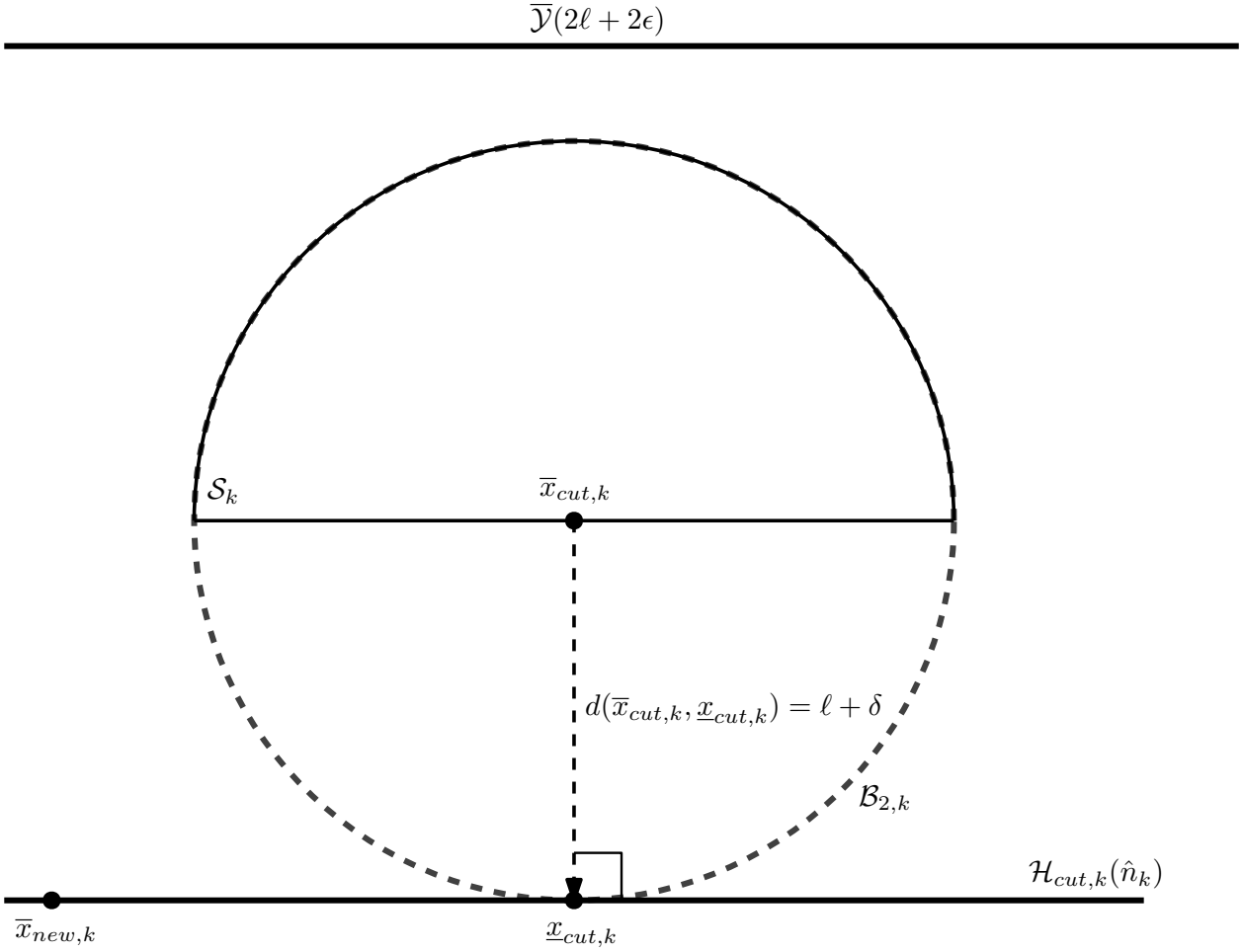


Figure 3.5: Illustration of the semi-hypersphere $\mathcal{S}_k$ and the hypersphere $\mathcal{B}_{2,k}$ at iteration k of the algorithm. Note that all $\overline{x}_{new,k} \in \overline{\mathcal{X}}_{new,k}$ lie outside of $\mathcal{B}_{2,k}$.

where $\overline{\mathcal{Y}}(z)$ is a convex, compact set and $\overline{\mathcal{X}}_k \subseteq \overline{\mathcal{Y}}\left(d(\overline{\mathcal{X}}_k, \mathcal{X})\right)$.

Thoerem 9. $\lim_{k\to\infty} d(\overline{\mathcal{X}}_k, \mathcal{X}) = 0$ *for any compact, convex, and full-dimensional* $\mathcal{X} \subset \mathbb{R}^n$.

Proof. We restrict our analysis to the monotonically decreasing subsequence that satisfies $d(\overline{\mathcal{X}}_{k+1}, \mathcal{X}) < d(\overline{\mathcal{X}}_k, \mathcal{X})$ from Corollary 6, and proceed by contradiction. Suppose $\lim_{k\to\infty} d(\overline{\mathcal{X}}_k, \mathcal{X}) = \ell > 0$, then $\forall\, k$, $\exists\ \overline{x}_{cut,k} \in \mathcal{V}(\overline{\mathcal{X}}_k)$ such that $d(\overline{x}_{cut,k}, \mathcal{X}) > \ell$. Further, since ℓ is the limit distance of the sequence, for each $\epsilon > 0$, $\exists\ \bar{k} \in \mathbb{N}$ such that $d(\overline{x}_{cut,k}, \mathcal{X}) < \ell + \epsilon$ for all $k \geq \bar{k}$. Consider any such $k > \bar{k}$ with corresponding $\overline{x}_{cut,k}$ such

that $d(\overline{x}_{cut,k}, \mathcal{X}) = \ell + \delta$ for some $\delta < \epsilon$ (clearly $\overline{\mathcal{X}}_k \subset \overline{\mathcal{Y}}\,(2\ell + 2\epsilon)$). Then, from Lemma 8, $\exists\, \mathcal{S}_k \subset \overline{\mathcal{Y}}\,(2\ell + 2\epsilon)$ such that $\mathcal{S}_k \cap \mathcal{S}_j = \emptyset$, $\forall\, j \neq k$ where $vol_n(\mathcal{S}_k) > \frac{1}{2} vol_n(\mathcal{B}_2(0, \ell)) > 0$. Since each cut removes $\mathcal{S}_k$ from $\overline{\mathcal{Y}}\,(2\ell + 2\epsilon)$, and infinitely many cuts are made, we can conclude that $\overline{\mathcal{Y}}\,(2\ell + 2\epsilon)$ has infinite volume in $\mathbb{R}^n$. However, $\overline{\mathcal{Y}}\,(2\ell + 2\epsilon)$ is a convex, compact set and therefore has finite volume. We have reached a contradiction; thus, $\lim_{k\to\infty} d(\overline{\mathcal{X}}_k, \mathcal{X}) = 0$. $\square$

3.5.5 3–Dimensional Illustrative Example with Convergence Guarantee

The same PDG problem that was solved in Section 3.4.4 is also solved here, this time with the algorithm described in this section. As in Section 3.4.4, the iterations are terminated after the 50th iteration. Both $\underline{\mathcal{X}}_k$ and $\overline{\mathcal{X}}_k$ are very similar to those in Figure 3.2 during the convergence process, so a similar figure is not repeated here. As expected, the Hausdorff distance between $\overline{\mathcal{X}}_k$ and $\mathcal{X}$ decreases monotonically as the algorithm iterates.

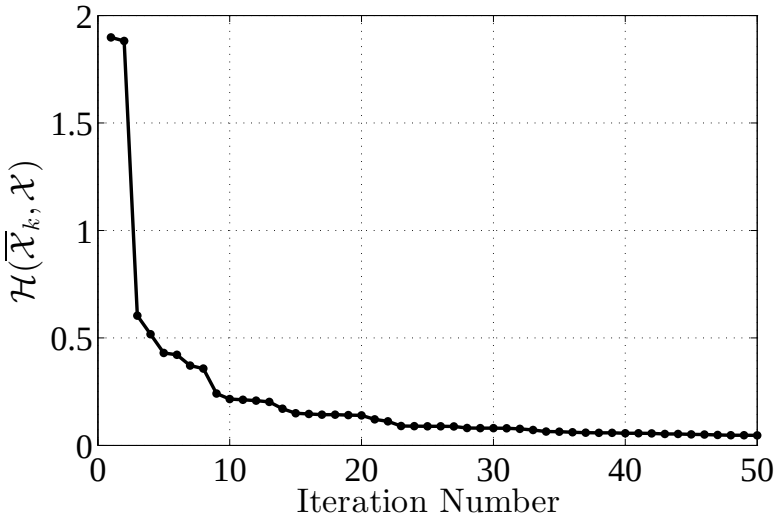


Figure 3.6: Convergence of $\overline{\mathcal{X}}_k$ to $\mathcal{X}$ via the algorithm in Section 3.5.3.

The volumes associated with $\underline{\mathcal{X}}_k$ and $\overline{\mathcal{X}}_k$ are also converging as shown in Figure 3.7. Since every iteration of the convergent subsequence shown to exist in Corollary 6 removes a finite and lower-bounded amount volume, the volumes of the inner and outer approximation

must converge.

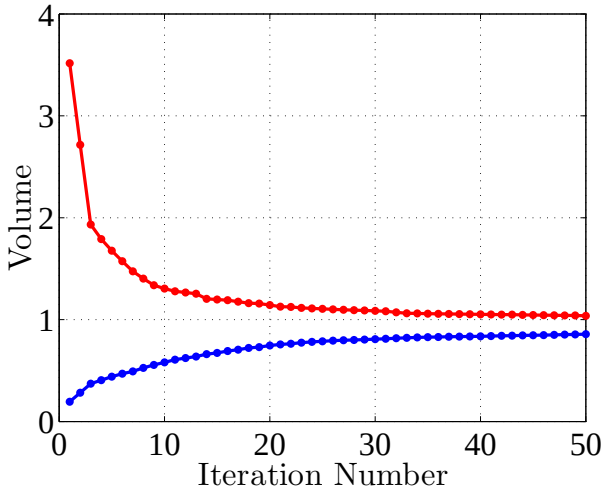


Figure 3.7: Volumetric convergence of $\overline{\mathcal{X}}_k$ (red) to $\underline{\mathcal{X}}_k$ (blue) via the algorithm in Section 3.5.3.

The optimization runtimes necessary for computing $\underline{\mathcal{X}}_k$ and $\overline{\mathcal{X}}_k$ are listed in Table 3.3. For the algorithm in Section 3.4, each Hausdorff call computes a point-to-set distance evaluated between a vertex of $\overline{\mathcal{X}}_k$ and the set $\underline{\mathcal{X}}_k$. On the other hand, Hausdorff calls in this algorithm compute the point-to-set distance between a vertex of $\overline{\mathcal{X}}_k$ and $\mathcal{X}$ itself. Unlike $\underline{\mathcal{X}}_k$, $\mathcal{X}$ need not be a polytope - a fact that is likely to make each call to the Hausdorff function more expensive. Indeed, the PDG formulation used in this example contains multiple (expensive) SOCP constraints, as we see via the increased Hausdorff runtime in Table 3.3 (up from 38 ms for the algorithm in Section 3.4). However, the runtime for each Hausdorff call is constant since the complexity of $\mathcal{X}$ does not change, unlike $\underline{\mathcal{X}}_k$ whose complexity grows as more vertices are added.

Computation Type	Runtime (ms)
Initialization $\times 2$	22
Hausdorff $\times 250$	1,481

Table 3.3: Total optimization runtimes for $n = 3$

The runtimes presented above were obtained by running the *generic* version of the solver described in Chapter 2. One advantage of computing the Hausdorff distance with respect to $\mathcal{X}$ itself is that a single custom solver can be generated for the task since the problem structure does not change from iteration to iteration. For the first algorithm presented in this chapter, the optimization to compute the Hausdorff distance is different with every iteration since $\underline{\mathcal{X}}_k$ is modified as part of the iteration step, so one would have to generate a family of custom solvers to compute the Hausdorff distances. With runtimes on the order of a second, both of the algorithms developed in this chapter are amenable to real-time computation by autonomous agents.

3.5.6 Mission Design Example

As a last example, we present results to determine the dependence and sensitivity of CRC sets to design parameters, including time of flight. The sort of CRC sets generated here are representative of the anaylses that are carried out during trade studies in the mission design phase of a vehicle (in this case, a vehicle capable of powered descent). Both algorithms discussed in this chapter can be used to generate the CRC sets presented in this section in about a second. Details on the meaning of the parameters can be found in Chapter 4 and [1]. The analysis is carried out for 3-dimensional controllability sets. First, we set the time of flight to 50 seconds and individually vary a parameter over an interval to observe its effect on the volume and shape of the CRC set. Then, we fix the parameters and vary the time of flight over an interval in order to expose any time of flight dependence.

Figure 3.8 presents the controllability sets with respect to different upper thrust bounds, initial vertical descent velocities, dry mass values, and thrust pointing constraint angles. As long as fuel is not the limiting factor, increasing the upper thrust bound enlarges the controllability set in every direction. Higher upper thrust bounds translate to more control authority, e.g., more lateral coverage, more tolerance to high initial potential energy, and more capability to nullify the initial velocity over a short distance. The analyses reveal that the initial vertical descent velocity does not have a major impact on the controllability set,

and the sensitivity of the controllability set to the vertical descent velocity is considerably lower below 130 m/s (which is the upper velocity bound of the vehicle in this example). The dry mass value (which can be translated to fuel mass because of a fixed wet mass) has an interesting effect on the shape of the controllability sets. From 1600 to 1680 kg of dry mass, the controllability sets are identical; however, after 1680 kg of dry mass, the controllability sets start to shrink to a cylindrical shape. As a final observation, increasing the thrust pointing constraint angle gives tbe vehicle more control authority in the lateral direction; therefore, as long as fuel is not a concern, it enlarges the set in the lateral directions with constraint-like (as the maximum angle for the thrust pointing constraint) inclinations on the sides.

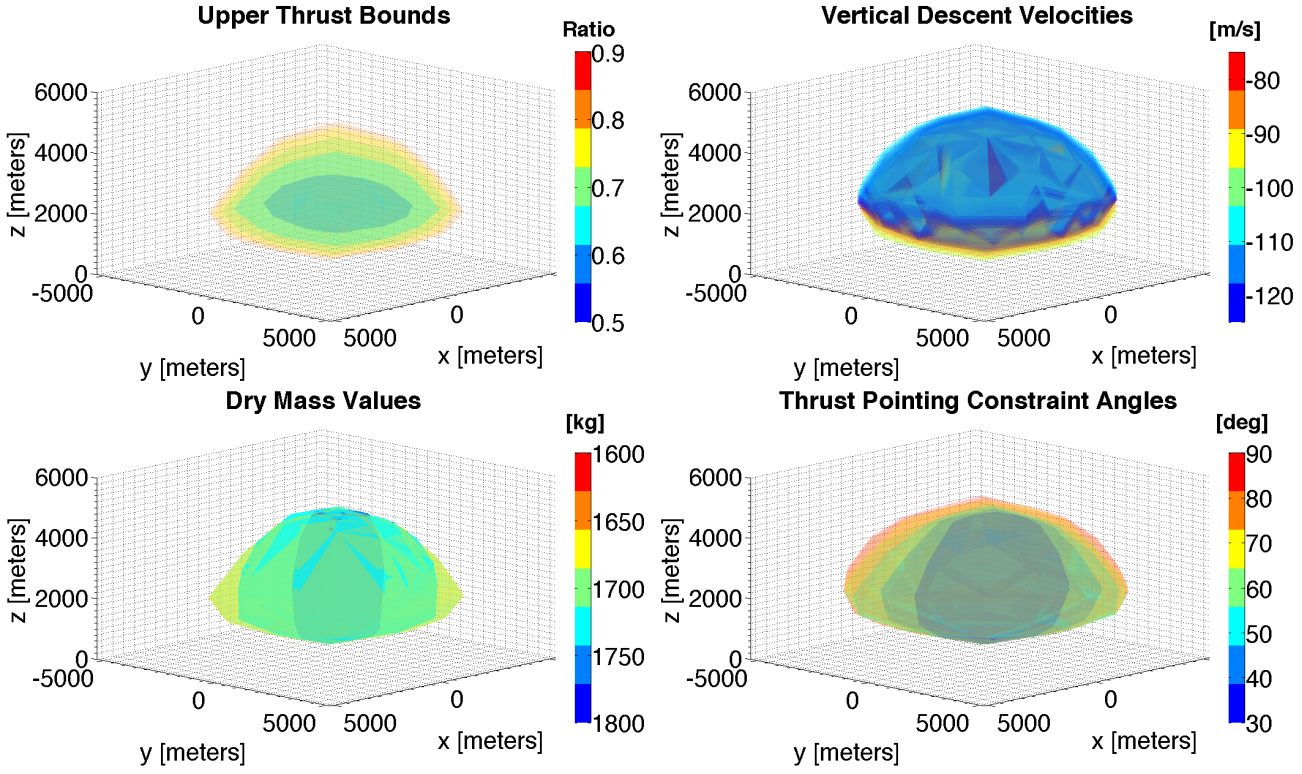


Figure 3.8: Controllability sets with varying: upper thrust bounds (top-left), downward velocity values (top-right), dry mass values (bottom-left) and maximum deviation of thrust pointing constraint (bottom-right).

Up until this point, we set the time of flight to 50 seconds. Now, we sample different time of flight values over an interval and observe how time of flight affects set propagations with fixed design parameters. All cases exhibit the same growth characteristics as time of flight is increased, except the case with a dry mass value of 1680 kg (Figure 3.9). For the dry mass case, increasing the time of flight caused the same shrinking characteristics that were caused by increasing dry mass values in the previous set of analyses. With a time of flight of 72 seconds, the amount of fuel (320 kg) onboard starts to be insufficient to cover the requisite lateral distances; therefore, the shape of the controllability set becomes more

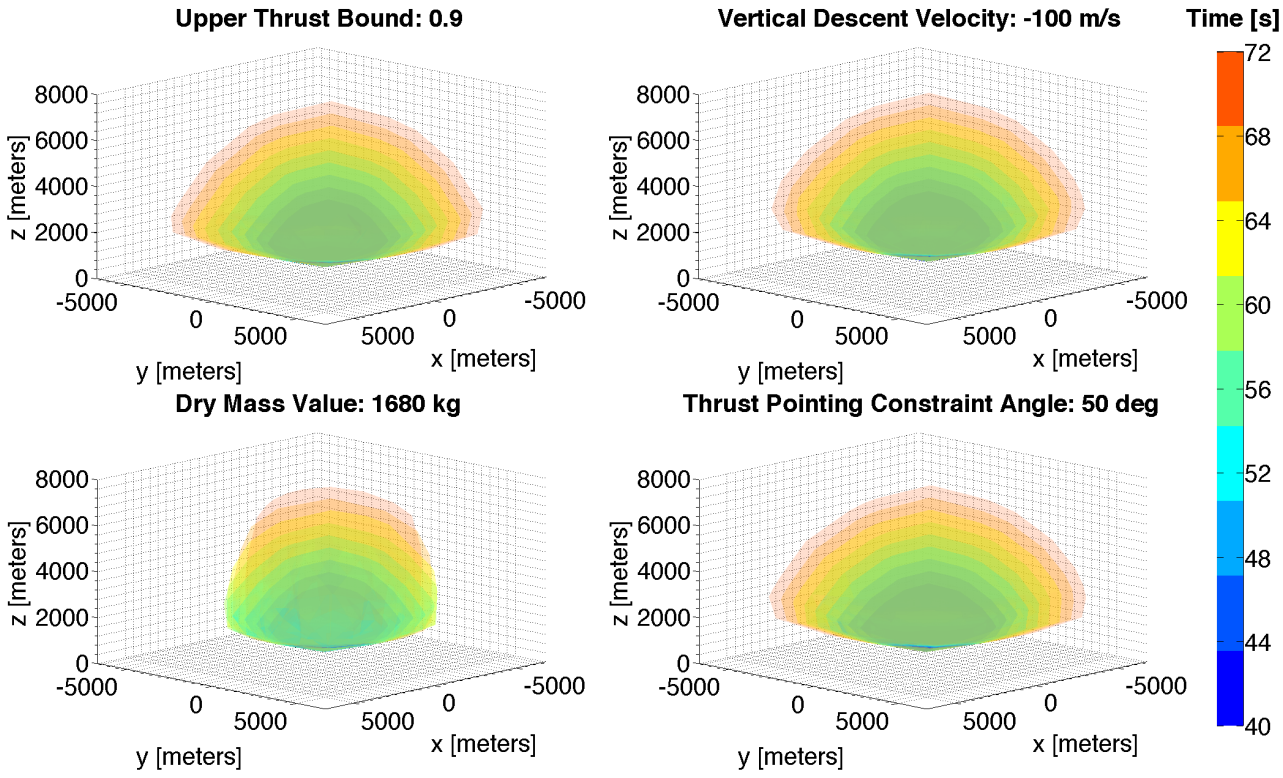


Figure 3.9: Four sets of analyses that employ 9 different time of flight values to check the sensitivities of the controllability sets to time of flight: the changes in the volume of the controllability sets with upper thrust bound: 0.9 (top-left), downward velocity value: -100 m/s (top-right), dry mass value: 1680 kg (bottom-left) and maximum deviation of thrust pointing constraint: 50 deg (bottom-right).

cylindrical and grows in the vertical direction.

