Computational Guidance and Control for Aerospace Systems

Taylor Patrick Reynolds

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

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Mehran Mesbahi, Chair

Behçet Açıkmeşe

Brian Fabien

Program Authorized to Offer Degree:
William E. Boeing Department of Aeronautics & Astronautics

University of Washington

Abstract

Computational Guidance and Control for Aerospace Systems

Taylor Patrick Reynolds

Chair of the Supervisory Committee:
Professor Mehran Mesbahi
William E. Boeing Department of Aeronautics & Astronautics

The objective of this dissertation is to develop new techniques that advance the state of the art in optimization-based trajectory generation. Two complementary techniques are studied. First, explicit trajectory generation computes a single path that connects two boundary conditions. For a general optimal control problem, sequential convex programming is used to design iterative algorithms that solve challenging aerospace problems. The limited power available on a spacecraft has long been at odds with the computationally demanding algorithms required to solve such problems, and so specialized techniques for developing real-time capable implementations of these algorithms are presented. Runtime analysis offers initial evidence that it is possible to solve explicit trajectory optimization problems reliably and fast enough to be considered a viable technology. As an alternative approach, implicit trajectory generation computes a set of functions that implicitly define an entire set of trajectories. By carrying out more extensive offline computations, it is shown that a feasible trajectory can be obtained from a wide array of initial conditions by using numerical integration. Consequently, the required real-time computations are significantly reduced compared to explicit trajectory optimization algorithms. Implicit trajectory generation methods can also offer a stronger theoretical, and offline-certifiable, guarantee that a feasible trajectory will be available for a prescribed set of vehicle conditions. Examples in powered descent and satellite attitude control are used to demonstrate each method.

To Jonathan Eric MacNaughton,

There's no simple explanation
For anything important any of us do
And yeah, the human tragedy
Consists in the necessity
Of living with the consequences
– Courage, The Tragically Hip

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

To Mom and Dad, thank you for creating a life for me where I could be free to go as far as my capabilities would let me. Thank you for pushing me when I needed to be pushed, and for being patient when patience was needed. None of this is possible without the support that you've given me, and I can't imagine thanking anyone but the two of you first and foremost. It will take me a lifetime to show you how truly thankful I am.

To my brother Michael, thank you for being my original and most influential motivator. From walking and talking all the way through to present day, you've been a role model to me, and many of things that I've done have been modeled off of your accomplishments. Thank you for always looking out for me, and running through life first so that my path was easier to walk. Your work ethic and perseverance are traits that I'd be lucky to emulate.

Thank you to my grandparents, Victor Alan and Leona Moore, and Norman and Margie Reynolds, for always encouraging us to pursue education. My maternal grandfather, Victor Alan Moore ("Pat" to many people, "Grandpa" to me), an engineer himself, advised me to pursue advanced mathematics while I could during my undergraduate education. So began my path towards control theory. Margie Reynolds ("Nana" to me) wrote a letter to the family near the end of her life, where she communicated her goals for each of us: to be hardworking and educated enough to have a decent life. I hope that I have made you each proud.

There are a few special teachers from my early education that stand out and deserve praise. Thank you Elizabeth Turcke for teaching the first course in mathematics that I truly enjoyed, you kickstarted a passion that persists to this day. Thank you David Loken for being so excited about physics that we the students had to follow suit. Thank you Stephanie Howie for helping me grow up. Thank you Saber Jafarpour for teaching the first course in

Control Theory that I took. I can recall very clearly the moment during one of your lectures that I realized that a career as a control systems engineer was what I wanted to do.

To Alec Johnston, Boris Baker, and Cole Byvelds, thank you for keeping me grounded and for being my best friends. Thank you for being roommates, drinking buddies, supporters, travel companions, and so much more. We've been through a lot together, and share bonds that will never be broken by time or distance. I look forward to a time when life brings us back together.

To Miki Szmuk, someone that I'm fortunate enough to consider both a mentor and friend, it's an understatement to say that I've benefited tremendously from working with you. Your fingerprints are all over this work. Above any technical matters, thank you for teaching me to be conscientious, to think analogically, and apparently some English grammar. I would need another few years to finish my degree without your guidance, and Renee thanks you as much as I do for that.

To Danylo Malyuta, the most capable learner and good-hearted person that I've had the opportunity to work with, thank you for being the best collaborator that I can image. I am consistently amazed at the breadth and depth of your interest and knowledge. It's both inspiring and humbling, and I'm truly lucky to have met and learned from you.

To Krish Kaycee, one of the best engineers that I have been fortunate enough to learn from, thank you for giving up your Saturdays for nearly two years, and more, to teach me spacecraft control from scratch. There are few people so generous, and I will do whatever I can to pay it forward. My work has been tremendously improved by the first principles approach to problem solving and narrative story telling that I learned from you. I'm lucky to consider you a friend.

To Charlie Kelly and James Rosenthal, thank you for making grad school fun. From card games until the clock and Rolling Rock ran out, to humoring me with interest in hockey games, I can't imagine these four years without you guys. Thank you for working hard to

make our dream of a building a satellite together a reality. Without you guys, I would still be complaining to Mehran that there should be a CubeSat program in our department. I'm looking forward to that three-family retirement home in Arlington where we'll finally start our satellite company.

To John Carson, thank you for providing me with the opportunity to work on a project that will contribute to something as awesome as SPLICE. I've always loved reading mentions of the "graduate students across the country" that worked on Apollo guidance, and it was amazing to play a small role in that capacity for SPLICE guidance. I will always appreciate the irony that these students became Draper, to whom I was the graduate student across the country. Thank you for your mentorship, it has meant a great deal to me.

To Behçet Açıkmeşe, thank you for your energy and constant influx of ideas – you have made my time here a great deal of fun. It was a highlight of my degree for us to do parallel "internships" and spend hours batting (sometimes obscure) ideas back and forth. It was basic research at its finest. I will look forward to you stumbling into the coffee shop where I'm working for many years to come.

To Mehran Mesbahi, thank you for taking a chance on me. I feel very privileged that you allowed me to explore the galaxy of control theory during my first few years (sometimes haphazardly), and earnestly encourage me to continuing doing so to this day. That freedom has allowed me to complete my Ph.D. without it ever feeling like work. I'm inspired by your gentle leadership style, a combination of humour, compassion, deceptively sharp insight, and unwavering support. You've often say that optimization is a lot like life, and I'm afraid that I may have learned more about life during my degree than I did about optimization. I could not ask for a better guide, and I am lucky to have had the opportunity to learn from you.

For that I also need to thank Bahman Gharesifard, profusely, as it was he who suggested that I forgo any other program that might admit me and go to Seattle to work with Mehran if I got the chance. I had no way of knowing that it was the best advice I would get during

that period of my life, and I will be forever grateful that I eventually listened.

To Renee, the love of my life, thank you for being my most ardent supporter throughout my degree. Thank you for putting up with the stress of a long distance relationship, a cross-country move, and constant trips back and forth to see each other. Thank you for being the emotional complement to my practicality, and for encouraging me to know the world outside of my academic bubble. Thinking about our future together motivated me to work harder in order to make it our present. You deserve it.

In a journey that has lasted 26 years and counting, I am certain to have missed thanking people that I should be thanking. I hope they will forgive me.

NOMENCLATURE

CGC: Computational Guidance and Control

DMI: Differential Matrix Inequality

DOF: Degree of Freedom

GNC: Guidance, Navigation, and Control

LMI: Linear Matrix Inequality

LOS: Line of Sight

PDG: Powered Descent Guidance

PTR: Penalized Trust Region

SCP: Sequential Convex Programming

SOC(P): Second-Order Cone (Program)

STC: State-Triggered Constraint

UW: University of Washington

Chapter 1

INTRODUCTION

The primary objective of this dissertation is to solve complex guidance problems for aerospace systems in ways that are conducive to onboard implementation. A focus is given in particular to problems that relate to spacecraft, such as orbiting satellites and planetary landers, but the methods can be applied much further afield. The term computational guidance and control (CGC) was recently coined to refer to techniques that are *algorithmic* in nature and rely on the real-time computation of control actions [1, 2]. The philosophy upheld in this dissertation is one that views the resulting architecture as the combination of new "computational guidance" methods and classical "control" methods, as a distinct view from "computational guidance" and "computational control". Classically, control refers to closed-loop disturbance rejection and the handling of model uncertainty, and feedback control is often designed to ensure stability about a reference path generated by a guidance routine.

The term guidance is used to refer to the process of generating a trajectory from a current state to some desired final state, and is synonymous with open-loop path planning. Trajectory optimization is a subset of guidance, and implies an additional desire for feasible trajectories that are near-optimal. Traditional guidance system design uses offline analysis to obtain a closed-form guidance law. The CGC methodology, by contrast, does not necessarily yield a closed-form mapping from state to control action. In other words, the guidance "laws" that result from CGC cause the reference state and control to be implicit functions of the problem data. This is, to varying degrees, a paradigm shift when it comes to control system design in the aerospace domain. We will show that this approach enables solutions to many difficult problems that are relevant to current and future aerospace vehicles and missions.

By difficult problems, one might think of highly nonlinear (i.e., more accurate) dynamic plant models and/or the explicit consideration of numerous state and control constraints. For these types of guidance problems, simply finding a feasible solution can be a challenge. Optimization-based methods happen to be quite good at finding feasible solutions, and have the obvious (and fortuitous) byproduct that the resulting trajectories are near-optimal. This thesis focuses on the class of CGC methods for which optimization is a core component, but adopts the viewpoint that feasibility is the driving objective. This interpretation is not new, yet it is not old either; but it is especially important when constructing real-time solutions to difficult problems.

There are, of course, some drawbacks to using optimization-based CGC. Foremost, it can only be used in practice when the solutions to the optimization problems can be computed fast enough relative to the timescale of the vehicle's motion that the control actions can be executed as intended. Ensuring real-time performance requires a bound on the algorithm's runtime for a desired numerical precision or convergence tolerance. This has long been (and continues to be) a fundamental challenge for algorithms designed to solve nonconvex optimal control problems. Second, the computational elements of CGC must inherit the rigorous mathematical analysis of classical and modern control. This means that algorithms should be designed intentionally using a first principles understanding of the problem at hand, convergence should be well understood both theoretically and numerically, and the relationship with additional feedback control well understood (which includes a systematic characterization of how robustness to system noise, time delays, and model uncertainty enter in to and are handled by the overall architecture). The field of computational guidance and control is just beginning to address these kinds of questions.

1.1 Statement of Contributions

This research focuses on two general approaches to CGC: explicit trajectory optimization and implicit trajectory optimization. An explicit trajectory optimization method, when implemented numerically, returns a complete trajectory from one state to another state by

solving an optimal control problem. In contrast, an implicit trajectory optimization method returns a number of functions that implicitly define a whole set trajectories. The delineation of these two approaches to trajectory design, as well as many of the definitions, problem statements, and solution methods presented for implicit trajectory optimization, are new contributions to the field of CGC. The two quadratic funnel synthesis algorithms that are presented for implicit trajectory generation offer two practical methods, and the convergence of each algorithm has been theoretically established.

Two central applications are carried throughout the text and used for numerical case studies: 6-DOF powered descent and spacecraft attitude control. Each problem has its own substantial literature and rich history of successful missions in which computational methods have played an important role, all of which will be reviewed thoroughly to place any new work in proper context. In the case of powered descent, a new theoretical solution is presented for a special (simplified) case of the 6-DOF landing problem, which we call the planar landing problem.

The numerical implementation of explicit trajectory optimization algorithms is then treated with real-time computation in mind. We provide a thorough and systematic approach to the transition from high-level algorithm development to low-level implementation, the specifics of which are a contribution of this work. It is with great pride that the work in this dissertation has contributed to technology demonstrations in both powered descent, aboard Blue Origin's New Shepard rocket, and in spacecraft attitude control, aboard a UW-built CubeSat. These demonstrations have provided a unique opportunity to advance the technology readiness level of specific optimization-based CGC algorithms to at least level six, and have paved the way for further use of CGC technologies in aerospace missions. Ultimately, we may conclude that feasible and near-optimal trajectories for both 6-DOF powered descent and constrained attitude control problems can be computed in real-time, while adhering to the standards of space flight software, and using hardware that is currently available.

1.2 Outline of Dissertation

This dissertation is organized as follows. First, Chapter 2 provides some background information that will serve as a common point of reference for each subsequent chapter. We introduce important assumptions and give formal statements of the tools from optimal control theory, convex optimization, and hypercomplex numbers that form core building blocks of our trajectory optimization methods. Next, Chapter 3 provides a thorough overview of powered descent guidance, including the 3-DOF problem, the planar landing problem, and the general 6-DOF problem. In the first two problems, we study and derive characteristics of the theoretically optimal solutions using the maximum principle. Chapter 3 also includes state-triggered constraints, and provides three examples that are relevant specifically to problems in powered descent.

Chapter 4 discusses the design of a sequential convex programming algorithm that is capable of solving nonconvex optimal control problems. We develop the algorithm using a general problem statement, and subsequently provide two case studies to demonstrate its application to the 6-DOF powered descent guidance problem with and without state-triggered constraints. Chapter 5 specializes the algorithm for real-time implementations that are designed to meet the standards of safety-critical space flight software. We provide a general strategy to transition from high-level algorithm development (e.g., the contents of Chapter 4) to low-level implementations with the performance metrics of runtime and computational memory utilization in mind. A case study in planar powered descent is provided.

Chapter 6 presents implicit trajectory optimization. We define the notion of a *funnel*, and demonstrate the use of time-varying quadratic Lyapunov functions in synthesizing a special class of funnels for nonlinear systems. Two algorithms are proposed to do this and each are proven to be convergent after a finite number of iterations. Case studies in powered descent and satellite attitude control are provided to demonstrate the utility of funnel synthesis and implicit trajectory optimization.

Lastly, Chapter 7 offers concluding remarks and several directions for future study.

1.3 Basic Notation

Most of the notation will be introduced as it is needed. This section will cover only the basic matrix-vector notation that is common to each chapter. The set of real numbers is denoted by $\mathbb{R}$. The sets of nonnegative and positive real numbers are denoted $\mathbb{R}_+$ and $\mathbb{R}_{++}$. Vectors are written using lowercase bold font, and $\boldsymbol{x} \in \mathbb{R}^n$ denotes an n-dimensional column vector of real numbers, with x_i as the i^{th} entry. When written in-line, a column vector may be denoted by $\boldsymbol{x} = (x_1, \ldots, x_n)$ to make good use of space. The transpose is denoted by $\boldsymbol{x}^\top$.

The cross product of two three-dimensional vectors $\boldsymbol{x}$, $\boldsymbol{y} \in \mathbb{R}^3$ is represented by $\boldsymbol{x}^\times \boldsymbol{y}$, where $\boldsymbol{x}^\times \in \mathbb{R}^{3\times 3}$ denotes a skew-symmetric matrix that when applied to the vector $\boldsymbol{y}$ produces the same mathematical result as the traditional expression $\boldsymbol{x} \times \boldsymbol{y}$. For two vectors $\boldsymbol{x}$, $\boldsymbol{y} \in \mathbb{R}^n$, the notation $\boldsymbol{x} \perp \boldsymbol{y}$ implies that $\boldsymbol{x}$ is orthogonal to $\boldsymbol{y}$ in the sense that $\boldsymbol{x}^\top \boldsymbol{y} = 0$. The notation $\boldsymbol{x} \parallel \boldsymbol{y}$ implies that the angle between the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is zero (they are parallel).

Matrices are written using uppercase symbols, and $M \in \mathbb{R}^{m\times n}$ denotes an m-by-n matrix of real numbers. The set of square symmetric matrices of dimension n is $\mathbb{S}^n$. The $n \times n$ identity matrix is denoted by I_n, while an m-by-n matrix of zeros is given by $0_{m\times n}$.

Derivatives with respect to time are denoted using the over-dot notation, so that $\frac{d\boldsymbol{x}}{dt} \equiv \dot{\boldsymbol{x}}$. The Jacobian of a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $\boldsymbol{x} \mapsto f(\boldsymbol{x})$ is denoted by $\nabla_{\boldsymbol{x}} f \in \mathbb{R}^{m\times n}$. The Hessian of a scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$ with $\boldsymbol{x} \mapsto f(\boldsymbol{x})$ is denoted by $\nabla^2_{\boldsymbol{xx}} f \in \mathbb{S}^n$. A vector norm is denoted by $\|\cdot\|_p$, where $p = 1, 2, \infty$.

Chapter 2
BACKGROUND

If in the first act you have hung a pistol on the wall,
then in the following one it should be fired.
– Anton Chekhov

The equations of motion for a physical system are expressed as differential equations. For problems in the field of CGC, these are differential equality constraints that all computed trajectories must obey. For guidance problems there are typically boundary conditions at the beginning and end of a trajectory that need to be met. Simply put, one aims to computes a trajectory from a current state to some final state while adhering to the physics that governs the system's motion. Once a suitable notion of "cost" is introduced, these are optimal control problems, and so this chapter begins with a review of optimal control theory in §2.1.

All numerical CGC methods in this work are based on convex optimization, the basics of which are introduced in §2.2. In §2.2.3, the connection between convex optimization and optimal control problems is introduced briefly, as this is a primary focus of Chapter 4. Hypercomplex numbers are then introduced in §2.3. Specifically, quaternions and dual quaternions are introduced, and it will be shown throughout this thesis that they are often an effective means to formulate optimal control problems for aerospace systems in ways that are conducive to the subsequent use of convex optimization to compute solutions.

2.1 Optimal Control Problems

The equations of motion for a physical system are assumed to be of the form

$$\dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\big), \quad t \in [t_0, t_f], \tag{2.1}$$

where $t_0, t_f \in \mathbb{R}_+$ represent the initial and final times respectively, $\boldsymbol{x}(\cdot) : \mathbb{R} \to \mathcal{X} \subset \mathbb{R}^{n_x}$ is the *state*, $\boldsymbol{u}(\cdot) : \mathbb{R} \to \mathcal{U} \subset \mathbb{R}^{n_u}$ is the *control* and $\boldsymbol{p} \in \mathcal{P} \subset \mathbb{R}^{n_p}$ is the *parameter vector*. When the state or control *vectors* are referred to, we mean $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ respectively. The sets $\mathcal{X}$, $\mathcal{U}$ and $\mathcal{P}$ represent the sets of admissible state, control, and parameter vectors. The function $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$ is referred to as the *equations of motion*, or the *dynamics of the system*. It is assumed that f is autonomous, in the sense that it only depends on time indirectly through the state and control vectors.

If the initial and/or final times are free variables then they are considered to be part of the parameter vector $\boldsymbol{p}$.

Assumption 2.1. *The following are assumed to hold:*

1 *The function $\boldsymbol{u}(\cdot)$ is piecewise continuous with respect to time and the set $\mathcal{U}$ is compact.*

2 *The function f is continuous with respect to $\boldsymbol{u}(t)$ and $\boldsymbol{p}$.*

3 *The Jacobian $\nabla_x f\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\big) \in \mathbb{R}^{n_x \times n_x}$ exists and is continuous with respect to $\boldsymbol{x}(t)$ for each $\{\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\} \in \mathcal{X} \times \mathcal{U} \times \mathcal{P}$.*

4 *The Jacobians $\nabla_u f\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\big) \in \mathbb{R}^{n_x \times n_u}$ and $\nabla_p f\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\big) \in \mathbb{R}^{n_x \times n_p}$ exist for each $\{\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\} \in \mathcal{X} \times \mathcal{U} \times \mathcal{P}$.*

Assumptions 2.1.2 and 2.1.3 are slightly stronger than simply requiring that f be Lipschitz with respect to state and control, but this reduced generality will be used in Chapter 4 when numerical solution strategies for optimal control problems are presented. Collectively, the four assumptions ensure that for each admissible function $\boldsymbol{u}(\cdot)$, there exists an absolutely continuous solution to (2.1) of the form

$$\boldsymbol{x}(t) = \boldsymbol{x}(t_0) + \int_{t_0}^{t} f\big(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta), \boldsymbol{p}\big)\, \mathrm{d}\zeta. \tag{2.2}$$

The initial condition $\boldsymbol{x}(t_0)$ is assumed to satisfy

$$S_0\big(\boldsymbol{x}(t_0), \boldsymbol{p}\big) = 0, \tag{2.3}$$

for some function $S_0 : \mathcal{X} \times \mathcal{P} \to \mathbb{R}^{n_0}$. We make the standard assumption that $n_0 \leq (n_x + n_p)$ and that the Jacobian matrix $\nabla S_0\big(\boldsymbol{x}(t_0), \boldsymbol{p}\big) \in \mathbb{R}^{n_0 \times (n_x + n_p)}$ exists and has full row rank. The initial boundary condition (2.3) effectively restricts the set of initial conditions to lie in some subset $\mathcal{X}_0 \subset \mathcal{X} \times \mathcal{P}$.

An optimal control problem typically aims to steer the state vector to some *target set* at the final time. The target set can be described in a similar way to $\mathcal{X}_0$ by using

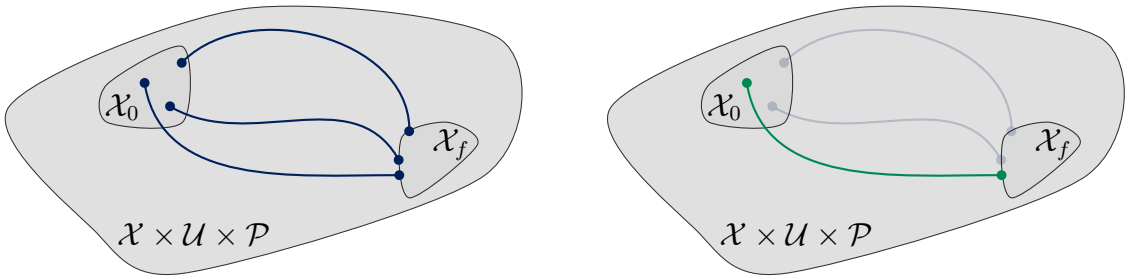
$$S_f\big(\boldsymbol{x}(t_f), \boldsymbol{p}\big) = 0, \tag{2.4}$$

for some function $S_f : \mathcal{X} \times \mathcal{P} \to \mathbb{R}^{n_f}$. Again, the standard assumption that $n_f \leq (n_x + n_p)$ and the Jacobian $\nabla S_f\big(\boldsymbol{x}(t_f), \boldsymbol{p}\big) \in \mathbb{R}^{n_f \times (n_x + n_p)}$ exists and has full row rank is made. The terminal boundary condition (2.4) effectively restricts the set of target (or final) conditions to lie in some subset $\mathcal{X}_f \subset \mathcal{X} \times \mathcal{P}$.

The term *feasible trajectory* will be used heavily throughout the text, and so we make a formal definition at this point. Because the state is a function of the control through (2.2), we can fully define a feasible trajectory in terms of a feasible control $\boldsymbol{u}(\cdot)$ and parameter vector $\boldsymbol{p}$ that together result in a state trajectory that satisfies the given equations of motion and boundary conditions. Nevertheless, we shall often explicitly carry the state when discussing feasible (and optimal) trajectories. This is formalized in Definition 1.

Definition 1 (Feasible Trajectory)**.** *A feasible trajectory is a triple* $\{\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), \boldsymbol{p}\}$ *that together satisfy the equations of motion* (2.1)*, the control* $\boldsymbol{u}(\cdot)$ *satisfies Assumption 2.1.1,* $\boldsymbol{u}(t) \in \mathcal{U}$ *and* $\boldsymbol{x}(t) \in \mathcal{X}$ *for all* $t \in [t_0, t_f]$*,* $\boldsymbol{p} \in \mathcal{P}$*, and the state and parameter vectors satisfy the boundary conditions* (2.3) *and* (2.4)*.*

A control problem, without considering any notion of optimality, is therefore the problem



(a) Control trajectories.

(b) An optimal control trajectory.

Figure 2.1: A control problem looks for a feasible trajectory, and there may be several possibilities. An optimal control problem selects a (or the) feasible trajectory that minimizes a cost functional.

of finding a feasible trajectory for a given set of equations of motion, boundary conditions and sets $\mathcal{X}$, $\mathcal{U}$ and $\mathcal{P}$. A depiction is given in Figure 2.1, and there may be several (even an infinite number of) feasible trajectories for any particular control problem.

Optimal control problems, on the other hand, associate some notion of *cost* to each feasible trajectory, and the objective becomes to compute a (or the) feasible trajectory that maximizes or minimizes this cost. The cost is described by using a functional – a scalar-valued function over a space of functions – and is assumed to have the general form

$$J\big(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}\big) = M\big(\boldsymbol{x}(t_0), \boldsymbol{x}(t_f), \boldsymbol{p}\big) + \int_{t_0}^{t_f} L\big(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta), \boldsymbol{p}\big)\, \mathrm{d}\zeta, \tag{2.5}$$

where $M : \mathcal{X}_0 \times \mathcal{X}_f \to \mathbb{R}$ is the *terminal cost* and $L : \mathcal{X} \times \mathcal{U} \times \mathcal{P} \to \mathbb{R}$ is the *running cost*. Explicit dependence on the state is present in the cost function J both because of the dependence of M on the initial state vector and for notational consistency (i.e., as a matter of taste). With reference to Figure 2.1a, each feasible trajectory can be thought of as being associated with a unique scalar that represents its cost. The act of solving an optimal control problem is equivalent to simply selecting the solution with the lowest or highest such cost, which is depicted in green in Figure 2.1b.

An optimal control problem with a cost functional of the form (2.5) is called a *Bolza*

problem. When $L \equiv 0$, the problem is called a *Mayer problem*, and when $M \equiv 0$ the problem is called a *Lagrange problem*. No form is more general than another, but different forms can be more convenient at different times. The process with which one passes between them is discussed in most textbooks on the subject [3, 4].

Having stated in words what an optimal control problem is, some mathematical notation is now adopted to formalize the statement. Optimal control problems can be stated so as to minimize or maximize the cost, and we will see examples of both at various points in the text. Because the maximization of a particular cost function is equivalent to the minimization of the negative of the cost function, we can assume minimization here without loss of generality. The mathematical statement of an optimal control problem is given in Problem 1, and is understood to carry with it all assumptions on the constituent functions/sets that have been made up to this point.