In order to analyze the changes in the CRC set volumes more closely, we combine results from the parameter variations and time of flight variations to create Figure 3.10. The first observation in all cases is that as long as the fuel constraint is not active, the CRC set volume increases with increasing time of flight. We vary the thrust upper bound and the time of flight first. A higher thrust upper bound means more control authority and as long as fuel is not a concern; therefore, increasing the thrust upper bound increases the volume of the controllability set. To analyze the effects of the vertical descent velocity, we consider eleven different initial vertical velocity values (with a 20 m/s lateral velocity). Although the difference in the volumes is small for the variations over the time of time of flight interval, the CRC volume dependence follows a trend. As one can see from the magnified frame in Figure 3.10, the volume of the controllability set reaches a maximum at a vertical descent velocity of 110 m/s. So, the volume increases until this value, then decreases with increasing downward velocity for a fixed time of flight in this interval. Also, a close examination of the magnified frame reveals that the -125 m/s line crosses the -95 m/s line, and that the -120 m/s line crosses the -100 m/s line; this means that the set volume reaches a maximum at some time of flight value and then it decreases again for each vertical descent velocity. Dry mass values are sampled between a 1600-1800 kg interval, which corresponds to 400-200 kg of fuel. The results show a strong sensitivity to dry mass variations since the fuel causes major limitations in the vehicle's control authority. The 1600 kg dry mass case shows a consistent volume increase with increasing time of flight, and it presents an upper bound for all other cases (since it has the most available fuel). However, beyond 1600 kg of dry mass, the volumes of the controllability set start to decrease with increasing time of flight since the fuel constraint becomes active. For the 1800 kg dry mass case, there is no feasible controllability set for $t_f > 40$ seconds. In the thrust pointing constraint case, more deviation from the vertical direction means more lateral control capability; hence, increasing the thrust pointing constraint angle results in wider CRC sets and therefore larger CRC set volumes.

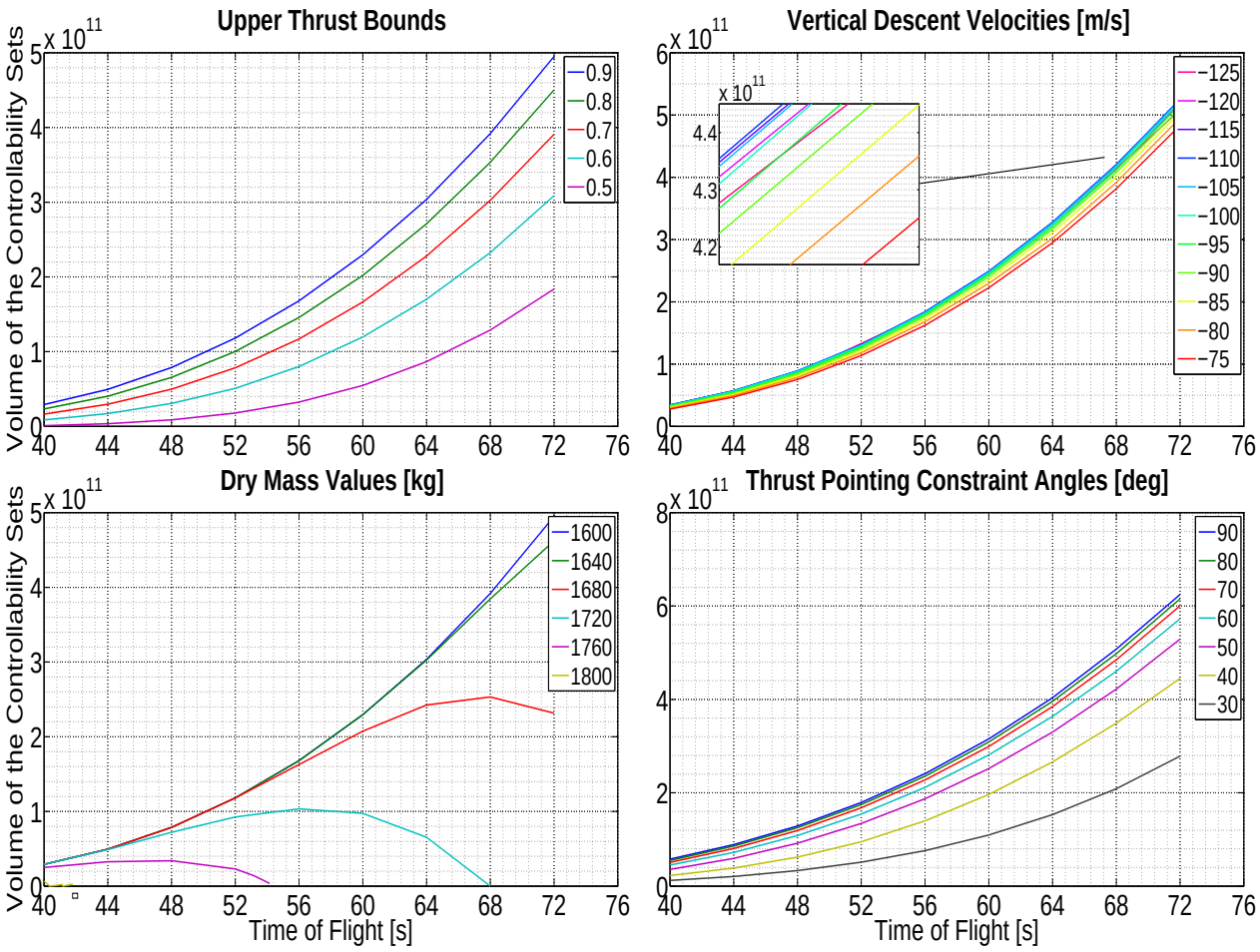


Figure 3.10: Four analyses using 9 different time of flight values to determine the sensitivities of the controllability sets to design parameters and initial conditions: changes in the volume of controllability sets with upper thrust bounds (top-left), downward velocity values (top-right), dry mass values (bottom-left) and maximum deviation of thrust pointing constraint (bottom-right).

Chapter 4

REAL-TIME OPTIMIZATION IN OPTIMAL CONTROL

As mentioned in Chapter 1, analytic solutions to trajectory optimization and control problems for PDG have not been found since the work by Meditch in 1964 for 1-D motion [18]. As a consequence, the only feasible path towards generic real-time trajectory optimization with realistic constraints is currently through numerical solvers. Casting optimal control problems into convex optimization problems enables the use of IPMs, which can be used to generate custom solvers, have excellent convergence properties, and compute solutions with polynomial time complexity, as described in Chapter 2. On an agent equipped with high-level autonomy based on the work in Chapter 3, a desired final state would be passed to a real-time trajectory optimization algorithm like the ones detailed in this chapter; the goal is then to generate a feasible trajectory that is optimal with respect to some objective function and pass this trajectory forward to the vehicle's closed loop control.

For the remainder of the chapter, two relevant aerospace applications are used to illustrate the potential for real-time trajectory optimization: PDG and Unmanned Aerial Vehicle (UAV) guidance in the presence of obstacles. For PDG, the primary objective is to cast its inherently non-convex constraints as convex ones either through lossless convexification or suitably accurate linearizations (see Section 4.1 for details). This chapter's first contribution deals with an extension of previous lossless convexification results for the type of constraints that were encountered during the G-FOLD flight testing campaign. With respect to the second example, the non-convex constraints introduced by avoiding obstacles cannot be handled with lossless convexification or linearization alone, so a novel approach called successive convexification is used [79, 80]. The chapter's second contribution is to leverage the successive convexification framework to avoid collisions with obstacles along the continuous arc from

the initial state to the final state with finitely many constraints.

Earlier convexification results [11–14, 16] do not ensure that convexification is lossless for trajectories with simultaneously active thrust pointing [13] and velocity upper bound constraints. Active state constraints, such as velocity bounds, complicate the form of Pontryagin's Maximum principle [81–83] used in proofs, thus making it difficult to establish a very general convexification theory with active state constraints. In [11, 12], the first lossless convexification results are established without incorporating thrust pointing constraints. These results are generalized to the case with thrust pointing constraints in [13]. In these results, the convexification is shown to be lossless for optimal trajectories without active state constraints or with momentarily active state constraints (on a finite number of time instances). In [14], these results are extended to optimal trajectories with active velocity constraints on finite time intervals. Finally, the theory is extended to allow for continuously active polytopic state constraints in [16]. Therefore, this chapter presents an extension of lossless convexification for trajectories with simultaneously active velocity bound constraints and thrust pointing constraints. This extension is important because the results for the test flights in 2013 [28, 84] make use of trajectories that are simultaneously limited by speed and thrust direction. More specifically, the test rocket was at its maximum velocity for more than 75% of its descent [50]. Thus, the theory presented herein is necessary to guarantee that the optimal solutions to the convexified PDG problem demonstrated in the test flights are also optimal for the original, non-convex PDG problem.

This chapter is adapted from [36, 39] and begins by summarizing the PDG formulation: starting with its natural, non-convex form, progressing towards a fully convexified version, and finally explaining the process for converting the convex optimization problem into one that can be used as an input to actual IPM solvers. Then, an extension to the theory of lossless convexification is made that supports the real-life trajectory of active constraints followed by the Xombie rocket during the G-FOLD flight test campaign. The chapter concludes with a real-time method for ensuring autonomous agents do not enter pre-defined keep out zones during the continuous arc from the initial state to the final state by using successive

convexification and finitely many constraints.

4.1 Fuel-Optimal PDG for Planetary Pinpoint Landing

This section summarizes the formulation of the optimal control problem for fuel-optimal PDG along with earlier convexification results [11–14]. The section concludes with the discretization of the convex optimal control problem, which produces a finite-dimensional SOCP problem [2]. This material is needed for the extended convexification results with active constraints.

4.1.1 Review of the Fuel-Optimal PDG Problem Formulation

The PDG problem is a finite-time horizon optimal control problem where fuel-optimal trajectories are sought that satisfy the relevant state and control constraints imposed by the vehicle capabilities and mission requirements. We assume that: (i) the only forces acting on the vehicle are due to the control thrust and gravity (i.e., there are negligible aerodynamic forces - particularly true on Mars due to its low atmospheric density), (ii) the vehicle is sufficiently close to the surface to warrant a flat planet model, where the acceleration due to gravity is constant, and (iii) the vehicle's attitude control is of sufficiently high bandwidth that the translational and rotational dynamics can be decoupled. As a consequence of the last assumption, we do not include the attitude dynamics into the formulation, and assume that any lander orientation needed to obtain the desired thrust vector can be obtained instantaneously. The translational equations of motion under these assumptions are [11],

$$\begin{aligned} \dot{r}(t) &= v(t), \\ \dot{v}(t) &= \frac{T(t)}{m(t)} - S(\omega)^2 r - 2S(\omega)v - g, \\ \dot{m}(t) &= -\alpha\|T(t)\|, \end{aligned} \tag{4.1}$$

where $r \in \mathbb{R}^3$ is the position of the vehicle relative to some planet-fixed coordinate frame, $v \in \mathbb{R}^3$ is the corresponding velocity, $m \in \mathbb{R}_+$ represents the vehicle's mass, $T \in \mathbb{R}^3$ is the thrust force, $g \in \mathbb{R}^3$ corresponds to gravitational acceleration, $\alpha \in \mathbb{R}_+$ is a positive constant that describes the mass flow rate, ω is the planet's rotational rate vector, and $S(\omega)$ is a skew symmetric matrix describing the cross-product operation. The components of r are defined as follows: r_1 corresponds to the downrange, r_2 corresponds to cross-range, and r_3 is the altitude. All of the initial conditions for the system are specified and denoted with a subscript zero. The vehicle is required to land softly on the surface; thus, the final positions and velocities are specified (without loss of generality) and denoted with a subscript f:

$$\begin{aligned} &r(t_0) = r_0, \qquad v(t_0) = v_0, \qquad m(t_0) = m_0, \\ &r_3(t_f) = 0, \qquad v(t_f) = 0. \end{aligned} \tag{4.2}$$

Thus, the final range, cross range, and mass are free; however, the final mass cannot be less than the dry mass, m_d, of the vehicle, i.e., $m(t_f) \geq m_d$. There are a number of additional constraints that any feasible trajectory must satisfy. A velocity upper bound constraint keeps the vehicle's thrusters operating in a safe (subsonic) regime. Given a velocity upper bound, V_c, the speed constraint is given by:

$$\|v(t)\| \leq V_c. \tag{4.3}$$

Second, the glide slope constraint keeps the vehicle in an inverted cone with its tip at the landing point [12, 13]:

$$\tan\gamma \|(r_1(t), r_2(t))\| \leq r_3(t), \tag{4.4}$$

where $\gamma \in [0, 90^\circ)$ is the minimum admissible glide slope angle.

The control constraints define the feasible thrust regime via an upper and a lower bound. A lower bound is needed because engines typically cannot operate reliably below a certain

throttle level. The upper bound exists because arbitrarily large thrusts are impossible.

$$0 < \rho_1 \leq \|T(t)\| \leq \rho_2. \tag{4.5}$$

This constraint is not convex due to the lower bound constraint (see Figure 4.1).

Finally, we consider the thrust pointing constraint given by:

$$\hat{n}^T T(t) \geq \|T(t)\| \cos\theta, \tag{4.6}$$

where $\hat{n} \in \mathbb{R}^3$ is a unit vector describing a desired pointing direction and $\theta \in [0, \pi]$ is a specified angle that represents the size of the feasible thrust pointing arc about $\hat{n}$. This constraint is convex when $\theta \leq \pi/2$ and non-convex when $\theta > \pi/2$. The unit vector and angle define a cone in which the thrust vector must reside. Such a constraint arises due to sensors (such as a camera) that must keep the ground in its field of view.

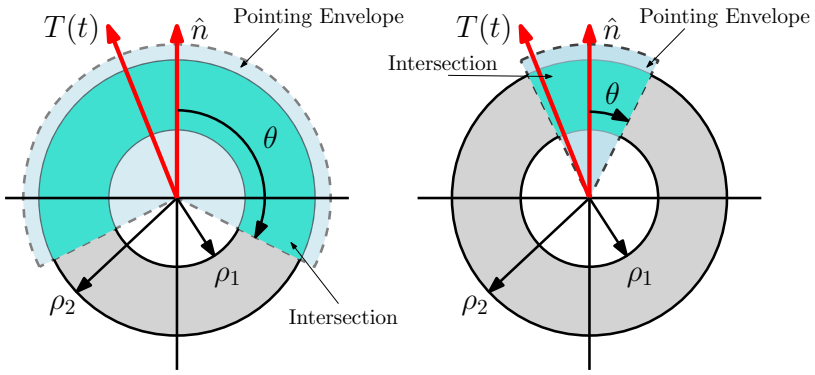


Figure 4.1: Feasible set of controls with pointing constraints and upper/lower norm bounds for a two dimensional control vector.

Obtaining a fuel-optimal trajectory for the PDG problem is equivalent to maximizing the mass of the vehicle at landing. Therefore, the fuel-optimal PDG problem is given by:

Problem 1. Non-Convex Fuel-Optimal PDG Problem

$$
\begin{aligned}
\max \quad & J = m(t_f) \\
\text{subj. to} \quad & \dot{r}(t) = v(t),\ r(t_0) = r_0,\ r_3(t_f) = 0 \\
& \dot{v}(t) = \frac{T(t)}{m(t)} - S(\omega)^2 r(t) - 2S(\omega)v(t) - g, \quad v(t_0) = v_0,\ v(t_f) = 0 \\
& \dot{m}(t) = -\alpha \|T(t)\|, \quad m(t_0) = m_0, \quad m_d \leq m(t_f) \\
& 0 < \rho_1 \leq \|T(t)\| \leq \rho_2, \quad \hat{n}^T T(t) \geq \|T(t)\| \cos\theta \\
& \|v(t)\| \leq V_c, \quad \tan\gamma \| \left(r_1(t), r_2(t)\right) \| \leq r_3(t).
\end{aligned}
\tag{4.7}
$$

4.1.2 Lossless Convexification of the Fuel-Optimal PDG Problem

As stated, Problem 1 is highly constrained and non-convex. There are three sources of non-convexity in the formulation above: (*i*) the lower bound on the thrust magnitude, (*ii*) the non-linear dynamics caused by the interaction between thrust and mass, and (*iii*) the thrust pointing constraint (if $\theta > \pi/2$). In general, such problems do not have closed form solutions and must be solved numerically. Because convex problems can be solved to global optimality using polynomial-time IPM algorithms, converting the problem to a convex form offers significant computational advantages, which motivate the following discussion on the convexification of the problem.

First, consider the non-convex control domain. Applying the convexification in [11, 12], the convexified PDG problem is presented in Problem 2 below, where Γ is a new slack variable that relaxes the control constraints by lifting the control space.

Problem 2. Fuel-Optimal PDG Problem with Losslessy Convexified Control Constraints

$$
\begin{aligned}
\max \quad & J = m(t_f) \\
\text{subj. to} \quad & \dot{r}(t) = v(t),\ r(t_0) = r_0,\ r_3(t_f) = 0 \\
& \dot{v}(t) = \frac{T(t)}{m(t)} - S(\omega)^2 r(t) - 2S(\omega)v(t) - g, \quad v(t_0) = v_0,\ v(t_f) = 0 \\
& \dot{m}(t) = -\alpha\Gamma(t), \quad m(t_0) = m_0, \quad m_d \leq m(t_f) \\
& \hat{n}^T T(t) \geq \Gamma\cos\theta, \quad \|T(t)\| \leq \Gamma(t), \quad 0 < \rho_1 \leq \Gamma(t) \leq \rho_2 \\
& \|v(t)\| \leq V_c, \quad \tan\gamma \| \left(r_1(t), r_2(t)\right) \| \leq r_3(t).
\end{aligned}
\tag{4.8}
$$

The objective of lossless convexification of the control constraints is to establish that solutions to the relaxed optimal control problem in Problem 2 are also solutions to the original formulation in Problem 1 – in other words, the above control relaxation has no impact on optimal solutions. Section 4.2 focuses on showing that this control relaxation remains lossless for simultaneously active maximum speed and thrust pointing constraints for trajectories that are representative of the Masten test flights.

4.1.3 Time-varying Mass

Two non-convexities remain in the formulation of Problem 2: the non-linear interaction between thrust and mass in the dynamics, and the non-convex thrust pointing constraint when $\theta \geq \pi/2$. While the non-linear mass dynamics are not problematic for the proof in Section 4.2, the dynamics must still be convexified to compute solutions by using convex programming methods. First a change of variables is performed, then the control constraints are approximated with a Taylor series. A very brief overview of this procedure is given here to present a purely convex trajectory optimization problem, with details given in Section IIIB of [11]. First, let:

$$
\sigma(t) \triangleq \frac{\Gamma(t)}{m(t)}, \qquad u(t) \triangleq \frac{T(t)}{m(t)}, \qquad z(t) \triangleq \ln m(t).
$$

Then, the mass dynamics become:

$$\dot{z}(t) = \frac{\dot{m}(t)}{m(t)} = -\alpha\sigma(t),$$

which results in a linear equality constraint. Up to this point, all operations have resulted in lossless convexifications. However, the upper and lower bound constraints on Γ in (4.5) become non-convex due to this choice of variables. Thus, a Taylor series approximation is applied to convexify these constraints. The resulting convex control constraints in the transformed variables are:

$$\left.\begin{aligned} &\|u(t)\| \leq \sigma(t),\ \hat{n}^T u(t) \geq \sigma(t)\cos\theta \\ &\rho_1 e^{-z_0}\big[1-(z(t)-z_0(t))\big] \leq \sigma(t) \\ &\sigma(t) \leq \rho_2 e^{-z_0}\big[1-(z(t)-z_0(t))\big] \end{aligned}\right\} \forall t \in [0, t_f],$$

where $z_0(t) = \ln(m_0 - \alpha\rho_2 t)$. An analysis of the error incurred through this approximation can be found in [50]. Combining all of the convexifications introduced, the fuel-optimal PDG problem becomes the following convex optimization problem:

Problem 3. Convexified Fuel-Optimal PDG Problem

$$\begin{aligned} \max \quad & J = z(t_f) \\ \text{subj. to} \quad & \dot{r}(t) = v(t),\ r(t_0) = r_0,\ r_3(t_f) = 0 \\ & \dot{v}(t) = u(t) - S(\omega)^2 r(t) - 2S(\omega)v(t) - g, \quad v(t_0) = v_0,\ v(t_f) = 0 \\ & \dot{z}(t) = -\alpha\sigma(t), \qquad z(t_0) = \ln(m_0), \qquad \ln(m_d) \leq z(t_f), \\ & \hat{n}^T u(t) \geq \sigma\cos\theta, \quad \|u(t)\| \leq \sigma(t) \\ & \rho_1 e^{-z_0(t)}\left[1-(z(t)-z_0(t))\right] \leq \sigma(t) \leq \rho_2 e^{-z_0(t)}\left[1-(z(t)-z_0(t))\right] \\ & \|v(t)\| \leq V_c, \quad \tan\gamma \| \left(r_1(t), r_2(t)\right) \| \leq r_3(t). \end{aligned} \tag{4.9}$$

Note that Problem 3, which is an optimal control problem, is solvable by convex programming methods after a discretization to convert it into a parameter optimization problem.

4.1.4 Canonicalization of the PDG Problem

In this section, Problem 3 is transformed into the standard form of a SOCP [2, 6], which is required to obtain a solution via IPMs. Problem 3 is first discretized to convert it into a finite-dimensional convex parameter optimization problem. The process involves a time discretization of the thrust (control) vector [11, 12]. Note that regardless of the type of time discretization employed, the resulting system will have linear discrete-time dynamics with linear inequality and convex quadratic inequality constraints.

Let the state vector, $\xi \in \mathbb{R}^7$, be:

$$\xi = \begin{bmatrix} z(t) \\ r(t) \\ v(t) \end{bmatrix},$$

where p is the number of time steps and Δt is the time step size. Then, the discretized dynamics in Problem 3 can be expressed as:

$$\chi_{k+1} = F\xi_k + Gu_k, \quad k = 1, \ldots, p-1,$$

where $F \in \mathbb{R}^{7\times 7}$ and $G \in \mathbb{R}^{7\times 3}$ and their explicit construction is quite straight forward and can be found in [11]. Thus, the dynamics form a series of $p-1$ linear equality constraints. Next, the feasible domains of the $p-1$ unknown states and $p-1$ unknown controls are expressed via two types of convex cones: the positive (or linear) cone, $\mathcal{K}_+$, and the second order cone, $\mathcal{K}_s$ [2, 6, 25]. All of the solution variables involved are constrained to be in one of these two types of cones, sometimes by introducing slack variables.

Definition 10. *A positive cone, $\mathcal{K}_+^n \subset \mathbb{R}^n$, is a convex set defined by:*

$$\mathcal{K}_+^n = \{w \in \mathbb{R}^n : w \geq 0\} \subseteq \mathbb{R}_+^n,$$

where the $\geq$ above refers to an element-wise inequality. The positive cone is used to de-

scribe all variables that are either unconstrained or linearly constrained and that are not quadratically constrained. For example if the variable w is unconstrained, it is replaced by two variables in the positive cone, $w = w_+ - w_-$ where $w_+ \geq 0$ and $w_- \geq 0$. If the variable is linearly constrained, i.e., $h^T w \leq d$, then we let $w = w_+ - w_-$ as before, and

$$h^T(w_+ - w_-) + w_s = d, \quad \text{where } \; w_+ \geq 0, \; w_- \geq 0, \; w_s \geq 0.$$

Definition 11. *A second order cone, $\mathcal{K}_S \subset \mathbb{R}^{n+1}$, formed by an n dimensional vector, q, and a non-negative scalar, w, is a convex set defined by:*

$$\mathcal{K}_S = \{(w, q) : \|q\| \leq w, \, q \in \mathbb{R}^n, w \in \mathbb{R}_+\}. \tag{4.10}$$

Any convex quadratic inequality or norm inequality constraint can be expressed as a variable in a second order cone, potentially by adding a slack variable similar to w. For example, expressing the maximum speed constraint is done as follows. First, a new slack variable, v_s, is prepended to the velocity vector to form a second order cone:

$$\begin{bmatrix} v_s \\ v \end{bmatrix} \in \mathcal{K}_S,$$

ans so $v_s \geq \|v\|$ by (4.10). Then, the maximum speed constraint is enforced by imposing:

$$v_s \leq V_c.$$

The inequality constraint is converted into an equality constraint by making use of one additional slack variable, $w_s \geq 0$:

$$v_s + w_s = V_c.$$

After assigning each PDG variable and necessary slack variable to a positive cone or second order cone, these variables are collected into a global solution variable, x. Finally, a straight-

forward, but tedious process produces the A matrix along with the b and c vectors such that Problem 3 is expressed in standard form as in (2.1) [2, 6].

4.2 Maximum-Divert Powered Descent Guidance

In this section, previous convexification results are extended to show that the control convexification presented in Section 4.1.2 is lossless when both the state constraints and the thrust pointing constraints are simultaneously active. The *primary* objective is to theoretically validate the control convexification used to obtain the trajectories that were flown in 2012-2014.

First, a theoretical extension of the PDG algorithm is presented. The need for this extension was encountered during the terrestrial test flights of the customized PDG algorithm [85]. Specifically, the speed of the test rocket in lateral flight was limited to 90 km/hr. When diverting a kilometer, the rocket would reach this speed limit and then remain at it, even for segments with active thrust pointing constraints. That is, the speed and thrust pointing constraints were simultaneously active. Previous results for lossless control convexification of the constrained PDG problem [14, 16] did not include simultaneously active thrust pointing and speed constraints. Further, the thrust-pointing constraint can also be continuously active. Hence, the lossless convexification results for the PDG problem, which theoretically prove that the solution of the convexified problem is also an optimal solution for the original non-convex PDG problem, must be extended to include continuously and simultaneously active thrust-pointing and speed constraints.

Second, near term planetary exploration missions may leverage existing landing systems, thereby limiting the amount of propellant available for diverts. For so-called Multi-X landing strategies [50], in which the divert goal is not to reach the center of a landing ellipse but simply to escape large, known hazards in the landing ellipse, it is beneficial to determine the largest divert possible given a fixed amount of propellant [14].

As a result, the extension of lossless convexification for the PDG problem with simultaneously and continuously active speed and thrust-pointing constraints is presented for the

maximum-divert PDG problem. A history of previous convexification results is presented in Table 4.1 below, with the result presented in this chapter concluding the table. Posing Problem 4 as a maximum divert PDG problem facilitates this extension and is valid for minimum fuel PDG problems with landing sites that require the entire fuel allocation to reach (i.e., landing sites that correspond to maximum diverts). This theoretical extension provides the analytical basis for the use of the convex PDG algorithm demonstrated in the terrestrial test flights [85] since the resulting trajectories use all of the onboard propellant to reach the maximum divert distance.

Year	Result
2007	Minimum-fuel control convexification without active state constraints or thrust pointing constraints [11].
2008	Minimum-landing-error control convexification with active state constraints for at most one instant and without thrust pointing constraints [12].
2013	Minimum-landing-error control convexification with thrust pointing constraints, but without active state constraints [13].
2013	Maximum divert control convexification with velocity constraints active over finite time intervals, but without thrust pointing constraints [14].
2014	Control convexification for general linear systems with active polytopic state constraints, but without thrust pointing constraints [16].
2015	Maximum divert control convexification with simultaneously active thrust pointing and velocity constraints (below).

Table 4.1: A History of Lossless Convexification Results

The convexified maximum divert problem is given next as Problem 4. The cost is changed with respect to Problem 2 to reflect the new objective. Also, two simplifications are made to the constraint set in Problem 4: i) the terms relating to planetary rotation are dropped

since both the time horizon and the initial altitude of the vehicle during the terrestrial tests are small; ii) the glide slope constraint is removed since the terrestrial trajectories flown by the test vehicle did not activate this constraint. Removing these constraints simplifies the theoretical analysis while still maintaining a constraint set that is representative of the terrestrial test trajectories. Indeed, having active state constraints complicates the convexification proofs (see [16]). This issue is further compounded by the fact that active thrust pointing constraints modify the feasible control space such that not all boundary points of the feasible control space are extremal points. In the maximum divert problem it is the velocity constraint that becomes active quite often together with the active control constraints, e.g., velocity bound and thrust pointing constraints. The subsequent proof establishes convexification for this case, which is the main theoretical contribution relevant to the tested landing maneuvers.

Problem 4. Convex Relaxation of the Controls in the Maximum Divert PDG Problem

$$
\begin{aligned}
\min \quad & J = -w_1 r_1(t_f) - w_2 r_2(t_f) \\
\text{subj. to} \quad & \dot{r}(t) = v(t), \qquad r(t_0) = r_0,\ r_3(t_f) = 0 \\
& \dot{v}(t) = T(t)/m(t) - g, \quad v(t_0) = v_0,\ v(t_f) = 0 \\
& \dot{m}(t) = -\alpha\Gamma(t), \qquad m(t_0) = m_0,\ m_d \leq m(t_f) \\
& \hat{n}^T T(t) \geq \Gamma\cos\theta, \quad \|v(t)\|^2 \leq V_c^2 \\
& \|T(t)\|^2 \leq \Gamma^2(t), \quad \rho_1 \leq \Gamma(t) \leq \rho_2
\end{aligned}
$$

As in Section 4.1.2, the goal is to show that any optimal solution of Problem 4 satisfies $\|T(t)\| = \Gamma(t)$, that is, that the convexification of the control constraints is lossless. In other words, optimal solutions of the convexified problem are optimal solutions of the original non-convex problem.

The proof relies on a maximum principle with state constraints as in [14, 81]. Note that, due to the increased complexity of the maximum principle, convexification results in the presence of active state constraints are harder to obtain. The maximum principle

with active state constraints states all of the necessary conditions for optimality, including the state/costate equations, pointwise maximum conditions, complementary slackness conditions, and transversality conditions. The Hamiltonian and the Lagrangian are:

$$\mathcal{H}[t] = p_r(t)^T v(t) + p_v(t)^T (T(t)/m(t) - g) - \alpha p_m(t)\Gamma(t), \tag{4.11}$$

$$\begin{aligned}\mathcal{L}[t] = \mathcal{H}[t] &+ \lambda_1(t)(\|T(t)\|^2 - \Gamma^2(t)) + \lambda_2(t)(\rho_1 - \Gamma(t)) + \lambda_3(t)(\Gamma - \rho_2) \\ &+ \lambda_4(\Gamma\cos\theta - n^T T) + \nu(t)(\|v(t)\|^2 - V_c^2),\end{aligned}$$

with costates p_r, p_v, and p_m and Lagrange multipliers λ_i and ν. The costate equations are:

$$\begin{aligned}\dot{p}_r(t) &= 0, \\ \dot{p}_v(t) &= -p_r(t) - 2\nu(t)v(t), \\ \dot{p}_m(t) &= p_v(t)^T T(t)/m^2(t).\end{aligned}$$

The transversality conditions are:

$$\begin{aligned}p_r(t_f) &= -e^{(1)} p_0 w_1 - e^{(2)} p_0 w_2 + e^{(3)} \xi_{r_3}, \\ p_v(t_f) &= \xi_v + 2\mu v(t_f), \\ p_m(t_f) &= -\zeta_m.\end{aligned}$$

The stationary conditions are:

$$\begin{aligned}\partial_T \mathcal{L}[t] &= p_v(t)/m(t) + 2\lambda_1(t)T(t) - \lambda_4 \hat{n} = 0, \\ \partial_\Gamma \mathcal{L}[t] &= -\alpha p_m(t) - 2\lambda_1(t)\Gamma(t) - \lambda_2(t) + \lambda_3(t) + \lambda_4(t)\cos\theta = 0.\end{aligned}$$

The pointwise maximum principle states the following:

$$\begin{aligned} \max \quad & p_v^T(t)T(t)/m(t) - \alpha p_m(t)\Gamma(t) \\ \text{subj. to} \quad & \rho_1 \leq \Gamma(t) \leq \rho_2, \quad \|T(t)\| \leq \Gamma(t), \quad \Gamma(t)\cos\theta \leq \hat{n}^T T(t). \end{aligned}$$

And lastly, since the final time is free and the Hamiltonian does not depend explicitly on time, it is identically zero along the optimal path,

$$\mathcal{H}[t] = 0 \ \ \forall\ t \in [0, t_f]. \tag{4.12}$$

Thoerem 10. *The thrust magnitude constraint holds with equality, i.e.,* $\|T(t)\| = \Gamma(t) \ \ \forall\, t$ *under the following three assumptions: (i) The angle* θ *defining the pointing cone is not* $\pi/2$*, i.e.,* $\cos\theta \neq 0$*. (ii) The unit vector,* $\hat{n}$*, defining the pointing cone is not orthogonal to the gravity vector* g*, i.e.,* $\hat{n} \not\perp g$*. (iii) The thrust magnitude constraints and vehicle weight satisfy the inequality* $\rho_1 \leq m(t)g \leq \rho_2$ *for all* t*.*

Proof. Proceeding by contradiction, assume there exists an interval $[t_1, t_2] \subset [t_0, t_f]$ where $\|T(t)\| < \Gamma(t)$ for all $t \in [t_1, t_2]$ under suitable restrictions on the control functions [14, 16]. The standard complementary slackness condition then implies that the multiplier $\lambda_1(t) = 0$ on the interval. The stationary conditions above then reduce to the following:

$$p_v(t)/m(t) = \lambda_4 \hat{n},$$

$$-\alpha p_m(t) - 2\lambda_1(t)\Gamma(t) - \lambda_2(t) + \lambda_3(t) + \lambda_4(t)\cos\theta = 0.$$

Note that if the pointing constraint is satisfied with strict inequality, then $\lambda_4(t) = 0$ and the proof in [14] is recovered. As such, it is assumed that the pointing constraint holds with equality throughout the remainder of the proof, that is:

$$\hat{n}^T T(t) = \Gamma(t)\cos\theta. \tag{4.13}$$

Then, using assumption (i) one obtains $\Gamma(t) = \hat{n}^T T(t)/\cos\theta$.

As a result, the pointwise maximum principle becomes,

$$\begin{aligned} \max \qquad & \left(p_v^T(t)/m(t) - \alpha p_m(t) \frac{\hat{n}^T}{\cos\theta} \right) T(t) \\ \text{subj. to} \quad & \rho_1 \leq \Gamma(t) \leq \rho_2, \quad \|T(t)\| \leq \Gamma(t). \end{aligned}$$

Upon defining the switching function,

$$\chi(t) = \frac{p_v^T(t)}{m(t)} - \alpha p_m(t) \frac{\hat{n}^T}{\cos\theta},$$

it is clear that $\|T(t)\| = \Gamma(t)$ whenever $\chi(t) \neq 0$ since the (maximizing) cost function drives optimal values of $T(t)$ to the $\|T(t)\| \leq \Gamma(t)$ constraint boundary – thus making it an equality. Thus assume $\chi(t) = 0$ on an interval which implies that the cost function equals zero. Solving for $p_v(t)$ when $\chi(t) = 0$ gives,

$$p_v(t) = \alpha m(t) p_m(t) \frac{\hat{n}}{\cos\theta},$$

along with:

$$\lambda_4(t) = \frac{\alpha p_m(t)}{\cos\theta} \quad \text{and} \quad \dot{\lambda}_4(t) = \frac{\alpha}{\cos\theta} \frac{p_v^T(t) T(t)}{m^2(t)}. \tag{4.14}$$

Differentiating p_v with respect to time gives,

$$\dot{p}_v(t) = \left(\dot{m}(t)\lambda_4(t) + m\dot{\lambda}_4(t) \right) \hat{n}. \tag{4.15}$$

This implies that $\dot{p}_v(t)$ is a vector along $\hat{n}$. Substituting (4.14) into (4.15) and taking the

norm yields,

$$\begin{aligned}
\|\dot{p}_v(t)\| &= \left|\dot{m}(t)\lambda_4(t) + m\dot{\lambda}_4(t)\right|, \\
&= \left|-\alpha^2\Gamma(t)\frac{p_m(t)}{\cos\theta} + \alpha\frac{p_v^T(t)T(t)}{\cos\theta m(t)}\right|, \\
&= \frac{\alpha}{|\cos\theta|}\left|-\alpha\Gamma(t)p_m(t) + \frac{p_v^T(t)T(t)}{m(t)}\right|. \qquad (4.16)
\end{aligned}$$

Next, note that $\chi(t) = 0 \implies \chi(t)T(t) = 0$, which combined with (4.13) implies that:

$$\begin{aligned}
\chi(t)T(t) &= \frac{p_v^T(t)T(t)}{m(t)} - \frac{\alpha p_m(t)\hat{n}^T T(t)}{\cos\theta}, \\
&= \frac{p_v^T(t)T(t)}{m(t)} - \frac{\alpha\Gamma(t)p_m(t)\cos\theta}{\cos\theta}, \\
&= \frac{p_v^T(t)T(t)}{m(t)} - \alpha\Gamma(t)p_m(t) = 0. \qquad (4.17)
\end{aligned}$$

Consequently, it is clear that $\|\dot{p}_v(t)\| = 0$ by substituting (4.17) into (4.16). Then, $\dot{p}_v(t) = 0$ and $\ddot{p}_v(t) = 0$ on the interval.

From [14], pre-multiply $\ddot{p}_v(t)$ with $\dot{v}(t)$, giving,

$$\begin{aligned}
\dot{v}^T(t)\ddot{p}_v(t) &= \dot{v}^T(t)(-\dot{p}_r(t) - 2\dot{\nu}(t)v(t) - 2\nu(t)\dot{v}(t)), \\
&= -2\dot{\nu}(t)\dot{v}^T(t)v(t) - 2\nu(t)\dot{v}^T(t)\dot{v}(t) = 0.
\end{aligned} \qquad (4.18)$$

If the velocity constraint is active, $d(v(t)^T v(t))/dt = 2\dot{v}(t)^T v(t) = 0$, in the interval, then the first term is zero and consequently $\dot{v}(t) = 0$ too. If it is inactive, then $\nu(t) = 0$ in the interval (due to complementary slackness) and hence $\dot{\nu}(t) = 0$. Thus, the first term is always zero. Hence, either $\nu(t) = 0$ or $\dot{v}(t) = 0$. Each case is now considered.

Case 1. Suppose that $\nu(t) = 0$. Then $p_r = 0$ since $\dot{p}_v(t) = 0$. The first transversality condition implies $p_0 = 0$. Additionally, the Hamiltonian can be rewritten as

$$\mathcal{H}[t] = \chi(t)T(t) - p_v^T(t)g = 0.$$

Because the switching function $\chi(t) = 0$, it must be that $p_v^T(t)g = 0$. Note that $p_v(t)$ is aligned with the unit vector $\hat{n}$. By assumption (ii), $\hat{n}$ is not orthogonal to g, and hence, $p_v^T(t)g = 0$ implies $p_v(t) = 0$. This subsequently implies that $p_m(t) = 0$, which violates the non-triviality condition of optimal control (see Theorem 15 in [16]). Hence, $\nu(t) \neq 0$.

Case 2. Suppose $\nu(t) \neq 0$, which implies that the velocity constraint is active. This means that $\dot{v}^T(t)v(t) = 0$, and hence $\dot{v} = 0$ (using (4.18)). Then $T(t) = m(t)g$. The pointwise maximum condition reduces to

$$\begin{aligned} &\max && -\alpha p_m(t)\Gamma(t) \\ &\text{subj. to} && \rho_1 \leq \Gamma(t) \leq \rho_2, \quad \|T(t)\| \leq \Gamma(t), \quad \hat{n}^T m(t)g = \Gamma(t)\cos\theta, \end{aligned}$$

by using (4.13). If $p_m(t) > 0$, then $\Gamma(t)$ should be made as small as possible, i.e., $\Gamma(t) = \|T(t)\|$. In the following, it is proven that this is the case. Combining (4.11), (4.12), and $T(t) = m(t)g$,

$$p_r^T v(t) = \alpha p_m(t)\Gamma(t).$$

Note that because $\dot{p}_v(t) = 0$, $p_r = -2\nu(t)v(t)$. Thus,

$$-2\nu(t)\|v(t)\|^2 = \alpha p_m(t)\Gamma(t).$$

Since $\nu(t) < 0$, $\|v(t)\| > 0$, $\alpha > 0$, and $\Gamma(t) > 0$, it must be that $p_m(t) > 0$. Thus, $\|T(t)\| = \Gamma(t)$ and $\rho_1 \leq \|T(t)\| \leq \rho_2$ by assumption (iii), which contradicts the original hypothesis. Thus, it has been shown by contradiction that the thrust magnitude constraint holds with equality everywhere along the optimal trajectory. □

Finally, as in Section 4.1.3, the time-varying mass dynamics can be rigorously approximated with convex dynamics and constraints [11–13]. The resulting convexified maximum divert PDG problem is then given by Problem 5.