Problem 1. *Find the piecewise continuous control signal* $\boldsymbol{u}(\cdot)$*, initial state* $\boldsymbol{x}(t_0)$ *and parameter vector* $\boldsymbol{p}$ *that solve the following problem:*

$$\min_{\boldsymbol{x}(t_0),\, \boldsymbol{u}(\cdot),\, \boldsymbol{p}} \quad J\big(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}\big) \tag{2.6a}$$

$$s.t. \quad \dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\big) \tag{2.6b}$$

$$S_0(\boldsymbol{x}(t_0), \boldsymbol{p}) = 0, \quad S_f(\boldsymbol{x}(t_f), \boldsymbol{p}) = 0, \tag{2.6c}$$

$$\boldsymbol{x}(t) \in \mathcal{X}, \quad \boldsymbol{u}(t) \in \mathcal{U}, \quad \boldsymbol{p} \in \mathcal{P}. \tag{2.6d}$$

where $t \in [t_0, t_f]$ *for each time-dependent constraint.*

2.1.1 The Maximum Principle

This section introduces the machinery required to discuss optimal solutions to Problem 1 under some further assumptions. Because we will not need anything more general when invoking the maximum principle, we assume for now that $\boldsymbol{p} = (t_0, t_f) \in \mathbb{R}^2$. That is, the only parameters are the initial and/or final times. This will be the case for all problems that

make use of the maximum principle to study *theoretically* optimal solutions. This assumption implies that we may take f and L to be independent of the parameter vector in this section. Moreover, we assume that $\mathcal{X} = \mathbb{R}^{n_x}$, and shall not explicitly address the maximum principle for problems with state constraints. The *numerical* solution strategies designed to solve optimal control problems that cannot be easily solved by using the maximum principle do not require or use either of these simplifying assumptions.

The existence of an optimal solution is assumed for all problems encountered in this work – but will be formally proven when possible (or when it does not already appear in the literature). For convenience, let the compact set $\mathcal{B} = \mathcal{X}_0 \times \mathcal{X}_f \subset \mathbb{R}^{2n_x+2}$ define the admissible boundary points for Problem 1 under the assumption that $\boldsymbol{p} = (t_0, t_f)$. Define the function $\mathcal{H} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R} \times \mathbb{R}^{n_x} \to \mathbb{R}$ as

$$\mathcal{H}\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \lambda_0, \boldsymbol{\lambda}(t)\big) \coloneqq \lambda_0 L\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) + \boldsymbol{\lambda}(t)^\top f\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big). \tag{2.7}$$

This function $\mathcal{H}$ is sometimes called the *Hamiltonian*, the function $\boldsymbol{\lambda}(\cdot) : \mathbb{R} \to \mathbb{R}^{n_x}$ is the *costate*, and the pair $\big(\lambda_0, \boldsymbol{\lambda}(t)\big)$ is called the *costate vector*. The version of the (pointwise) maximum principle that we use is stated in Theorem 2.2 and is taken primarily from [3].

Theorem 2.2 (Maximum Principle)**.** *Assume that $\mathcal{U}$ does not change with time, and that Assumption 2.1 holds. Let $[t_0^*, t_f^*]$ denote the optimal time interval, $\boldsymbol{u}^* : [t_0^*, t_f^*] \to \mathcal{U}$ be an optimal control and $\boldsymbol{x}^* : [t_0^*, t_f^*] \to \mathcal{X}$ be the corresponding optimal state from* (2.2)*. Then:*

1. *There exists a function $\boldsymbol{\lambda}^* : [t_0^*, t_f^*] \to \mathbb{R}^{n_x}$ and a constant ${\lambda_0}^* \leq 0$ such that $\big({\lambda_0}^*, \boldsymbol{\lambda}^*(t)\big) \neq (0, 0)$ for all $t \in [t_0^*, t_f^*]$.*

2. *Define $\pi(t) \coloneqq \big(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), {\lambda_0}^*, \boldsymbol{\lambda}^*(t)\big)$, so that the following canonical equations hold:*

$$\dot{\boldsymbol{x}}^*(t) = \nabla_\lambda \mathcal{H}\big(\pi(t)\big) \quad \textit{and} \quad \dot{\boldsymbol{\lambda}}^*(t) = -\nabla_x \mathcal{H}\big(\pi(t)\big). \tag{2.8}$$

3. For each fixed $t \in [t_0^*, t_f^*]$,

$$\mathcal{H}\big(\pi(t)\big) = \max_{\boldsymbol{w}(t)\,\in\,\mathcal{U}} \quad \mathcal{H}\big(\boldsymbol{x}^*(t), \boldsymbol{w}(t), {\lambda_0}^*, \boldsymbol{\lambda}^*(t)\big). \tag{2.9}$$

4. The $(2n+2)$*-vector*

$$\begin{aligned}\big[\mathcal{H}\big(\pi(t_0^*)\big) - \lambda_0 \nabla_{t_0} M,\, -\boldsymbol{\lambda}(t_0^*) - \lambda_0 \nabla_{\boldsymbol{x}(t_0)} M,& \\ - \mathcal{H}\big(\pi(t_f^*)\big) - \lambda_0 \nabla_{t_f} M,\, \boldsymbol{\lambda}(t_f^*) - \lambda_0 \nabla_{\boldsymbol{x}(t_f)} M\big]&\end{aligned} \tag{2.10}$$

is orthogonal to the tangent space of $\mathcal{B}$ *at the endpoint* $\big(t_0^*, \boldsymbol{x}^*(t_0^*), t_f^*, \boldsymbol{x}^*(t_f^*)\big)$*. All partial derivatives in* (2.10) *are evaluated at* $\big(t_0^*, \boldsymbol{x}^*(t_0^*), t_f^*, \boldsymbol{x}^*(t_f^*)\big)$.

Condition 1 is referred to as the *non-degeneracy* condition, and condition 4 is referred to as the *transversality* condition. The maximum principle was developed in Russia during the 1960s by Pontryagin and his students, and the theorem often bears his name. The name "maximum principle" is a direct consequence of condition 3 [5]. The theorem provides a necessary condition; meaning optimal solutions must satisfy its conditions, but satisfaction of these conditions alone does not guarantee that a solution is optimal (in the sense of the minimization that is considered here). We will not address sufficient conditions of optimality. The proof of Theorem 2.2 is also not given here, as there are several good resources for it already. Liberzon [4] provides a very accessible proof, while Berkovitz [3, 6] provides a much more rigorous and general treatment. The original work of Pontryagin et al., however, is still be one of the best resources on the subject [5].

2.2 Convexity & Numerical Optimization

A large portion of Chapter 4 is dedicated to transcribing optimal control problems into parameter optimization problems for which the cost function and all constraints are *convex* functions of the solution variable(s). Some of the key aspects of convex optimization are therefore reviewed here.

For several reasons, the use of convex optimization in safety critical aerospace applications should come as no surprise. An incomplete list of reasons for this might include: an increasingly complete theoretical understanding of optimal solutions, a lower information-based complexity than more general classes of optimization, a bounded number of operations to achieve a desired accuracy, reliability (i.e., determinism), and rapidly maturing solvers that are designed for embedded systems. Convex optimization problems also result in *globally* optimal solutions; a feasible solution that yields a strictly lower cost does not exist. This final property lies in contrast to nonconvex optimization methods designed to solve nonconvex optimization problems. Nonconvex optimization problems may have several *locally* optimal solutions that are not globally optimal.

Convex analysis and optimization theory are vast subjects with rich histories, and a comprehensive treatment is not attempted here. Only the key details that are used to discuss subsequent CGC methods for aerospace systems are reviewed.

2.2.1 Convex Sets & Convex Functions

Convex optimization is based the notions of convex sets and convex functions. A set $\mathcal{C} \subset \mathbb{R}^{n_z}$ is convex if (and only if) for any two elements $\boldsymbol{z}_1$, $\boldsymbol{z}_2 \in \mathcal{C}$ their *convex combination* $\lambda \boldsymbol{z}_1 + (1-\lambda)\boldsymbol{z}_2$ lies in $\mathcal{C}$ for any $\lambda \in [0,1]$. In other words, the line segment connecting the points $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ lies entirely in the set $\mathcal{C}$. If it is true that for any two elements $\boldsymbol{z}_1$, $\boldsymbol{z}_2 \in \mathcal{C}$ their *conic combination* $\lambda_1 \boldsymbol{z}_1 + \lambda_2 \boldsymbol{z}_2 \in \mathcal{C}$ for any λ_1, $\lambda_2 \geq 0$, then $\mathcal{C}$ is called a *convex cone*.

The most important convex sets used in this work are polyhedra, norm balls and norm cones. A polyhedron can be described by a set of linear equalities and/or inequalities, and generally takes the form

$$\mathcal{P} = \{\boldsymbol{z} \mid A\boldsymbol{z} = \boldsymbol{b},\, G\boldsymbol{z} \leq \boldsymbol{h}\}, \tag{2.11}$$

for matrices $A \in \mathbb{R}^{n_e \times n_z}$, $G \in \mathbb{R}^{n_i \times n_z}$ and vectors $\boldsymbol{b} \in \mathbb{R}^{n_e}$, $\boldsymbol{h} \in \mathbb{R}^{n_i}$. A polyhedron can be thought of as the intersection of n_e hyperplanes and n_i halfspaces, or as the set of solutions to a finite number of linear equations.

Each norm $\|\cdot\|_p$ for $p = 1, 2, \ldots, \infty$ in the Euclidean space $\mathbb{R}^{n_z}$ defines a norm ball of radius ϵ centered at $\boldsymbol{z}_c \in \mathbb{R}^{n_z}$ in the form of the set

$$\mathcal{B}_\epsilon(\boldsymbol{z}_c) = \{\boldsymbol{z} \mid \|\boldsymbol{z} - \boldsymbol{z}_c\|_p \leq \epsilon\}. \tag{2.12}$$

This set is a convex cone, a fact that can be established by using the basic properties of a norm.

Similarly, the norm cone is the set

$$\mathcal{C} = \{(w, \boldsymbol{z}) \mid \|\boldsymbol{z}\|_p \leq w\} \subset \mathbb{R}^{n_z+1}. \tag{2.13}$$

A special case arises when the two-norm, $p = 2$, is used in the definition of the norm cone. In this case, the set in (2.13) is called a *second-order cone*.

Lastly, a positive semidefinite matrix $M \in \mathbb{R}^{n_z \times n_z}$ is defined to be a symmetric matrix whose eigenvalues are nonnegative. We use $M \succeq 0$ to denote M as a positive semidefinite matrix. The set

$$\mathbb{S}_+^{n_z} = \{M \in \mathbb{R}^{n_z \times n_z} \mid M \succeq 0\} \tag{2.14}$$

is referred to as the positive semidefinite cone. If the eigenvalues of M are strictly positive, then we write $M \in \mathbb{S}_{++}^{n_z}$ or $M \succ 0$. Both $\mathbb{S}_+^{n_z}$ and $\mathbb{S}_{++}^{n_z}$ are convex cones.

A function $f : \mathcal{D} \to \mathbb{R}$ is said to have domain $\mathcal{D} \subset \mathbb{R}^{n_z}$ and co-domain $\mathbb{R}$. The most popular definition of a convex function uses the so-called Jensen's inequality: A function f is convex if $\mathcal{D}$ is a convex set and for any $\boldsymbol{z}_1$, $\boldsymbol{z}_2 \in \mathcal{D}$ the inequality

$$f\big(\lambda \boldsymbol{z}_1 + (1-\lambda)\boldsymbol{z}_2\big) \leq \lambda f(\boldsymbol{z}_1) + (1-\lambda) f(\boldsymbol{z}_2) \tag{2.15}$$

holds for all $\lambda \in [0, 1]$. Two important characterizations can be added if it is assumed that f is twice differentiable.[1] In this case, either of the following can be used to establish convexity

[1]The notion of subdifferentiability will not be used in any serious way and is therefore omitted.

of the function f: the set $\mathcal{D}$ is a convex set and for all $\boldsymbol{z}_1$, $\boldsymbol{z}_2 \in \mathcal{D}$

$$f(\boldsymbol{z}_2) \geq f(\boldsymbol{z}_1) + \nabla_z f\big(\boldsymbol{z}_1\big)(\boldsymbol{z}_2 - \boldsymbol{z}_1), \tag{2.16a}$$

$$\nabla^2_{zz} f(\boldsymbol{z}_1) \succeq 0. \tag{2.16b}$$

The statement in (2.16a) can be interpreted as saying that each vector $\boldsymbol{z}_2$ in a tangent hyperplane of a convex function *underestimates* the function value at the point $\boldsymbol{z}_2$. In other words, it is saying that the first-order Taylor series of f centered at $\boldsymbol{z}_1$ defines a supporting hyperplane to the function f at $\boldsymbol{z}_1$. All affine functions are convex, because (2.16a) is valid with equality. The statement (2.16b) means that f has some "upwards" curvature over the domain $\mathcal{D}$. Loosely speaking, this paints the geometric picture that convex functions are somehow "bowl-like" – a good intuitive picture to maintain when we discuss *minimization* of a convex function. All quadratic functions with positive definite Hessians are therefore convex.

2.2.2 Convex Optimization

Convex optimization refers to the minimization of a convex function over a convex set. The definition of a convex function can be used to show that all optimal solutions of such problems are globally optimal. This means that once a solution is obtained, there are no feasible solutions that result in a strictly lower cost.[2] There are several different classes of convex optimization problems; and these are sometimes called *convex programs*.

In the 1940s, Dantzig first popularized the *simplex method* to solve linear programs. Linear programs are the class of convex optimization problems that minimize a linear function over a polyhedral set of the form (2.11). The solution of a linear program always occurs at a vertex of the polyhedron, and this fact is exploited by the simplex algorithm in that the algorithm iterates between vertices in search of the optimal one. The fact that the simplex

[2] A feasible solution in this setting is a vector that lies in the convex domain, $\mathcal{D}$, over which the function is defined and the minimization takes place.

method was developed at the same time as early computers contributed to its ubiquitous influence on the field of numerical optimization [7, 8]. Convex analysis, on the other hand, was already an established mathematical field by this time [9]. One of the great innovations in optimization theory was the realization that interior point methods can be used to solve linear programming problems [8, 10]. Interior point methods are fundamentally different from Dantzig's simplex algorithm; and their names divulge the reason. An interior point method follows a path that lies strictly inside the feasible region, approaching the boundary (a vertex) only in the limit. The simplex method, as mentioned, travels around the vertices of the feasible polyhedron. While interior point methods proved useful for the practical solution of linear programming problems, their most important property is that they can be used for more general problems than linear programming (the simplex method cannot). This fact continues to play a critical role in the development of reliable solvers for problems encountered in the aerospace domain, which are often beyond the limited scope of linear programming.

The two classes of convex optimization problems that will be encountered most in Chapters 4 through 6 are *second-order cone programs* (SOCPs) and *semidefinite programs* (SDPs). An SOCP resembles a linear program, except that some of its variables are constrained to lie in a second-order cone of the form (2.13). A general SOCP can be cast in the following standard form:

$$\min_{\boldsymbol{z}} \quad \boldsymbol{c}^\top \boldsymbol{z} \tag{2.17a}$$

$$\text{s.t.} \quad A\boldsymbol{z} = \boldsymbol{b} \tag{2.17b}$$

$$\boldsymbol{z} \in \mathbb{R}^\ell_+ \times \mathcal{C}_{Q_1} \times \cdots \times \mathcal{C}_{Q_m} \tag{2.17c}$$

where the $\mathcal{C}_{Q_i}$ are convex second-order cones of dimension $d_i + 1$. The problem data $\{A, \boldsymbol{b}, \boldsymbol{c}, \ell, d_1, \ldots, d_m\}$ fully define the optimal solution of the SOCP.

An SDP is more general than an SOCP, and introduces matrix variables that may live

in the cone of positive semidefinite matrices. A standard form SDP can be written as:

$$\min_{Z} \quad \text{tr}\left(C^\top Z\right) \tag{2.18a}$$

$$\text{s.t.} \quad AZ = B \tag{2.18b}$$

$$Z \succeq 0 \tag{2.18c}$$

where now the problem's solution variable is the matrix Z. Note that both linear cones and second-order cones can be embedded into an SDP via suitable definition of A and B. (For example, constraining all off-diagonal elements of a single row and column of Z to be zero and setting the entry on the diagonal to one would restrict the diagonal entry to lie in the linear cone.)

A piece of code designed specifically to solve a given class of optimization problems is called a *solver*. Solvers are developed using a specific programming language, commonly Matlab®, Python or C/C++ [11, 12, 13, 14, 15, 16, 17, 18]. A useful software tool that is often paired with these solvers is a *parser*. A parser, roughly speaking, translates what a user would write down on paper into the standard form of whatever class of problems that the solver is designed to handle, for example either (2.17) or (2.18). The automation of the parsing process greatly reduces the time required to prototype an optimization-based algorithm, and also permits the testing of a variety of solvers for a given problem. Parsers have been developed for high-level languages such as Matlab® or Python [19, 20], while low-level languages such as C or C++ typically require "hand parsing" a problem into its standard form [14]. A notable exception is the C/C++ based solver BSOCP developed by Dueri et al. [15, 16] that has an accompanying parser written in C++.

For algorithm implementations designed to be run in real-time on an aerospace vehicle, hand parsing is a necessary final step. We explore this in more detail in Chapter 5, which is devoted to a systematic procedure to hand parse the class of SOCPs that arise most often during the numerical solution of optimal control problems in the aerospace domain.

2.2.3 Transcription Methods

One of the central themes of this dissertation is to solve optimal control problems in the form of Problem 1 by solving a sequence of convex optimization problems of the form of (2.17). The primary difference between Problem 1 and (2.17) is that the feasible domain of the former is an infinite-dimensional space of functions, while the feasible domain of the latter is a finite-dimensional vector space. Moreover, the constraints in (2.17) have a very specific form; they must either be affine or representable as a second-order cone. Problem 1, however, contains nonlinear *differential* constraints of the form (2.6b) and possibly nonconvex constraints (2.6c) and/or (2.6d).

A transcription method bridges the gap between optimal control problems and parameter optimization problems (convex or nonconvex). This is done by approximating the infinite-dimensional space of functions with a finite-dimensional space of functions by making certain assumptions about how the control varies as a function of time. In doing so we are reducing the "size" of the set of feasible trajectories, and understand that it is therefore only possible to compute *sub-optimal* solutions this way. (Unless the control function that is optimal in the sense of the maximum principle also happens to be a member of this finite-dimensional space of functions.)

Moreover, a continuous-time constraint of the form $\boldsymbol{x}(t) \in \mathcal{X}$, as present in (2.6d), implies an infinite number of constraints; one for each instantaneous value of time in the interval $[t_0, t_f]$. To achieve a finite-dimensional representation of this constraint, we merely impose the constraint at a finite number of times in the interval $[t_0, t_f]$, thereby *approximating* feasibility. Only in special circumstances will there be a theoretical guarantee that enforcing a constraint at finitely many times in the interval $[t_0, t_f]$ will imply that the constraint is in fact satisfied for all times in the interval.

Transcription methods are likely the most important tool in the art of trajectory optimization. In addition to the solver that is used, the transcription method drives properties such as algorithm convergence, numerical robustness and computer memory storage require-

ments. Some textbooks that cover the subject are [21] and [22], and several articles have surveyed the field with a particular focus on aerospace applications [23, 24, 25, 26, 27].

2.3 Hypercomplex Numbers: Quaternions & Dual Quaternions

A *coordinate frame* is denoted by $\mathcal{F}$ and represents a right-handed orthonormal triad of *basis* vectors $\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$. When a subscript $\mathcal{F}_1$ is used, the corresponding basis vectors inherit the subscript $\{\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_1\}$. A vector that is given the same subscript shall be understood to mean "resolved in the coordinates of frame $\mathcal{F}_1$". Mathematically, this may be expressed for $\boldsymbol{r}_1 \in \mathbb{R}^3$ as

$$\boldsymbol{r}_1 = r_x \boldsymbol{x}_1 + r_y \boldsymbol{y}_1 + r_z \boldsymbol{z}_1, \tag{2.19}$$

or simply written as $\boldsymbol{r}_1 = \big(r_x, r_y, r_z\big) \in \mathbb{R}^3$.

The set of possible orientations of one coordinate frame relative to another is given by the special orthogonal group in three dimensions,

$$SO(3) \coloneqq \{C \in \mathbb{R}^{3\times 3} \mid \det C = +1,\, C^{-1} = C^\top\}. \tag{2.20}$$

The elements of this set are commonly referred to as rotation matrices and have six free parameters. If the vector $\boldsymbol{r}$ has coordinates $\boldsymbol{r}_1$ in $\mathcal{F}_1$ and coordinates $\boldsymbol{r}_2$ in $\mathcal{F}_2$, then the relationship $\boldsymbol{r}_2 = C_{21}\boldsymbol{r}_1$ holds for a rotation matrix $C_{21} \in SO(3)$.

2.3.1 Quaternions

Quaternions are used to parameterize the orientation of one coordinate frame with respect to another, and serve as an alternative to rotation matrices. To avoid unecessary pedantry, quaternions shall be considered four-dimensional elements of the vector space $\mathbb{R}^4$. A more rigorous construction of quaternions and the algebraic rules that they obey can be found in Appendix A. A quaternion $\boldsymbol{q} \in \mathbb{R}^4$ is composed of a vector part, $\boldsymbol{q}_v \in \mathbb{R}^3$, and a scalar part, $q_4 \in \mathbb{R}$, such that $\boldsymbol{q} = \big(\boldsymbol{q}_v, q_4\big)$. According to the multiplcation rules derived in Appendix A,

the product of two quaternions $\boldsymbol{q}, \boldsymbol{p} \in \mathbb{R}^4$ can be written as

$$\boldsymbol{q} \otimes \boldsymbol{p} = [\boldsymbol{q}]_{\otimes}\boldsymbol{p} = [\boldsymbol{p}]^*_{\otimes}\boldsymbol{q}, \tag{2.21}$$

where,

$$[\boldsymbol{q}]_{\otimes} = \begin{bmatrix} q_4 I_3 + \boldsymbol{q}_v^{\times} & \boldsymbol{q}_v \\ -\boldsymbol{q}_v^{\top} & q_4 \end{bmatrix}, \quad \text{and} \quad [\boldsymbol{p}]^*_{\otimes} = \begin{bmatrix} p_4 I_3 - \boldsymbol{p}_v^{\times} & \boldsymbol{p}_v \\ -\boldsymbol{p}_v^{\top} & p_4 \end{bmatrix}. \tag{2.22a}$$

To illustrate how quaternions are related to attitude and the set $SO(3)$, a third attitude parameterization is required. Using Euler's famous theorem that a general displacement of a rigid body with one fixed point is a rotation about an axis through that point [28, 29], each rotation matrix can be identified with an *axis-angle pair* $(\boldsymbol{a}, \varphi)$. Here, $\boldsymbol{a} \in \mathbb{R}^3$ denotes the unit vector pointing along the fixed axis of rotation and $\varphi \in [0, 2\pi]$ is the angle that is rotated through. The rotation matrix is related to these parameters by

$$C = \cos\varphi I_3 + (1 - \cos\varphi)\,\boldsymbol{a}\boldsymbol{a}^{\top} - \sin\varphi \boldsymbol{a}^{\times}, \tag{2.23}$$

where,

$$\boldsymbol{a}^{\times} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}. \tag{2.24}$$

Consider now three coordinate frames $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ and the associated rotation matrices C_{21}, C_{32} and $C_{31} = C_{32}C_{21}$. By associating the axis-angle pairs $(\boldsymbol{a}_1, \varphi_1) \equiv C_{21}$ and $(\boldsymbol{a}_2, \varphi_2) \equiv C_{32}$, we can then show by using (2.23) that the axis-angle pair $(\boldsymbol{a}_3, \varphi_3) \equiv C_{31}$ can be written as

$$\sin\frac{\varphi_3}{2}\,\boldsymbol{a}_3 = \sin\frac{\varphi_1}{2}\cos\frac{\varphi_2}{2}\boldsymbol{a}_1 + \cos\frac{\varphi_1}{2}\sin\frac{\varphi_2}{2}\boldsymbol{a}_2 + \sin\frac{\varphi_1}{2}\sin\frac{\varphi_2}{2}\boldsymbol{a}_1^{\times}\boldsymbol{a}_2, \tag{2.25a}$$

$$\cos\frac{\varphi_3}{2} = \cos\frac{\varphi_1}{2}\cos\frac{\varphi_2}{2} - \sin\frac{\varphi_1}{2}\sin\frac{\varphi_2}{2}\boldsymbol{a}_1^{\top}\boldsymbol{a}_2. \tag{2.25b}$$

By defining the quaternions

$$\boldsymbol{q}_{21} \coloneqq \begin{bmatrix} \sin\frac{\varphi_1}{2}\boldsymbol{a}_1 \\ \cos\frac{\varphi_1}{2} \end{bmatrix} \quad \text{and} \quad \boldsymbol{q}_{32} \coloneqq \begin{bmatrix} \sin\frac{\varphi_2}{2}\boldsymbol{a}_2 \\ \cos\frac{\varphi_2}{2} \end{bmatrix} \tag{2.26}$$

the multiplication rule (2.21) turns out to be exactly equivalent to the composition rule in (2.25). Thus for a general axis-angle pair $(\boldsymbol{a}, \varphi)$ that describes the rotation of a rigid body with a single fixed point, the corresponding quaternion is $\boldsymbol{q} = \left(\sin\frac{\varphi}{2}\boldsymbol{a}, \cos\frac{\varphi}{2}\right) \in \mathbb{R}^4$. The fact that $\boldsymbol{a}^\top\boldsymbol{a} = 1$ translates to a unit length condition on the quaternion:

$$\boldsymbol{q}^\top\boldsymbol{q} = \sin^2\frac{\varphi}{2}\boldsymbol{a}^\top\boldsymbol{a} + \cos^2\frac{\varphi}{2} = 1. \tag{2.27}$$

It is therefore the set of *unit quaternions* that describe rotations of a rigid body. The set of unit quaternions is denoted by $\mathbb{R}^4_u \coloneqq \{\boldsymbol{q} \in \mathbb{R}^4 \,|\, \boldsymbol{q}^\top\boldsymbol{q} = 1\}$ and can be thought of as the three-dimensional sphere (three-sphere) embedded within $\mathbb{R}^4$. Using the expression (2.23) in addition to the definition of the quaternion in terms of the axis-angle pair leads to the expression

$$C = q_4^2 I_3 - 2q_4\boldsymbol{q}_v^\times + \boldsymbol{q}_v^\times\boldsymbol{q}_v^\times + \boldsymbol{q}_v\boldsymbol{q}_v^\top. \tag{2.28}$$

The quaternion based analogue of the rotation operation $\boldsymbol{r}_2 = C_{21}\boldsymbol{r}_1$ can then be derived using (2.28) as follows:

$$\begin{aligned}
\boldsymbol{r}_2 &= \left(q_4^2 I_3 - 2q_4\boldsymbol{q}_v^\times + \boldsymbol{q}_v^\times\boldsymbol{q}_v^\times + \boldsymbol{q}_v\boldsymbol{q}_v^\top\right)\boldsymbol{r}_1, \\
\begin{bmatrix} \boldsymbol{r}_2 \\ 0 \end{bmatrix} &= \begin{bmatrix} q_4 I_3 - \boldsymbol{q}_v^\times & \boldsymbol{q}_v \\ -\boldsymbol{q}_v^\top & q_4 \end{bmatrix} \begin{bmatrix} q_4 I_3 - \boldsymbol{q}_v^\times & -\boldsymbol{q}_v \\ \boldsymbol{q}_v^\top & q_4 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 \\ 0 \end{bmatrix}, \\
\begin{bmatrix} \boldsymbol{r}_2 \\ 0 \end{bmatrix} &= [\boldsymbol{q}^*]_\otimes[\boldsymbol{q}]^*_\otimes \begin{bmatrix} \boldsymbol{r}_1 \\ 0 \end{bmatrix} = [\boldsymbol{q}^*]_\otimes \left(\begin{bmatrix} \boldsymbol{r}_1 \\ 0 \end{bmatrix} \otimes \boldsymbol{q}\right) = \boldsymbol{q}^* \otimes \begin{bmatrix} \boldsymbol{r}_1 \\ 0 \end{bmatrix} \otimes \boldsymbol{q},
\end{aligned} \tag{2.29}$$

where $\boldsymbol{q}^*$ denotes the quaternion conjugate that will be defined shortly in (2.30).