For cases in which the target landing site uses all available fuel (as is the case with the terrestrial test flights), the lossless convexification has been proven when the speed and

thrust pointing constraints are simultaneously active. Thus, Problem 5 produces the same solutions as Problem 1.

Problem 5. Convex Relaxation of the Maximum Divert PDG Problem

$$
\begin{aligned}
\min \quad & J = -w_1 r_1(t_f) - w_2 r_2(t_f) \\
\text{subj. to} \quad & \dot{r}(t) = v(t), \qquad r(t_0) = r_0,\ r_3(t_f) = 0 \\
& \dot{v}(t) = u(t) - g, \quad v(t_0) = v_0,\ v(t_f) = 0 \\
& \dot{z}(t) = -\alpha\sigma(t), \qquad z(t_0) = \ln m_0,\ \ln m_d \leq z(t_f) \\
& \hat{n}^T u(t) \geq \sigma(t)\cos\theta, \quad \|v(t)\|^2 \leq V_c^2 \\
& \|u(t)\|^2 \leq \sigma^2(t), \quad \rho_1 e^{-z_0}\big[1 - (z(t) - z_0(t))\big] \leq \sigma(t) \\
& \sigma(t) \leq \rho_2 e^{-z_0}\big[1 - (z(t) - z_0(t))\big]
\end{aligned}
$$

4.3 Non-convex Trajectory Optimization for Obstacle Avoidance

The problem of avoiding obstacles with autonomous vehicles has been attracting immense research interest, and with the proliferation of self-driving cars, autonomous drone delivery systems, and the increasing availability of commercial off–the–shelf UAVs, interest will only continue growing. It is well known that convex optimization problems can be solved to global optimality very efficiently using a variety of methods [2, 4, 5, 7]. However, incorporating avoidance regions or obstacles inherently makes the state space non-convex. As a consequence, a variety of methods have been used to find feasible solutions to the non-convex optimal control problem of finding obstacle avoidance trajectories [86–89].

Random sampling methods have been used to quickly find geometrically feasible trajectories [88]; these trajectories are then smoothed in order to make them feasible with respect to vehicle dynamics and actuator constraints. More recently, sequences of quadratic programs [90], or more generally sequences of convex programs are solved to obtain feasible solutions to non-convex optimal control problems [79, 80, 91]. These methods have the advantage of dealing with non-convex constraints while simultaneously ensuring that convex

vehicle constraints are satisfied.

The goal of this section is to develop a framework for obtaining physically feasible trajectories that avoid cylindrical or ellipsoidal obstacles. On a system equipped with the high-level autonomy described in Chapter 3, this type of guidance would allow an agent to safely navigate around buildings or avoid dense residential areas where noise is a concern. At a high level, this is accomplished by solving a convergent sequence of convex optimization problems, where the non-convex collision avoidance constraints in each problem are linearized about the solution of the previous iteration (a process called *successive convexification*). Earlier work in this area showed that such sequences of optimization problems converge to a locally optimal fixed point when the system dynamics are nonlinear [80] or when every constraint function is convex [92]. However, no general convergence results exist for problems with non-convex constraint functions solved via successive approximation methods.

There are two main contributions presented in this section. First, the section develops a method for incorporating inter-sample collision avoidance constraints into discrete trajectory optimization problems. Enforcing collision avoidance constraints solely at the discrete time epochs may lead to computed trajectories passing through obstacles between time epochs. One approach to mitigating this risk is to over-sample the time discretization, thereby reducing the duration of the inter-sample epochs. However, such approaches do not guarantee that the obstacle avoidance constraints will be satisfied between discrete states, and unnecessarily introduce more solution variables. The methods presented in here provide a systematic approach for eliminating inter-sample collisions by introducing appropriate state and control constraints. Second, this section extends previous successive convexification convergence results to include a more general class of non-convex state constraints. These theoretical guarantees ensure the successful execution of the successive convexification process, which is critical for safe autonomous operations.

For the remainder of the section, $\mathcal{C}^q$ for some $q \in \mathbb{N}$ is the set of q continuously differentiable functions. Additionally, $I_{n\times n}$ is the multiplicative identity in $\mathbb{R}^{n\times n}$ for $n \in \mathbb{N}$.

4.3.1 Problem Formulation

Consider the discrete–time, finite–horizon, constrained trajectory optimization problem specified as follows:

$$\begin{aligned} \underset{x_k,u_k}{\text{minimize:}} \quad & \ell(x_k,u_k) \\ \text{subject to:} \quad & \forall k \in \mathcal{N}_0^{N-1},\ x_{k+1} = f(x_k,u_k,t_{k+1}), \\ & \forall k \in \mathcal{N}_0^{N-1},\ u_k \in \mathcal{U}, \\ & \forall k \in \mathcal{N}_1^{N}, \quad x_k \in \mathcal{X}, \\ & \qquad\qquad\quad x_0 \in \mathcal{X}_0 \subset \mathcal{X}, \\ & \qquad\qquad\quad x_N \in \mathcal{X}_f \subset \mathcal{X}, \end{aligned} \tag{4.19}$$

where $\ell : \mathbb{R}^{N(n_x+n_u)} \to \mathbb{R}$ is the objective function, N is the number of discrete time steps, each $x_k \in \mathbb{R}^{n_x}$ is the state at time t_k, each $u_k \in \mathbb{R}^{n_u}$ is the control at time t_k, $f(x,u,t)$ maps the state x and control u to the state $x(t)$ (the state at time t), $\mathcal{U} \subset \mathbb{R}^{n_u}$ is the set of permissible controls, $\mathcal{X} \subset \mathbb{R}^{n_x}$ is the set of feasible states, $\mathcal{X}_0 \subseteq \mathcal{X}$ is the set of initial states, and $\mathcal{X}_f \subseteq \mathcal{X}$ is the set of final states. Moreover, we break the feasible set of states $\mathcal{X}$ into

$$\mathcal{X} := \mathcal{X}_C \cap \mathcal{X}_{NC}, \tag{4.20}$$

where $\mathcal{X}_C$ is a convex, compact set and $\mathcal{X}_{NC}$ is the non-convex set of states that avoids obstacles.

Assumptions

This paper partly deals with extending previous trust-region-based convergence results [80] to allow for a more general class of non-convex state and control constraints. We make the following assumptions to obtain these results:

1. $f(x,u,t) \in \mathcal{C}^2$.

2. $\ell(x_k,u_k)$ is convex with respect to x_k and u_k.

3. $\mathcal{X}_0$, $\mathcal{X}_f$, and $\mathcal{U}$ are convex, compact sets.

4. Linear Independence Constraint Qualification (LICQ): For any fixed point of the algorithm $(x^*, u^*) \in \mathcal{X} \times \mathcal{U}$, the gradient of active constraint functions evaluated at (x^*, u^*) has full row rank.

The first assumption is quite reasonable, since most dynamical systems of interest have smooth, continuous dynamics. Assumption 2 does not impose any practical limitations because non-convexities in the objective function can be moved to the constraints without loss of generality. The third assumption is also very mild, and a large class of non-convexities in the control space can be handled via lossless convexification [11, 16]. Satisfying LICQ can impose some practical limitations, but LICQ is always satisfied when the boundaries of the obstacles are not in contact with any other constraint boundaries (sufficient condition).

We make one final assumption in order to obtain the inter-sample collision avoidance constraints presented in this paper:

5. For any $(x_k, u_k) \in \mathcal{X} \times \mathcal{U}$, the state transition matrix admits finitely many $t^* \in [t_k, t_{k+1}]$ such that
$$\frac{\partial f(x_k, u_k, t^*)}{\partial t} = 0.$$

Assumption 5 is clearly satisfied for systems with state transition matrices that are polynomial (e.g., double integrator dynamics), sinusoidal, or exponential functions of time (sufficient conditions).

4.3.2 Collision Avoidance

For the work presented in this paper, the obstacles are represented as cylinders with infinite height; this has the effect of avoiding a circular region in the projection of the trajectory onto the ground plane. Using this obstacle specification, the set $\mathcal{X}_{NC}$ in (4.20) is specified by:

$$\mathcal{X}_{NC} := \left\{ x \in \mathbb{R}^{n_x} : \ \|Tx - p_{c,i}\|_2 \geq r_i, \ \forall i \in \mathcal{N}_1^{n_{cyl}} \right\},$$

where $T : \mathbb{R}^{n_x} \to \mathbb{R}^2$ is a projection that maps the position state to the ground plane and satisfies $TT^T = I_{2\times 2}$, $p_{c,i} \in \mathbb{R}^2$ is the position of the center of cylinder i in the ground plane, $r_i \in \mathbb{R}_+$ is the corresponding radius, and n_{cyl} is the number of cylinders that need to be avoided. For the context of this paper, a collision occurs if any $x \notin \mathcal{X}_{NC}$ is a part of a trajectory.

Inter-sample Collision Avoidance

In order to avoid collisions for real vehicles, it is not sufficient to impose $x_k \in \mathcal{X},\ k \in \mathcal{N}_1^N$ since it only ensures that the discrete points x_k do not collide with obstacles. The continuous path between any x_k and x_{k+1} may still intersect an obstacle (see Figure 4.2).

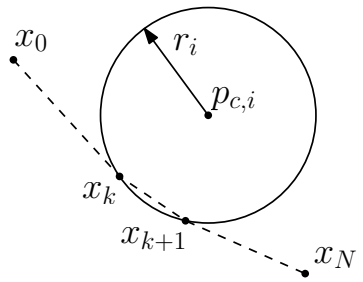


Figure 4.2: Inter-sample collision between x_k and x_{k+1}.

Knowledge about the continuous path between any two discrete points is derived from the state transition matrix, $f(x, u, t)$. The function, $h_i(x, u, t)$, can be used to determine whether a collision occurs with cylinder i at any time t, and is defined as follows:

$$h_i(x, u, t) := \|Tf(x, u, t) - p_{c,i}\|_2^2 - r_i^2.$$

Then, the set of states and controls that avoid collisions along the path from time t_A to t_B

is given by:

$$\mathcal{H}(t_A, t_B) := \big\{(x,u) \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} :\ h_i(x,u,t) \geq 0, \\ \forall t \in (t_A,\ t_B),\ \forall i \in \mathcal{N}_1^{n_{cyl}}\big\}. \tag{4.21}$$

Thus, the following constraint guarantees that all continuous sub-arcs between discrete points remain feasible:

$$\forall k \in \mathcal{N}_0^{N-1},\ (x_k, u_k) \in \mathcal{H}(t_k, t_{k+1}). \tag{4.22}$$

The set described in (4.21) imposes a constraint for each of the infinitely many time instances between t_A and t_B, and must therefore be re-expressed into a form that is suitable for numerical optimization. This is accomplished by identifying the times at which $h_i(x,u,t)$ attains its minima with respect to time. The set $\mathcal{T}_i^*(x,u,t_A,t_B)$ describes the times in the range $(t_A,\ t_B) \in \mathbb{R}_+$ at which $h_i(x,u,t)$ attains a minimum with respect to time, and is defined as

$$\mathcal{T}_i^*(x,u,t_A,t_B) := \Bigg\{t \in (t_A,\ t_B) :\ \frac{\partial h_i(x,u,t)}{\partial t} = 0, \\ \frac{\partial^2 h_i(x,u,t)}{\partial t^2} > 0\Bigg\}.$$

Note that $\mathcal{T}_i^*(x,u,t_A,t_B)$ is a finite set by Assumption 5. Moreover, if $\mathcal{T}_i^*(x,u,t_A,t_B) = \emptyset$, then no minima occur on the continuous arc from state x using control u over the horizon $[t_A,\ t_B]$ (since $x_k \in \mathcal{X},\ k \in \mathcal{N}_1^N$ ensures the endpoints are feasible). On the other hand, if $\mathcal{T}_i^*(x,u,t_A,t_B) \neq \emptyset$, a constraint must be imposed on the value of each of the finitely many distinct minima, leading to the following equivalent reformulation:

$$\mathcal{H}(t_A, t_B) = \big\{(x,u) \in \mathbb{R}^{n_x} \times \mathbb{R}n_u :\ h_i(x,u,t^*) \geq 0, \\ \forall\, t^* \in \mathcal{T}_i^*(x,u,t_A,t_B),\ \forall i \in \mathcal{N}_1^{n_{cyl}}\big\}. \tag{4.23}$$

As a result of the above reformulation, only a finite number of (non-convex) constraints need to be imposed in order to guarantee that the continuous path between two consecutive discrete states avoids collisions. Note that finding the set $\mathcal{T}_i^*(x,u,t_A,t_B)$ may involve using numerical techniques (e.g., Newton-Raphson). Also note that, there are analytic expressions

for the roots of polynomials of degree four or less (see Abel–Ruffini theorem).

Non-Convex Obstacle Avoidance Formulation

Combining the problem formulation in (4.19) and the inter-sample constraints in (4.22) and (4.23), we arrive at the non-convex obstacle avoidance problem formulation in (4.24).

$$\begin{aligned}
\underset{x_k,u_k}{\text{minimize:}} \quad & \ell(x_k, u_k) \\
\text{subject to:} \quad & \forall k \in \mathcal{N}_0^{N-1},\ x_{k+1} = f(x_k, u_k, t_{k+1}), \\
& \forall k \in \mathcal{N}_0^{N-1},\ u_k \in \mathcal{U}, \\
& \forall k \in \mathcal{N}_1^{N}, \quad x_k \in \mathcal{X}_C, \\
& \forall k \in \mathcal{N}_1^{N}, \quad x_k \in \mathcal{X}_{NC}, \\
& \forall k \in \mathcal{N}_0^{N-1},\ (x_k, u_k) \in \mathcal{H}(t_k, t_{k+1}), \\
& \qquad\qquad x_0 \in \mathcal{X}_0 \subset \mathcal{X}, \\
& \qquad\qquad x_N \in \mathcal{X}_f \subset \mathcal{X},
\end{aligned} \tag{4.24}$$

where the fourth and fifth constraints in (4.24) represent non-convex state and control constraints.

4.3.3 Successive Convexification

We begin by expressing non-convex optimization problems in a more general setting. Since non-convexities in state and control are treated in the same fashion, we introduce a joint variable $z := \{x_1^T, \ldots, x_N^T, u_0^T, \ldots, u_{N-1}^T\}^T \in \mathbb{R}^M$ where $M = N(n_x + n_u)$, and let $\mathcal{Z}$ be the appropriately repeated Cartesian product of $\mathcal{X}_C$, $\mathcal{U}$, $\mathcal{X}_0$, and $\mathcal{X}_f$ such that every $z \in \mathcal{Z}$ satisfies the convex constraints in (4.24). Note that $\mathcal{Z}$ is a convex, compact set by Assumption 3. Similarly, we denote $g_0(z) = \ell(x_k, u_k)$ and stack the non-convex constraints to form the column vector $g(z)$. Using these definitions, a general non-convex optimal control problem

can be expressed as:

$$\begin{aligned} \underset{z}{\text{minimize:}} \quad & g_0(z) \\ \text{subject to:} \quad & g_j(z) = 0, \quad j \in \mathcal{N}_1^e, \\ & g_j(z) \leq 0, \quad j \in \mathcal{N}_{e+1}^s, \\ & g_j(z) = 0, \quad j \in \mathcal{N}_{s+1}^r, \\ & g_j(z) \leq 0, \quad j \in \mathcal{N}_{r+1}^q, \end{aligned} \tag{4.25}$$

where $g_j(z),\ j \in \mathcal{N}_1^s$ represent non-convex constraints, and $\mathcal{Z}$ can be equivalently expressed as

$$\mathcal{Z} = \left\{ z \in \mathbb{R}^M : g_j(z) = 0, j \in \mathcal{N}_{s+1}^r, g_j(z) \leq 0, j \in \mathcal{N}_{r+1}^q \right\}.$$

Convex Subproblems

A natural way to convexify (4.25) is to linearize the non-convex constraints $g_j(z),\ j \in \mathcal{N}_1^s$ by using first order Taylor approximations. In general, the solution of the linearized problem will be different from that of the original, non-convex problem. Therefore, the objective is to utilize an algorithm whose solution at least satisfies the first order optimality conditions of the original problem. As it turns out, one can achieve this by conducting the linearization successively. Let the solution of the $(k-1)^{th}$ succession be z^{k-1} and denote $d = z - z^{k-1}$. Then, the (convex) linearized subproblem is given by:

$$\begin{aligned} \underset{d}{\text{minimize:}} \quad & g_0(d + z^{k-1}) \\ \text{subject to:} \quad & g_j(z^{k-1}) + \nabla g_j(z^{k-1})\, d = 0, \quad j \in \mathcal{N}_1^e, \\ & g_j(z^{k-1}) + \nabla g_j(z^{k-1})\, d \leq 0, \quad j \in \mathcal{N}_{e+1}^s, \\ & d + z^{k-1} \in \mathcal{Z}, \end{aligned} \tag{4.26}$$

where $\nabla g_j(z)$ is the gradient of the j^{th} constraint evaluated at z. The linearization renders the problem convex, but also introduces two issues: artificial infeasibility and approxima-

tion error. These issues are addressed by introducing penalty functions and trust regions, respectively.

Artificial infeasibility refers to cases in which (4.25) has feasible solutions, yet (4.26) is infeasible. This issue may arise when z^{k-1} is not a feasible solution of (4.25), and is addressed by relaxing the linearized non-convex constraints in (4.26) and penalizing their violation in the objective function [80]. Note that this is an implicit way to introduce slack variables. The 1-norm used in [80] degenerates to a finite sum of absolute values. Therefore, with the addition of inequality constraints, the penalty function used in [80] is extended to become

$$\begin{aligned} L(d,\lambda) := g_0(d+z^{k-1}) + \sum_{j=1}^{e} \lambda_j \left| g_j(z^{k-1}) + \nabla g_j(z^{k-1})\, d \right| \\ + \sum_{j=e+1}^{s} \lambda_j \max\left\{0, g_j(z^{k-1}) + \nabla g_j(z^{k-1})\, d\right\}, \end{aligned} \tag{4.27}$$

where the scalars $\lambda_j \geq 0$ are penalty weights and $\lambda := [\lambda_1, \lambda_2, \ldots, \lambda_s]$. Note that the corresponding penalty function prior to linearization is specified by

$$J(z,\lambda) := g_0(z) + \sum_{j=1}^{e} \lambda_j |g_j(z)| + \sum_{j=e+1}^{s} \lambda_j \max\left\{0, g_j(z)\right\}. \tag{4.28}$$

$J(z,\lambda)$ is also the most commonly used exact penalty function in constrained optimization literature.

As previously mentioned, the second issue introduced by the linearization is the approximation error. If left unchecked, it has the potential to render the optimization unbounded. To mitigate this risk, we ensure that the z^k does not deviate significantly from z^{k-1}, via a *trust region* on the new step, $\|d\| \leq \Delta$. The rationale is that the linear approximation is only accurate near the point the constraints were linearized about. Thus, the final formulation of

the convex subproblem is specified in (4.29).

$$\begin{aligned} \underset{d}{\text{minimize:}} \quad & L(d,\lambda) \\ \text{subject to:} \quad & \|d\| \leq \Delta, \\ & d + z^{k-1} \in \mathcal{Z}. \end{aligned} \tag{4.29}$$

Collision Avoidance Algorithm

The overall successive-convexification-based collision avoidance algorithm is presented in this section. The goal here is to obtain feasible, locally optimal trajectories to the non-convex trajectory optimization problem.

Data: $\alpha, \epsilon, \Delta, \rho_0, \rho_1, \rho_2 \in \mathbb{R}_+$ such that $\rho_0 < \rho_1 < \rho_2$ and $\alpha > 1$, N, λ, and the implicit form of the sets $\mathcal{X}_C$, $\mathcal{X}_{NC}$, $\mathcal{U}$, $\mathcal{X}_0$, and $\mathcal{X}_f$
Result: a feasible, locally optimal sequence of states and controls, z^*
begin
  Solve (4.19) with $\mathcal{X} = \mathcal{X}_C$ to get z^0 (ignore obstacles);
  if $x_k \in \mathcal{X}_{NC},\ k \in \mathcal{N}_1^N$ *and* $(x_k, u_k) \in \mathcal{H}(t_k, t_{k+1}),\ k \in \mathcal{N}_0^{N-1}$ **then**
    No collisions occurred, return $z^* = z_0$;
  else
    $\Delta L^1 = 10\epsilon$;
    $k = 1$;
    while $\Delta L^k \geq \epsilon$ **do**
      Solve (4.29) to get d and $z^k = z^{k-1} + d$;
      Compute $J(z^k, \lambda)$ using (4.28);
      Compute $\Delta L^k := J(z^{k-1}, \lambda) - L(d, \lambda)$;
      Compute $\Delta J^k := J(z^{k-1}, \lambda) - J(z^k, \lambda)$;
      Compute $\rho := \Delta L^k / \Delta J^k$;
      if $\rho < \rho_0$ **then**
        Reject this step and $\Delta = \Delta/\alpha$;
      else
        $$\Delta = \begin{cases} \Delta/\alpha, & \rho < \rho_1, \\ \Delta, & \rho_1 \leq \rho < \rho_2, \\ \alpha\Delta, & \rho_2 \leq \rho. \end{cases} ;$$
        $k = k + 1$;
    return $z^* = z^k$;

Here, the initial reference states and controls are obtained by solving (4.19) without considering the obstacles, thus the initial references may be infeasible with respect to the obstacles. After each successful iteration of the loop, the reference states and controls are set to be the most recent solution of (4.29). The success of each iteration is based on tbe criteria commonly used in the trust region methods literature (e.g. [93]).

4.3.4 Convergence Analysis

In this section, we extend the convergence result from [80] to include a broader class of state constraints. The procedure for showing the convergence of this successive convexification method is largely the same as [80]. Thus, in the interest of compactness, only statements that need modification are presented here.

First, the non-convex penalty problem is expressed as

$$\begin{aligned} \text{minimize:} \quad & J(z,\lambda) \\ \text{subject to:} \quad & g_j(z) = 0, \quad j \in \mathcal{N}_{s+1}^{r}, \\ & g_j(z) \leq 0, \quad j \in \mathcal{N}_{r+1}^{q}. \end{aligned} \tag{4.30}$$

The difference between (4.30) and the conventional penalty formulation is that the convex constraints are not relaxed. As a result, (4.30) is still a constrained problem, but since $g_j(z) = 0$ and $g_j(z) \leq 0$ for $j \in \mathcal{N}_{s+1}^{q}$ are convex constraints, they can be incorporated into the convex programming framework without any approximation.

The first step in the proof is to show the exactness of the penalty function, $J(z,\lambda)$. Given a feasible point of (4.30) z^*, we define the index set of active convex constraints at z^* to be

$$\mathcal{I}_{ac}(z^*) = \big\{ j \in \mathcal{N}_{s+1}^{q} : \, g_j(z^*) = 0 \big\}.$$

Then, since LICQ is satisfied by Assumption 4, the tangent cone of these active convex

constraints is represented by

$$T(z^*) = \left\{ \alpha d \in \mathbb{R}^M : \ \forall \alpha \geq 0, \ \nabla g_j(z^*)^T d \leq 0, \ j \in \mathcal{I}_{ac}(z^*) \right\}.$$

With $T(z^*)$ defined, the first order necessary conditions for a point z^* to be a local optimum of the original, non-convex problem in (4.25) are given by Theorem 11.

Thoerem 11 (**Karush–Kuhn–Tucker (KKT)**). *If the original problem in* (4.25) *satisfies LICQ, and z^* is a local optimum of that problem, then there exist Lagrange multipliers μ_j and $\sigma_j \geq 0$ such that*

$$\sigma_j g_j(z^*) = 0, \ j \in \mathcal{N}_{e+1}^s,$$

and $\forall d \in T(z^*)$,

$$\left(\nabla g_0(z^*) + \sum_{j=1}^{e} \mu_j \nabla g_j(z^*) + \sum_{j=e+1}^{s} \sigma_j \nabla g_j(z^*) + \sum_{j=s+1}^{r} \mu_j \nabla g_j(z^*) + \sum_{j=r+1}^{q} \sigma_j \nabla g_j(z^*) \right) d \geq 0.$$

We call such a z^* a KKT point.

Theorem 12 provides the exactness of the penalty function $J(z, \lambda)$. Its proof follows the same reasoning as that used in [80], so it is omitted.

Thoerem 12. *If z^* is a KKT point of the original problem in* (4.25) *with multipliers $\bar{\mu}_j$, and $\bar{\sigma}_j$, and if the penalty weight λ satisfies*

$$\lambda_j \geq |\mu_j|, \ \forall \ j \in \mathcal{N}_1^e,$$
$$\lambda_j \geq \sigma_j, \quad \forall \ j \in \mathcal{N}_{e+1}^s,$$

then z^* is a constrained stationary point of the penalty problem in (4.30). *Conversely, if a constrained stationary point of the penalty problem z^* is feasible for the original problem,*

then it is also a KKT point of the original problem.

The definition of a constrained stationary point can be found in [80]. From Theorem 12, one can see that the objective of the successive convexification algorithm is to find such constrained stationary points. In fact, the algorithm produces a sequence of points $\{z^k\}$, and the goal is to prove that this sequence converges to constrained stationary points. This convergence result has been established in Theorem 3 and Theorem 4 from [80], and their proofs are not affected by the introduction of the non-convex inequality constraints added in this paper. Thus, we present the final result in the next theorem without re-stating its proof.

Thoerem 13. *The "predicted" changes ΔL^k defined inside the algorithm in Section 4.3.3 are non-negative for any k.*

Also, $\Delta L^{k^*} = 0$ implies that z^{k^*} is a constrained stationary point of the penalty problem in (4.30).

Furthermore, if $\forall k$, $\Delta L^k > 0$, the algorithm generates an infinite sequence $\{z^k\}$. Then, $\{z^k\}$ has limit points, and any limit point z^* is a constrained stationary point of the penalty problem in (4.30).

4.3.5 Illustrative Example

In this section, we use double integrator dynamics to illustrate the process of imposing (4.22) in practice. For each $k \in \mathcal{N}_0^{N-1}$ & $i \in \mathcal{N}_1^{n_{cyl}}$, the objective is to impose $h_i(x_k, u_k, \tau) \geq 0, \forall\ \tau \in \mathcal{T}_i^*(x_k, u_k, t_k, t_{k+1})$. The first step is to linearize this non-convex constraint about $(\bar{x}_k, \bar{u}_k, \bar{t}^*)$, $\forall\, \bar{t}^* \in \mathcal{T}_i^*(\bar{x}_k, \bar{u}_k, t_k, t_{k+1})$ via a first order Taylor Series approximation:

$$\bar{h}_i(x_k, u_k, t^*) = h_i(\bar{x}_k, \bar{u}_k, \bar{t}^*) + \left.\frac{\partial h_i}{\partial x_k}\right|_{\substack{x_k=\bar{x}_k \\ u_k=\bar{u}_k}} (x_k - \bar{x}_k) + \left.\frac{\partial h_i}{\partial u_k}\right|_{\substack{x_k=\bar{x}_k \\ u_k=\bar{u}_k}} (u_k - \bar{u}_k),$$

and incorporate $\bar{h}_i(x_k, u_k, t^*)$ into (4.27). Let the state be given by $x = [p^T, v^T]^T$, where $p, v \in \mathbb{R}^2$. Then for $t \in (t_k, t_{k+1})$, the position of the vehicle is given by

$$Tf(x_k, u_k, t) = p(t) = p_k + v_k(t - t_k) + \frac{1}{2}(u_k + g)(t - t_k)^2.$$

For these dynamics, $\mathcal{T}_i^*(\bar{x}_k, \bar{u}_k, t_k, t_{k+1})$ is found by noting that $h_i(\bar{x}_k, \bar{u}_k, t)$ can be expressed as

$$h_i(\bar{x}_k, \bar{u}_k, t) = a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0,$$

where $a_j,\ j \in \mathcal{N}_0^4$ are known coefficients (since $\bar{x}_k, \bar{u}_k$, and t_k are known). Thus, finding the set $\mathcal{T}_i^*(\bar{x}_k, \bar{u}_k, t_k, t_{k+1})$ is equivalent to finding the roots of a cubic polynomial with known coefficients, which can be done analytically using Cardano's formula. As a final example, we denote $\tau = \bar{t}^* - t_k$ and $\delta = \bar{p}_k - p_{c,i}$ to express the partial of h_i with respect to control as

$$\frac{\partial h_i}{\partial u_k} = 2\delta^T \tau^2 + (u_k + g)^T \tau^3 + \frac{\partial h_i}{\partial t^*} \frac{\partial t^*}{\partial u_k},$$

where the last term reflects the dependence of $\mathcal{T}_i^*$ on u_k, and $\frac{\partial t^*}{\partial u_k}$ can be approximated via the finite difference method.

4.3.6 Numerical Results

This section describes several numerical examples which illustrate the obstacle avoidance capability and numerical convergence properties of the proposed planning algorithm. The application scenario considered for these examples is that of trajectory optimization for multi-rotor unmanned aerial vehicles in obstacle-rich environments. For onboard trajectory planning in such applications, the main requirement is that of rapid convergence to obstacle free and dynamically feasible trajectories.

Multi-Rotor Trajectory Planning Problem

For trajectory planning purposes, the motion of the multi-rotor is modeled using double integrator dynamics. The state at time t_k is defined to be $x_k = [p_k^T, v_k^T]^T$ where $p_k \in \mathbb{R}^3$ is the vehicle position and $v_k \in \mathbb{R}^3$ is the corresponding velocity. The discrete time model is given by

$$x_{k+1} = f(x_k, u_k, t_{k+1}) = Ax_k + B(u_k + g)$$

where A and B correspond to double integrator dynamics with time step Δt, and are given by:

$$A = \begin{bmatrix} I_{3\times3} & \Delta t \cdot I_{3\times3} \\ 0_{3\times3} & I_{3\times3} \end{bmatrix},\ B = \begin{bmatrix} \frac{1}{2}\Delta t^2 \cdot I_{3\times3} \\ \Delta t \cdot I_{3\times3} \end{bmatrix}.$$

As in Section 4.3.5, the above dynamics render $h_i(x, u, t)$ a quartic with respect to time.

The convex set of feasible states, $\mathcal{X}_C$ is given by:

$$\mathcal{X}_C := \left\{(p,\ v) \in \mathbb{R}^6\ :\ \|v\|_2 \leq V_{max}\right\},$$

where V_{max} is a prescribed velocity norm upper bound. The convex set of controls is given by:

$$\mathcal{U} := \left\{u \in \mathbb{R}^3 : \hat{n}^T u \geq \|u\| \cos(\theta_{cone}),\ \|u\|_2 \leq T_{max}/m\right\},$$

where θ_{cone} defines a thrust cone angle that confines the thrust vector to a cone pointing towards the ceiling, T_{max} is the maximum thrust capacity, and m is the vehicle mass.

For the obstacle avoidance scenario, three cylindrical shapes are used to represent obstacles present in the environment, with centers and radii defined by the parameters $p_{c,i}, r_i,\ i = 1, 2, 3$. The objective function is a minimum time heuristic chosen to promote contact between the trajectory and the boundary of the obstacles. A summary of the simulation parameters selected for the scenario can be found in Table 4.2.

Table 4.2: Simulation Parameters

Parameter	Value	Parameter	Value
N	17	$p_{c,1}$	$[-3,\ 0]^T$ m
V_{max}	2 m/s	r_1	3 m
T_{max}	40 N	$p_{c,2}$	$[4,\ -1]^T$ m
m	3 kg	r_2	2 m
g	$[0,\ 0,\ -9.81]^T$ m/s^2	$p_{c,3}$	$[8,\ 1]^T$ m
θ_{cone}	30 deg	r_3	1 m
Δ	3.15	α	1.2
ρ_0	0	ρ_1	0.25
ρ_2	2	ϵ	1×10^{-5}

Simulation Results

The problem defined in Section 4.3.6 is solved using CVX [51] in MATLAB with the following initial and final conditions as test cases:

$$p_0 = [-7,\ 0,\ 0]^T m,\ v_0 = [0,\ 0,\ 0]^T m/s,$$

$$p_f = [8,\ -0.1,\ 0.7]^T m,\ v_f = [0,\ 0,\ 0]^T m/s.$$

Figure 4.3 shows a 3-D view of the converged trajectory, note that there is an altitude change. The vehicle path skims the cylinders but does not pass into the circular area, even in the inter-sample regions. Note that the velocity constraint is active for most of the trajectory (since the objective function was minimum time of flight), and the remaining convex constraints are also satisfied (Figure 4.4).

The algorithm convergence properties, as discussed in Section 4.3.4, are demonstrated by the plots shown in Figure 4.5. In particular, the improvement in linearized objective function (ΔL), and the state and control sequence norms converge quickly in a small number

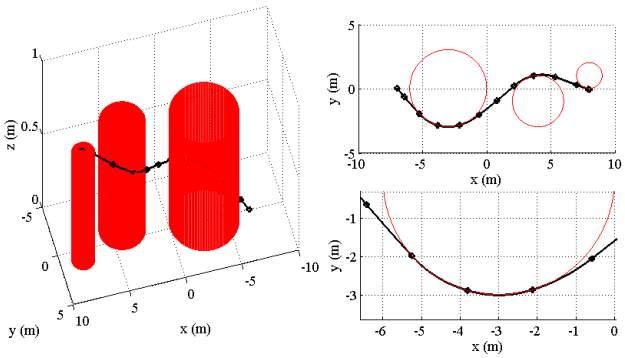


Figure 4.3: Converged Feasible Path Views: Isometric, XY-Plane, Inter-sample Region Zoom

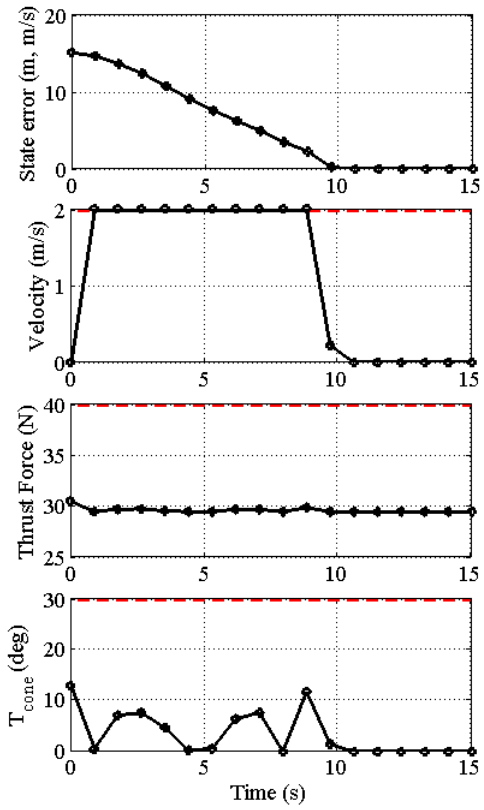


Figure 4.4: Constraint Overview

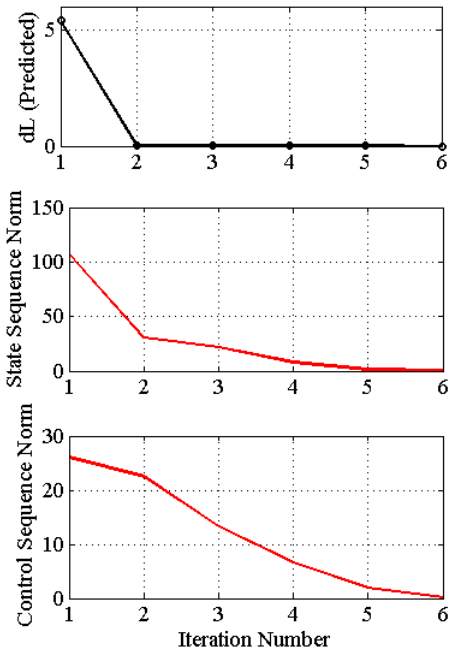


Figure 4.5: Convergence Overview

of iterations. The state and control sequence norms were determined by finding the norm of the change in state and control between successive iterations of the optimization algorithm, defined as

$$\Delta X^k = \sum_{j=1}^{N} \|x_j^k - x_j^{k-1}\|_2, \;\; \Delta U^k = \sum_{j=0}^{N-1} \|u_j^k - u_j^{k-1}\|_2,$$

where $\{x_j^k\}$, $\{u_j^k\}$ are the state and control sequences computed in iteration k. For most simulations, algorithm convergence is achieved within 5 to 11 iterations.

Chapter 5

REAL-TIME OPTIMIZATION IN CONTROL

Once a guidance trajectory has been computed via the methods described in Chapter 4, the burden falls on a closed loop control law to follow the desired trajectory to the final destination. At any given time step, the desired control determined by the closed loop must be mapped onto the vehicle's actuators. This chapter focuses on an algorithm that improves the performance of such control allocations with a focus on spacecraft attitude control systems, and is adapted from [35].

Control allocation is defined as the mapping from a physical command, such as a desired control torque or force, to actuator level commands (e.g., thruster on-off times or flywheel accelerations). Attitude control systems with momentum-based actuators (RWAs and CMGs) typically use closed form solutions such as the Moore-Penrose Pseudo-inverse (MP) to allocate torques among actuators. Variations of MP are used to avoid the geometric singularities inherent to the single-gimbal CMG problem and are shown to be locally optimal with respect to the minimum torque error and desired gimbal rate error [94]. However, these closed form solutions do not explicitly consider actuator dynamics or constraints, and therefore may produce infeasible solutions. Furthermore, since the MP solution does not consider actuator constraints, saturation and rate constraints are imposed within the inner-loop of the actuator motors, thus adversely affecting the allocation accuracy and power usage.

Many innovative approaches have been used to find solutions to the momentum allocation problem in the literature [95–99]; however, the method presented in this chapter, Minimum Error Dissipative Power Reduction Control (ME-DPRC), is the first to explicitly incorporate actuator rate limits into the allocation problem and make use of friction to reduce power. ME-DPRC leverages the custom IPM algorithms developed in Chapter 2, enabling us to

solve for optimal allocations more than three orders of magnitude faster than generic IPM algorithms ($<$ 1 ms per solution); this is a dramatic leap in computational efficiency and is the key to enabling real-time implementations. Unlike adaptive control techniques for power reduction [100–102], ME-DPRC produces a minimum torque error solution that does not violate constraints, utilizes known friction characteristics (i.e., does not estimate friction parameters), and is altered at the command level (not solely the actuator inner-loop).

5.1 Lexicographic Optimization (LO) Control Allocation

LO is a form of Lexicographic Goal Programming (LGP) for multiple, potentially conflicting goals [103] that are naturally sorted into a prioritized sequence of objectives from the problem's inherent structure. One can first optimize the most important cost function, and then use this optimal cost as a constraint for the remainder of the optimizations while repeating the same procedure for all remaining costs. Consider the optimization over the following ordered set of q cost functions, $f\colon \min_{x\in\mathbb{X}} \ (f_1; f_2; \ldots; f_q)$, where $x \in \mathbb{R}^n$ is the solution variable, $\mathbb{X}$ is the set of all feasible solutions, and (;...;) refers to an ordered set of cost functions (in descending order). Using LGP, this problem can be broken down into q sub-problems. First, one solves the highest priority optimization problem due to, $f_1\colon \min_{x\in\mathbb{X}} c_1{=}f_1(x)$. Suppose an optimal solution, x_1^*, is found and that $c_1^* = f_1(x_1^*)$. Then, the second optimization is: $\min_{x\in\mathbb{X}} c_2{=}f_2(x)$ subject to $f_1 \leq c_1^*$. Continuing in this fashion, the final problem is given by:

$$\min_{x\in\mathbb{X}} c_q = f_q(x) \ \text{ subject to } \ f_i \leq c_i^*,\ i = 1, \ldots, q-1.$$

The resulting solution, $x_q^* \in \mathbb{X}$, is the optimal solution of the ordered set of costs. If $\mathbb{X}$ is a convex set and f_i are convex functions, then IPMs can be used to quickly find solutions with polynomial-time convergence guarantees [5]. A single, weighted cost $\sum_{i=1}^{q} w_i f_i(x)$ with $w_{i+1} \leq w_i, i{=}1, \ldots, q-1$, does not in general produce the same solution as LO.

5.1.1 A Lexicographic Optimization View of MP

The MP solution in (5.4) can be identically cast as a particular 2-step LO problem. First, an *unconstrained* optimization problem is solved to minimize the 2-norm of the torque error (e.g., for a reaction wheel system):

$$\min_{\dot{\Omega}_k} \; J_1 = \left\| \dot{h}_{des,k} - I^W A_s \dot{\Omega}_k \right\|_2 . \tag{5.1}$$

where $\dot{\Omega}_k$ is the reaction wheel acceleration at time step t_k, $\dot{h}_{des,k}$ is the desired torque at time step t_k, I^W is the moment of inertia matrix of each wheel (assumed to be identical here), and A_S is a matrix containing the unit vectors for each wheel's spin axis in its columns. Given a minimal cost, J_1^*, the 2-norm of the rotor rates is then minimized while enforcing minimum torque error:

$$\min_{\dot{\Omega}_k} J_2 = \left\| \dot{\Omega}_k \right\|_2 \;\; \text{s.t.} \;\; \left\| \dot{h}_{des,k} - I^W \mathbf{A}_s \dot{\Omega}_k \right\|_2 \leq J_1^*, \tag{5.2}$$

where the quadratic inequality in Equation (5.2) leads to an SOCP [2]. This LO problem exposes the optimizations that are truly taking place any time one uses a MP pseudo-inverse. Pseudo-inverse solutions (including MP) prioritizes unconstrained problems naturally. Casting a 2-step LO problem has two advantages: (i) enables the incorporation of constraints explicitly (MP does not consider any constraints); (ii) use of specialized costs for the second optimization (e.g., power).