The most important operation using quaternions is the product defined in (2.21). An

important characteristic is that the product of two unit quaternions remains a unit quaternion [30]. Two other operations are also used. First, the quaternion conjugate is defined to be

$$\boldsymbol{q}^* = \begin{bmatrix} -\boldsymbol{q}_v \\ q_4 \end{bmatrix}. \tag{2.30}$$

Given two quaternions $\boldsymbol{q}, \boldsymbol{p} \in \mathbb{R}^4_u$, the quaternion conjugate permits the definition of the *multiplicative error quaternion* as $\delta\boldsymbol{q} = \boldsymbol{q}^* \otimes \boldsymbol{p}$. The (multiplicative) identity quaternion is defined as $\boldsymbol{q}_{\text{id}} = \boldsymbol{q}^* \otimes \boldsymbol{q} = (0_{3\times 1},\, 1)$.

Next, the quaternion cross product is defined as

$$\boldsymbol{q} \oslash \boldsymbol{p} = [\boldsymbol{q}]_{\oslash}\boldsymbol{p} = [\boldsymbol{p}]^*_{\oslash}\boldsymbol{q}, \tag{2.31}$$

where,

$$[\boldsymbol{q}]_{\oslash} = \begin{bmatrix} q_4 I_3 + \boldsymbol{q}_v^{\times} & \boldsymbol{q}_v \\ 0_{1\times 3} & 0 \end{bmatrix} = \begin{bmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{2.32a}$$

$$[\boldsymbol{p}]^*_{\oslash} = \begin{bmatrix} p_4 I_3 - \boldsymbol{p}_v^{\times} & \boldsymbol{p}_v \\ 0_{1\times 3} & 0 \end{bmatrix} = \begin{bmatrix} p_4 & p_3 & -p_2 & p_1 \\ -p_3 & p_4 & p_1 & p_2 \\ p_2 & -p_1 & p_4 & p_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.32b}$$

All three-parameter representations of rigid body rotations suffer from the presence of a singularity [29]. A singularity here refers to an ambiguity whereby a given set of parameters may correspond to numerous rotations.[3] Unit quaternions provide a *minimal singularity-free* parameterization of the set of rotation matrices and form a double cover of $SO(3)$. The

[3]The most common singularity encountered is that of gimbal-lock when a set of three Euler angles are used to describe a rotation.

double cover arises from the fact that $\boldsymbol{q}$ and $-\boldsymbol{q}$ correspond to the same rotation matrix, which can be seen from (2.28).

2.3.2 Dual Quaternions

Dual quaternions extend the quaternion parameterization to capture both the relative orientation and relative position of two coordinate frames. A dual quaternion is defined as an eight-dimensional vector $\tilde{\boldsymbol{q}} \in \mathbb{R}^8$ and is composed of a real part, $\boldsymbol{q}_1 \in \mathbb{R}^4$, and a dual part $\boldsymbol{q}_2 \in \mathbb{R}^4$, such that $\tilde{\boldsymbol{q}} = \big(\boldsymbol{q}_1, \boldsymbol{q}_2\big)$. In the same way as for quaternions, a derivation of dual quaternions and the algebraic rules that govern their use (as well as the origin for the nomenclature) is given in Appendix A.

Much like unit quaternions represent rotational motion, it is the set of *unit* dual quaternions that are used to represent the more general rotation and translation of a rigid body [31]. As shown in Appendix A, a unit dual quaternion satisfies $\boldsymbol{q}_1^\top \boldsymbol{q}_1 = 1$ and $\boldsymbol{q}_1^\top \boldsymbol{q}_2 = 0$. Consequently, the set of unit dual quaternions is defined as

$$\mathbb{R}^8_u \coloneqq \left\{ \tilde{\boldsymbol{q}} = \big(\boldsymbol{q}_1, \boldsymbol{q}_2\big) \in \mathbb{R}^8 \mid \boldsymbol{q}_1 \in \mathbb{R}^4_u,\ \boldsymbol{q}_1^\top \boldsymbol{q}_2 = 0 \right\}. \tag{2.33}$$

Unit dual quaternions therefore form a six-dimensional submanifold within $\mathbb{R}^8$. Note that the first constraint forces the real part of a unit dual quaternion to be a unit quaternion, or an element of the three-sphere $\mathbb{R}^4_u$. The second constraint forces the dual part of a unit dual quaternion to be an element of the (three-dimensional) tangent hyperplane of the three-sphere at the point $\boldsymbol{q}_1$. As such, the set of unit dual quaternions in (2.33) can be interpreted as the union over the three-sphere of the Cartesian products of each $\boldsymbol{q}_1$ and the corresponding three-dimensional tangent hyperplane to the three-sphere at $\boldsymbol{q}_1$.

Let $\tilde{\boldsymbol{q}}$, $\tilde{\boldsymbol{p}} \in \mathbb{R}^8_u$ be two unit dual quaternions. Dual quaternion multiplication is defined using the quaternion multiplcation rule (2.21) according to

$$\tilde{\boldsymbol{q}} \otimes \tilde{\boldsymbol{p}} = [\tilde{\boldsymbol{q}}]_\otimes \tilde{\boldsymbol{p}} = [\tilde{\boldsymbol{p}}]^*_\otimes \tilde{\boldsymbol{q}}, \tag{2.34}$$

where,

$$[\tilde{\boldsymbol{q}}]_{\otimes} \coloneqq \begin{bmatrix} [\boldsymbol{q}_1]_{\otimes} & 0_{4\times 4} \\ [\boldsymbol{q}_2]_{\otimes} & [\boldsymbol{q}_1]_{\otimes} \end{bmatrix} \quad \text{and} \quad [\tilde{\boldsymbol{p}}]^*_{\otimes} \coloneqq \begin{bmatrix} [\boldsymbol{p}_1]^*_{\otimes} & 0_{4\times 4} \\ [\boldsymbol{p}_2]^*_{\otimes} & [\boldsymbol{p}_1]^*_{\otimes} \end{bmatrix}. \tag{2.35}$$

Note that the product $\tilde{\boldsymbol{q}} \otimes \tilde{\boldsymbol{p}}$ remains a unit dual quaternion. Similar to the quaternion case, the dual quaternion conjugate can be defined as

$$\tilde{\boldsymbol{q}}^* = \begin{bmatrix} \boldsymbol{q}_1^* \\ \boldsymbol{q}_2^* \end{bmatrix} \tag{2.36}$$

where $\boldsymbol{q}_1^*$, $\boldsymbol{q}_2^*$ are given by (2.30). Lastly, the dual quaternion cross product is defined using the quaternion cross product as

$$\tilde{\boldsymbol{q}} \oslash \tilde{\boldsymbol{p}} = [\tilde{\boldsymbol{q}}]_{\oslash}\, \tilde{\boldsymbol{p}} = [\tilde{\boldsymbol{p}}]^*_{\oslash} \tilde{\boldsymbol{q}}, \tag{2.37}$$

where,

$$[\tilde{\boldsymbol{q}}]_{\oslash} \coloneqq \begin{bmatrix} [\boldsymbol{q}_1]_{\oslash} & 0_{4\times 4} \\ [\boldsymbol{q}_2]_{\oslash} & [\boldsymbol{q}_1]_{\oslash} \end{bmatrix} \quad \text{and} \quad [\tilde{\boldsymbol{p}}]^*_{\oslash} \coloneqq \begin{bmatrix} [\boldsymbol{p}_1]^*_{\oslash} & 0_{4\times 4} \\ [\boldsymbol{p}_2]^*_{\oslash} & [\boldsymbol{p}_1]^*_{\oslash} \end{bmatrix}. \tag{2.38}$$

2.3.3 Dual Quaternions & Rigid Body Motion

Consider the two coordinate frames $\mathcal{F}_1$ and $\mathcal{F}_2$. Assume that $\mathcal{F}_1$ is a non-rotating frame with a fixed origin, and $\mathcal{F}_2$ is a possibly rotating frame whose origin is the center of mass of a rigid body. The geometric position vector pointing from the origin of $\mathcal{F}_1$ to the origin of $\mathcal{F}_2$ is denoted by $\boldsymbol{r}$. The coordinates of this position vector in each frame are therefore denoted by $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$. Assume that the orientation of $\mathcal{F}_2$ with respect to $\mathcal{F}_1$ is given by the unit quaternion $\boldsymbol{q}$. A pure rotation by the unit quaternion $\boldsymbol{q}$ is represented by the unit dual quaternion $\tilde{\boldsymbol{q}} = \big(\boldsymbol{q},\, 0_{4\times 1}\big)$. A pure translation may be described either by $\tilde{\boldsymbol{q}} = \big(\boldsymbol{q}_{\text{id}},\, \frac{1}{2}\boldsymbol{r}_1\big)$ or by $\tilde{\boldsymbol{q}} = \big(\boldsymbol{q}_{\text{id}},\, \frac{1}{2}\boldsymbol{r}_2\big)$. Note that $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are assumed to have a zero scalar part here (i.e., they are *pure* quaternions). A general displacement is given by *either* a rotation by $\boldsymbol{q}$ followed by a translation by $\boldsymbol{r}_2$ or a translation by $\boldsymbol{r}_1$ followed by a rotation $\boldsymbol{q}$. These two sequences are

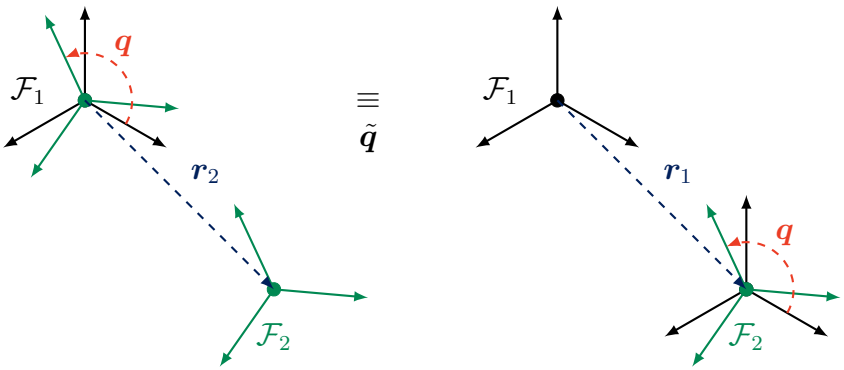


Figure 2.2: Rotation followed by translation is equivalent to a translation followed by a rotation.

geometrically equivalent, as shown in Figure 2.2.

The composition of a rotation and translation is given by dual quaternion multiplication. Using the representations of a pure rotation and a pure translation, it follows that the unit dual quaternion representing the difference between these two frames is

$$\tilde{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{q} \\ \frac{1}{2}\boldsymbol{r}_1 \otimes \boldsymbol{q} \end{bmatrix} = \begin{bmatrix} \boldsymbol{q} \\ \frac{1}{2}\boldsymbol{q} \otimes \boldsymbol{r}_2 \end{bmatrix} \in \mathbb{R}^8_u. \tag{2.39}$$

The first expression describes a translation by $\boldsymbol{r}_1$ followed by a rotation $\boldsymbol{q}$, whereas the second expression describes a rotation by $\boldsymbol{q}$ followed by a translation $\boldsymbol{r}_2$. The equivalence of the two expressions in (2.39) also follows from the relationship derived in (2.29).

Velocities can be represented using dual quaternions by appending a zero scalar part to the usual angular and linear velocities. Let $\boldsymbol{\omega}_2, \boldsymbol{v}_2 \in \mathbb{R}^4$ respectively denote the angular and linear velocities resolved in the frame $\mathcal{F}_2$. We take $\boldsymbol{v}_2 = \dot{\boldsymbol{r}}_2 + \boldsymbol{\omega}_2^{\times}\boldsymbol{r}_2$, and define the *dual velocity* to be

$$\tilde{\boldsymbol{\omega}} = \begin{bmatrix} \boldsymbol{\omega}_2 \\ \boldsymbol{v}_2 \end{bmatrix} \in \mathbb{R}^8. \tag{2.40}$$

This definition permits the use of the expression $\dot{\boldsymbol{r}}_1 = \boldsymbol{q}^* \otimes \boldsymbol{v}_2 \otimes \boldsymbol{q}$ while remaining consistent

with the transport theorem.

The definition of the dual quaternion in (2.39) in conjunction with the quaternion triple identity (see [32]) leads to the following lemma, which can be proven by construction and is instrumental in expressing constraint functions in terms of the unit dual quaternion.

Lemma 2.3. *Let* $\boldsymbol{q} \in \mathbb{R}^4_u$ *be the unit quaternion that maps coordinates from frame* $\mathcal{F}_1$ *to* $\mathcal{F}_2$*. Let* $\boldsymbol{r}$ *denote the vector from the origin of* $\mathcal{F}_1$ *to the origin of* $\mathcal{F}_2$*. Consider the pure quaternions* $\boldsymbol{c}, \boldsymbol{d}, \in \mathbb{R}^4$ *whose vector parts have coordinates* $\boldsymbol{c}_1, \boldsymbol{d}_1$ *in* $\mathcal{F}_1$ *and* $\boldsymbol{c}_2, \boldsymbol{d}_2$ *in* $\mathcal{F}_2$*. Using* (2.39)*, the following equations hold:*

$$\boldsymbol{r}_1^\top \boldsymbol{c}_1 = \tilde{\boldsymbol{q}}^\top M_1 \tilde{\boldsymbol{q}}, \quad \textit{where} \quad M_1 = \begin{bmatrix} 0_{4\times4} & [\boldsymbol{c}_1]_\otimes^\top \\ [\boldsymbol{c}_1]_\otimes & 0_{4\times4} \end{bmatrix}, \tag{2.41a}$$

$$\boldsymbol{r}_2^\top \boldsymbol{c}_2 = \tilde{\boldsymbol{q}}^\top M_2 \tilde{\boldsymbol{q}}, \quad \textit{where} \quad M_2 = \begin{bmatrix} 0_{4\times4} & [\boldsymbol{c}_2]_\otimes^{*\top} \\ [\boldsymbol{c}_2]_\otimes^* & 0_{4\times4} \end{bmatrix}, \tag{2.41b}$$

$$\boldsymbol{c}_1^\top \boldsymbol{d}_1 = \tilde{\boldsymbol{q}}^\top M_3 \tilde{\boldsymbol{q}}, \quad \textit{where} \quad M_3 = \begin{bmatrix} [\boldsymbol{c}_1]_\otimes [\boldsymbol{d}_2]_\otimes^* & 0_{4\times4} \\ 0_{4\times4} & 0_{4\times4} \end{bmatrix}, \tag{2.41c}$$

$$\|\boldsymbol{r}_1\|_2 = \|2E_d \tilde{\boldsymbol{q}}\|_2, \quad \textit{where} \quad E_d = \begin{bmatrix} 0_{4\times4} & 0_{4\times4} \\ 0_{4\times4} & I_4 \end{bmatrix}. \tag{2.41d}$$

Chapter 3

POWERED DESCENT GUIDANCE

The rocket worked perfectly, except for landing on the wrong planet.

– Wernher von Braun

Powered descent guidance refers to the problem of transferring a vehicle from an estimated initial state to a target state using rocket-powered engines and/or reaction control systems. A guidance trajectory, in this context, is understood as a trajectory that will *autonomously* achieve this objective using the available actuation methods. In the (theoretical) presence of zero environmental disturbances, zero model error, and perfect state estimation and actuation, the guidance trajectory will be precisely the trajectory followed by the vehicle.

Powered descent guidance is inherently a challenging problem due to the physics of the problem,[1] and this difficulty is compounded by the addition of precision landing requirements, atmospheric interactions and constraints that arise due to navigation, communication, and/or thermal system requirements. While each celestial body targeted for landing presents unique design considerations, all powered descent methods are universally subject to a set of such constraints. The payoffs for overcoming these challenges are numerous: increased scientific return from a given mission, lower cost-per-kilogram to launch spacecraft from Earth, and the possibility of establishing permanent human outposts elsewhere in our solar system [33, 34].

The need for autonomy is clear, as it is not feasible for a rocket booster returning to the Earth to be piloted and landed by humans, and (robotic) missions to other celestial bodies suffer time delays that prohibit direct control from humans on Earth. Despite the historic success of piloted lunar landings during the Apollo program, autonomous powered descent

[1] Just try to balance your pen upright in the palm of your hand.

and precision landing have been identified as key enabling technologies for future robotic and human missions [34]. This need for autonomy in conjunction with the definition of guidance above imply the need to compute trajectories in real-time onboard the landing vehicle, which for the case of powered descent equates roughly to an algorithm runtime on the order of one second or less. We will explore the details of a real-time algorithm implementation in Chapter 5, while this chapter focuses on the optimal control problem statements relevant to precision landing and their theoretical solutions.

In designing a real-time algorithm that is intended to provide an "optimal" solution, it is helpful to understand exactly what optimal means. Even the (sub-optimal) Apollo guidance system was verified to produce descent braking phase trajectories that consumed as near as 16 kg to the optimal solution [35]. Theoretical results are therefore discussed first, and used as a basis from which subsequent numerical results in Chapters 4 and 5 can be validated.

This chapter is organized as follows. First, a review of the substantial literature in powered descent is provided in §3.1. Next, a technical discussion of the 3-DOF translation-only landing problem is provided in §3.2. The 3-DOF problem has been studied since (at least) the days of the Apollo program, and its development arc provides a natural starting point to frame our extensions to more general landing problems. Section 3.3 provides a formulation and solution of the planar landing problem that begins to consider how the inclusion of attitude variables affect the optimal solution of powered descent guidance problems. Section 3.4 provides a general problem statement for the 6-DOF landing problem. Given the complexity of the 6-DOF problem, no analytical solutions exist, to the best of our knowledge, in the same way that they do for the 3-DOF and planar landing problems. We will therefore turn to iterative numerical algorithms in Chapter 4 to solve 6-DOF powered descent guidance problems.

3.1 Review of Past Work

Research in the area of powered descent guidance began in earnest when the Apollo program set its sights on achieving a piloted lunar landing. Until that point, no rocket-powered

spacecraft had achieved a controlled soft landing on a celestial body. While the Apollo lunar modules were piloted by humans, the guidance system was capable of landing the spacecraft autonomously [35]. Apollo guidance is a 3-DOF translation guidance method, with attitude commands computed separately using the generated thrust commands in a cascaded architecture. Despite the popularity of powered descent guidance as a modern research topic, Apollo-derived methods still form the core of nearly all guidance methods that are flown on real missions [36, 37, 38].

While Apollo guidance is not, strictly speaking, optimal in any sense, its designers were aware of theoretical work that studied the problem of propellant-optimal soft landings [39, 40, 41]. The work of Lawden in [40] is often credited as the originator of several key insights; he developed analytical expressions for the optimal trajectories of the general 3-DOF guidance problem. Some of these results are summarized in §3.2. At the time, however, numerical solutions to such problems were difficult to obtain. The first computationally tractable solutions sufficient for a flight implementation were limited to 1-DOF vertical descent trajectories [41]. These early results, obtained by using the maximum principle, were quite promising, but were not developed in time for use during the Apollo program. The end of the Apollo program came with it a quiescent period for powered descent research that began in the late 1970s. The major shift to the Shuttle program and its use of the reliable PEG guidance [42, 43] meant that only a few authors continued to develop methods for rocket-powered descent for non-Earth bodies [44, 45, 46, 47]. It would be nearly 40 years before the next major results appeared.

Research on propellant-optimal powered descent garnered renewed interest due to robotic Martian landing missions around the turn of the century. D'Souza investigated a simplified (no mass variable) 3-DOF landing problem using a combination of minimum acceleration and final time as the cost [48]. No path constraints were imposed on state or control variables, but an analytic feedback control law was realized. Topcu, Casoliva and Mease presented results for the 3-DOF landing problem using the maximum principle [49, 50]. Their results largely mirror Lawden's, though numerical trials using newly available software were provided to

confirm theoretical results. Several authors have continued to develop analytical solutions to the first-order necessary conditions for the 3-DOF problem [51, 52, 53]. Very efficient numerical routines for obtaining optimal thrust programs have been developed [51, 54], though these algorithms can in general only handle a very limited set of state and control constraints.

At the same time, Açıkmeşe and Ploen published work with an alternate viewpoint on the 3-DOF problem [55, 56, 57]. These works took a convex programming approach, and showed that a nonconvex lower bound on thrust magnitude can be relaxed to a convex constraint by using a slack variable. Using the maximum principle, they showed that in fact this relaxation is lossless, in the sense that the relaxed problem yields the same optimal solution as the original problem. A subsequent change of variables and an approximation led to a fully convex problem formulation that can be solved efficiently (see §3.2). This lossless convexification result bridged the remaining gap between theoretical understanding of the 3-DOF guidance problem and the ability to obtain numerical solutions quickly and efficiently. The theory of lossless convexification has since been expanded to nonconvex thrust pointing constraints [58, 59, 60], minimum-landing error problems [61] and more general optimal control problems [62, 63, 64, 65]. These efforts culminated in the development of a custom second-order cone programming solver [15, 16] and successful tests of the guidance system in terrestrial flight tests [66, 67].

Each of the references given thus far present guidance methods where it is assumed that the attitude commands are computed separately using a cascaded architecture (i.e., thrust commands feed an inner-loop attitude controller). More recent work has explored the generalized 6-DOF guidance problem that considers both translational and rotational motion of a rigid body. A driving requirement for this generalization is the need to model vision-based pointing constraints for modern navigation sensors [34, 68]. Such constraints couple the rotation and translation of the vehicle, and cannot be easily enforced when the two are considered separately.

A theoretical characterization of the solution(s) to the propellant-optimal 6-DOF problem

remains an active area of research. As a result, we define and analyze an intermediate problem, the *planar landing problem*, that can be understood to be a special case of the 6-DOF problem, but one for which we are able to derive theoretical properties of the optimal solution using the maximum principle [69]. The planar landing problem has translational motion that is restricted to a two-dimensional plane, and a single attitude variable. The results can be used to (partially) validate numerical algorithms designed to solve the more general 6-DOF problem, and represent an incremental step forward in our understanding of propellant-optimal powered descent.

Previous work on 6-DOF powered descent include the use of Lyapunov techniques [70], model predictive control [32, 71] and more recently feedforward trajectory generation techniques [72, 73, 74, 75, 76, 77, 78]. Each of these references present numerical solution techniques to the 6-DOF landing problem. The state variables selected for the 6-DOF problem formulation can be used to classify various methods. For example, one may use standard Cartesian variables in conjunction with unit quaternions, homogeneous transformation matrices, dual quaternions, or another parameterization. There are relevant advantages and disadvantages between the various parameterizations, but this distinction is largely a design choice, and we shall focus on the use of unit dual quaternions.