5.1.2 Minimum Torque Error MP for CMGs

The goal of control allocation algorithms is to map a commanded torque to a specific set of rotor or gimbal rates in real-time. Thus, control allocation typically lies below a higher tier spacecraft controller that generates torque commands, but above inner loop controllers that regulate gimbal and rotor rates. In this framework, $\dot{\delta}_{i,k}$ denotes the gimbal rate of the i^{th} gimbal at the k^{th} time step, and $\dot{\delta}_k$ denotes the vector of all gimbal rates at the k^{th} time step. Typical assumptions when using pseudo-inverse solution for CMGs are [94]:

(i) Uniform rotor angular momentum magnitude across all CMGs: $h_0 = I^{Wi}\Omega_i = I^W\Omega$; (ii) Only rotor angular momentum is non-negligible: $h = h_0 A_{s,k}$, where $A_{s,k}$ is a matrix with rotor spin directions in its columns; (iii) Only the gyroscopic internal torque contribution is non-negligible: $\dot{h}_k = h_0 A_{t,k}\dot{\delta}_k$, where $A_{t,k}$ is a matrix with gyroscopic torque directions in its columns. For simplicity, $A_k = A_{t,k}$ for CMGs henceforth.

The primary objective of allocation is to minimize the torque error, $\tau_{err,k}$, between the actuated and the desired torques,

$$\tau_{err,k} = \dot{h}_{des,k} - h_0 A_{k+1}\dot{\delta}_{k+1},$$

where $\dot{h}_{des,k}$ is the desired torque at the next time step, A_{k+1} is the Jacobian matrix evaluated using the gimbal angles of the next time step, and $\dot{\delta}_{k+1}$ is a vector of n gimbal rates at the next time step. For sufficiently short time steps, $A_{k+1} \approx A_k$ can temporarily be assumed to formulate an analytic solution. The standard MP solution is obtained by minimizing [94],

$$J = \frac{1}{2}(\dot{h}_{des,k} - h_0 A_k\dot{\delta}_{k+1})^T(\dot{h}_{des,k} - h_0 A_k\dot{\delta}_{k+1}), \tag{5.3}$$

and is given by,

$$\dot{\delta}^*_{k+1} = \frac{1}{h_0}(A_k^T A_k)^{-1} A_k^T\, \dot{h}_{des,k}. \tag{5.4}$$

Equation (5.4) yields gimbal rates that exactly match the desired torque when the desired torque lies in the image of A_k; otherwise, the solution represents the minimum 2-norm torque error solution. In either case, the solution contains no null-space component and does not consider actuator inequality constraints. Thus, the solution may be infeasible. Furthermore, friction torques can be used to reduce the power usage of control allocation; however, the solution provided by (5.4) has no apparent mechanism for prioritizing gimbal decelerations.

5.1.3 CMG Minimum Torque Error/Power Allocation

The primary objective is to minimize torque error while minimizing power usage whenever possible without violating actuator constraints. Gimbals change the direction of the flywheel spin vector, and therefore, the Jacobian matrix, $A_{t,k} = A_k$, is a function of time. Once gimbal rate commands are computed, a lower level closed-loop controller is assumed to track them instantaneously. As the gimbal rate evolves, the actual gimbal angles and therefore the Jacobian matrix vary as well. Hence, CMGs have an internal state impacting their actuation capability - a property that makes the CMG allocation more challenging. Since the ME-DPRC algorithm greedily minimizes torque error, it is unlikely that the final torque error represents a global minimum. There are situations where a small amount of initial torque error saves large amounts of error later. This issue can be mitigated by allocating CMG commands over a longer time horizon, and is the subject of future research.

The first optimization problem in the CMG ME-DPRC allocation is:

$$\min_{\dot{\delta}_{k+1},\ddot{\delta}_k} J_1 = \left\| \dot{h}_{des,k} - h_0 A_{k+1} \dot{\delta}_{k+1} \right\|_2 \text{ s.t., (5.6)}, \tag{5.5}$$

with

$$\begin{gathered} \dot{\delta}_{k+1} = \dot{\delta}_k + \ddot{\delta}_k \Delta t, \\ -\dot{\delta}_{max} \leq \dot{\delta}_{k+1} \leq \dot{\delta}_{max}, \quad -\ddot{\delta}_{max} \leq \ddot{\delta}_k \leq \ddot{\delta}_{max}, \end{gathered} \tag{5.6}$$

where $\dot{\delta}_{max}$ is the upper bound on gimbal rates, $\ddot{\delta}_{max}$ is the upper bound on gimbal accelerations, and

$$A_{k+1} = \Theta \begin{bmatrix} -s(\delta_{1,k+1}) & -s(\delta_{2,k+1}) & s(\delta_{3,k+1}) & s(\delta_{4,k+1}) \\ -c(\delta_{1,k+1}) & -c(\delta_{2,k+1}) & -c(\delta_{3,k+1}) & -c(\delta_{4,k+1}) \\ -s(\delta_{1,k+1}) & s(\delta_{2,k+1}) & s(\delta_{3,k+1}) & -s(\delta_{4,k+1}) \end{bmatrix} \tag{5.7}$$

where Θ is a constant matrix describing the CMG arrangement, $c(\cdot) = \cos(\cdot)$, and $s(\cdot) = \sin(\cdot)$. The cost function in (5.5) contains both sinusoids and bilinear relations of gimbal rates ($\dot{\delta}_{k+1}$) when (5.7) is substituted into the cost function - thus making this cost non-convex. A Taylor series expansion of the sinusoids about the k^{th} time step for the i^{th} CMG is used:

$$\begin{aligned} c(\delta_{i,k+1}) &\approx c(\delta_{i,k}) - s(\delta_{i,k})\dot{\delta}_{i,k}\Delta t, \\ s(\delta_{i,k+1}) &\approx s(\delta_{i,k}) + c(\delta_{i,k})\dot{\delta}_{i,k}\Delta t. \end{aligned} \tag{5.8}$$

Hence A_{k+1} can be approximated as follows,

$$A_{k+1} \approx \Theta\left(A_k + \dot{A}_k \Delta t\right) = \hat{A}_k. \tag{5.9}$$

By using the above approximation, the minimum torque error problem in (5.5) is redefined as:

$$\min_{\dot{\delta}_{k+1}, \ddot{\delta}_k} J_1 = \left\| \dot{h}_{des,k} - h_0 \hat{A}_k \dot{\delta}_{k+1} \right\|_2 \text{ s.t. (5.6).} \tag{5.10}$$

As before, the goal is to minimize power usage over the set of minimum torque error solutions by making use of friction. After extensive simulations, we observed that globally prioritizing the minimization of gimbal rate tends to produce the most robust power savings. Ultimately, we find that following second optimization is the best greedy method to save power:

$$\begin{aligned} \min_{\dot{\delta}_{k+1}, \ddot{\delta}_k} J_2 &= \left\| \dot{\delta}_{k+1} \right\|_2 \\ \text{s.t.} \quad & \left\| \dot{h}_{des,k} - h_0 \hat{A}_k \dot{\delta}_{k+1} \right\|_2 \leq J_1^*, \ \text{(5.6)}. \end{aligned} \tag{5.11}$$

5.1.4 Spectral Scaling

To maintain tracking performance in the presence of singularities, a Singular Value Decomposition (SVD) based approach is employed. While SVD has been used to avoid CMG singularities [104], here it is used to mitigate the effects of singularities by adjusting the desired torque alone, rather than perturbing the pseudo-inverse matrix. The Jacobian matrix, $\hat{A}_k$, maps gimbal rates to torques; therefore, if $\hat{A}_k$ loses rank, a subspace of torques is no longer in the image of $\hat{A}_k$. In this case, it is important to still maintain allocation performance, while minimizing power. Consider the SVD of the Jacobian $\hat{A}_k \in \mathbb{R}^{3\times n}$ (for n CMGs),

$$\hat{A}_k = U \left[\Lambda \,|\, \mathbf{0}\right] V^T, \tag{5.12}$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$ with $\lambda_1 \geq \lambda_2 \geq \lambda_3$, and U and V are unitary matrices. As λ_3 approaches zero, gimbal contributions from one of the basis vectors in V has a diminishing impact on the output torque. Until this torque contribution is zero, it is still a part of the torque necessary to get the minimum torque error via (5.10). Consequently, large gimbal angle rates are necessary to provide the minimum torque error solution - generally causing the gimbal rates to quickly saturate. This may lead to loss of control authority in other directions, thereby prolonging the duration of the singularity. Moreover, this effort only *marginally* affects the output torque. Therefore, we propose the following method for reducing the negative impact of singularities on minimizing torque error. First, a threshold, $\bar{\lambda}$, is introduced to determine the level of actuation near singularities. The lower the value of $\bar{\lambda}$, the higher the actuation near singularity. Next, define T, which transforms the desired torque vector:

$$T = \begin{cases} I_{3\times3} & : \lambda_3 \geq \bar{\lambda} \\ I_{3\times3} - \frac{\bar{\lambda}-\lambda_3}{\bar{\lambda}} \hat{u}_3 \hat{u}_3^T & : \lambda_3 < \bar{\lambda} \\ I_{3\times3} - \frac{\bar{\lambda}-\lambda_3}{\bar{\lambda}} \hat{u}_3 \hat{u}_3^T - \frac{\bar{\lambda}-\lambda_2}{\bar{\lambda}} \hat{u}_2 \hat{u}_2^T & : \lambda_2 < \bar{\lambda} \end{cases} \tag{5.13}$$

where $I_{3\times3}$ is a 3×3 identity matrix, $\hat{u}_3$ is the basis vector that corresponds to λ_3 , and $\hat{u}_2$ is the basis vector that corresponds to λ_2. Finally, the desired torque vector is transformed to require less actuation: $\dot{h}'_{des,k} = T\dot{h}_{des,k}$, where $\dot{h}'_{des,k}$ is used for the optimization problems in (5.10) and (5.11) in lieu of the larger $\dot{h}_{des,k}$. We empirically observed that using modest values for $\bar{\lambda}$ (e.g., 0.3) reduces the torque error by causing the gimbal trajectory to spend less time at singularity. However, in cases with singularities, the use of spectral scaling results in a net power reduction due to the reduced actuation for the torque subspaces associated with the lower control authority.

5.1.5 Numerical Simulation Examples

Numerical simulations are presented to assess the performance of ME-DPRC for the allocation of feed-forward (open-loop) control torques. A fifth order Runge-Kutta integrator is used with a fixed time-step of 0.005 s, and rotor actuation commands are updated ev-

ery 0.04 s. In all color figures, blue represents the results for the first CMG, then red, green, and finally cyan for the others. This section presents simulation results for 4 single gimbal CMGs in a box array. ME-DPRC is compared with Singular Direction Avoidance (SDA) [104] and a saturated MP solution. the MP solution is computed via (5.4) and the SDA solution is computed as in [104], then both are saturated if actuator constraints are violated. The CMG gimbal angle time series vary with allocation algorithms due to the different commanded gimbal rate, i.e., each evolves its own Jacobian matrix, $\hat{A}_k$, independently. Simulation parameters are given in Table 5.1.

Variable	Value	Units	Variable	Value	Units
h_0	1.8626	N-ms	I_{gw}	0.06286	Kgm^2
$\dot{\delta}_{max}$	1	rad/s	$\ddot{\delta}_{max}$	5	rad/s^2
θ	$\pi/4$	rad	a_g	0.2	N-m
b_g	8	–	c_g	0.4	N-m
α	0.075	–	$\bar{\lambda}$	0.35	–

Table 5.1: CMG Simulation Parameters

Single-Axis Satellite Slew

The desired torque profile is given in Equation (5.14), which is consistent with a rest-to-rest single-axis slew maneuver:

$$\dot{h}_{des}(t) = \begin{cases} 0.4[t, -t, t]^T & : 0 \leq t \leq 1 \\ 0.4[1, -1, 1]^T & : 1 < t \leq 10 \\ 0.4[-1, 1, -1]^T & : 20 \leq t \leq 30 \\ [0, 0, 0]^T & : \text{otherwise} \end{cases} \tag{5.14}$$

First, the saturated analytic MP method results are presented (see Figure 5.1) to serve as a baseline.

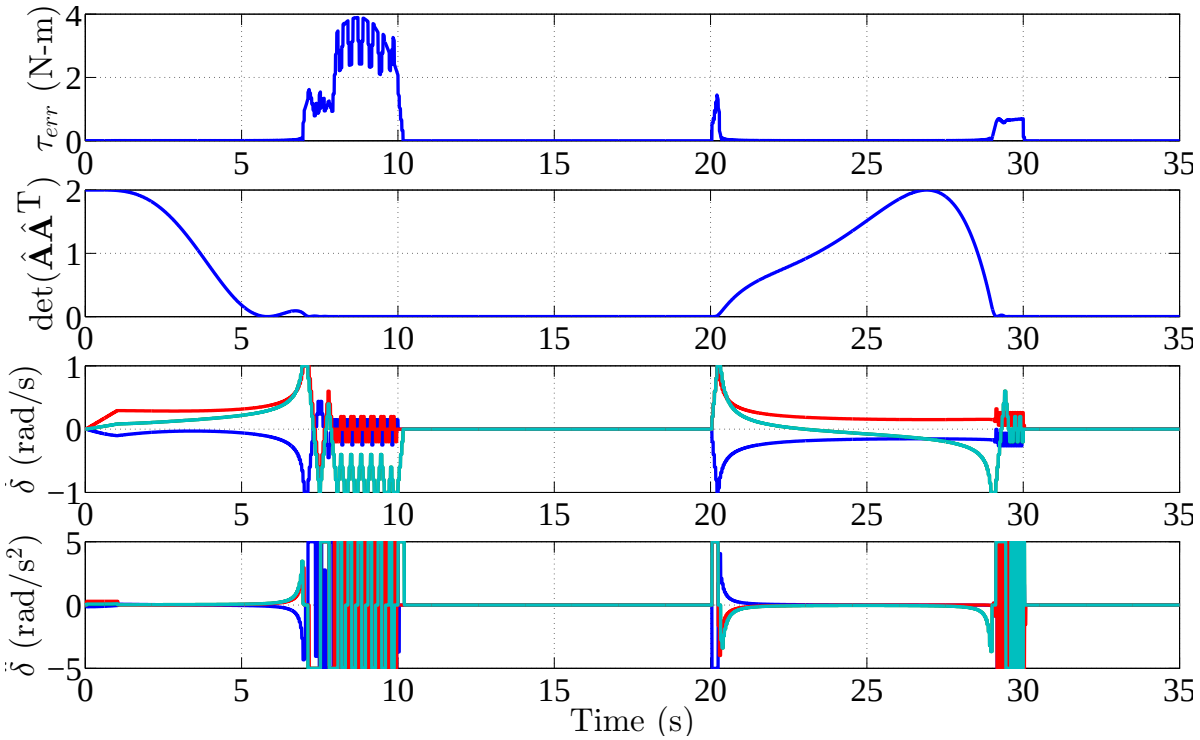


Figure 5.1: Saturated-MP solutions for $\dot{h}_{des}$ in (5.14). Total angular momentum error: 8.36 N-ms. Total energy: 13.72 J.

Although the analytic MP solutions represent minimum torque error solutions, this only applies when there are no actuator constraints. Once the MP solution is saturated, it is no longer a minimum torque error solution - as shown by the large angular momentum error over the 35 second simulation (8.36 N-ms of integrated torque error). Moreover, as the Jacobian matrix loses rank, the oscillations in the commanded gimbal rates become large since the assembly is losing control authority. Due to these oscillations, the energy used to actuate the gimbals is 13.72 J.

Second, the same slew is tested on the SDA steering law with $k_\sigma = 10$ and $\alpha_0 = 0.1$.

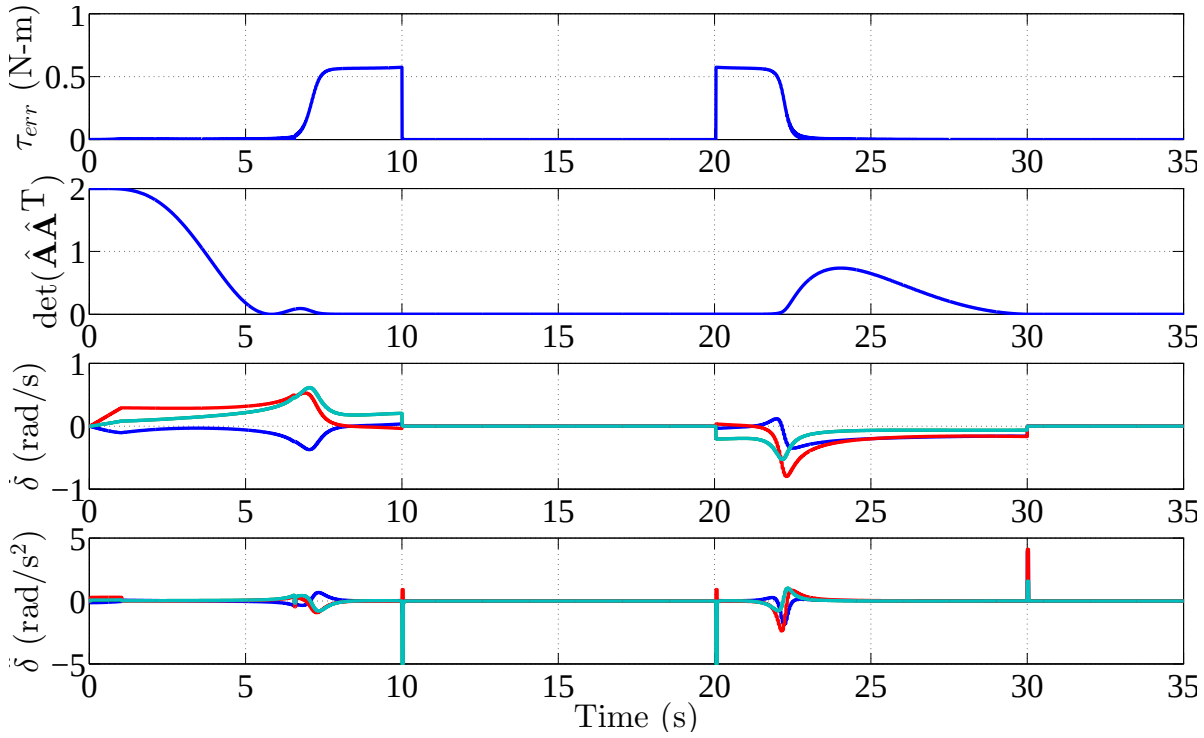


Figure 5.2: SDA solutions for $\dot{h}_{des}$ in (5.14). Total angular momentum error: 2.94 N-ms. Total energy: 5.23 J.

SDA performs well for this slew (Fig. 5.2), reducing the momentum error by 65% and energy use by 62% despite not explicitly considering the constraints.

Finally, the performance of the constrained LO-based MP is assessed after the commanded torque has gone through spectral scaling. The spectral scaling results in a 48% power reduction over the MP solution. The Jacobian does not lose rank at the 30 second mark (Fig. 5.3), further improving torque tracking performance. Overall, ME-DPRC yields 37% better torque tracking performance than SDA at a cost of 33% more power consumption. In general, a greedy algorithm like ME-DPRC cannot predict or control the long term behavior of the Jacobian. The oscillations seen in Figure 5.3 are lower frequency than the actuation frequency, thus the gimbals are not experiencing uncontrolled 'chatter'.

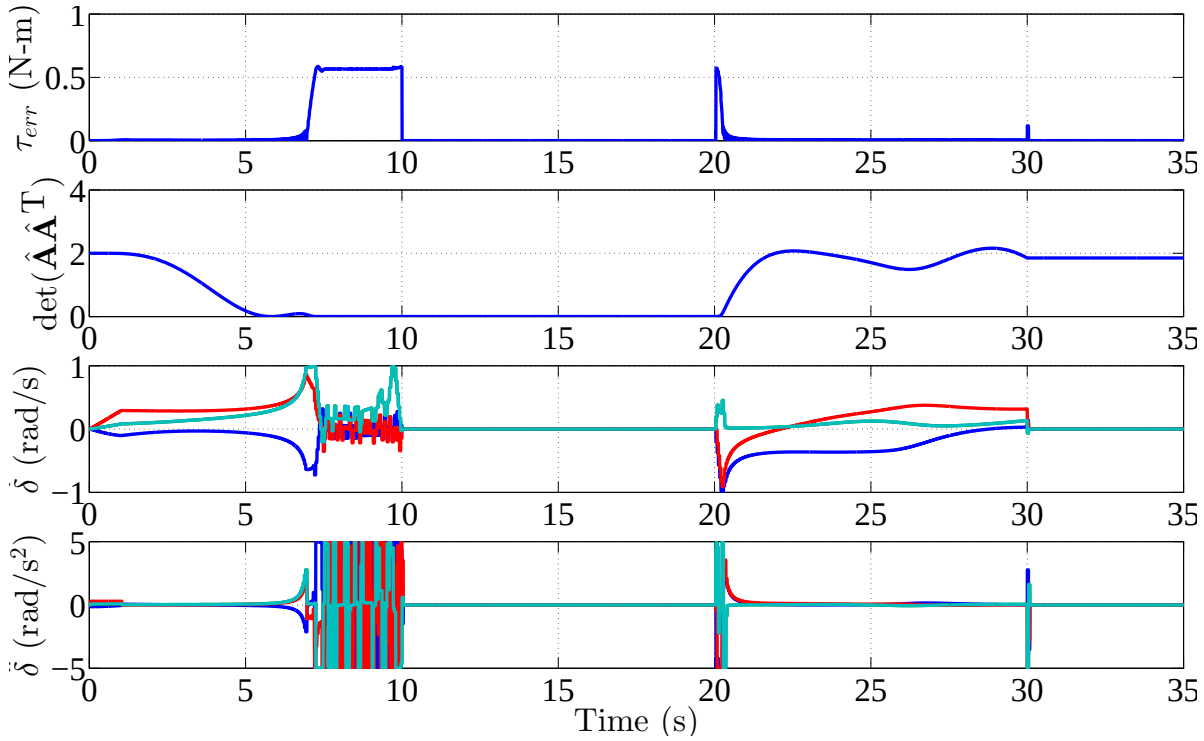


Figure 5.3: ME-DPRC results for $\dot{h}_{des}$ in (5.14). Total angular momentum error: 1.83 N-ms. Total energy used: 7.84 J.

High-amplitude Sinusoidal Torque Tracking

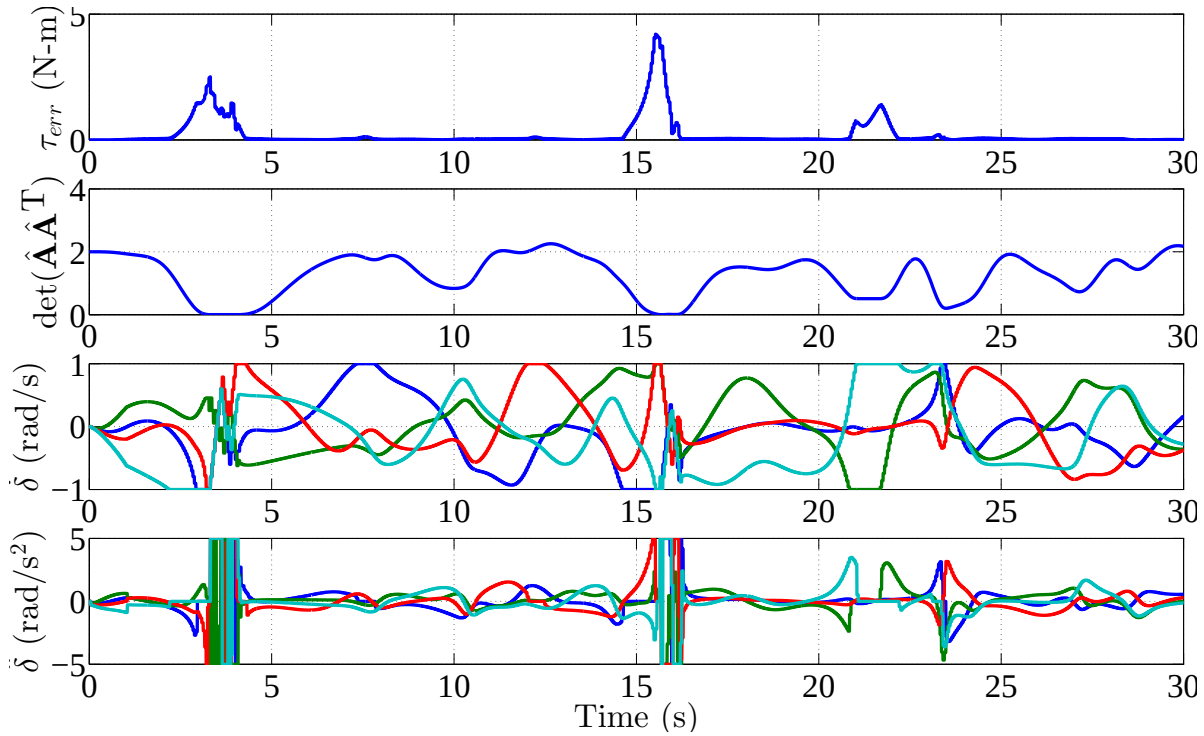


Figure 5.4: Saturated MP solutions for $\dot{h}_{des}$ in (5.15). Angular momentum error: 6.22 N-ms. Total energy: 14.58 J.

The following high-amplitude sinusoidal torque command is used to stress the allocation algorithm:

$$\dot{h}_{des}=\begin{cases}1.4t[\sin(\omega_0 t),\cos(2\omega_0 t),-\sin(1.5\omega_0 t)]^T\text{: } 0\leq t\leq 1\\ 1.4[\sin(\omega_0 t),\cos(2\omega_0 t),-\sin(1.5\omega_0 t)]^T : t>1\end{cases} \tag{5.15}$$

where $\omega_0=\frac{\pi}{5}$ rad/s.

The saturated analytic MP results are presented first. Saturation causes the overall solution to accumulate a significant amount (6.22 N-ms) of angular momentum error (Fig. 5.4) while consuming 14.58 J. Next, the SDA results are presented, which did not perform as well as the MP solution for the sinusoidal torque profile (Fig. 5.5), taking 22% more power

and accruing 23% more angular momentum error than MP. The ME-DPRC results show that the angular momentum error is significantly reduced (68%), but 3% more energy than the analytic MP is used to attain the improved torque-tracking (Fig. 5.6). If the gimbals had not saturated or approached singularities, both ME-DPRC and SDA produce the same control allocation as the analytic MP solution.

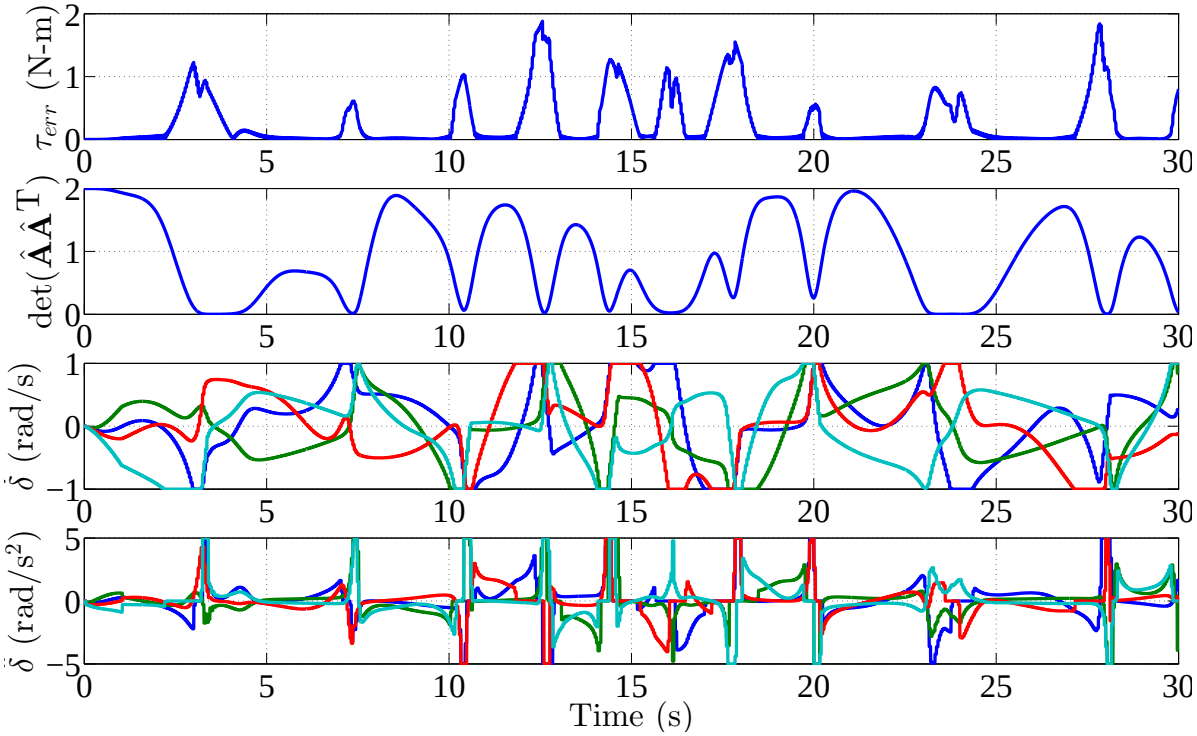


Figure 5.5: SDA solutions for $\dot{h}_{des}$ in (5.15). Total angular momentum error: 8.02 N-ms. Total energy: 18.57 J.

5.1.6 Assessment of computational efficiency

Custom solvers were used to obtain runtimes for the CMG examples above, resulting in an average runtime of 0.2 ms (5000 Hz update rate) of computation time per allocation on a laptop with 2.1 GHz Intel i7 processor and 8 GB of RAM. - making ME-DPRC real-time implementable.

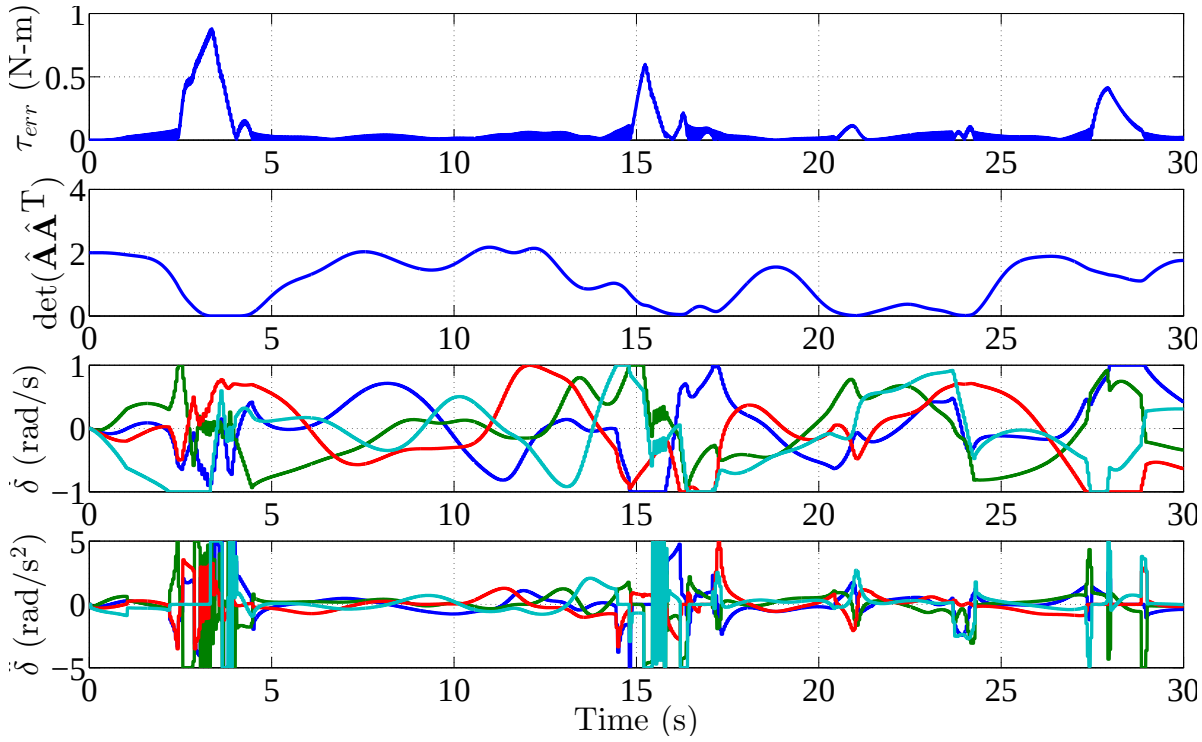


Figure 5.6: ME-DPRC results for $\dot{h}_{des}$ in (5.15). Total angular momentum error: 1.99 N-ms. Total energy used: 15.00 J.

Chapter 6

CONCLUSION

The overarching goal of this dissertation is to develop the tools necessary to democratize optimization-based approaches in the application space of highly autonomous systems. Much of the resistance to this type of approach in the aerospace industry, for instance, stems from a need to thoroughly verify any novel methods. To that end, this dissertation begins by presenting the details of an IPM solver implementation that has undergone a tremendous amount of verification on an aerospace system. Moreover, this document develops a framework for generating custom C solvers designed specifically for real-time use on safety-critical systems with limited computational resources. This solver customization research culminated with a set of flight tests using Masten's Xombie rocket, where it was used to compute fuel-optimal trajectories in real-time onboard the vehicle. The ability to solve optimization problems with realistic engineering constraints rapidly is a powerful tool that makes up the backbone of much of the results in the rest of the dissertation, and is one of its biggest contributions.

In order for a vehicle to be considered a system with high-level autonomy, it must be able to characterize its feasible state space and use it to identify an achievable goal state. Two methods are introduced in Chapter 3 that are capable of characterizing a vehicle's reachable and controllable sets via upper and lower bounding polytopic approximations. One of the major contributions of this work is that the second algorithm in Chapter 3 is mathematically shown to converge to the desired set; to the author's knowledge, this is the first such proof for iterative polytopic approximations. Thus, given top-level mission objectives, an autonomous agent would be capable of quickly determining which objectives are feasible, and then choose the feasible objective that maximizes some reward. Although the methods for computing

reachable and controllable sets shown here were implemented with a mix of C++ solvers (from Chapter 2) and MATLAB code, runtimes were on the order of a second for three dimensional state spaces. Thus, an implementation written entirely in C/C++ would be capable of better performance still. Note also, that selecting an objective is not a task that an autonomous system is likely to carry out at a high frequency, so the threshold for real-time implementability is more lenient than, say that of an inner control loop.

Once a goal state is identified, either through the high-level autonomy proposed in Chapter 3 or through direct operator inputs, processors onboard the vehicle can then compute an optimal trajectory that takes the vehicle from its current state to the goal state. Chapter 4 summarizes methods to accomplish such onboard trajectory generation and expands lossless convexification results in the context of the relevant aerospace application of planetary descent guidance. Further, it presents a method for leveraging successive convexification to avoid obstacles along the continous arc between discrete states with finitely many constraints. These methods are enough to encompass a broad range of the optimal control problems encountered in practice.

The final contributions of this dissertation focus on the role of optimization on a vehicle's low level control loops. More specifically, Chapter 5 develops a method for mapping desired control actions into individual actuator commands, while taking into account vehicle and actuator constraints. This capability allows control allocations to adhere more closely to the desired control actions, and also minimize power usage whenever possible. Furthermore, by leveraging the custom solvers described in Chapter 2, it shown that these allocations can be computed in 0.2 ms, or at a rate of 5000 Hz. Therefore this sort of method is very well suited to real-time, low level control loops.

6.1 Future Work

Due in part to the trend that processors are becoming faster and cheaper, and to larger industry trends towards valuing results over heritage, the future for real-time optimization seems promising. Further research needs to be carried out to develop custom solvers that

handle larger problem sizes without adversely impacting performance (the custom solvers in Chapter 2 are limited to problem classes with about 1000 solution variables). This can be accomplished either by directly generating assembly code that avoids cache misses, or by modifying IPM algorithms to more easily leverage the growing multi-core trend. Both avenues of exploration have the potential to dramatically increase the application space of real-time optimization in exciting ways.

One of the major drawbacks of the algorithms described in Chapter 3 is that the dynamics, and more generally the state constraints, must be linear in order to be cast as convex optimization problems. While a great many applications fall into this category (e.g., rockets and multi-rotor UAVs), there are some applications that cannot be accurately cast in this manner, such as applications that require the Dubins model for their dynamics. Thus, research should be carried out to develop a framework capable of handling non-convex state constraints. This capability would also be useful for situations in which obstacles must be avoided, similar to the problem tackled towards the end of Chapter 4.

Finally, although successive convexification is a very powerful tool for solving optimization problems with non-convex state constraints, it is only guaranteed to find a local optimum. The specific local optimum that the algorithm returns depends on the initial guess used to begin iterations. Further research should be carried out to characterize the state space in a manner that can return a global optimum after sampling a finite number of initial guesses.

BIBLIOGRAPHY

[1] Utku Eren, Daniel Dueri, and Behçet Açıkmeşe. Constrained reachability and controllability sets for planetary precision landing via convex optimization. *Journal of Guidance, Control, and Dynamics*, in press, 2015.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] Sanjay Mehrotra and Jie Sun. A method of analytic centers for quadratically constrained convex quadratic programs. *SIAM Journal on Numerical Analysis*, 28(2):529–544, 1991.

[4] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992.

[5] Y. Nesterov and A. Nemirovsky. *Interior-point Polynomial Methods in Convex Programming*. SIAM, 1994.

[6] J. Peng, C. Roos, and T. Terlaky. *Self-Regularity: A New Paradigm for Primal-Dual Interior-Point Algorithms*. Princeton Series in Applied Mathematics, 2001.

[7] Daniel Dueri, Jing Zhang, and Behçet Açikmese. Automated Custom Code Generation for Embedded, Real-time Second Order Cone Programming. In *IFAC Proceedings Volumes*, volume 47, pages 1605–1612. Elsevier, 2014.

[8] Alexander Domahidi, Eric Chu, and Stephen Boyd. ECOS: An SOCPSolver for Embedded Systems. In *European Control Conference (ECC)*, pages 3071–3076. IEEE, July 2013.

[9] Bixiang Wang. Implementation of Interior Point Methods for Second order Conic Optimization. Master's thesis, McMaster University, 2003.

[10] D. Bersekas. *Nonlinear Programming*. Athena Scientific, 2 edition, 1999.

[11] B. Açıkmeşe and S. R. Ploen. Convex programming approach to powered descent guidance for Mars landing. *AIAA Journal of Guidance, Control and Dynamics*, 30(5):1353–1366, 2007.

[12] L. Blackmore, B. Açıkmeşe, and D. P. Scharf. Minimum landing error powered descent guidance for Mars landing using convex optimization. *AIAA Journal of Guidance, Control and Dynamics*, 33(4), 2010.

[13] B. Açıkmeşe, J.M. Carson, and L. Blackmore. Lossless convexification of non-convex control bound and pointing constraints of the soft landing optimal control problem. *IEEE Transactions on Control Systems Technology*, 21(6):2104–2113, 2013.

[14] Matthew W Harris and Behçet Açıkmeşe. Maximum divert for planetary landing using convex optimization. *Journal of Optimization Theory and Applications*, 162(3):975–995, 2014.

[15] B. Açıkmeşe and L. Blackmore. Lossless convexification of a class of optimal control problems with non-convex control constraints. *Automatica*, 47(2):341–347, 2011.

[16] M.W. Harris and B. Açıkmeşe. Lossless convexification of non-convex optimal control problems for state constrained linear systems. *Automatica*, 50(9):2304–2311, 2014.

[17] Lars Blackmore. Autonomous precision landing of space rockets. *The Bridge*, 4(46):15–20, 2016.

[18] J. S. Meditch. On the problem of optimal thrust programming for a lunar soft landing. *IEEE Transactions on Automatic Control*, AC-9(4):477–484, 1964.

[19] A. R. Klumpp. Apollo lunar descent guidance. *Automatica*, 10:133–146, 1974.

[20] F. Najson and K. D. Mease. Computationally inexpensive guidance algorithm for fuel-efficient terminal descent. *Journal of Guidance, Control, and Dynamics*, 29(4):955–964, 2006.

[21] U. Topcu, J. Casoliva, and K. D. Mease. Minimum-fuel powered descent for mars pinpoint landing. *Journal of Spacecraft and Rockets*, 44(2):324–331, 2007.