3.2 3-DOF Landing Problems

The 3-DOF guidance problem focuses solely on the translational aspects of the powered descent problem; an assumption that is equivalent to assuming that the vehicle itself is represented as a point mass for the sake of trajectory generation. We assume that the attitude of the landing vehicle may be controlled well enough to follow the computed thrust commands. A surface-fixed reference frame, $\mathcal{F}_{\mathcal{L}}$, is assumed to have its origin at the intended

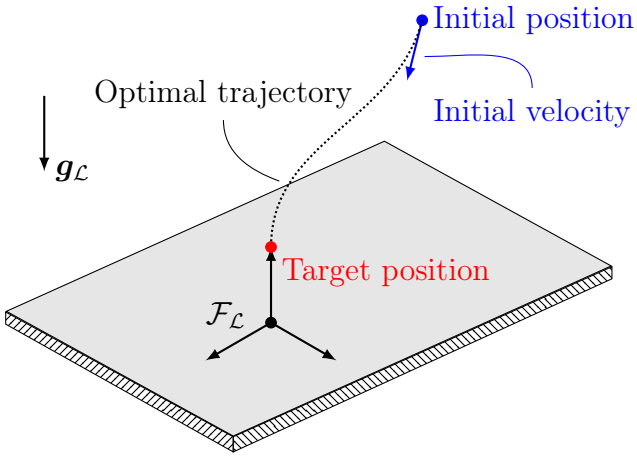


Figure 3.1: A notional 3-DOF landing scenario where the vehicle is approximated as a point mass for the sake of trajectory generation.

landing site with basis vectors $\{\boldsymbol{x}_{\mathcal{L}}, \boldsymbol{y}_{\mathcal{L}}, \boldsymbol{z}_{\mathcal{L}}\}$. The vehicle's state vector is taken to be

$$\boldsymbol{x}(t) = \begin{bmatrix} m(t) \\ \boldsymbol{r}_{\mathcal{L}}(t) \\ \boldsymbol{v}_{\mathcal{L}}(t) \end{bmatrix} \in \mathbb{R}^7. \tag{3.1}$$

Here, $m(t) \in \mathbb{R}_{++}$ denotes the vehicle's mass, $\boldsymbol{r}_{\mathcal{L}}(t) \in \mathbb{R}^3$ denotes the surface-fixed position vector and $\boldsymbol{v}_{\mathcal{L}}(t) \in \mathbb{R}^3$ denotes the surface-fixed velocity vector. For the present study of optimal solutions, the following assumptions are made: the rotation rate of the planet can be ignored and the frame $\mathcal{F}_{\mathcal{L}}$ may be approximated as inertial, aerodynamic forces (if present) can be ignored for trajectory design and subsequently treated as a disturbance, the landing takes place sufficiently close to the surface that the gravitational acceleration vector may be taken as a constant. A notional landing scenario is depicted in Figure 3.1.

Remark 3.1. *Each of these assumptions may be relaxed to a varying degree and analytic optimal trajectories still obtained. For example, [40, 79] provide solutions to the same 3-DOF landing problem with linear and central gravity models. For the lunar descent scenarios that motivated this work, the added fidelity of these gravity models was verified to be negligible*

for terminal descent maneuvers, and is hence not considered further. References [58, 60, 66] discuss the addition of planetary rotation, a feature that is required when the distance traveled during a descent maneuver is non-negligible compared to the planetary radius.

The mass is assumed to vary as a linear function of the thrust magnitude according to

$$\dot{m}(t) = -\alpha \|\boldsymbol{u}_{\mathcal{L}}(t)\|_2, \qquad \alpha \coloneqq \frac{1}{I_{sp} g_e}, \tag{3.2}$$

where $\boldsymbol{u}_{\mathcal{L}}(t) \in \mathbb{R}^3$ is the thrust vector, I_{sp} is the vacuum specific impulse and g_e is the acceleration due to gravity at sea level on Earth. The kinematics and dynamics governing the position and velocity states are then

$$\dot{\boldsymbol{r}}_{\mathcal{L}}(t) = \boldsymbol{v}_{\mathcal{L}}(t), \quad \dot{\boldsymbol{v}}_{\mathcal{L}}(t) = \frac{1}{m(t)} \boldsymbol{u}_{\mathcal{L}}(t) + \boldsymbol{g}_{\mathcal{L}} \tag{3.3}$$

where $\boldsymbol{g}_{\mathcal{L}} \in \mathbb{R}^3$ is the (constant) gravitational acceleration vector. The thrust magnitude is constrained according to

$$u_{\min} \leq \|\boldsymbol{u}_{\mathcal{L}}(t)\|_2 \leq u_{\max} \tag{3.4}$$

where $0 < u_{\min} \leq u_{\max}$ defines the permissible range for the thrust magnitude. The boundary conditions enforce a given position and velocity at both the initial and final times, but only the initial mass is fixed:

$$m(t_0) = m_{ic}, \quad \boldsymbol{r}_{\mathcal{L}}(t_0) = \boldsymbol{r}_{\mathcal{L},ic}, \quad \boldsymbol{v}_{\mathcal{L}}(t_0) = \boldsymbol{v}_{\mathcal{L},ic}, \tag{3.5a}$$

$$\boldsymbol{r}_{\mathcal{L}}(t_f) = \boldsymbol{r}_{\mathcal{L},f}, \quad \boldsymbol{v}_{\mathcal{L}}(t_f) = \boldsymbol{v}_{\mathcal{L},f}. \tag{3.5b}$$

The initial time $t_0 \in \mathbb{R}_+$ is assumed to be fixed and without loss of generality may be assumed to be zero. The final time $t_f \in \mathbb{R}_{++}$ is a free variable.

A minimum propellant objective is used, and is traditionally formulated in Lagrange form using the two-norm of the thrust $\boldsymbol{u}_{\mathcal{L}}(t)$. It is therefore appropriate to take $L\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\big) =$

$\|\boldsymbol{u}_{\mathcal{L}}(t)\|_2$ in (2.5). In some cases it can be useful to leverage (3.2) and note that

$$\begin{aligned}\int_{t_0}^{t_f} \|\boldsymbol{u}_{\mathcal{L}}(t)\|_2 \,\mathrm{d}t &= \int_{t_0}^{t_f} -\frac{1}{\alpha}\dot{m}(t)\,\mathrm{d}t, \\ &= -\frac{1}{\alpha}\left(m(t_f) - m(t_0)\right), \\ &\propto -m(t_f), \end{aligned} \tag{3.6}$$

because $m(t_0)$ and α are both given constants. The minimum propellant objective can therefore be equivalently cast in Mayer form by using $M\big(\boldsymbol{x}(t_0), \boldsymbol{x}(t_f), \boldsymbol{p}\big) = -m(t_f)$. The distinction is primarily one of taste, and all theoretical results hold regardless of which form is used. Other cost functions, such as minimum-landing error [51, 61] or minimum-time problems are also possible.

Given the dynamics, constraints, boundary conditions and cost, the 3-DOF powered descent guidance problem can be stated as the following optimal control problem.

Problem 2. *Find the piecewise continuous control function $\boldsymbol{u}_{\mathcal{L}}(\cdot)$ and final time $t_f \in \mathbb{R}_{++}$ that solve the following problem:*

$$\min_{\boldsymbol{u}_{\mathcal{L}}(\cdot), t_f} \quad \int_{t_0}^{t_f} \|\boldsymbol{u}_{\mathcal{L}}(t)\|_2 \,\mathrm{d}t \tag{3.7a}$$

$$s.t. \quad \dot{m}(t) = -\alpha\|\boldsymbol{u}_{\mathcal{L}}(t)\|_2, \quad \dot{\boldsymbol{r}}_{\mathcal{L}}(t) = \boldsymbol{v}_{\mathcal{L}}(t), \quad \dot{\boldsymbol{v}}_{\mathcal{L}}(t) = \frac{1}{m(t)}\boldsymbol{u}_{\mathcal{L}}(t) + \boldsymbol{g}_{\mathcal{L}}, \tag{3.7b}$$

$$m(t_0) = m_{ic}, \quad \boldsymbol{r}_{\mathcal{L}}(t_0) = \boldsymbol{r}_{\mathcal{L},ic}, \quad \boldsymbol{v}_{\mathcal{L}}(t_0) = \boldsymbol{v}_{\mathcal{L},ic}, \tag{3.7c}$$

$$\boldsymbol{r}_{\mathcal{L}}(t_f) = \boldsymbol{r}_{\mathcal{L},f}, \quad \boldsymbol{v}_{\mathcal{L}}(t_f) = \boldsymbol{v}_{\mathcal{L},f}, \tag{3.7d}$$

$$u_{\min} \leq \|\boldsymbol{u}_{\mathcal{L}}(t)\|_2 \leq u_{\max} \tag{3.7e}$$

For clarity of exposition in discussing the optimal solutions to Problem 2, the argument of time, t, shall be dropped unless it is needed to explicitly refer to a boundary condition.

The Hamiltonian for Problem 2 is given by

$$\mathcal{H} = \lambda_0 \|\boldsymbol{u}_{\mathcal{L}}\|_2 - \alpha\lambda_m \|\boldsymbol{u}_{\mathcal{L}}\|_2 + \boldsymbol{\lambda}_r^\top \boldsymbol{v}_{\mathcal{L}} + \boldsymbol{\lambda}_v^\top \left(\frac{1}{m}\boldsymbol{u}_{\mathcal{L}} + \boldsymbol{g}_{\mathcal{L}}\right). \tag{3.8}$$

The optimal thrust can be decomposed into a direction and magnitude using $\boldsymbol{u}_{\mathcal{L}} = \Gamma \boldsymbol{p}$, where now $\Gamma = \|\boldsymbol{u}_{\mathcal{L}}\|_2 \in \mathbb{R}$ must satisfy the constraint $u_{\min} \leq \Gamma \leq u_{\max}$ and $\boldsymbol{p} \in \mathbb{R}^3$ is a unit vector. The inputs $\Gamma, \boldsymbol{p}$ can be thought of as replacing the input $\boldsymbol{u}_{\mathcal{L}}$, provided that the additional constraint $\|\boldsymbol{p}\|_2 = 1$ is satisfied. The Hamiltonian (3.8) can be rewritten in terms of these two inputs as

$$\mathcal{H} = (\lambda_0 - \alpha\lambda_m)\,\Gamma + \frac{\Gamma}{m}\boldsymbol{\lambda}_v^\top \boldsymbol{p} + \boldsymbol{\lambda}_r^\top \boldsymbol{v}_{\mathcal{L}} + \boldsymbol{\lambda}_v^\top \boldsymbol{g}_{\mathcal{L}}. \tag{3.9}$$

Invoking the maximum principle – in particular (2.9) – the optimal thrust *direction* is the choice of unit vector $\boldsymbol{p}$ that maximizes (3.9). Because Γ and m are both positive numbers, this corresponds to an optimal direction of

$$\boldsymbol{p}^* = \frac{\boldsymbol{\lambda}_v}{\|\boldsymbol{\lambda}_v\|_2}. \tag{3.10}$$

The vector $\boldsymbol{p}^*$ is so important to the solution of this problem that Lawden termed it the *primer vector* [40]. The optimal thrust vector is always in the direction of the primer vector for the 3-DOF problem. Using (3.10), the Hamiltonian can be simplified to

$$\mathcal{H} = \left(\lambda_0 - \alpha\lambda_m + \frac{1}{m}\|\boldsymbol{\lambda}_v\|_2\right)\Gamma + \boldsymbol{\lambda}_r^\top \boldsymbol{v}_{\mathcal{L}} + \boldsymbol{\lambda}_v^\top \boldsymbol{g}_{\mathcal{L}}. \tag{3.11}$$

The costate equations given by (2.8) are then

$$\dot{\lambda}_m = \frac{\Gamma}{m^2}\|\boldsymbol{\lambda}_v\|_2, \quad \dot{\boldsymbol{\lambda}}_r = 0, \quad \dot{\boldsymbol{\lambda}}_v = -\boldsymbol{\lambda}_r, \tag{3.12}$$

and the transversality conditions are found by applying (2.10) to be

$$\lambda_m(t_f) = 0, \quad \mathcal{H}\big(\pi(t_f)\big) = 0. \tag{3.13}$$

Recall that the definition of $\pi(t_f)$ is given in Theorem 2.2. An immediate consequence of (3.12) is that

$$\boldsymbol{\lambda}_v = -\boldsymbol{\lambda}_r t + \boldsymbol{\lambda}_{v,0}, \tag{3.14}$$

where $\boldsymbol{\lambda}_r, \boldsymbol{\lambda}_{v,0} \in \mathbb{R}^3$ are both constant vectors. That $\boldsymbol{\lambda}_v$ is a linear function of time will prove crucial in determining the time-profile of the thrust magnitude Γ along optimal solutions. The main result for this problem is summarized below, various accounts of which can be found in [40, 47, 50, 51, 56, 80].

Theorem 3.2. *The optimal thrust magnitude for Problem 2, $\Gamma^* = \|\boldsymbol{u}^*_{\mathcal{L}}\|_2$, satisfies either $\Gamma^* = u_{\min}$ or $\Gamma^* = u_{\max}$ and has at most three subarcs in the order of max-min-max.*

Proof. Using (3.11), define the *switching function*

$$S = \lambda_0 - \alpha\lambda_m + \frac{1}{m}\|\boldsymbol{\lambda}_v\|_2. \tag{3.15}$$

According to the maximization criteria (2.9) of the maximum principle, the optimal thrust magnitude is given by

$$\Gamma^* = \begin{cases} u_{\min}, & S < 0, \\ u_{\max}, & S > 0, \\ \in (u_{\min}, u_{\max}), & S = 0. \end{cases} \tag{3.16}$$

The third case where $S = 0$ is referred to as a *singular arc*. The statement of this theorem says that singular arcs are not possible along optimal solutions. To see why, assume for a contradiction that $S = 0$ for some interval of time $[t_1, t_2] \subset [t_0, t_f]$ with $t_1 < t_2$. On this interval, $\dot{S} = 0$ must also hold, where

$$\dot{S} = -\alpha\dot{\lambda}_m - \frac{\dot{m}}{m^2}\|\boldsymbol{\lambda}_v\|_2 - \frac{1}{m\|\boldsymbol{\lambda}_v\|_2}\boldsymbol{\lambda}_v^\top\boldsymbol{\lambda}_r \tag{3.17a}$$

$$= -\frac{\alpha\Gamma^*}{m^2}\|\boldsymbol{\lambda}_v\|_2 + \frac{\alpha\Gamma^*}{m^2}\|\boldsymbol{\lambda}_v\|_2 - \frac{1}{m\|\boldsymbol{\lambda}_v\|_2}\boldsymbol{\lambda}_v^\top\boldsymbol{\lambda}_r \tag{3.17b}$$

$$= \frac{1}{m\|\boldsymbol{\lambda}_v\|_2}\left(\boldsymbol{\lambda}_r^\top\boldsymbol{\lambda}_r t - \boldsymbol{\lambda}_r^\top\boldsymbol{\lambda}_{v,0}\right) \tag{3.17c}$$

If $\dot{S} = 0$ over an interval of time, then $\boldsymbol{\lambda}_r^\top \boldsymbol{\lambda}_r = 0$ and $\boldsymbol{\lambda}_r^\top \boldsymbol{\lambda}_{v,0} = 0$ because each have constant values. Both $\boldsymbol{\lambda}_r$ and $\boldsymbol{\lambda}_{v,0}$ cannot both be zero, since (3.14) would imply that $\boldsymbol{\lambda}_v = 0$ for all times $t \in [t_0, t_f]$. If this were the case, then $\lambda_m(t_f) = 0$ in conjunction with (3.12) would imply that $\lambda_m = 0$ for all times as well. The final transversality condition $\mathcal{H}(\pi(t_f)) = 0$ would then imply that $\lambda_0 = 0$, and thus the entire vector $(\lambda_0, \boldsymbol{\lambda})$ would vanish, violating the maximum principle's non-degeneracy clause. Therefore the singular conditions $\boldsymbol{\lambda}_r^\top \boldsymbol{\lambda}_r = 0$ and $\boldsymbol{\lambda}_r^\top \boldsymbol{\lambda}_{v,0} = 0$ imply only that $\boldsymbol{\lambda}_r = 0$. Through (3.14), the primer vector $\boldsymbol{\lambda}_v = \boldsymbol{\lambda}_{v,0}$ must be constant in time. The transversality condition (3.13) then leads to

$$\mathcal{H}\big(\pi(t_f^*)\big) = \boldsymbol{\lambda}_{v,0}^\top \boldsymbol{g}_{\mathcal{L}} = 0, \tag{3.18}$$

which implies that the primer vector (and hence thrust direction) is orthogonal to the gravitational acceleration everywhere along the optimal solution. Based on the assumption of constant and vertical gravity, this leads to the conclusion that in the $\boldsymbol{z}_{\mathcal{L}}$ direction the vehicle is in free fall. Any boundary conditions for which the final vertical velocity is smaller than the initial vertical velocity (in absolute value) are therefore not achievable. This contradicts nearly all boundary conditions of interest in powered descent. Hence a singular arc is not optimal and $S \neq 0$ for almost all $t \in [t_0, t_f]$.[2] See [56] for a more general proof of this contradiction for any boundary conditions.

To establish the max-min-max structure of Γ^*, consider the function $\dot{S}$ given by (3.17c). The sign of this derivative is governed by the bracketed term, and may only switch once from negative to positive as $t \to t_f$ because $\ddot{S} > 0$. This implies that the value of S may change sign at most twice, leading to at most three subarcs where either $\Gamma^* = u_{\min}$ or $\Gamma^* = u_{\max}$. Because the sign of $\ddot{S}$ is positive, the sign of S can only go from positive to negative to positive, and therefore the most general switching structure is max-min-max. □

It is worth noting that even with more general gravitational models, up to and including

[2]The "almost all" here refers specifically to cases where the function S may have a zero-crossing, so that $S = 0$ can occur for a single instance of time, but not an interval of time.

a central Keplerian model, the result that $\Gamma^* = u_{\min}$ or $\Gamma^* = u_{\max}$ holds. However, these more general models can lead to a different switching structure and result in more subarcs than simply max-min-max [40, 51].

3.2.1 Lossless Convexification

The knowledge of the switching structure for the 3-DOF powered descent guidance problem permits the design of numerical algorithms that solve the necessary conditions of optimality. The most mature technology that does so is likely the *universal powered guidance* algorithm detailed in [51, 54]. This approach is limited however, to the problem statement given in Problem 2 and the use of nonlinear root finding techniques. Additional input or state constraints cannot typically be added without reproving a new version of Theorem 3.2, and adjusting the numerical algorithm. A potentially better approach is that of *lossless convexification*, whereby the proof of Theorem 3.2 is used to establish the equivalence of Problem 2 and a relaxed problem whose control constraint set is convex. Note that the set of feasible controls in Problem 2 is nonconvex due to (3.7e). This result was first demonstrated in [56] and a brief review of the main idea is presented here.

Consider the following relaxed version of Problem 2.

Problem 3. *Find the piecewise continuous functions* $\Gamma(\cdot)$ *and* $\boldsymbol{u}_{\mathcal{L}}(\cdot)$ *and final time* $t_f \in \mathbb{R}_{++}$ *that solve the following problem:*

$$\min_{\Gamma(\cdot),\boldsymbol{u}_{\mathcal{L}}(\cdot),t_f} \quad \int_{t_0}^{t_f} \Gamma(t)\,\mathrm{d}t \tag{3.19a}$$

$$s.t. \quad \dot{m}(t) = -\alpha\Gamma(t), \quad \dot{\boldsymbol{r}}_{\mathcal{L}}(t) = \boldsymbol{v}_{\mathcal{L}}(t), \quad \dot{\boldsymbol{v}}_{\mathcal{L}}(t) = \frac{1}{m(t)}\boldsymbol{u}_{\mathcal{L}}(t) + \boldsymbol{g}_{\mathcal{L}}, \tag{3.19b}$$

$$m(t_0) = m_{ic}, \quad \boldsymbol{r}_{\mathcal{L}}(t_0) = \boldsymbol{r}_{\mathcal{L},ic}, \quad \boldsymbol{v}_{\mathcal{L}}(t_0) = \boldsymbol{v}_{\mathcal{L},ic}, \tag{3.19c}$$

$$\boldsymbol{r}_{\mathcal{L}}(t_f) = \boldsymbol{r}_{\mathcal{L},f}, \quad \boldsymbol{v}_{\mathcal{L}}(t_f) = \boldsymbol{v}_{\mathcal{L},f}, \tag{3.19d}$$

$$u_{\min} \leq \Gamma(t) \leq u_{\max} \tag{3.19e}$$

$$\|\boldsymbol{u}_{\mathcal{L}}(t)\|_2 \leq \Gamma(t) \tag{3.19f}$$

The new constraint (3.19f) is a second-order cone, and hence the control $\big(\boldsymbol{u}_{\mathcal{L}}(t), \Gamma(t)\big)$ takes its values in a convex set $\mathcal{U}$ for all times $t \in [t_0, t_f]$. The Hamiltonian for Problem 3 has exactly the form of (3.9) but with $\boldsymbol{p}$ replaced by $\boldsymbol{u}_{\mathcal{L}}$. The same invocation of the maximum principle results in $\|\boldsymbol{u}^*_{\mathcal{L}}\|_2 = \Gamma^*$. This implies that $u_{\min} \leq \|\boldsymbol{u}^*_{\mathcal{L}}\|_2 \leq u_{\max}$ and hence the input $\boldsymbol{u}^*_{\mathcal{L}}$ is a *feasible* solution for Problem 2 as well. The exact same steps given in the proof of Theorem 3.2 lead to the conclusion that either $\Gamma^* = u_{\min}$ or $\Gamma^* = u_{\max}$, and there is at most three subarcs with a max-min-max order. Hence $\boldsymbol{u}^*_{\mathcal{L}}$ is *optimal* for Problem 2.

What stands between Problem 3 and simply applying a transcription method to obtain a convex optimization problem is the nonlinear dynamics that govern $\boldsymbol{v}_{\mathcal{L}}$ in (3.19b). This is remedied by the change of variables:

$$\sigma \coloneqq \frac{\Gamma}{m}, \quad \boldsymbol{\nu}_{\mathcal{L}} \coloneqq \frac{\boldsymbol{u}_{\mathcal{L}}}{m}, \quad z \coloneqq \ln m. \tag{3.20}$$

The variable z replaces the mass m in the problem formulation, and has the dynamics $\dot{z} = -\alpha\sigma$. The previously nonlinear dynamics for $\boldsymbol{v}_{\mathcal{L}}$ become a linear function of $\boldsymbol{\nu}_{\mathcal{L}}$, and the convex constraint (3.19f) is equivalent to $\|\boldsymbol{\nu}_{\mathcal{L}}\|_2 \leq \sigma$. However, the constraint (3.19e) becomes nonconvex in z and σ:

$$u_{\min} e^{-z} \leq \sigma \leq u_{\max} e^{-z}. \tag{3.21}$$

This constraint can be approximated to within 2% for powered descent maneuvers that take less than 70 s with the following convex constraints [56]:

$$\sigma \leq u_{\max}\big(1 - (z - z_0)\big), \tag{3.22a}$$

$$\sigma \geq u_{\min}\left(1 - (z - z_0) + \frac{1}{2}(z - z_0)^2\right), \tag{3.22b}$$

where $z_0 = \ln(m_{ic} - \alpha u_{\max} t)$. The constraints (3.22) represent a Taylor series approximation to the constraint (3.21) about z_0, which is a lower bound on z everywhere. The additional

constraint $z_0 \leq z \leq \ln(m_{ic} - \alpha u_{\min} t)$ ensures that the new variable z satisfies the physical constraints required of the vehicle's mass. In summary, the relaxed problem with the change of variables and subsequent approximation can be stated as follows.

Problem 4. *Find the piecewise continuous functions $\sigma(\cdot)$ and $\boldsymbol{\nu}(\cdot)$ and final time $t_f \in \mathbb{R}_{++}$ that solve the following problem:*

$$
\begin{aligned}
\min_{\sigma(\cdot),\boldsymbol{\nu}_{\mathcal{L}}(\cdot),t_f} \quad & \int_{t_0}^{t_f} \sigma(t)\,\mathrm{d}t \\
s.t. \quad & \dot{z}(t) = -\alpha\sigma(t), \quad \dot{\boldsymbol{r}}_{\mathcal{L}}(t) = \boldsymbol{v}_{\mathcal{L}}(t), \quad \dot{\boldsymbol{v}}_{\mathcal{L}}(t) = \boldsymbol{\nu}_{\mathcal{L}}(t) + \boldsymbol{g}_{\mathcal{L}}, \\
& z(t_0) = z_0, \quad \boldsymbol{r}_{\mathcal{L}}(t_0) = \boldsymbol{r}_{\mathcal{L},ic}, \quad \boldsymbol{v}_{\mathcal{L}}(t_0) = \boldsymbol{v}_{\mathcal{L},ic}, \\
& \qquad\qquad \boldsymbol{r}_{\mathcal{L}}(t_f) = \boldsymbol{r}_{\mathcal{L},f}, \quad \boldsymbol{v}_{\mathcal{L}}(t_f) = \boldsymbol{v}_{\mathcal{L},f}, \\
& u_{\min}\left(1 - (z(t) - z_0(t)) + \frac{1}{2}(z(t) - z_0(t))^2\right) \leq \sigma(t) \leq u_{\max}\left(1 - \left(z(t) - z_0(t)\right)\right) \\
& \ln(m_0 - \alpha u_{\max} t) \leq z(t) \leq \ln(m_0 - \alpha u_{\min} t), \\
& \|\boldsymbol{\nu}_{\mathcal{L}}(t)\|_2 \leq \sigma(t)
\end{aligned}
$$

Problem 4 is now a convex optimal control problem, whose solution closely approximates that of the nonconvex optimal control problem given by Problem 2.

The (approximate) equivalence of Problem 4 and Problem 2 can be extended to the case of a minimum-landing-error cost function [61], and the addition of thrust pointing constraints [58, 60]. Similar lossless convexification results can also be derived for more general linear and nonlinear systems [63, 64, 81]. The use of convex optimization to obtain numerical solutions for a variety of problem statements, its demonstration in a real flight test (G-FOLD, see [66, 67]) and the fairly complete theoretical understanding of the problem, all support the conclusion that the 3-DOF problem is a solved problem in most any sense of the term.

3.3 Planar Landing Problems

Until recently, the 3-DOF problem was the most general powered descent problem for which the structure of the optimal control solution, in the sense of Theorem 3.2, was understood. The 6-DOF problem that we present in §3.4 is a much broader problem statement for a rocket-powered rigid body – but a theoretical characterization of the propellant-optimal solution(s) to this problem is a very challenging area of active research. Rather than attempting to directly solve the more general 6-DOF problem, this section takes an incremental step beyond the 3-DOF scenario by introducing a single attitude variable and restricting the translational motion to a plane that is fixed in the landing frame.

We call this intermediate problem the *planar landing problem*, and it can be understood as a special case of the 6-DOF problem [69]. Fortunately, we are able to derive characteristics of the propellant-optimal solutions to this problem using the maximum principle, and relate these characteristics to what we know from the 3-DOF analysis in §3.2. This understanding of the planar landing problem can be used to (partially) validate numerical algorithms designed to solve the 6-DOF problem. By choosing the problem data to align with the assumptions of the planar landing problem formulation, we would expect the numerical algorithm to return trajectories that are in line with the to-be-derived theoretical properties of the planar landing problem.

Because we now consider the orientation of the (non-point mass) vehicle, we introduce a body-fixed coordinate frame $\mathcal{F}_{\mathcal{B}}$ with origin at the center of mass of the (rigid) vehicle and coordinate vectors $\{\boldsymbol{x}_{\mathcal{B}}, \boldsymbol{y}_{\mathcal{B}}, \boldsymbol{z}_{\mathcal{B}}\}$. We describe the vehicle's motion in the same landing frame $\mathcal{F}_{\mathcal{L}}$ as used in §3.2, and we maintain the same assumptions that planetary rotation and any atmospheric effects can be ignored for the study of optimal descent trajectories. We further assume that the acceleration due to gravity is constant, so that $\boldsymbol{g}_{\mathcal{L}} = -g\boldsymbol{z}_{\mathcal{L}}$, where g is the acceleration due to gravity at the surface of the planet under consideration.

The additional assumption that leads to planar motion is that the initial position and velocity vectors, in the landing frame, lie in a plane that contains the landing site. In this

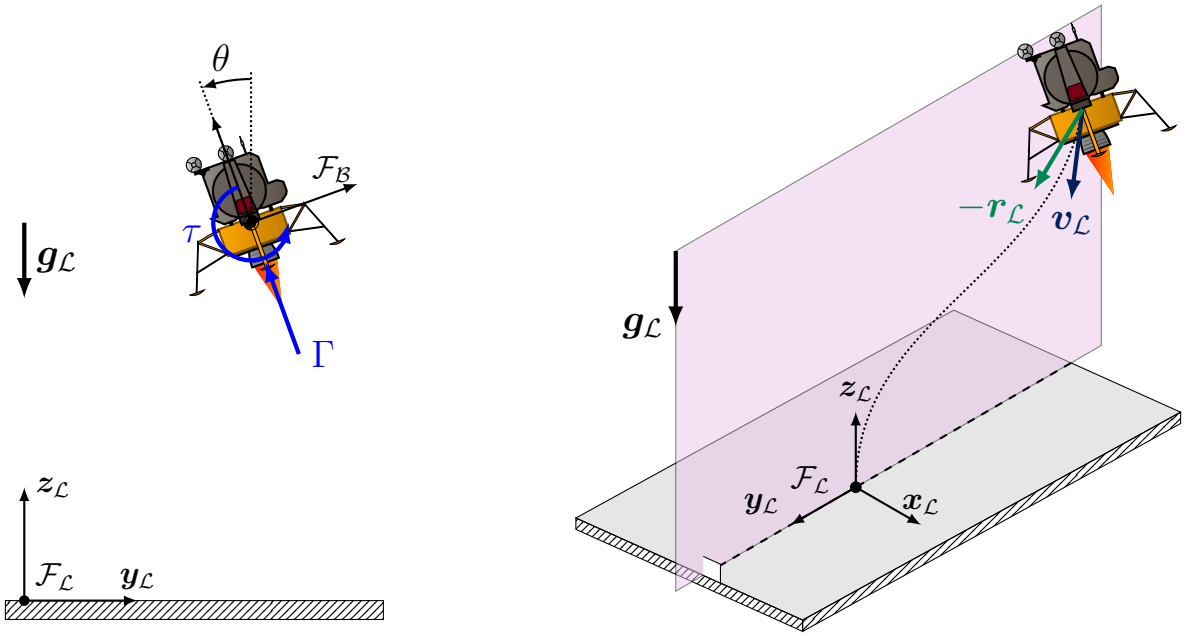


Figure 3.2: A notional planar landing scenario, where the vehicle's motion is restricted to a plane and attitude is now considered as part of the problem.

case, we may assume without loss of generality that the vehicle's motion evolves in the $\boldsymbol{y}_\mathcal{L}$-$\boldsymbol{z}_\mathcal{L}$ plane.[3] See Figure 3.2 for a depiction. The inertial position and velocity vectors can then be written as $\boldsymbol{r}_\mathcal{L}(t) = \big(y(t),\, z(t)\big) \in \mathbb{R}^2$ and $\boldsymbol{v}_\mathcal{L}(t) = \big(v_y(t),\, v_z(t)\big) \in \mathbb{R}^2$, and the attitude of the vehicle can be represented by a single angle, which we denote by $\theta(t) \in \mathbb{R}$, and define to be the angle formed between $\boldsymbol{z}_\mathcal{B}$ and $\boldsymbol{z}_\mathcal{L}$.

We assume that a single main engine and separate torque generating actuators provide the control authority during the descent. The main engine is assumed to be rigidly affixed to the vehicle so that the thrust direction is fixed in the body frame and passes through the center of mass. The main engine has a variable thrust magnitude denoted by $\Gamma(t) \in \mathbb{R}$, and the independent torque input is denoted by $\tau(t) \in \mathbb{R}$. See Fig. 3.2 for an illustration of the control configuration.

[3] Provided that there are no state constraints that couple the rotational and translational motion, see [73] or §4.3 for an example where out-of-plane motion can be induced by such a constraint.

The thrust vector can be expressed in the landing frame as the product of the thrust magnitude and the (unit) direction vector that is a function of the vehicle's attitude:

$$\boldsymbol{u}_{\mathcal{L}}(t) = \Gamma(t)\boldsymbol{d}(t), \quad \boldsymbol{d}(t) \coloneqq \begin{bmatrix} -\sin\theta(t) \\ \cos\theta(t) \end{bmatrix} \in \mathbb{R}^2. \tag{3.24}$$

The control variables are the scalar-valued functions $\Gamma(\cdot)$ and $\tau(\cdot)$, each of which are assumed to be piecewise continuous. The state vector is taken to be

$$\boldsymbol{x}(t) = \begin{bmatrix} m(t) \\ \boldsymbol{r}_{\mathcal{L}}(t) \\ \boldsymbol{v}_{\mathcal{L}}(t) \\ \theta(t) \\ \omega(t) \end{bmatrix} \in \mathbb{R}^7, \tag{3.25}$$

where $\omega(t) \in \mathbb{R}$ is the angular velocity of the body frame with respect to the landing frame. Using the assumptions of the planar landing problem, the equations of motion are:

$$\begin{aligned} &\dot{m}(t) = -\alpha\Gamma(t)\,, \quad \dot{\boldsymbol{r}}_{\mathcal{L}}(t) = \boldsymbol{v}_{\mathcal{L}}(t)\,, \quad \dot{\boldsymbol{v}}_{\mathcal{L}}(t) = \frac{\Gamma(t)}{m(t)}\boldsymbol{d}(t) + \boldsymbol{g}_{\mathcal{L}}\,, \\ &\dot{\theta}(t) = \omega(t)\,, \quad \dot{\omega}(t) = \frac{\tau(t)}{J}\,, \end{aligned} \tag{3.26}$$

where α is defined in (3.2) and J is the inertia about the $\boldsymbol{x}_{\mathcal{B}}$ axis in the body frame. Both control inputs are assumed to be bounded according to

$$\Gamma_{\min} \leq \Gamma(t) \leq \Gamma_{\max}, \tag{3.27a}$$

$$-\tau_{\max} \leq \tau(t) \leq \tau_{\max}, \tag{3.27b}$$

where $\Gamma_{\min}, \Gamma_{\max}, \tau_{\max} \in \mathbb{R}_{++}$ are positive real numbers.

We constrain the initial and final states according to the following boundary conditions:

$$m(t_0) = m_{ic}, \quad \boldsymbol{r}_{\mathcal{L}}(t_0) = \boldsymbol{r}_{\mathcal{L},ic}, \quad \boldsymbol{v}_{\mathcal{L}}(t_0) = \boldsymbol{v}_{\mathcal{L},ic}, \qquad \omega(t_0) = 0, \tag{3.28a}$$

$$\boldsymbol{r}_{\mathcal{L}}(t_f) = \boldsymbol{r}_{\mathcal{L},f}, \quad \boldsymbol{v}_{\mathcal{L}}(t_f) = \boldsymbol{v}_{\mathcal{L},f}, \quad \theta(t_f) = 0, \quad \omega(t_f) = 0. \tag{3.28b}$$

Note that the initial attitude is not constrained, a feature whose importance we shall discuss in §3.4 for the 6-DOF problem, and that plays an important role in the results of the planar landing problem as well.

The minimum-propellant planar landing problem is summarized in Problem 5.

Problem 5. *Given an initial time $t_0 \in \mathbb{R}_+$, find the piecewise continuous functions $\Gamma(\cdot)$, $\tau(\cdot)$ and final time $t_f \in \mathbb{R}_{++}$ that solve the following problem:*

$$\min_{t_f, \Gamma(\cdot), \tau(\cdot)} \quad \int_{t_0}^{t_f} \Gamma(t)\, \mathrm{d}t \tag{3.29a}$$

$$s.t. \quad \dot{m}(t) = -\alpha\Gamma(t), \quad \dot{\boldsymbol{r}}_{\mathcal{L}}(t) = \boldsymbol{v}_{\mathcal{L}}(t), \quad \dot{\boldsymbol{v}}_{\mathcal{L}}(t) = \frac{\Gamma(t)}{m(t)}\boldsymbol{d}(t) + \boldsymbol{g}_{\mathcal{L}}, \tag{3.29b}$$

$$\dot{\theta}(t) = \omega(t), \quad \dot{\omega}(t) = \frac{\tau(t)}{J}, \tag{3.29c}$$

$$\Gamma_{\min} \leq \Gamma(t) \leq \Gamma_{\max}, \quad -\tau_{\max} \leq \tau(t) \leq \tau_{\max}, \tag{3.29d}$$

$$m(t_0) = m_{ic}, \quad \boldsymbol{r}_{\mathcal{L}}(t_0) = \boldsymbol{r}_{\mathcal{L},ic}, \quad \boldsymbol{v}_{\mathcal{L}}(t_0) = \boldsymbol{v}_{\mathcal{L},ic}, \qquad \omega(t_0) = 0, \tag{3.29e}$$

$$\boldsymbol{r}_{\mathcal{L}}(t_f) = \boldsymbol{r}_{\mathcal{L},f}, \quad \boldsymbol{v}_{\mathcal{L}}(t_f) = \boldsymbol{v}_{\mathcal{L},f}, \quad \theta(t_f) = 0, \quad \omega(t_f) = 0. \tag{3.29f}$$

Existence of Optimal Solutions to Problem 5

Before discussing the characteristics of an optimal solution to Problem 5, let us first establish the existence of such a solution. To simplify the notation, we henceforth suppress the argument of time, t, except when discussing boundary conditions for which the appropriate time must be specified to avoid ambiguity.

The dynamics in (3.26) can be written in the form $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ for a suitably defined function f by using the definitions of the state vector in (3.25) and the control vector $\boldsymbol{u} =$

(Γ, τ). Moreover, the set of feasible controls given by (3.27) can be denoted by the compact set $\mathcal{U} \subset \mathbb{R}^{n_u}$ for $n_u = 2$. Let the set

$$\mathcal{B} \coloneqq \Big\{ \big(\boldsymbol{x}(t_0), t_0, \boldsymbol{x}(t_f), t_f\big) \;\big|\; t_0 = 0, \text{ (3.28a) and (3.28b) are satisfied} \Big\} \subset \mathbb{R}^{2n_x+2} \tag{3.30}$$

describe the set of admissible endpoints for Problem 5 for $n_x = 7$.

Suppose that there exists a feasible solution to Problem 5. Let $\mathcal{I} \subset \mathbb{R}$ be a compact interval and $\mathcal{X} \subset \mathbb{R}^{n_x}$ and $\mathcal{V} \in \mathbb{R}^{n_u}$ be suitably defined open regions such that any feasible solution satisfies $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{u} \in \mathcal{V}$. The composite function $\big(L\,, f\big) : \mathcal{I} \times \mathcal{X} \times \mathcal{V} \to \mathbb{R}^{n_x+1}$, defined by the (Lagrange) cost function in Problem 5 and the dynamics (3.26), is continuous for the planar landing problem. Furthermore, by assuming that the initial attitude satisfies $\theta(t_0) \in [-\pi, \pi]$, the final time is bounded by some $t_{f,\max}$, and the final mass satisfies $m(t_f) > 0$,[4] the set of admissible endpoints $\mathcal{B}$ is a closed set of points in $\mathbb{R}^{2n_x+2}$ and we may say that $\big(t_0,\, \boldsymbol{x}(t_0)\big) \in \mathcal{I} \times \mathcal{X}$ and $\big(t_f,\, \boldsymbol{x}(t_f)\big) \in \mathcal{I} \times \mathcal{X}$.

The set of admissible controls $\mathcal{U}$ is a compact and convex subset of $\mathcal{V}$ because it is defined by (3.27). Because the dynamics (3.26) are affine in the control vector and $\mathcal{U}$ is convex, there exists a compact set $\mathcal{R} \subset \mathcal{I} \times \mathcal{X}$ such that all feasible trajectories are contained in $\mathcal{R}$. This follows from Filippov's Theorem [4]. Based on these observations, if the set

$$\mathcal{Q}^+(t, \boldsymbol{x}) = \Big\{ \hat{\boldsymbol{y}} = \big(y^0\,, \boldsymbol{y}\big) \mid y^0 \geq L(\boldsymbol{w}),\, \boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{w}),\, \boldsymbol{w} \in \mathcal{U} \Big\} \tag{3.31}$$

is convex, then by [6, Theorem 4.4.2], an optimal solution to Problem 5 exists. But a vector $(y^0,\, \boldsymbol{y}) \in \mathcal{Q}^+(t, \boldsymbol{x})$ – where $\boldsymbol{y} = \big(y_1,\, \boldsymbol{y}_2,\, \boldsymbol{y}_3,\, y_4,\, y_5\big)$ for $y_1,\, y_4,\, y_5 \in \mathbb{R}$ and $\boldsymbol{y}_2,\, \boldsymbol{y}_3 \in \mathbb{R}^2$ – if and only if

$$y^0 \geq w_1, \quad y_1 = -\alpha w_1, \quad \boldsymbol{y}_2 = \boldsymbol{v}_{\mathcal{L}}, \quad \boldsymbol{y}_3 = \frac{w_1}{m}\boldsymbol{d} + \boldsymbol{g}_{\mathcal{L}}, \tag{3.32a}$$

[4] Similar to [56], the limited fuel carried by the vehicle naturally leads to the existence of a $t_{f,\max}$ and non-zero lower bound on $m(t_f)$.

$$y_4 = \omega, \quad y_5 = \frac{w_2}{J}, \quad \Gamma_{\min} \leq w_1 \leq \Gamma_{\max}, \quad -\tau_{\max} \leq w_2 \leq \tau_{\max}. \tag{3.32b}$$

These constitute a set of linear equalities and inequalities, and hence the set $\mathcal{Q}^+(t, \boldsymbol{x})$ is a convex polyhedron. As such, an optimal solution to Problem 5 exists provided that a feasible one does.

Optimal Solutions for Problem 5

The Hamiltonian associated with Problem 5 is

$$\mathcal{H} = \lambda_0 \Gamma - \lambda_m \alpha \Gamma + \boldsymbol{\lambda}_r^\top \boldsymbol{v}_{\mathcal{L}} + \boldsymbol{\lambda}_v^\top \left(\frac{\Gamma}{m} \boldsymbol{d} + \boldsymbol{g}_{\mathcal{L}} \right) + \lambda_\theta \omega + \lambda_\omega \frac{\tau}{J}, \tag{3.33}$$

where $\lambda_0 \leq 0$ and $\boldsymbol{\lambda} \coloneqq \big(\lambda_m, \boldsymbol{\lambda}_r, \boldsymbol{\lambda}_v, \lambda_\theta, \lambda_\omega\big) \in \mathbb{R}^7$ is the costate vector. From (2.8) the following equations govern the evolution of the optimal costate vector,

$$\dot{\lambda}_m = \frac{\Gamma}{m^2} \boldsymbol{\lambda}_v^\top \boldsymbol{d}, \quad \dot{\boldsymbol{\lambda}}_r = 0, \quad \dot{\boldsymbol{\lambda}}_v = -\boldsymbol{\lambda}_r, \quad \dot{\lambda}_\theta = -\frac{\Gamma}{m} \boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta, \quad \dot{\lambda}_\omega = -\lambda_\theta, \tag{3.34}$$

where $\boldsymbol{d}_\theta = \big(-\cos\theta, -\sin\theta \big)$ is the derivative of $\boldsymbol{d}$ with respect to θ. The transversality conditions derived from (2.10) are

$$\mathcal{H}\big(\pi(t_f)\big) = 0, \quad \lambda_\theta(t_0) = 0, \quad \lambda_m(t_f) = 0. \tag{3.35}$$

where the definition of $\pi(t_f)$ is given in Theorem 2.2. Recall that the unit vector in the direction of $\boldsymbol{\lambda}_v$ was termed the primer vector, $\boldsymbol{p}^*$, for the 3-DOF problem, and defined the optimal thrust direction. A key intuitive observation that will guide the analysis of the planar landing problem is that when $\tau_{\max} \to \infty$, the translational components of the optimal planar solution approach those of the optimal 3-DOF solution. This can be seen by noting that $\boldsymbol{d}$ can be made to track $\boldsymbol{p}^*$ with arbitrary precision given an infinite control authority via the input τ. The desire to point $\boldsymbol{d}$ in this direction follows from the integral form of the maximum principle given in [3, Theorem 3.1]. However, finite limitations on τ and boundary

conditions on the attitude variables render the solutions to the 3-DOF and planar problems distinct in general. The unit direction $\boldsymbol{p}^*$ may not be consistent with the attitude boundary conditions and/or change too rapidly for the attitude control system to follow by using the bounded torque input (as captured by $\tau_{\max}$ and J). We might still expect, however, that the primer vector will be tracked as closely as possible. This intuition is formalized in the remainder of this section.

Importantly, we can see that the costate expressions for $\boldsymbol{\lambda}_r$ and $\boldsymbol{\lambda}_v$ in (3.34) match those that occur for the 3-DOF problem in (3.12). Namely, their solutions are

$$\boldsymbol{\lambda}_r(t) = \boldsymbol{\lambda}_r\,, \quad \text{and} \quad \boldsymbol{\lambda}_v(t) = -\boldsymbol{\lambda}_r t + \boldsymbol{\lambda}_{v,0}\,, \tag{3.36}$$

where $\boldsymbol{\lambda}_r$, $\boldsymbol{\lambda}_{v,0} \in \mathbb{R}^2$ are constant vector quantities. Note that these vectors do not have the same numerical value between the 3-DOF problem and a planar landing problem; only the analytic expressions match. (Even if the 3-DOF solution is restricted to a plane and the vectors become effectively two-dimensional.)

For future reference, we define the switching functions associated with the thrust and torque as follows

$$S_\Gamma \coloneqq \frac{1}{m}\boldsymbol{\lambda}_v^\top \boldsymbol{d} - \alpha\lambda_m + \lambda_0, \tag{3.37a}$$

$$S_\tau \coloneqq \frac{1}{J}\lambda_\omega. \tag{3.37b}$$

We first establish the fact that the costate vector corresponding to the velocity is non-zero almost everywhere along an optimal trajectory. This preliminary result is key to subsequent analysis, just as the corresponding statement for the 3-DOF problem was instrumental in proving Theorem 3.2.

Lemma 3.3. *The vector $\boldsymbol{\lambda}_v$ associated with Problem 5 is non-zero for almost all $t \in [t_0, t_f]$.*

Proof. The proof is by contradiction. Suppose that $\boldsymbol{\lambda}_v = 0$ for some interval $t \in [t_1, t_2]$, where $t_0 \leq t_1 < t_2 \leq t_f$. Then (3.36) implies that $\boldsymbol{\lambda}_v = 0$ and $\boldsymbol{\lambda}_r = 0$ for all $t \in [t_0, t_f]$.

By virtue of the second and third transversality conditions in (3.35) and the corresponding costate dynamics in (3.34), we then have that $\lambda_m = 0$ and $\lambda_\theta = 0$ for all $t \in [t_0, t_f]$. Hence $\lambda_\omega = \lambda_{\omega,0}$ for some constant $\lambda_{\omega,0} \in \mathbb{R}$. The transversality condition $\mathcal{H}\big(\pi(t_f)\big) = 0$ then leads to $\lambda_0 = -\tau\lambda_{\omega,0}/J\Gamma$. If $\lambda_{\omega,0}$ is zero, then $\lambda_0 = 0$ and the non-degeneracy clause of the maximum principle is violated, so $\lambda_{\omega,0} \neq 0$. The switching function S_τ in this case implies that the optimal torque must be $\tau = \text{sign}\,(\lambda_{\omega,0})\,\tau_{\max} = \pm\tau_{\max}$ for all $t \in [t_0, t_f]$. The dynamic equation governing the angular velocity from (3.26) then leads to

$$\omega(t_f) - \omega(t_0) = \int_{t_0}^{t_f} \dot{\omega}\, dt = \pm \int_{t_0}^{t_f} \frac{\tau_{\max}}{J} dt \neq 0, \tag{3.38}$$

which contradicts the boundary conditions $\omega(t_f) = \omega(t_0) = 0$ assumed in (3.28). Hence we cannot have $\boldsymbol{\lambda}_v(t) = 0$ along an optimal trajectory. □

The following theorem provides the first of two main results for the planar landing problem, and establishes one of the key connections between the results of the planar landing problem and the results of the 3-DOF problem (c.f. Theorem 3.2).

Theorem 3.4. *The thrust magnitude is non-singular almost everywhere along an optimal solution of Problem 5.*

Proof. The proof is performed in two steps, first by assuming that the torque input is singular, and then by assuming that it is non-singular. If the torque is singular for some interval of time $t \in [t_1,\, t_2]$, where $t_0 \leq t_1 < t_2 \leq t_f$, then the switching function (3.37b) implies that we must have $\lambda_\omega = 0$ for all $t \in [t_1,\, t_2]$. On this same time interval we must then have $\dot{\lambda}_\omega = 0$ and hence $\lambda_\theta = \dot{\lambda}_\theta = 0$. Using the costate equations (3.34) this means that $0 = \boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta$, because Γ and m are strictly positive. Therefore the vector $\boldsymbol{\lambda}_v$ is orthogonal to $\boldsymbol{d}_\theta$ for all times when the torque is singular. Because $\boldsymbol{d}$ and $\boldsymbol{d}_\theta$ are also orthogonal, we conclude that $\boldsymbol{d}$ and $\boldsymbol{\lambda}_v$ must be either parallel or anti-parallel, or in other words, that the optimal thrust direction is aligned with the primer vector and $\boldsymbol{d} = \pm\boldsymbol{p}^*$.

The switching function associated with the thrust magnitude is given by (3.37a). For

$t \in [t_1,\, t_2]$ we then have

$$\dot{S}_\Gamma = \frac{\alpha\Gamma}{m^2}\boldsymbol{\lambda}_v^\top \boldsymbol{d} - \frac{1}{m}\boldsymbol{\lambda}_r^\top \boldsymbol{d} + \frac{\omega}{m}\boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta - \frac{\alpha\Gamma}{m^2}\boldsymbol{\lambda}_v^\top \boldsymbol{d}, \tag{3.39a}$$

$$= -\frac{1}{m}\boldsymbol{\lambda}_r^\top \boldsymbol{d}, \tag{3.39b}$$

because $\boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta = 0$. Note that if $\boldsymbol{\lambda}_r^\top \boldsymbol{d} = 0$, then by virtue of $\boldsymbol{d} = \pm \boldsymbol{p}^*$ we know that $\boldsymbol{\lambda}_r^\top \boldsymbol{\lambda}_v = 0$. From (3.36) this implies that $-\|\boldsymbol{\lambda}_r\|_2^2 t + \boldsymbol{\lambda}_r^\top \boldsymbol{\lambda}_{v,0} = 0$ for all $t \in [t_1, t_2]$. From Lemma 3.3 we cannot have both $\boldsymbol{\lambda}_r$ and $\boldsymbol{\lambda}_{v,0}$ zero, and thus it must be that $\boldsymbol{\lambda}_r = 0$. Hence $\dot{S}_\Gamma = 0$ only if $\boldsymbol{\lambda}_r = 0$ on the $[t_1, t_2]$. We can show that $S_\Gamma \neq 0$ when $\boldsymbol{\lambda}_r = 0$ to complete the proof, since even if $\dot{S}_\Gamma = 0$ we will have established that a singular arc is not possible.

Assume for a contradiction that $\boldsymbol{\lambda}_r = 0$ and $S_\Gamma = 0$ for $t \in [t_1, t_2]$. Using the transversality condition $\mathcal{H}\big(\pi(t_f)\big) = 0$, it follows that $\mathcal{H}\big(\pi(t_f)\big) = \boldsymbol{\lambda}_v^\top \boldsymbol{g}_\mathcal{L} = 0$. Because $\boldsymbol{\lambda}_v$ and $\boldsymbol{d}$ are either parallel or anti-parallel, this implies that $\boldsymbol{d}^\top \boldsymbol{g}_\mathcal{L} = 0$. However this condition says that the attitude is orthogonal to the gravity vector, which means so is the thrust direction. Hence there is no ability to thrust in the vertical direction and the vehicle is in vertical free fall. Just as in the proof of Theorem 3.2, this contradicts nearly all boundary conditions of interest in powered descent. Hence $S_\Gamma \neq 0$ almost everywhere when $t \in [t_1, t_2]$ if $\boldsymbol{\lambda}_r = 0$. As a result, the thrust must be non-singular.