[22] B. A. Steinfeld, M. J. Grant, D. A. Matz, R. D. Braun, and G. H. Barton. Guidance, navigation and control system performance trades for mars pinpoint landing. *Journal of Spacecraft and Rockets*, 47(1):188–198, 2010.

[23] S.R. Ploen, B. Açıkmeşe, and A. Wolf. A comparison of powered descent guidance laws for Mars pinpoint landing. *AIAA GNC Conference and Exhibit, Keystone, CO*, 2006.

[24] Aron A Wolf, Jeff Tooley, Scott Ploen, Mark Ivanov, Behcet Acikmese, and Konstantin Gromov. Performance trades for mars pinpoint landing. In *Aerospace Conference, 2006 IEEE*, pages 16–pp. IEEE, 2006.

[25] Daniel Dueri, Jing Zhang, and Behçet Açikmese. Automated Custom Code Generation for Embedded, Real-time Second Order Cone Programming. In *IFAC Proceedings Volumes*, volume 47, pages 1605–1612. Elsevier, 2014.

[26] J. Mattingley and S. Boyd. Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[27] B. Açıkmeşe, M. Aung, J. Casoliva, S. Mohan, A. Johnson, D. Scharf, D. Masten, J. Scotkin, A. Wolf, and M. W. Regehr. Flight testing of trajectories computed by G-FOLD: Fuel optimal large divert guidance algorithm for planetary landing. In *AAS/AIAA Spaceflight Mechanics Meeting*, 2013.

[28] D. P. Scharf, M. W. Regehr, B. Açıkmeşe, D. Dueri, G. M. Vaughan, J. Benito, H. Ansari, A. Johnson M. Aung, D. Masten, S. Nietfeld, J. Casoliva, and S. Mohan. ADAPT demonstrations of onboard large-divert guidance with a vtvl rocket. *IEEE Aerospace Conference*, 2014.

[29] JPL, Masten Space Systems, and University of Texas. First flight testing of real-time G-FOLD, Guidance for Fuel Optimal Large Divert, validation. `http://www.jpl.nasa.gov/news/news.php?release=2013-247`, August 2013.

[30] P. Lu and X. Liu. Autonomous trajectory planning for rendezvous and proximity operations by conic optimization. *AIAA Journal of Guidance, Control and Dynamics*, 36(2), March-April 2013.

[31] Y. Kim and M. Mesbahi. Quadratically constrained attitude control via semidefinite programming. *IEEE Transactions on Automatic Control*, 49(5), May 2004.

[32] U. Lee and M. Mesbahi. Quaternion-based optimal spacecraft reorientation under complex attitude constrained zones. In *AAS/AIAA Astrodynamics Specialist Conference*. AAS/AIAA, 2013.

[33] B. Açıkmeşe, D. Scharf, E. Murray, and F. Hadaegh. A convex guidance algorithm for formation reconfiguration. In *AIAA Guidance, Navigation, and Control*, Keystone, USA, 08/2006 2006.

[34] B. Açıkmese. Application of lexicographic goal programming with convex optimization in control systems. *AIAA Guidance, Navigation, and Control Conference*, 2013.

[35] Daniel Dueri, Frederick Leve, and Behçet Açıkmeşe. Minimum error dissipative power reduction control allocation via lexicographic convex optimization for momentum control systems. *IEEE Transactions on Control Systems Technology*, 24(2):678–686, 2016.

[36] Daniel Dueri, Behçet Açıkmeşe, Daniel P Scharf, and Matthew W Harris. Customized real-time interior-point methods for onboard powered-descent guidance. *Journal of Guidance, Control, and Dynamics*, pages 1–16, 2016.

[37] Daniel Dueri, Saša V Raković, and Behçet Açıkmeşe. Consistently improving approximations for constrained controllability and reachability. In *Control Conference (ECC), 2016 European*, pages 1623–1629. IEEE, 2016.

[38] Daniel P Scharf, Behçet Açıkmeşe, Daniel Dueri, Joel Benito, and Jordi Casoliva. Implementation and experimental demonstration of onboard powered-descent guidance. *Journal of Guidance, Control, and Dynamics*, pages 1–17, 2016.

[39] Daniel Dueri, Yuanqi Mao, Zohaib Mian, Jerry Ding, and Behçet Açikmeşe. Trajectory optimization with inter-sample obstacle avoidance via successive convexification. In *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*, pages 1150–1156. IEEE, 2017.

[40] Gavin F Mendeck and Lynn E Craig. Entry guidance for the 2011 mars science laboratory mission. In *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, pages 8–11, 2011.

[41] P. O. M. Scokaert, D. Q. Mayne, and J. B. Rawlings. Suboptimal model predictive control (feasibility implies stability). *IEEE Transactions on Automatic Control*, 44(3):648–654, Mar 1999.

[42] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[43] P. Lu. Entry guidance: A unified method. *AIAA Journal of Guidance, Control, and Dynamics*, 37(3):713–728, 2014.

[44] Colin N Jones and Manfred Morari. Polytopic approximation of explicit model predictive controllers. *Automatic Control, IEEE Transactions on*, 55(11):2542–2553, 2010.

[45] Alberto Bemporad and Carlo Filippi. An algorithm for approximate multiparametric convex programming. *Computational optimization and applications*, 35(1):87–108, 2006.

[46] Colin Neil Jones, Miroslav Barić, and Manfred Morari. Multiparametric linear programming with applications to control. *European Journal of Control*, 13(2):152–170, 2007.

[47] J. Mattingley and S. Boyd. Real-time convex optimization in signal processing. *IEEE Signal Processing Magazine*, 27(3):50–61, 2010.

[48] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for Embedded Systems. In *Proceedings European Control Conference*, 2013.

[49] Damian Frick, Alexander Domahidi, and Manfred Morari. Embedded optimization for mixed logical dynamical systems. *Computers & Chemical Engineering*, 72(0):21 – 33, 2015. A Tribute to Ignacio E. Grossmann.

[50] D.P. Scharf, B. Açikmeşe, D. Dueri, J. Casoliva, and J. Benito. Implementation and experimental demonstration of onboard powered descent guidance. to be submitted to *AIAA Journal of Guidance, Control, and Dynamics*, 2015.

[51] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[52] J Löfberg. YALMIP 3. *http://control.ee.ethz.ch/ joloef/yalmip.msql*, 2004.

[53] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[54] Stephen J Wright. *Primal-dual interior-point methods*, volume 54. Siam, 1997.

[55] Jorge Nocedal and Stephen J Wright. Springer series in operations research. numerical optimization, 1999.

[56] L Vandenberghe. The cvxopt linear and quadratic cone program solvers. *Online: http://www.seas.ucla.edu/ vandenbe*, 2010.

[57] Y. Nesterov and A. Nemirovski. *Interior Point Polynomial Algorithms in Convex Programming*. SIAM, 1994.

[58] Yu E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, February 1997.

[59] William W. Hager. Updating the inverse of a matrix. *SIAM*, 31(2):221–239, June 1989.

[60] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):381–388, September 2004.

[61] B. Açıkmeşe and L. Blackmore. Lossless convexification for a class of optimal control problems with nonconvex control constraints. *Automatica*, 47(2):341–347, 2011.

[62] R. H. Tutuncu, K. C. Toh, and Michael J. Todd. Solving semidefinite-quadratic-linear problems using SPDT3. *Mathematical Programming*, 2002.

[63] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. Version 1.05 available from `http://fewcal.kub.nl/sturm`.

[64] D.P. Scharf, S.R. Ploen, and B. Açikmeşe. Interpolation-enhanced powered descent guidance for onboard nominal, off-nominal, and multi-x scenarios. In to appear in *AIAA Guidance, Navigation, and Control Conf.*, number 2015-0850, 2015.

[65] F. C. Schweppe. *Uncertain Dynamic Systems*. Prentice Hall, Englewood Cliffs, NJ, 1973.

[66] A. Kurzhanski and I. Vályi. *Ellipsoidal Calculus for Estimation and Control*. Systems & Control: Foundations & Applications. Birkhauser, Boston, Basel, Berlin, 1997.

[67] J. P. Aubin. *Viability theory*. Systems & Control: Foundations & Applications. Birkhauser, Boston, Basel, Berlin, 1991.

[68] F. Blanchini and S. Miani. *Set–Theoretic Methods in Control*. Systems & Control: Foundations & Applications. Birkhäuser, Boston, Basel, Berlin, 2008.

[69] Saša V. Raković and Mirko Fiacchini. Approximate reachability analysis for linear discrete time systems using homothety and invariance. In *Proceedings of the 17th IFAC World Congress IFAC 2008*, Seoul, Korea, 2008.

[70] Saša V Raković, Eric C Kerrigan, David Q Mayne, and John Lygeros. Reachability analysis of discrete-time systems with disturbances. *Automatic Control, IEEE Transactions on*, 51(4):546–561, 2006.

[71] Sasa V Rakovic, Ion Matei, and John S Baras. Reachability analysis for linear discrete time set-dynamics driven by random convex compact sets. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 4751–4756. IEEE, 2012.

[72] L. Blackmore, M. Ono, and B.C. Williams. Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics*, 27(6):1080–1094, 2011.

[73] L. Blackmore, M. Ono, A. Bektassov, and B.C. Williams. A probabilistic particle-control approximation of chance-constrained stochastic predictive control. *Robotics, IEEE Transactions on*, 26(3):502–517, 2010.

[74] M. Ono, L. Blackmore, and B. C. Williams. Chance constrained finite horizon optimal control with nonconvex constraints. In *in Proceedings of the American Control Conference*, 2010.

[75] Daniel Dueri, Behcet Acikmese, Morgan Baldwin, and R Scott Erwin. Finite-horizon controllability and reachability for deterministic and stochastic linear control systems with convex constraints. In *American Control Conference (ACC), 2014*, pages 5016–5023. IEEE, 2014.

[76] Jun Liu, N. Ozay, U. Topcu, and R.M. Murray. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Transactions on Automatic Control*, 58(7):1771–1785, 2013.

[77] J. Mattingley and S. Boyd. *Automatic Code Generation for Real-Time Convex Optimization. Convex Optimization in Signal Processing and Communications, Y. Eldar and D. Palomar, Eds.* Cambridge University Press, 2010.

[78] GD Chakerian and JR Sangwine-Yager. Synopsis and exercises for the theory of convex sets. 2009.

[79] Michael Szmuk and Behçet Açıkmeşe. Successive convexification for fuel-optimal powered landing with aerodynamic drag and non-convex constraints. In *AIAA Guidance, Navigation, and Control Conference*, page 0378, 2016.

[80] Yuanqi Mao, Michael Szmuk, and A Acikmese Behcet. Successive convexification of non-convex optimal control problems and its convergence properties. In *IEEE Conference on Decision and Control*, December 2016.

[81] L. S. Pontryagin, R. V. Gamkrelidze V. G. Boltyanskii, and E. F. Mischenko. *The Mathematical Theory of Optimal Processes.* Pergamon Press, 1964.

[82] L. D. Berkovitz. *Optimal Control Theory.* Springer-Verlag, 1975.

[83] A.A. Milyutin and N.P. Osmolovskii. *Calculus of Variations and Optimal Control.* American Mathematical Socity, 1998.

[84] Masten Space Systems JPL and University of Texas at Austin. 750 meter divert Xombie test flight for G-FOLD, Guidance for Fuel Optimal Large Divert, validation. `http://www.jpl.nasa.gov/video/?id=1270`, September 2013.

[85] Behçet Açıkmese, Lars Blackmore, Daniel P Scharf, and Aron Wolf. Enhancements on the convex programming based powered descent guidance algorithm for mars landing. In *AIAA/AAS Astrodynamics Specialist Conf.*, number AAS 2008-6426. AAS/AIAA, 2008.

[86] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104, 2010.

[87] Liang Yang, Juntong Qi, Jizhong Xiao, and Xia Yong. A literature review of uav 3d path planning. In *Intelligent Control and Automation (WCICA), 2014 11th World Congress on*, pages 2376–2381. IEEE, 2014.

[88] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[89] Arthur Richards, Tom Schouwenaars, Jonathan P How, and Eric Feron. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *Journal of Guidance, Control, and Dynamics*, 25(4):755–764, 2002.

[90] Christof Büskens and Helmut Maurer. Sqp-methods for solving optimal control problems with control and state constraints: adjoint variables, sensitivity analysis and real-time control. *Journal of computational and applied mathematics*, 120(1):85–108, 2000.

[91] Xinfu Liu and Ping Lu. Solving nonconvex optimal control problems by convex optimization. *Journal of Guidance, Control, and Dynamics*, 37(3):750–765, 2014.

[92] Yuanqi Mao, Daniel Dueri, Michael Szmuk, and A Acikmese Behcet. Successive convexification of non-convex optimal control problems with state constraints. In *submitted to International Federation of Automatic Control*, November 2016.

[93] Andrew R Conn, Nicholas IM Gould, and Ph L Toint. *Trust region methods*, volume 1. Siam, 2000.

[94] Frederick Leve. Evaluation of steering algorithm optimality for single-gimbal control moment gyroscopes. *IEEE Transactions on Control System Technology (to appear)*, 2013.

[95] David S Bayard. An optimization result with application to optimal spacecraft reaction wheel orientation design. In *American Control Conference*, volume 2, pages 1473–1478. IEEE, 2001.

[96] SR Vadali and JL Junkins. Spacecraft large angle rotational maneuvers with optimal momentum transfer. *Journal of the Astronautical Sciences*, 31(2):217–235, 1983.

[97] SB Skaar and LG Kraige. Large-angle spacecraft attitude maneuvers using an optimal reaction wheel power criterion. *Journal of the Astronautical Sciences*, 32(1):47–61, 1984.

[98] Xipu Li and Santosh Ratan. Optimal speed management for reaction wheel control system and method, April 3 2007. US Patent 7,198,232.

[99] Hanspeter Schaub and Vaios J Lappas. Redundant reaction wheel torque distribution yielding instantaneous l2 power-optimal spacecraft attitude control. *Journal of guidance, control, and dynamics*, 32(4):1269–1276, 2009.

[100] Dohee Kim, William MacKunis, N Fitz-Coy, and Warren E Dixon. Precision ipacs in the presence of dynamic uncertainty. In *Conference on Decision and Control. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 5959–5964. IEEE, 2009.

[101] D Kim, FA Leve, N Fitz-Coy, and WE Dixon. Integrated power reduction and adaptive attitude control system of a vscmg-based satellite. *Spaceflight Mechanics*, 140, February 2011.

[102] Hyungjoo Yoon and Panagiotis Tsiotras. Spacecraft adaptive attitude and power tracking with variable speed control moment gyroscopes. *Journal of Guidance, Control, and Dynamics*, 25(6):1081–1090, 2002.

[103] M. Tamiz, D. F. Jones, and C. Romero. Goal programming for decision making: An overview of the current state-of-the-art. *European Journal of Operation Research*, 111:569–581, 1998.

[104] Kevin A Ford and Christopher D Hall. Singular direction avoidance steering for control-moment gyros. *Journal of Guidance, Control, and Dynamics*, 23(4):648–656, 2000.

Appendix A
BSOCP USER'S GUIDE

A.1 Installing BSOCP

Before attempting to install `BSOCP`, make sure you are able to compile code on your operating system. For Linux and Mac, this means having up-to-date versions of `g++` and `gcc` installed. For Windows, this means either having Cygwin or the Linux Subsystem for Windows installed. Additional steps may be required for use with MATLAB, as discussed below. In the following sections, specific instructions will be given for different operating systems.

The solver comes in two flavors: 1) a stand-alone generic solver that will solve any properly parsed SOCP; 2) a stand-alone generic solver that generates a custom solver for the problem class defined by the input problem. The latter uses an identical interface to the former, so it can easily be swapped out for the generic solver when the time comes to generate a customized solver.

A.1.1 MATLAB

`BSOCP` interfaces with MATLAB via its built in `MEX` functionality. As a consequence, it is necessary to first configure your MATLAB installation with a compatible `MEX` compiler. See `https://www.mathworks.com/support/compilers.html` for details on how to set up a compiler and which compilers are suitable for your operating system and MATLAB version. Once this step is done, however, `BSOCP` will be available for use regardless of your operating system.

To begin, run `compile_Bsocp.m` to compile the generic stand-alone solver and `compile_Bsocp_gen.m` to compile the stand-alone generic solver that generates custom code. These functions create either `.mexw64` files or `.mexw32` files depending on the architecture of

your computer. Place these files in your MATLAB path or in the specific directories you will be calling them from. Once a custom solver has been generated, a `compile_Bsocp_custom.m` function has been provided for convenience, which will compile the generated solvers into an executable in a similar fashion.

A.1.2 Linux and Mac

A `makefile` is included with `Bsocp` to facilitate compilation on Unix environments. Typing `make` in the directory containing `Bsocp` will create `libCSOCP.a` for use in the linking stage of programs that use the solver. Similarly, running `make libgen` will create `libCSOCPgen.a` for use in the linking stage of programs that generate custom solvers.

A.1.3 Windows

The solver does not support a native Windows installation, but can be compiled for use on MATLAB via the `MEX` functionality as described in Section A.1.1. For stand-alone use without MATLAB, `BSOCP` can be run via Cygwin or the Linux Subsystem for Windows (available in Windows 10) - in either case, the Linux and Mac instructions in section A.1.2 apply for installation purposes.

A.2 Convergence Parameters

There are a number of convergence parameters that define the termination conditions of the solver. This section defines their purpose and proposes some default values that should work in a majority of numerical conditions. As always, pre-conditioning the problem definition (i.e., ensuring the dynamic range of values in A, b, c is small) improves convergence performance. For example, this can be accomplished via problem-specific non-dimensionalizations (see [1] for more details).

A.2.1 Primary Convergence Parameters

This section deals with `RhoAb`, `RhoPb`, `RhoDb`, `RhoGb`, `RhoIb`, `Reps`, and `RelEps`. `RhoAb` is related to r_g and β from (2.4) in Chapter 2 and is a measure of the numeric tolerance on the constraints. While the following relationship is not exactly correct, the following expression is a reasonable approximation: $\|Ax^* - b\|_1 \leq$ `RhoAb` for optimal solution x^* and constraints given by A, b as in (2.1). r_p and r_d from (2.4) are residuals that are zero for feasible solutions of the primal and dual respectively. Since zero is never achieved exactly in numeric algorithms, `RhoPb` and `RhoDb` are the tolerances below which the corresponding residual is considered zero. Similarly, `RhoGb` is a measure of the tolerance on the duality gap residual, r_g. By Theorem 1, τ and κ in (2.4) cannot both go to zero or the problem was ill posed - `RhoIb` is the tolerance on how low both terms can get before the solver stops iterating. `Reps` is a measure of numerical instability and corresponds to the smallest value ν in (2.4) can attain before the solver terminates with a numeric infeasibility flag. This is important because a number of operations involve dividing by ν. In most cases, the solution here is to reduce the dynamic range of A, b, c. The final convergence parameter, `RelEps` deals with the ratio of the residual terms in (2.4) (e.g., r_p, r_d, r_g, etc) at the current iteration and their value after the first iteration - once any given ratio falls below `RelEps`, it is considered to be zero by the solver.

A.2.2 Newton Step Parameters

The parameters in this section are used to take Newton steps during each iteration: `BIG`, `gammaC`, `alphaB`, and `alpha_max`. `BIG` should be a large number compared to the values in A, b, c. `gammaC` is a centering parameter that scales the size of the computed Newton step, α such that $\alpha = \min($`gammaC`$\cdot\alpha$, `alpha_max`$)$. `alphaB` is used to initialize the Newton step, where 0.98`alphaB` is the initial value used for every primal and dual step size.

A.2.3 Central Path Parameter

The parameter `betaC` should always be set to $m + l + 1$ (where m is the number of second order cones and l is the number of linear cones), and is the denominator of μ_0 in (2.5). It helps define the central path that the algorithm will follow.

A.2.4 Suggested Parameters

Table A.1 is a list of suggested parameters that should cause the solver to converge in most cases. In the author's experience, appropriate scaling of A, b, c is sufficient to enable the use of these suggested parameters. However, depending on the dynamic range of the inputs, it may be necessary to increase parameters like `RhoAb` to values as large as 0.001. In such cases, it is recommended to scale `RhoAb`, `RhoPb`, and `RhoDb` together. Similarly, `RhoGb` and `RhoIb` should be scaled together. Parameters in parenthesis typically do not need to be changed.

Parameter	Value
`RhoAb`	1.0e-6
`RhoPb`	1.0e-7
`RhoDb`	1.0e-7
`RhoGb`	1.0e-9
`RhoIb`	1.0e-9
`Reps`	1.0e-10
`RelEps`	(1.0e-3)
`BIG`	(1.0e+6)
`gammaC`	(0.999)
`alphaB`	(0.999)
`alpha_max`	(1.0)

Table A.1: Suggested Solver Parameters