Assume now that the torque is non-singular, so that $\tau = \pm\tau_{\max}$, for some interval of time $t \in [t_1,\, t_2]$, where $t_0 \leq t_1 < t_2 \leq t_f$. In this case, one may directly solve the attitude state equations on this interval to yield

$$\theta(t) = \theta(t_1) + \omega(t_1)(t - t_1) \pm \frac{1}{2}\frac{\tau_{\max}}{J}(t - t_1)^2, \quad \forall t \in [t_1, t_2]. \tag{3.40}$$

The switching function for the thrust magnitude is still given by S_Γ in (3.37a). If the thrust is singular, then $S_\Gamma = 0$ for all $t \in [t_1, t_2]$. Accordingly, we would have $\dot{S}_\Gamma = 0$ for $t \in [t_1, t_2]$,

and the expression in (3.39a) becomes

$$\dot{S}_\Gamma = -\frac{1}{m}\boldsymbol{\lambda}_r^\top \boldsymbol{d} + \frac{\omega}{m}\boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta \tag{3.41a}$$

$$0 = \frac{1}{m}\frac{d}{dt}(\boldsymbol{\lambda}_v^\top \boldsymbol{d}), \tag{3.41b}$$

which implies that $\boldsymbol{\lambda}_v^\top \boldsymbol{d}$ is constant. However, since we know the form of $\boldsymbol{\lambda}_v$ as an explicit function of time via (3.36), we can combine this with (3.40) to obtain $\boldsymbol{\lambda}_v^\top \boldsymbol{d}$ as an explicit function of time. The resulting function can be neither zero nor constant over an interval of time. Essentially, the vector $\boldsymbol{\lambda}_v$ varies as a linear function of time, while the vector $\boldsymbol{d}$ varies sinusoidally with a frequency that increases quadratically in time. They cannot be perpendicular for an interval of time, but rather only at finitely many instants of time between t_1 and t_2. This implies that $\dot{S}_\Gamma \neq 0$ for almost all $t \in [t_1, t_2]$. As a result, $S_\Gamma = 0$ is not possible except at a finite number of instants over the interval $[t_1, t_2]$. Hence the thrust magnitude is non-singular almost everywhere on this interval. □

Theorem 3.4 shows that the thrust magnitude must always be non-singular for an optimal solution, regardless of the value of the torque input. Since singular torque arcs may occur, we now characterize what these singular torques can look like in terms of the optimal state and costate vectors. The switching function for the torque input is given by (3.37b), and along a torque-singular arc, we have $S_\tau = 0$. Taking Lie derivatives along extremal solutions results in

$$JS_\tau^{(1)} = -\lambda_\theta, \tag{3.42a}$$

$$JS_\tau^{(2)} = \frac{\Gamma}{m}\boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta, \tag{3.42b}$$

$$JS_\tau^{(3)} = \frac{\alpha\Gamma^2}{m^2}\boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta - \frac{\Gamma}{m}\boldsymbol{\lambda}_r^\top \boldsymbol{d}_\theta - \frac{\Gamma\omega}{m}\boldsymbol{\lambda}_v^\top \boldsymbol{d}, \tag{3.42c}$$

$$JS_\tau^{(4)} = \left(\frac{\alpha^2\Gamma^3}{m^3} - \frac{\Gamma\omega^2}{m^2}\right)\boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta - \frac{2\alpha\Gamma^2}{m^2}\boldsymbol{\lambda}_r^\top \boldsymbol{d}_\theta + \frac{2\Gamma\omega}{m}\boldsymbol{\lambda}_r^\top \boldsymbol{d} - \left(\frac{2\alpha\Gamma^2\omega}{m^2} + \frac{\Gamma}{mJ}\tau\right)\boldsymbol{\lambda}_v^\top \boldsymbol{d}, \tag{3.42d}$$

each of which must be zero along torque-singular arcs. The singular arc is therefore *second-*

order. As is already known from the proof of Theorem 3.4 the condition (3.42b) reveals that $\boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta = 0$. Substituting this into (3.42c) yields

$$\boldsymbol{\lambda}_r^\top \boldsymbol{d}_\theta = -\omega \boldsymbol{\lambda}_v^\top \boldsymbol{d}, \tag{3.43}$$

along torque-singular arcs. Plugging (3.42b) and (3.43) into (3.42d) then leads to

$$\tau = 2J\omega \frac{\boldsymbol{\lambda}_r^\top \boldsymbol{d}}{\boldsymbol{\lambda}_v^\top \boldsymbol{d}}. \tag{3.44}$$

Lastly, the generalized (Legendre-Clebsch) convexity condition reveals that

$$\frac{1}{J} \boldsymbol{\lambda}_v^\top \boldsymbol{d} \geq 0, \tag{3.45}$$

which in addition to the fact that $\boldsymbol{\lambda}_v^\top \boldsymbol{d}_\theta = 0$ implies that $\boldsymbol{d} = \boldsymbol{p}^*$ along torque-singular arcs! This confirms the initial intuition that the planar landing solution can resemble the 3-DOF solution – we see here that when (and only when) the torque is singular, the analytic expressions that govern the optimal thrust direction are the same. This also shows that in cases where the torque is *always* singular, the most general thrust magnitude profile is max-min-max, because the equations that govern the optimal planar solutions recover exactly those of the 3-DOF problem. However, it will not always be the case that $\boldsymbol{d} = \boldsymbol{p}^*$ for an entire planar landing trajectory due to boundary conditions on the vehicle attitude, vehicle characteristics (e.g., inertia) and torque limitations. The following result establishes the more general switching structure of the optimal thrust input for planar landing problems.

Lemma 3.5. *The optimal solution of Problem 5 has at most five subarcs in the order of min-max-min-max-min.*

Proof. Note that when $\boldsymbol{d} = \boldsymbol{p}^*$, there can be up to three subarcs in the order of max-min-max, inherited from the 3-DOF solution. To establish the result, we shall look at how minimum thrust arcs can be possible at the beginning and end of a trajectory. For cases where $\boldsymbol{d} \neq \boldsymbol{p}^*$

at the initial time, the torque input will be used to steer the attitude towards $\boldsymbol{d} = \boldsymbol{p}^*$. According to the integral form of the maximum principle [3], this ought to happen as quickly as possible, resulting in a non-singular torque arc. Hence the value of the inner product $\boldsymbol{\lambda}_v^\top \boldsymbol{p}^*$ will be *increasing* during this time, implying that its time derivative is positive. If the torque is non-singular, the sign of (3.41b) implies that a switch from minimum thrust to maximum thrust is possible. When the initial attitude is free, as in Problem 5, we can expect that this case would arise infrequently; the optimal solution is free to be chosen so that its initial attitude satisfies $\boldsymbol{d} = \boldsymbol{p}^*$. A constrained initial attitude could therefore induce an initial minimum thrust arc.

On the other hand, the terminal attitude boundary condition may impose that $\boldsymbol{d} \neq \boldsymbol{p}^*$ at the final time. It is therefore possible that the inner product $\boldsymbol{\lambda}_v^\top \boldsymbol{p}^*$ *decreases* in value at the end of the trajectory. Hence the sign of (3.41b) implies that a switch from maximum to minimum thrust is possible at the end of the trajectory if the torque is again non-singular during this period. □

The presence of an initial or terminal minimum thrust arc is governed by the boundary conditions and control constraints of the problem, as well as the physical vehicle parameters. Of course, we note that simply constraining the attitude at either endpoint does not *guarantee* the presence of a minimum thrust arc, but does represent a necessary condition.

In Chapter 5, we provide an example solution for a planar landing problem that is solved both by a commercial optimal control solver and a (customized) real-time numerical solution algorithm. More details are contained in [69] as well. It's worth pointing out the planar landing problem using a gimbaled thrust vector and no additional torque input remains an open problem – results analogous to those presented in this section have proven to be more challenging to obtain.

3.4 6-DOF Landing Problems

We now present the more general 6-DOF powered descent guidance problem. This class of problems is more complex than both the 3-DOF and planar landing problems that have just been discussed, and as a result has not yet enjoyed the same thorough theoretical understanding. The main challenge lies in the consideration of the complete attitude dynamics that were ignored for the 3-DOF problem and restricted to a single dimension for the planar landing problem, as well as the consideration of a much richer set of state and control constraints. While results analogous to Theorem 3.2 and Theorem 3.4 are challenging, significant progress has been made from a numerical perspective.

Figure 3.3 provides a depiction of a notional 6-DOF landing scenario, where the vehicle is no longer considered a point mass as in Figure 3.1, and the full orientation of the vehicle is to be considered part of the trajectory generation problem. We will use dual quaternions to represent the state of the vehicle, which permits both an elegant formulation of the equations of motion and the ability to express several state constraints as convex functions. We also discuss state-triggered constraints: a new type of constraint that emulates a binary decision to impose-or-not-to-impose a specific constraint depending on the value of a specified trigger function. The goal of this section is to state a general 6-DOF problem formulation, while numerical techniques designed to solve said problem are deferred to Chapter 4.

Assumptions

It is assumed throughout this section that all powered descent maneuvers occur close enough to the landing site and for short enough durations that gravity can be approximated as constant vector. As mentioned previously, the landing accuracy lost by making this assumption has been verified against a central field model to be small for the problems considered, though this must be evaluated on a case-by-case basis. It is further assumed that all maneuvers take place at speeds sufficiently below orbital speeds so that effects of planetary rotation may be neglected. If necessary, both of these assumptions can be removed with relative ease by

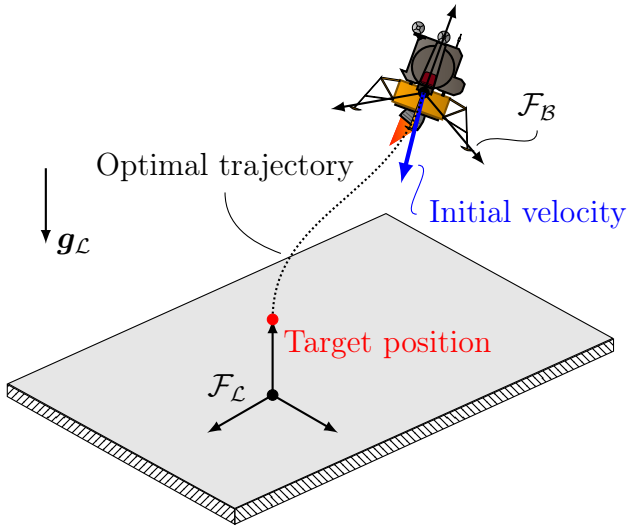


Figure 3.3: A notional 6-DOF landing scenario where the vehicle's attitude is now fully accounted for during optimal trajectory generation.

suitable modification of the dynamics introduced in §3.4.1. For cases where the atmospheric density is non-zero, it is assumed that the ambient pressure is roughly constant.

The inertia matrix, center of mass and center of pressure of the vehicle are all assumed to be constant. In previous work, the effects of variable inertia on the trajectories generated were investigated [72]. In the scenarios studied in [72], trajectories did not deviate significantly from those obtained with a constant inertia matrix, and so these variations are ignored for trajectory design.

The translational control capabilities of landing vehicles are typically dominated by the rocket engine(s), whereas the attitude control capabilities are dictated by both the rocket engine(s) and/or a reaction control system (RCS). The planar landing problem presented in §3.3 assumed that two independent actuation mechanisms existed. For 6-DOF trajectory optimization, it is henceforth assumed that the rocket engine(s) provide *both* the translation and attitude control authority. An RCS system is assumed to be available only for closed-loop attitude control. However, RCS can be easily added to the guidance problem formulation, and this will be demonstrated in Chapter 6.

3.4.1 Baseline Problem Formulation

Consider again the surface-fixed landing frame $\mathcal{F}_{\mathcal{L}}$ with origin at the intended landing site and constructed from the orthonormal vectors $\{\boldsymbol{x}_{\mathcal{L}}, \boldsymbol{y}_{\mathcal{L}}, \boldsymbol{z}_{\mathcal{L}}\}$. These basis vectors are oriented such that $\boldsymbol{x}_{\mathcal{L}}$ represents the downrange direction, $\boldsymbol{y}_{\mathcal{L}}$ represents the crossrange direction, and $\boldsymbol{z}_{\mathcal{L}}$ points locally up. Similarly, a body frame $\mathcal{F}_{\mathcal{B}}$ has its origin at the vehicle's center of mass and is constructed from the orthonormal vectors $\{\boldsymbol{x}_{\mathcal{B}}, \boldsymbol{y}_{\mathcal{B}}, \boldsymbol{z}_{\mathcal{B}}\}$, where $\boldsymbol{z}_{\mathcal{B}}$ is chosen to point along the vehicle's vertical axis.

This section derives a *baseline* problem formulation that consists of the dual quaternion-based equations of motion and a nominal set of state and control constraints and boundary conditions.

Equations of Motion

The state vector depends on the parameterization that is assumed. Because the reader may be more familiar with the use of Cartesian variables, we present the development of the dual quaternion-based equations of motion in parallel with the more standard Cartesian variables and unit quaternions. The following state vectors are defined

$$\text{Dual quaternions:} \quad \boldsymbol{x}(t) = \big(m(t),\, \tilde{\boldsymbol{q}}(t),\, \tilde{\boldsymbol{\omega}}(t)\big) \in \mathbb{R}^{17}, \tag{3.46a}$$

$$\text{Cartesian:} \quad \boldsymbol{x}(t) = \big(m(t),\, \boldsymbol{r}_{\mathcal{L}}(t),\, \boldsymbol{v}_{\mathcal{L}}(t),\, \boldsymbol{q}(t),\, \boldsymbol{\omega}_{\mathcal{B}}(t)\big) \in \mathbb{R}^{14}. \tag{3.46b}$$

Here, $m(t) \in \mathbb{R}_{++}$ denotes the vehicle's mass, $\boldsymbol{r}_{\mathcal{L}}(t) \in \mathbb{R}^3$ the position vector in the surface-fixed landing frame, $\boldsymbol{v}_{\mathcal{L}}(t) \in \mathbb{R}^3$ the velocity in the surface-fixed landing frame, $\boldsymbol{q}(t) \in \mathbb{R}^4_u$ the attitude quaternion, $\boldsymbol{\omega}_{\mathcal{B}}(t) \in \mathbb{R}^3$ the angular velocity, $\tilde{\boldsymbol{q}}(t) \in \mathbb{R}^8_u$ the vehicle's dual quaternion and $\tilde{\boldsymbol{\omega}}(t) \in \mathbb{R}^8$ the vehicle's dual velocity. The quaternion and dual quaternion definitions are given in §2.3.2.

In both cases, the mass is now assumed to vary as an affine function of the thrust magnitude because we may account for atmospheric effects. We therefore augment the mass

dynamics from (3.2) to be

$$\dot{m}(t) = -\alpha \|\boldsymbol{u}_{\mathcal{B}}(t)\|_2 - \beta, \qquad \alpha \coloneqq \frac{1}{I_{sp} g_e}, \qquad \beta \coloneqq \alpha P_{amb} A_{noz} \tag{3.47}$$

where $\boldsymbol{u}_{\mathcal{B}}(t) \in \mathbb{R}^3$ is the thrust vector *in the body frame*, I_{sp} is the vacuum specific impulse, g_e is the acceleration due to gravity at sea level on Earth, P_{amb} is the ambient atmospheric pressure, and A_{noz} is the exit area of the engine's nozzle [82].

The dual quaternion kinematics relate the "velocity" states, $\tilde{\boldsymbol{\omega}}$, to the time rate of change of "position" states, $\tilde{\boldsymbol{q}}$. These are given for the Cartesian variable and dual quaternion state variables by

$$\text{Dual quaternions:} \quad \dot{\tilde{\boldsymbol{q}}}(t) = \frac{1}{2}\tilde{\boldsymbol{q}}(t) \otimes \tilde{\boldsymbol{\omega}}(t), \tag{3.48a}$$

$$\text{Cartesian:} \quad \dot{\boldsymbol{r}}_{\mathcal{L}}(t) = \boldsymbol{v}_{\mathcal{L}}(t) \quad \text{and} \quad \dot{\boldsymbol{q}}(t) = \frac{1}{2}\boldsymbol{q}(t) \otimes \boldsymbol{\omega}_{\mathcal{B}}(t). \tag{3.48b}$$

The dual quaternion equations can be derived by directly computing the time derivative of (2.39) using (3.48b):

$$\frac{d\tilde{\boldsymbol{q}}}{dt} = \frac{d}{dt}\begin{bmatrix} \boldsymbol{q} \\ \frac{1}{2}\boldsymbol{r}_{\mathcal{L}} \otimes \boldsymbol{q} \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{q}} \\ \frac{1}{2}(\dot{\boldsymbol{r}}_{\mathcal{L}} \otimes \boldsymbol{q} + \boldsymbol{r}_{\mathcal{L}} \otimes \dot{\boldsymbol{q}}) \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\boldsymbol{q} \otimes \boldsymbol{\omega}_{\mathcal{B}} \\ \frac{1}{2}(\boldsymbol{q} \otimes \boldsymbol{v}_{\mathcal{B}} + \frac{1}{2}\boldsymbol{r}_{\mathcal{L}} \otimes \boldsymbol{q} \otimes \boldsymbol{\omega}_{\mathcal{B}}) \end{bmatrix} = \frac{1}{2}\tilde{\boldsymbol{q}} \otimes \tilde{\boldsymbol{\omega}}.$$

The dynamics can be derived using the Newton-Euler equations. The Cartesian and dual quaternion approaches differ in their treatment of the translational dynamics. Here, the Cartesian approach views the equations in the (inertial) surface-fixed frame, whereas the dual quaternion approach views the equations in the rotating body frame. In each case, the dynamics are obtained by taking a derivative of the linear momentum:

$$\frac{d}{dt}\big(m(t)\boldsymbol{v}_{\mathcal{L}}(t)\big) = m(t)\dot{\boldsymbol{v}}_{\mathcal{L}}(t) = m(t)\boldsymbol{g}_{\mathcal{L}} + \boldsymbol{a}_{\mathcal{L}}(t) + \boldsymbol{u}_{\mathcal{L}}(t), \tag{3.49a}$$

$$\frac{d}{dt}\big(m(t)\boldsymbol{v}_{\mathcal{B}}(t)\big) = m(t)\dot{\boldsymbol{v}}_{\mathcal{B}}(t) + \boldsymbol{\omega}_{\mathcal{B}}(t)^{\times}\big(m(t)\boldsymbol{v}_{\mathcal{B}}(t)\big) = m(t)\boldsymbol{g}_{\mathcal{B}}(t) + \boldsymbol{a}_{\mathcal{B}}(t) + \boldsymbol{u}_{\mathcal{B}}(t), \tag{3.49b}$$

where $\boldsymbol{g}_{\mathcal{L}}$ and $\boldsymbol{g}_{\mathcal{B}}(t)$ represent the acceleration due to gravity and $\boldsymbol{a}_{\mathcal{L}}(t)$ and $\boldsymbol{a}_{\mathcal{B}}(t)$ represent the aerodynamic forces in the surface-fixed and body frames respectively. The vector $\boldsymbol{g}_{\mathcal{B}}(t)$ is time-varying due to its dependence on the attitude of the vehicle (whereas $\boldsymbol{g}_{\mathcal{L}}$ was assumed to be constant). Note that momentum changes due to mass variability are accounted for in (3.49) by the definition of the mass depletion dynamics in (3.47) and $\boldsymbol{u}_{\mathcal{B}}(t)$; see [83] for details.

The rotational dynamics are treated equivalently between the two approaches. A temporal derivative of the angular momentum in the rotating body frame leads to Euler's equations

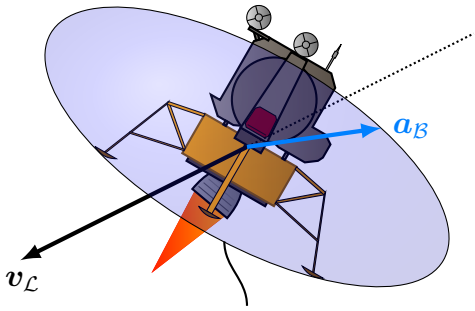
$$\frac{d}{dt}\big(J\boldsymbol{\omega}_{\mathcal{B}}(t)\big) = J\dot{\boldsymbol{\omega}}_{\mathcal{B}}(t) + \boldsymbol{\omega}_{\mathcal{B}}(t)^{\times} J\boldsymbol{\omega}_{\mathcal{B}}(t) = \boldsymbol{r}_{u,\mathcal{B}}^{\times}\boldsymbol{u}_{\mathcal{B}}(t) + \boldsymbol{r}_{cp,\mathcal{B}}^{\times}\boldsymbol{a}_{\mathcal{B}}(t)\,, \tag{3.50}$$

where $\boldsymbol{r}_{u,\mathcal{B}}$, $\boldsymbol{r}_{cp,\mathcal{B}} \in \mathbb{R}^3$ denote the constant body-frame vectors from the vehicle's center of mass to the point where the thrust is applied and to the center of pressure.

Atmospheric forces and torques are typically ignored for the purposes of trajectory generation, and treated as a disturbance in subsequent closed-loop control design. The predominant effect of atmosphere on a descent trajectory is to produce a drag force that acts to slow the vehicle. This can actually be desirable for landing scenarios where linear momentum needs to be reduced anyway – drag forces can serve as "free thrust". For 6-DOF problems, the attitude of the vehicle may be further used to create a lift force that is orthogonal to the velocity vector of the vehicle. This complex coupling of attitude and translation states has not been fully explored in the literature to-date, and so an aerodynamic model tractable for trajectory optimization was introduced in [78]. The model expresses the aerodynamic force in $\mathcal{F}_{\mathcal{B}}$ coordinates as follows

$$\begin{aligned}\boldsymbol{a}_{\mathcal{B}}(t) &= -\frac{1}{2}\rho A C_a \boldsymbol{v}_{\mathcal{B}}(t)\|\boldsymbol{v}_{\mathcal{B}}(t)\|_2\,, \quad C_a \in \mathbb{S}^3_{++}\,, \\ &= -\frac{1}{2}\rho A C_a C_{\mathcal{LB}}(t)\boldsymbol{v}_{\mathcal{L}}(t)\|\boldsymbol{v}_{\mathcal{L}}(t)\|_2\end{aligned} \tag{3.51}$$

where ρ is the ambient atmospheric density, and $A \in \mathbb{R}_{++}$ is a (constant) reference area. The



$$\mathcal{E}_{aero} \coloneqq \left\{ \boldsymbol{a}_{\mathcal{B}} : \boldsymbol{a}_{\mathcal{B}} = -\tfrac{1}{2}\rho A C_a V^2 \hat{\boldsymbol{v}}_{\mathcal{B}} \,,\ \|\hat{\boldsymbol{v}}_{\mathcal{B}}\|_2 = 1 \right\}$$

Figure 3.4: Illustration of the ellipsoidal aerodynamic model designed to capture the effects of both lift and drag forces on a vehicle maneuvering in atmosphere.

parameter $C_a \in \mathbb{S}^3_{++}$ is defined as the aerodynamic coefficient matrix, a symmetric positive definite matrix that generalizes the standard scalar definition. The definition of C_a ensures that for any $V \in \mathbb{R}_+$ the set

$$\mathcal{E}_{aero} \coloneqq \left\{ \boldsymbol{a}_{\mathcal{B}} \mid \boldsymbol{a}_{\mathcal{B}} = -\tfrac{1}{2}\rho A C_a V^2 \hat{\boldsymbol{v}}_{\mathcal{B}} \,,\ \|\hat{\boldsymbol{v}}_{\mathcal{B}}\|_2 = 1 \right\} \tag{3.52}$$

defines an ellipsoid in $\mathcal{F}_{\mathcal{B}}$ coordinates. Because most rocket-powered vehicles are approximately axisymmetric, it can be assumed that $C_a = \mathbf{diag}\left\{c_{a,xy},\, c_{a,xy},\, c_{a,z}\right\}$, where $c_{a,xy}$ and $c_{a,z}$ are positive scalars. This assumption aligns the principal axes of $\mathcal{E}_{aero}$ with the axes of $\mathcal{F}_{\mathcal{B}}$. If $c_{a,xy} \neq c_{a,z}$, then $\boldsymbol{a}_{\mathcal{B}}(t)$ can have components orthogonal to $\boldsymbol{v}_{\mathcal{B}}(t)$. In this case, $\boldsymbol{a}_{\mathcal{B}}(t)$ may be interpreted as the vector sum of the aerodynamic drag and lift forces. Furthermore, if $c_{a,z} < c_{a,xy}$, the vehicle experiences minimum drag when $\boldsymbol{v}_{\mathcal{B}}$ is aligned with $\boldsymbol{z}_{\mathcal{B}}$, and the lift component of $\boldsymbol{a}_{\mathcal{B}}$ points in the correct direction. Since the set $\mathcal{E}_{aero}$ corresponding to this choice of C_a defines an ellipsoid, the corresponding model is referred to as the *ellipsoidal aerodynamic model*, and is illustrated in Figure 3.4.

Note that if $c_{a,xy} = c_{a,z}$, then $\boldsymbol{a}_{\mathcal{B}}$ is always anti-parallel to $\boldsymbol{v}_{\mathcal{B}}$. In this case, $\boldsymbol{a}_{\mathcal{B}}$ may be interpreted as a pure drag force, and the model recovers the traditional definition of an aerodynamic drag force. Since the set $\mathcal{E}_{aero}$ corresponding to this choice of C_a defines a

sphere, the corresponding model is referred to as the *spherical aerodynamic model*. Under the assumptions of the spherical model, the product $C_{\mathcal{BL}}C_aC_{\mathcal{LB}}$ simplifies to $c_{a,z}I_3$, rendering $\boldsymbol{a}_{\mathcal{L}} = C_{\mathcal{BL}}\boldsymbol{a}_{\mathcal{B}}$ independent of the vehicle's attitude.

Remark 3.6. *A notable advantage of the dual quaternion model in regards to aerodynamic effects is the inclusion of $\boldsymbol{v}_{\mathcal{B}}$ as a state. The Cartesian model requires the rotation of $\boldsymbol{v}_{\mathcal{L}}$ into the body frame to properly account for the aerodynamic torque, an additional nonlinearity that is not required by the dual quaternion model. This feature will be exploited later to convexify an angle of attack constraint using dual quaternions.*

The Cartesian dynamics are formed by combining each of (3.47), (3.48b), (3.49a) and (3.50) along with the aerodynamic model to write

$$\dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = \begin{bmatrix} -\alpha\|\boldsymbol{u}_{\mathcal{B}}(t)\|_2 - \beta \\ \boldsymbol{v}_{\mathcal{L}}(t) \\ \frac{1}{m(t)}\left[C_{\mathcal{LB}}(t)\boldsymbol{u}_{\mathcal{B}}(t) + \boldsymbol{a}_{\mathcal{L}}(t)\right] + \boldsymbol{g}_{\mathcal{L}} \\ \frac{1}{2}\boldsymbol{q}(t) \otimes \boldsymbol{\omega}_{\mathcal{B}}(t) \\ J^{-1}\left[\boldsymbol{r}^{\times}_{u,\mathcal{B}}\boldsymbol{u}_{\mathcal{B}}(t) + \boldsymbol{r}^{\times}_{cp,\mathcal{B}}\boldsymbol{a}_{\mathcal{B}}(t) - \boldsymbol{\omega}_{\mathcal{B}}(t)^{\times}J\boldsymbol{\omega}_{\mathcal{B}}(t)\right] \end{bmatrix}, \tag{3.53}$$

where $\boldsymbol{x}$ is the appropriate state vector from (3.46) and $\boldsymbol{u} \equiv \boldsymbol{u}_{\mathcal{B}}$.

Similarly, the dual quaternion dynamics are formed by combining (3.47), (3.48a), (3.49b) and (3.50), (2.40) and the aerodynamic model to write

$$\dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = \begin{bmatrix} -\alpha\|\boldsymbol{u}_{\mathcal{B}}(t)\|_2 - \beta \\ \frac{1}{2}\tilde{\boldsymbol{q}}(t) \otimes \tilde{\boldsymbol{\omega}}(t) \\ \boldsymbol{J}^{-1}(t)\left[\Phi\boldsymbol{u}_{\mathcal{B}}(t) + \Phi_a\boldsymbol{a}_{\mathcal{B}}(t) - \tilde{\boldsymbol{\omega}}(t) \oslash \boldsymbol{J}(t)\tilde{\boldsymbol{\omega}}(t)\right] + \tilde{\boldsymbol{g}}_{\mathcal{B}}(t) \end{bmatrix}, \tag{3.54}$$

where, $\boldsymbol{x}$ is the appropriate state vector from (3.46), $\boldsymbol{u} \equiv \boldsymbol{u}_{\mathcal{B}}$, and

$$\boldsymbol{J}(t) \coloneqq \left[\begin{array}{cc|cc} & & m(t)I_3 & 0_{3\times 1} \\ \multicolumn{2}{c|}{0_{4\times 4}} & 0_{1\times 3} & 1 \\ \hline J & 0_{3\times 1} & \multicolumn{2}{c}{\multirow{2}{*}{$0_{4\times 4}$}} \\ 0_{1\times 3} & 1 & & \end{array}\right]_{8\times 8} \qquad \tilde{\boldsymbol{g}}_{\mathcal{B}}(t) \coloneqq \left[\begin{array}{c} \boldsymbol{g}_{\mathcal{B}}(t) \\ \hline 0_{4\times 1} \end{array}\right]_{8\times 1}$$

$$\Phi \coloneqq \left[\begin{array}{c} I_3 \\ 0_{1\times 3} \\ \hline \boldsymbol{r}^{\times}_{u,\mathcal{B}} \\ 0_{1\times 3} \end{array}\right]_{8\times 3} \qquad \Phi_a \coloneqq \left[\begin{array}{c} I_3 \\ 0_{1\times 3} \\ \hline \boldsymbol{r}^{\times}_{cp,\mathcal{B}} \\ 0_{1\times 3} \end{array}\right]_{8\times 3}$$

Note that the *dual inertia* matrix $\boldsymbol{J}(t)$ is always invertible. More details on rigid body kinematics and dynamics using dual quaternions may be found in [31, 32, 71, 73, 84, 85, 86, 87]. Note that the dynamics that correspond to the dual scalar part of the dual quaternion $\tilde{\boldsymbol{\omega}}$ – the fourth and eighth entries – will always be zero in (3.54). Therefore, it is possible to omit these entries and treat $\tilde{\boldsymbol{\omega}}$ as a six-dimensional vector in implementation. This is the standard practice, and it reduces the state dimension from $n_x = 17$ to $n_x = 15$.

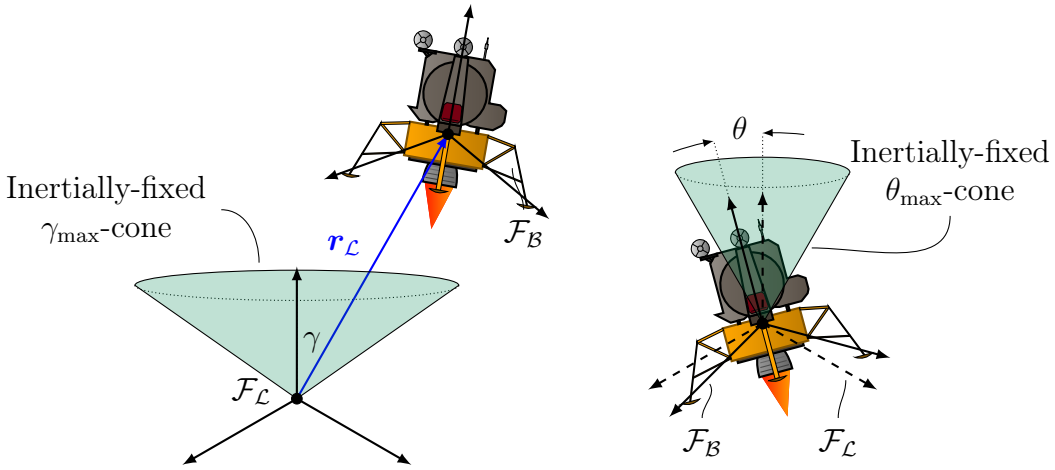
State and Control Constraints

This section details the state and control constraints considered part of the baseline problem formulation. The argument of time is henceforth omitted for the sake of clarity, except when discussing boundary conditions. To ensure that trajectories do not use more fuel than is stored onboard, a constraint is enforced on the mass of the vehicle according to

$$m \geq m_{dry}, \tag{3.55}$$

where $m_{dry} \in \mathbb{R}_{++}$ is the dry mass of the vehicle.

The *approach angle* is defined to be the angle formed between $\boldsymbol{r}_{\mathcal{L}}$ and $\boldsymbol{z}_{\mathcal{L}}$. An approach



(a) Approach angle. (b) Tilt angle.

Figure 3.5: A depiction of the approach cone and tilt angle constraints that form part of the baseline powered descent guidance problem.

cone constraint is used to ensure the vehicle's position lies above the surface of the planet, while also ensuring sufficient elevation at large distances from the landing site. The approach cone constraint is depicted in Figure 3.5a, and is expressed in Cartesian variables as

$$-\boldsymbol{r}_{\mathcal{L}}^{\top}\boldsymbol{z}_{\mathcal{L}} + \|\boldsymbol{r}_{\mathcal{L}}\|_2 \cos\gamma_{\max} \leq 0, \tag{3.56}$$

where it is assumed that $\gamma_{\max} \in [0,\, 90]$ deg. The approach cone constraint (3.56) can be equivalently expressed in terms of the dual quaternion by using (2.41a) and (2.41d) to write

$$c_\gamma(\tilde{\boldsymbol{q}}) \coloneqq -\tilde{\boldsymbol{q}}^{\top} M_\gamma \tilde{\boldsymbol{q}} + \|2E_d\tilde{\boldsymbol{q}}\|_2 \cos\gamma_{\max} \leq 0, \quad M_\gamma \coloneqq \begin{bmatrix} 0_{4\times4} & [\boldsymbol{z}_{\mathcal{L}}]_{\otimes}^{\top} \\ [\boldsymbol{z}_{\mathcal{L}}]_{\otimes} & 0_{4\times4} \end{bmatrix}. \tag{3.57}$$

It was shown in [32, 71] that $c_\gamma : \mathbb{R}^8_u \to \mathbb{R}$ is convex over the bounded domain $\mathbf{dom}\, c_\gamma = \left\{\tilde{\boldsymbol{q}} \in \mathbb{R}^8_u \,|\, \tilde{\boldsymbol{q}}^{\top}\tilde{\boldsymbol{q}} \leq 1 + \frac{1}{4}\Delta^2\right\}$, where $\|\boldsymbol{r}_{\mathcal{L}}\|_2 \leq \Delta$ is an upper bound on the distance from the landing site.

The vehicle is also subject to a tilt angle constraint that limits the angle formed between

the vehicle's vertical axis $\boldsymbol{z}_{\mathcal{B}}$ and the inertial direction, $\boldsymbol{z}_{\mathcal{L}}$. This constraint is depicted in Figure 3.5b. When both vectors are viewed inertially, the tilt angle constraint is expressed as

$$-\boldsymbol{z}_{\mathcal{L}}^{\top}\left(\boldsymbol{q}\otimes\boldsymbol{z}_{\mathcal{B}}\otimes\boldsymbol{q}^{*}\right)+\cos\theta_{\max}\leq 0, \tag{3.58}$$

where $\theta_{\max}\in[0,90]$ deg is the maximum allowable tilt angle. The constraint (3.58) is equivalently formulated in terms of the dual quaternion by using (2.41c) to write

$$c_{\theta}(\tilde{\boldsymbol{q}})\coloneqq\tilde{\boldsymbol{q}}^{\top}M_{\theta}\tilde{\boldsymbol{q}}+\cos\theta_{\max}\leq 0,\quad M_{\theta}=\begin{bmatrix}[\boldsymbol{z}_{\mathcal{L}}]_{\otimes}[\boldsymbol{z}_{\mathcal{B}}]_{\otimes}^{*} & 0_{4\times4}\\ 0_{4\times4} & 0_{4\times4}\end{bmatrix}. \tag{3.59}$$

It was shown in [71, 72] that for any $\theta_{\max}\in[0,90]$ deg, the function c_{θ} is convex for all $\tilde{\boldsymbol{q}}\in\mathbb{R}_{u}^{8}$.

The last state constraints that are considered part of the baseline problem bound the allowable angular rates and linear velocities by enforcing

$$\|\boldsymbol{\omega}_{\mathcal{B}}\|_{\infty}\leq\omega_{\max}\quad\text{and}\quad\|\boldsymbol{v}_{\mathcal{B}}\|_{2}\leq v_{\max}, \tag{3.60}$$

where $\omega_{\max},v_{\max}\in\mathbb{R}_{++}$ are the maximum allowable angular velocity about any axis and maximum vehicle speed. By defining the matrices $E_{w}\coloneqq\begin{bmatrix}0_{4\times4} & I_{4}\end{bmatrix}\in\mathbb{R}^{4\times8}$ and $E_{v}\coloneqq\begin{bmatrix}I_{4} & 0_{4\times4}\end{bmatrix}\in\mathbb{R}^{4\times8}$, the constraints (3.60) can be written in terms of the dual velocity simply as

$$\|E_{w}\tilde{\boldsymbol{\omega}}\|_{\infty}\leq\omega_{\max}\quad\text{and}\quad\|E_{v}\tilde{\boldsymbol{\omega}}\|_{2}\leq v_{\max}. \tag{3.61}$$

It was assumed that a single rocket engine provides both the translation and attitude control authority. At times, rocket engines necessitate operation in restricted thrust and gimbal angle regimes. For example, the main engines of Apollo-era landers could either operate at 93% thrust or in the permitted interval of 11% to 65% of the rated thrust value [35]. The forbidden thrust regions were avoided to prevent cavitation in the propulsion system and limit wear on engine components. To model permitted thrust regions, restrictions are

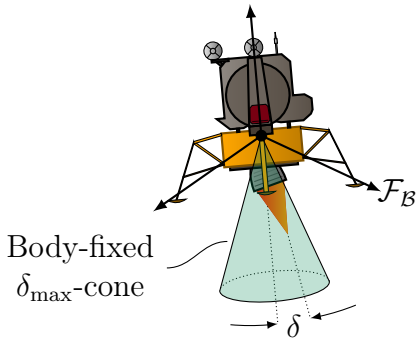


Figure 3.6: A depiction of the gimbal angle constraint.

placed on the norm of the thrust vector according to

$$u_{\min} \leq \|\boldsymbol{u}_\mathcal{B}\|_2 \leq u_{\max}, \tag{3.62}$$

where $[u_{\min}, u_{\max}] \subset \mathbb{R}_{++}$ denotes the permitted thrust interval.

The engine is assumed to be capable of being gimbaled symmetrically about two axes and the gimbal angle is defined as the total angular deflection of the thrust vector from its nominal position. This constraint is depicted in Figure 3.6 and the mechanical limitations of the engine are modeled using the gimbal angle constraint

$$\|\boldsymbol{u}_\mathcal{B}\|_2 \leq \sec \delta_{\max} \boldsymbol{z}_\mathcal{B}^\top \boldsymbol{u}_\mathcal{B}. \tag{3.63}$$

In practice, the maximum gimbal angle typically satisfies $\delta_{\max} \ll 90$ deg, and hence (3.63) represents a second-order cone in $\boldsymbol{u}_\mathcal{B}$.

The last constraints imposed on the control are rate constraints that ensure commanded thrust vectors do not change too rapidly for the engine to follow. There are two important rate constraints; throttle rate and gimbal rate. The precise expressions for each can be computed by taking a temporal derivative of the appropriate constraint expressions (using

the fact that $u_{\min} > 0$)

$$\frac{d}{dt}\|\boldsymbol{u}_{\mathcal{B}}\|_2 = \frac{\boldsymbol{u}_{\mathcal{B}}^\top \dot{\boldsymbol{u}}_{\mathcal{B}}}{\|\boldsymbol{u}_{\mathcal{B}}\|_2}$$
$$\frac{d}{dt}\left(\cos\delta = \frac{\boldsymbol{u}_{\mathcal{B}}^\top \boldsymbol{z}_{\mathcal{B}}}{\|\boldsymbol{u}_{\mathcal{B}}\|_2}\right) \quad \Rightarrow \quad \dot{\delta}\sin\delta = \left(\cos\delta\frac{\boldsymbol{u}_{\mathcal{B}}}{\|\boldsymbol{u}_{\mathcal{B}}\|_2} - \boldsymbol{z}_{\mathcal{B}}\right)^\top \frac{\dot{\boldsymbol{u}}_{\mathcal{B}}}{\|\boldsymbol{u}_{\mathcal{B}}\|_2}$$

and upper bounding the expressions using constants for the throttle rate limit and gimbal rate limit. These resulting expressions, however, are quite nonconvex. Instead of using the precise equations, we approximate the throttle rate and gimbal rate limits using the convex constraints

$$-\dot{u}_{z,\max} \leq \boldsymbol{z}_{\mathcal{B}}^\top \dot{\boldsymbol{u}}_{\mathcal{B}} \leq \dot{u}_{z,\max}, \tag{3.64a}$$
$$\|E_{xy}\dot{\boldsymbol{u}}_{\mathcal{B}}\|_2 \leq \dot{\delta}_{\max}\boldsymbol{z}_{\mathcal{B}}^\top \boldsymbol{u}_{\mathcal{B}}, \tag{3.64b}$$

where $E_{xy} \in \mathbb{R}^{2\times 3}$ selects the thrust vector components in the $\boldsymbol{x}_{\mathcal{B}}$-$\boldsymbol{y}_{\mathcal{B}}$ plane, $\dot{\delta}_{\max}$ is the maximum gimbal rate, and $\dot{u}_{z,\max} \in \mathbb{R}_{++}$ approximates the throttle rate limit.

Boundary Conditions

To complete the baseline problem formulation, a notional set of boundary conditions are introduced. We allow the initial maneuver time to be selected as part of the optimization, a feature that was introduced in [78]. We assume that the vehicle follows some known trajectory prior to the initiation of the powered descent maneuver, and define the *coast time*, $t_c \in \mathbb{R}_+$, to be the time during which the vehicle follows this known trajectory before the beginning of the descent maneuver.

The initial maneuver time remains denoted by t_0 for consistency, and Figure 3.7 provides an illustration of the resulting descent timeline. The variables t_c, $\boldsymbol{r}_{\mathcal{L}}(t_0)$ and $\boldsymbol{v}_{\mathcal{L}}(t_0)$ are selected as part of the optimization process.

The initial angular velocity is assumed to be fixed. The optimization process selects the initial attitude for two reasons. First, typical lunar descent sequences have a short pitch-up

attitude maneuver that occurs immediately prior to the final approach phase [35, 38]. A guidance scheme like the one described here would select the attitude to be achieved at the end of the pitch-up maneuver, thereby ensuring the initial attitude satisfies any constraints that are enforced on the subsequent powered descent maneuver. Second, extensive numerical testing has shown that leaving this variable free offers better convergence behavior over a wide range of initial conditions.

The boundary conditions are enforced as equality constraints according to

$$\begin{aligned}
\text{Dual Quaternions:} \quad m(t_0) &= m_{ic} - \alpha \int_0^{t_c} \Gamma_{\mathrm{c}} \, \mathrm{d}t - \beta t_c, \\
\tilde{\boldsymbol{q}}(t_0) &= \boldsymbol{b}_{\tilde{\boldsymbol{q}}}(\boldsymbol{q}(t_0), t_c) \coloneqq \begin{bmatrix} \boldsymbol{q}(t_0) \\ \frac{1}{2}\boldsymbol{b}_{\boldsymbol{r},0}(t_c) \otimes \boldsymbol{q}(t_0) \end{bmatrix}, \\
\tilde{\boldsymbol{\omega}}(t_0) &= \boldsymbol{b}_{\tilde{\boldsymbol{\omega}}}(\boldsymbol{q}(t_0), t_c) \coloneqq \begin{bmatrix} \boldsymbol{\omega}_{\mathcal{B},ic} \\ \boldsymbol{q}^*(t_0) \otimes \boldsymbol{b}_{\boldsymbol{v},0}(t_c) \otimes \boldsymbol{q}(t_0) \end{bmatrix}, \qquad (3.65a) \\
\text{Cartesian:} \quad m(t_0) &= m_{ic} - \alpha \int_0^{t_c} \Gamma_{\mathrm{c}} \, \mathrm{d}t - \beta t_c \,, \\
\boldsymbol{r}_{\mathcal{L}}(t_0) &= \boldsymbol{b}_{\boldsymbol{r},0}(t_c) \,, \quad \boldsymbol{v}_{\mathcal{L}}(t_0) = \boldsymbol{b}_{\boldsymbol{v},0}(t_c), \\
\boldsymbol{\omega}_{\mathcal{B}}(t_0) &= \boldsymbol{\omega}_{\mathcal{B},ic} \,, \qquad (3.65b)
\end{aligned}$$

where m_{ic} and $\boldsymbol{\omega}_{\mathcal{B},ic}$ represent the specified initial mass and angular velocity. The function $\Gamma_{\mathrm{c}} : \mathbb{R} \to \mathbb{R}$ denotes the known profile of the engine's throttle from time $t_0 - t_c$ to time t_0 when the maneuver starts. The functions $\boldsymbol{b}_{\tilde{\boldsymbol{q}}} : \mathbb{R}^4_u \times \mathbb{R} \to \mathbb{R}^8_u$ and $\boldsymbol{b}_{\tilde{\boldsymbol{\omega}}} : \mathbb{R}^4_u \times \mathbb{R} \to \mathbb{R}^8_u$ map the (free) initial attitude quaternion and the coast time to the corresponding initial dual quaternions. Lastly, $\boldsymbol{b}_{\boldsymbol{r},0} : \mathbb{R} \to \mathbb{R}^3$ and $\boldsymbol{b}_{\boldsymbol{v},0} : \mathbb{R} \to \mathbb{R}^3$ are vector-valued polynomial functions of t_c that describe the known trajectory prior to the initiation of the powered descent maneuver. These polynomials can be chosen, for example, to represent an aerodynamics-free engine-off (ballistic) trajectory by using

$$\boldsymbol{b}_{\boldsymbol{r},0}(t_c) \coloneqq \boldsymbol{r}_{\mathcal{L},ic} + \boldsymbol{v}_{\mathcal{L},ic}\, t_c + \frac{1}{2}\boldsymbol{g}_{\mathcal{L}} t_c^2, \quad \boldsymbol{b}_{\boldsymbol{v},0}(t_c) \coloneqq \boldsymbol{v}_{\mathcal{L},ic} + \boldsymbol{g}_{\mathcal{L}} t_c, \qquad (3.66)$$

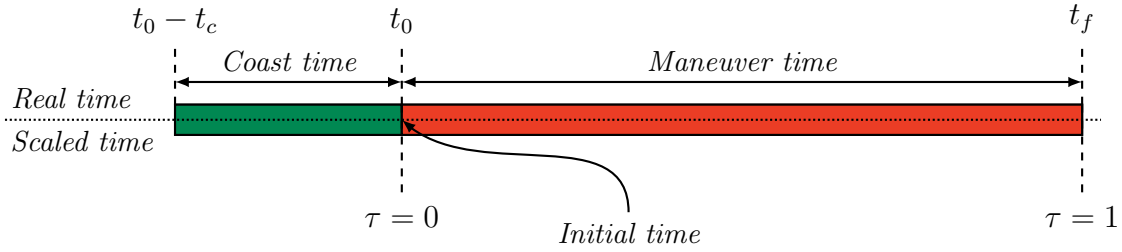


Figure 3.7: An illustration of the powered descent timeline using a free initial time. The scaled time $\tau \in [0, 1]$ is introduced in Chapter 4 and is included for later reference.

where $\boldsymbol{r}_{\mathcal{L},ic}$ and $\boldsymbol{v}_{\mathcal{L},ic}$ are prescribed initial position and velocity vectors.

At the final time, both the dual quaternion and dual velocity are fixed, whereas the final mass is free (but still constrained by (3.55)). These are enforced as equality constraints according to

$$\text{Dual Quaternions:} \quad \tilde{\boldsymbol{q}}(t_f) = \tilde{\boldsymbol{q}}_f, \quad \tilde{\boldsymbol{\omega}}(t_f) = \tilde{\boldsymbol{\omega}}_f, \tag{3.67a}$$

$$\text{Cartesian:} \quad \boldsymbol{r}_{\mathcal{L}}(t_f) = \boldsymbol{r}_{\mathcal{L},f}, \quad \boldsymbol{v}_{\mathcal{L}}(t_f) = \boldsymbol{v}_{\mathcal{L},f}, \quad \boldsymbol{q}(t_f) = \boldsymbol{q}_f, \quad \boldsymbol{\omega}_{\mathcal{B}}(t_f) = \boldsymbol{\omega}_{\mathcal{B},f}, \tag{3.67b}$$

where $\tilde{\boldsymbol{q}}_f$ and $\tilde{\boldsymbol{\omega}}_f$ are constructed from the target inertial position, $\boldsymbol{r}_{\mathcal{L},f}$, attitude $\boldsymbol{q}_f$, inertial velocity, $\boldsymbol{v}_{\mathcal{L},f}$, and angular velocity, $\boldsymbol{\omega}_{\mathcal{B},f}$ using the definitions in §2.3.2.

Baseline Problem Statement

The baseline problem, stated using the dual quaternion parameterization, is summarized as Problem 6. A minimum-fuel objective function is used in Mayer form, as this is more straightforward for numerical solutions. Problem 6 is a nonconvex, free-final-time optimal control problem. The convexity of each problem constraint is denoted to the right of its expression. The nonlinear dynamics, lower bound on thrust magnitude, and boundary conditions are the sources of nonconvexity. The lossless convexification technique for the 3-DOF problem used a relaxation and a change of variables to replace the nonconvex thrust constraint, and subsequently remove the nonlinearities present in the dynamics of that problem with an ap-

proximation (see (3.20)). A similar technique is not possible for the 6-DOF problem, because (i) the same relaxation of the control constraint set is not guaranteed to produce optimal solutions identical to those of the original problem, and (ii) there is no known change of variables that renders the problem affine in both state and control. Moreover, it is quite challenging to apply Theorem 2.2 to Problem 6 – indeed an altered version of the theorem is even required to deal with the additional state constraints that may transition from active to inactive (and vise-versa) along an optimal trajectory [88]. As such, solutions to Problem 6 shall be obtained numerically in Chapter 4.

Problem 6: Dual Quaternion Baseline Problem

Cost Function:

$\underset{\boldsymbol{u}_{\mathcal{B}}(\cdot),\, t_c,\, t_f}{\text{minimize}} \quad -m(t_f)$	(3.6)	*convex*

Dynamics:

$\dot{m} = -\alpha \lVert \boldsymbol{u}_{\mathcal{B}} \rVert_2 - \beta$	(3.47)	*nonconvex*
$\dot{\tilde{\boldsymbol{q}}} = \frac{1}{2} \tilde{\boldsymbol{q}} \otimes \tilde{\boldsymbol{\omega}}$	(3.48a)	*nonconvex*
$\dot{\tilde{\boldsymbol{\omega}}} = \boldsymbol{J}^{-1} \left(\Phi \boldsymbol{u}_{\mathcal{B}} + m \tilde{\boldsymbol{g}}_{\mathcal{B}} + \Phi_a \boldsymbol{a}_{\mathcal{B}} - \tilde{\boldsymbol{\omega}} \oslash \boldsymbol{J} \tilde{\boldsymbol{\omega}} \right)$	(3.54)	*nonconvex*

State Constraints:

$m_{dry} \leq m$	(3.55)	*convex*
$0 \geq -\tilde{\boldsymbol{q}}^\top M_\gamma \tilde{\boldsymbol{q}} + \lVert 2 E_d \tilde{\boldsymbol{q}} \rVert_2 \cos \gamma_{\max}$	(3.57)	*convex*
$0 \geq \tilde{\boldsymbol{q}}^\top M_\theta \tilde{\boldsymbol{q}} + \cos \theta_{\max}$	(3.59)	*convex*
$\omega_{\max} \geq \lVert E_w \tilde{\boldsymbol{\omega}} \rVert_\infty$	(3.61)	*convex*
$v_{\max} \geq \lVert E_v \tilde{\boldsymbol{\omega}} \rVert_2$	(3.61)	*convex*

Control Constraints:

$u_{\min} \leq \lVert \boldsymbol{u}_{\mathcal{B}} \rVert_2 \leq u_{\max}$	(3.62)	*nonconvex*
$0 \geq \lVert \boldsymbol{u}_{\mathcal{B}} \rVert_2 - \sec \delta_{\max} \boldsymbol{z}_{\mathcal{B}}^\top \boldsymbol{u}_{\mathcal{B}}$	(3.63)	*convex*
$0 \geq \lVert E_{xy} \dot{\boldsymbol{u}}_{\mathcal{B}} \rVert_2 - \dot{\delta}_{\max} \boldsymbol{z}_{\mathcal{B}}^\top \boldsymbol{u}_{\mathcal{B}}$	(3.64b)	*convex*
$-\dot{u}_{z,\max} \leq \boldsymbol{z}_{\mathcal{B}}^\top \dot{\boldsymbol{u}}_{\mathcal{B}} \leq \dot{u}_{z,\max}$	(3.64a)	*convex*

Boundary Conditions:

$m(t_0) = m_{ic} - \alpha \int_0^{t_c} \Gamma_{\mathrm{c}} \, \mathrm{d}t - \beta t_c$	(3.65a)	*convex*
$\tilde{\boldsymbol{q}}(t_0) = \boldsymbol{b}_{\tilde{\boldsymbol{q}}}(\boldsymbol{q}(t_0), t_c), \quad \tilde{\boldsymbol{q}}(t_f) = \tilde{\boldsymbol{q}}_f$	(3.65a), (3.67a)	*nonconvex*
$\tilde{\boldsymbol{\omega}}(t_0) = \boldsymbol{b}_{\tilde{\boldsymbol{\omega}}}(\boldsymbol{q}(t_0), t_c), \quad \tilde{\boldsymbol{\omega}}(t_f) = \tilde{\boldsymbol{\omega}}_f$	(3.65a), (3.67a)	*nonconvex*

3.4.2 State-Triggered Constraints

The most common type of constraints in the optimal control literature are enforced either over the entire maneuver time interval $[t_0, t_f]$, or at a predetermined subset of this interval (for example in multi-phase optimal control problems). These constraints can be classified as *temporally-scheduled constraints*. In contrast, a *state-triggered constraint* (STC) is enforced only when a state-dependent condition is satisfied, and the time interval over which the constraint is satisfied becomes a function of the trajectory itself (and is not predetermined). STCs allow discrete decisions to be formulated into a continuous optimization framework. Thus, an optimal control problem containing an STC determines its solution variables with a simultaneous understanding of how the constraint affects the optimization, and of how the optimization enables or disables the constraint. The "state" referred to in STC is not limited to the traditional definition from linear systems theory (i.e., (3.46)). Rather, it connotes a (set of) variable(s) from a larger optimization problem, and applies equally to control-, parameter- or time-triggered constraints.

To motivate the utility of STCs in powered descent, consider the scenario depicted in Figure 3.8. Here, we have a sensor whose boresight vector passes through the center of mass of vehicle and has a relatively small field of view. The sensor's boresight is canted from the vertical down direction because of the need to avoid pointing the sensor directly into the main engine's thrust plume while providing a direct line of sight to the ground [89]. Suppose that we try to enforce the line of sight constraint throughout the maneuver, while also ensuring that the vehicle is upright when it reaches the landing site. As shown in the rightmost graphic of Figure 3.8, this could result in an infeasible optimal control problem because these two constraints can become inconsistent near the end of the maneuver.

In the previous example, we can reacquire a feasible problem by asking that the line of sight constraint only be enforced for the first few seconds of the descent, or is not enforced once the vehicle is within a given distance from the landing site. That is, we would like the constraint to be *disabled* once a certain time- or state-based condition is met. This particular

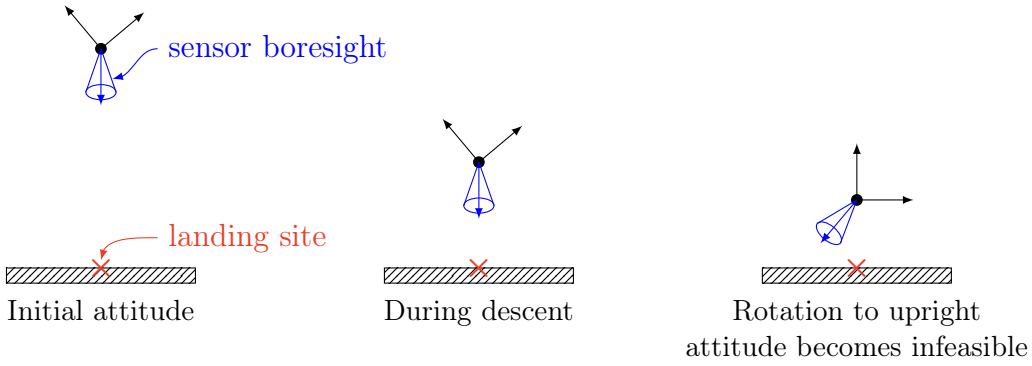


Figure 3.8: A simple landing scenario where maintaining the landing site in a sensor's field of view is incompatible with an upright orientation towards the end of the descent maneuver.

sensing example is not as pathological as it may seem; indeed there are several examples in powered descent where it is beneficial to have the ability to either stop or start enforcing a constraint once a particular set of criteria is met. It certainly appears that this extends to trajectory optimization in general. Some other examples (including an elaboration on this motivating example) are provided after the definition of a state-triggered constraint and their continuous-variable formulation are introduced.

Formally, an STC is defined as an inequality constraint that is enforced conditionally according to a logical statement. Each STC is composed of a *trigger condition* and a *constraint condition*. The trigger condition is given by the inequality $g(\boldsymbol{z}) < 0$, where $\boldsymbol{z} \in \mathbb{R}^{n_z}$ denotes the solution variable of a parent optimization problem, and $g(\cdot) : \mathbb{R}^{n_z} \to \mathbb{R}$ is called the *trigger function*. The constraint condition is given by the inequality $c(\boldsymbol{z}) \leq 0$, where $c(\cdot) : \mathbb{R}^{n_z} \to \mathbb{R}$ is called the *constraint function*. Both $g(\cdot)$ and $c(\cdot)$ are assumed to be differentiable with respect to their arguments. An STC is then defined as the following logical implication:

$$g(\boldsymbol{z}) < 0 \;\Rightarrow\; c(\boldsymbol{z}) \leq 0\,. \tag{3.68}$$

The practical value of (3.68) is also evident from the contrapositive implication: the constraint condition may be violated *only if* the trigger condition is not satisfied.

Continuous Formulation

The STC in (3.68) represents a binary decision: if the trigger condition evaluates to true, then the constraint condition is enforced, otherwise, the constraint condition is not enforced. This type of constraint is not readily incorporated into a continuous optimal control or parameter optimization problem. Mixed-integer techniques are the most common way to implement constraints like (3.68), and require the introduction of discrete decision variables. Despite the existence of efficient branch and bound algorithms, these approaches suffer from worst-case exponential computational complexity [90, 91]. This computational complexity is further compounded by the iterative nature of the solution process required to solve Problem 6; each combination of discrete decisions in the branch and bound sequence would require multiple iterations to converge. Consequently, we argue that mixed-integer solution strategies are not well suited for solving powered descent guidance problems in ways that will be conducive to real-time implementation.

The role of the continuous formulation is therefore to represent the binary nature of (3.68) as the outcome of a set of equations composed solely of continuous (non-integer) variables. To this end, consider the set of equations

$$g(\boldsymbol{z}) + \sigma \geq 0, \quad \sigma \geq 0, \quad \sigma\big(g(\boldsymbol{z}) + \sigma\big) = 0, \tag{3.69a}$$

$$\sigma c(\boldsymbol{z}) \leq 0. \tag{3.69b}$$

The formulation in (3.69) ensures that σ is strictly positive if the trigger condition is satisfied. Indeed, if $g(\boldsymbol{z}) < 0$, then (3.69a) implies that $\sigma > 0$, and therefore (3.69b) is true if and only if $c(\boldsymbol{z}) \leq 0$, which is the constraint condition. Therefore (3.68) and (3.69) are logically equivalent.

To ensure the uniqueness of σ (a feature that is very useful for numerical implementation), note that for a given $\boldsymbol{z}$, the three constraints (3.69a) form a *linear complementarity problem* in the variable σ [92]. This problem has a unique solution σ^*, a property that is only made possible through the addition of the third constraint in (3.69a), and this unique solution

varies continuously as a function of $\boldsymbol{z}$. The analytical solution of this linear complementarity problem, σ^*, is given by

$$\sigma^*(\boldsymbol{z}) \coloneqq -\min\big(g(\boldsymbol{z}),0\big). \tag{3.70}$$

Substituting (3.70) into (3.69) guarantees the satisfaction of (3.69a), and therefore we need only enforce the following inequality constraint to capture the intended logical decision:

$$h(\boldsymbol{z}) \coloneqq -\min\big(g(\boldsymbol{z}),0\big)c(\boldsymbol{z}) \leq 0. \tag{3.71}$$

The constraint (3.71) is a logically equivalent formulation of (3.68). Moreover, by using the solution (3.70), the slack variable that was added to capture the binary nature of the STC, σ, was eliminated altogether; erasing the need to add a new set of solution variables to the parent optimization problem.

Compound State-Triggered Constraints

Certain applications may have one constraint condition that is enforced when a combination of trigger conditions are satisfied or, conversely, have multiple constraint conditions that are enforced when one trigger condition is satisfied. While the STC defined in (3.68) assumes scalar-valued trigger and constraint functions, the natural generalization to vector-valued *compound state-triggered constraints* (CSTC) can be formulated as well [77]. The trigger and constraint conditions of compound state-triggered constraints are composed by using Boolean *and* and *or* operations. The scalar definition of an STC in (3.68) can therefore be generalized to one of the following expressions

$$\textit{Or-Or}: \qquad \bigvee_{i=1}^{n_g}\Big(g_i(\boldsymbol{z})<0\Big) \quad \Rightarrow \quad \bigvee_{i=1}^{n_c}\Big(c_i(\boldsymbol{z})\leq 0\Big), \tag{3.72a}$$

$$\textit{Or-And}: \qquad \bigvee_{i=1}^{n_g}\Big(g_i(\boldsymbol{z})<0\Big) \quad \Rightarrow \quad \bigwedge_{i=1}^{n_c}\Big(c_i(\boldsymbol{z})\leq 0\Big), \tag{3.72b}$$

$$\textit{And-Or}: \quad \bigwedge_{i=1}^{n_g} \Big(g_i(\boldsymbol{z}) < 0 \Big) \quad \Rightarrow \quad \bigvee_{i=1}^{n_c} \Big(c_i(\boldsymbol{z}) \leq 0 \Big), \tag{3.72c}$$

$$\textit{And-And}: \quad \bigwedge_{i=1}^{n_g} \Big(g_i(\boldsymbol{z}) < 0 \Big) \quad \Rightarrow \quad \bigwedge_{i=1}^{n_c} \Big(c_i(\boldsymbol{z}) \leq 0 \Big), \tag{3.72d}$$

where now $g(\cdot) : \mathbb{R}^{n_z} \to \mathbb{R}^{n_g}$ and $c(\cdot) : \mathbb{R}^{n_z} \to \mathbb{R}^{n_c}$ denote vector-valued functions. As before, we assume that the Jacobians $\nabla_{\boldsymbol{z}} g$ and $\nabla_{\boldsymbol{z}} c$ exist. The *Or-Or* CSTC (3.72a) enforces at least one constraint condition $c_i(\boldsymbol{z}) \leq 0$ if any of the trigger conditions $g_i(\boldsymbol{z}) < 0$ are satisfied. The *Or-And* CSTC (3.72b) enforces all constraint conditions if any one of the trigger conditions are satisfied. The *And-Or* CSTC (3.72c) enforces at least one constraint condition only if all of the trigger conditions are satisfied. Finally, the *And-And* CSTC (3.72d) enforces all constraint conditions when all trigger conditions are satisfied.

The continuous formulations corresponding to the CSTCs defined in (3.72) can be derived by repeating the same steps that were used for the scalar case:

$$\textit{Or-Or}: \quad h_{\vee\vee}(\boldsymbol{z}) \coloneqq \left[\sum_{i=1}^{n_g} \sigma_i^*(\boldsymbol{z}) \right] \left[\prod_{i=1}^{n_c} c_i(\boldsymbol{z}) + \varsigma_i \right] = 0, \quad \varsigma_i \geq 0. \tag{3.73a}$$

$$\textit{Or-And}: \quad h_{\vee\wedge}(\boldsymbol{z}) \coloneqq \left[\sum_{i=1}^{n_g} \sigma_i^*(\boldsymbol{z}) \right] c_j(\boldsymbol{z}) \leq 0, \qquad j = 1, \ldots, n_c. \tag{3.73b}$$

$$\textit{And-Or}: \quad h_{\wedge\vee}(\boldsymbol{z}) \coloneqq \left[\prod_{i=1}^{n_g} \sigma_i^*(\boldsymbol{z}) \right] \left[\prod_{i=1}^{n_c} c_i(\boldsymbol{z}) + \varsigma_i \right] = 0, \quad \varsigma_i \geq 0. \tag{3.73c}$$

$$\textit{And-And}: \quad h_{\vee\wedge}(\boldsymbol{z}) \coloneqq \left[\prod_{i=1}^{n_g} \sigma_i^*(\boldsymbol{z}) \right] c_j(\boldsymbol{z}) \leq 0, \qquad j = 1, \ldots, n_c. \tag{3.73d}$$

The slack variable $\boldsymbol{\varsigma} \in \mathbb{R}^{n_c}$ has been introduced to account for the inequality form of the CSTC, and as a result the formulation (3.73) is slightly different from the case where the constraint conditions are given by equalities [77].

Examples in Powered Descent

Several interesting examples of STCs and CSTCs have been identified for problems in powered descent, autonomous rendezvous, quadrotor path planning, and autonomous driving [72, 77, 78, 93, 94, 95]. In this section, we explore three examples of state-triggered constraints as they apply to powered descent guidance problems.

Let's revisit the motivating example from Figure 3.8 and see how to formulate the constraint now that we have introduced the machinery to do so. We'll generalize this example to use a compound state-triggered constraint. Consider a slant-range-triggered line of sight (LOS) constraint whereby a vehicle is constrained to point the optical sensor at the landing site only during a portion of the descent when its slant range lies between two predetermined values. This scenario is illustrated in Figure 3.9 and is particularly relevant for future autonomous landing systems that employ Hazard or Terrain Relative Navigation (HRN/TRN) and/or vision-based safe site selection and redesignation [34, 68]. Because the constraint is enforced conditionally on the slant range taking values in an interval, this is an example of a compound state-triggered constraint. The compound trigger condition is given in inertial (Cartesian) coordinates as

$$\bigwedge_{i=1}^{2} \big(g_{\xi,i}(\boldsymbol{r}_{\mathcal{L}}) < 0\big) \coloneqq \big(\rho_{min} < \|\boldsymbol{r}_{\mathcal{L}}\|_2\big) \wedge \big(\|\boldsymbol{r}_{\mathcal{L}}\|_2 < \rho_{max}\big)$$

and is designed to trigger when the vehicle has a slant range between ρ_{min} and ρ_{max} to the landing site. Expressing the trigger functions in terms of each state vector by using (2.41d) gives

$$g_{\xi,1}(\boldsymbol{x}) \coloneqq \begin{cases} \rho_{min} - \|2E_d\tilde{\boldsymbol{q}}\|_2, & \text{Dual quaternion,} \\ \rho_{min} - \|\boldsymbol{r}_{\mathcal{L}}\|_2, & \text{Cartesian.} \end{cases} \tag{3.74a}$$

$$g_{\xi,2}(\boldsymbol{x}) \coloneqq \begin{cases} \|2E_d\tilde{\boldsymbol{q}}\|_2 - \rho_{max}, & \text{Dual quaternion,} \\ \|\boldsymbol{r}_{\mathcal{L}}\|_2 - \rho_{max}, & \text{Cartesian.} \end{cases} \tag{3.74b}$$

To develop the constraint condition, consider a sensor that is installed with a fixed offset from the origin of the body frame, denoted by $\boldsymbol{d}_{\mathcal{B}} \in \mathbb{R}^3$, and a constant orientation with respect the body frame, expressed by the rotation matrix $C_{\mathcal{BS}}$. If the sensor's boresight direction in the sensor frame is given by $\boldsymbol{p}_{\mathcal{S}}$, then the sensor's boresight in the body frame is given by

$$\boldsymbol{p}_{\mathcal{B}} = C_{\mathcal{BS}}\boldsymbol{p}_{\mathcal{S}} + \boldsymbol{d}_{\mathcal{B}}. \tag{3.75}$$

An LOS constraint ensures that the angle between the vector $\boldsymbol{p}_{\mathcal{B}}$ and the sensor-to-landing-site vector, $-\boldsymbol{r}_{\mathcal{B}} - \boldsymbol{d}_{\mathcal{B}}$ is less than $\xi_{\max} \in [0, 90]$ deg. In Cartesian coordinates, this is represented by

$$(\boldsymbol{r}_{\mathcal{B}} + \boldsymbol{d}_{\mathcal{B}})^\top \boldsymbol{p}_{\mathcal{B}} + \|\boldsymbol{r}_{\mathcal{B}} + \boldsymbol{d}_{\mathcal{B}}\|_2 \cos\xi_{\max} \leq 0 \tag{3.76}$$

We assume that $\|\boldsymbol{r}_{\mathcal{B}}\|_2 \gg \|\boldsymbol{d}_{\mathcal{B}}\|_2$, so that we can approximate the constraint as

$$\boldsymbol{r}_{\mathcal{B}}^\top \boldsymbol{p}_{\mathcal{B}} + \|\boldsymbol{r}_{\mathcal{B}}\|_2 \cos\xi_{\max} \leq \varepsilon_\xi \tag{3.77}$$

where $\varepsilon_\xi = -\boldsymbol{d}_{\mathcal{B}}^\top \boldsymbol{p}_{\mathcal{B}}$ is a constant quantity and is zero if the sensor is located at the center of mass of the vehicle (the origin of the body frame). Note that in general, the additional modeling fidelity afforded by considering constant sensor offsets is small, in particular when the distances covered during landing far exceed the length-scale of the vehicle.

The left-hand side of (3.77) can be expressed as a function of the dual quaternion $\tilde{\boldsymbol{q}}$ by using (2.41b) and (2.41d) to write

$$c_\xi(\boldsymbol{x}) \equiv c_\xi(\tilde{\boldsymbol{q}}) \coloneqq \tilde{\boldsymbol{q}}^\top M_\xi \tilde{\boldsymbol{q}} + \|2E_d\tilde{\boldsymbol{q}}\|_2 \cos\xi_{\max} \leq \varepsilon_\xi, \quad M_\xi = \begin{bmatrix} 0_{4\times4} & [\boldsymbol{p}_{\mathcal{B}}]_\otimes^{*\top} \\ [\boldsymbol{p}_{\mathcal{B}}]_\otimes^* & 0_{4\times4} \end{bmatrix}. \tag{3.78}$$

References [32, 71] showed that the function $c_\xi(\tilde{\boldsymbol{q}})$ in (3.78) is a convex function over the same domain as (3.57). A compound state-triggered constraint using an *And* trigger

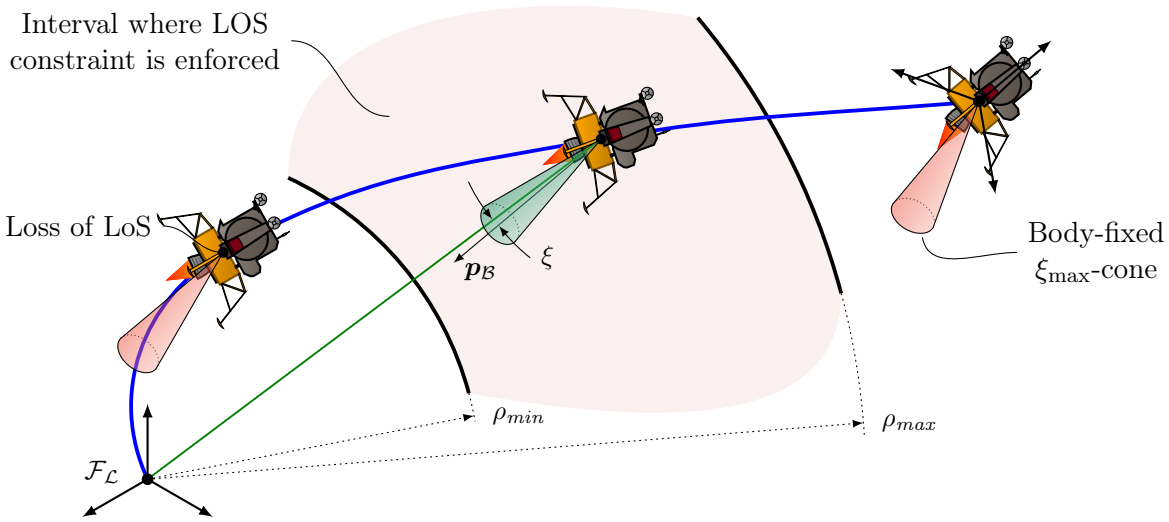


Figure 3.9: An illustration of the slant-range-triggered line of sight compound state-triggered constraint for a case where $\boldsymbol{d}_{\mathcal{B}} = 0$.

condition is obtained by using (3.74) and (3.78) to write:

$$h_{\xi}(\boldsymbol{x}) \coloneqq \sigma^{*}_{\xi,1}(\boldsymbol{x})\sigma^{*}_{\xi,2}(\boldsymbol{x})c_{\xi}(\boldsymbol{x}) \leq 0, \tag{3.79}$$

where $\sigma^{*}_{\xi,i}(\boldsymbol{x}) = -\min\left(g_{\xi,i}(\boldsymbol{x}),\, 0\right)$ for each of the two trigger functions $g_{\xi,i}$.

A related example is a slant-range-triggered approach angle constraint, whereby the approach cone constraint (3.57) is enforced only if the vehicle is within a certain distance of the landing site. This particular constraint has applications in both powered descent and rendezvous and proximity operations. In the case of powered descent, consider a vehicle at low altitude but with a nearly-horizontal velocity (traveling nearly parallel to the ground) at a large distance from the landing site – where the approach angle may be quite large. In this case, it may be desirable to compute a trajectory that steers the vehicle toward a descent with an approach angle that is close to 0 deg (for a vertical descent) *only* near the end of the maneuver. This scenario is typical of lunar landings, where the altitude-to-horizontal-velocity ratio is quite low relative to other landing scenarios. See Figure 3.10a for an illustration. In

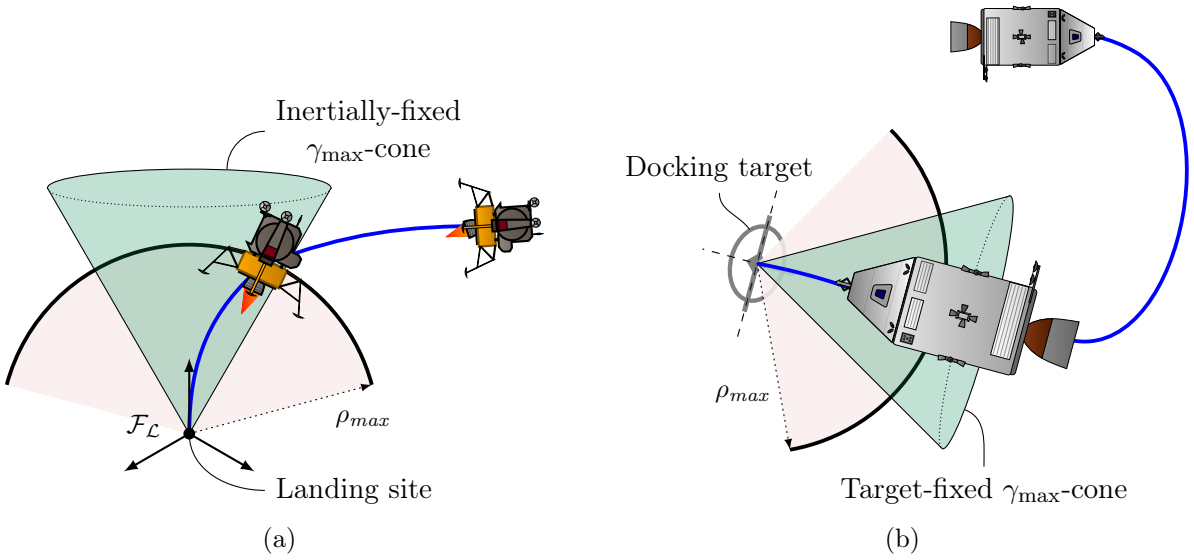


Figure 3.10: An illustration of the slant-range-triggered approach cone constraint for both (a) powered descent and (b) rendezvous and proximity operations.

the context of rendezvous and proximity operations, this type of constraint would allow a vehicle to compute a feasible trajectory from outside of the approach cone, and also ensures that the "chaser" vehicle does not come within a specified distance of the docking target unless it has a sufficient approach angle. See Figure 3.10b for an illustration.

The trigger function is formed by (3.74b), which results in a trigger condition that evaluates to true once the vehicle has a slant-range of less than $\rho_{\max}$ to the landing site or docking target. The constraint condition is given by the approach angle constraint (3.57). The state-triggered constraint can then be formulated as

$$h_\gamma(\boldsymbol{x}) \equiv h_\gamma(\tilde{\boldsymbol{q}}) = \sigma^*_{\xi,2} c_\gamma(\tilde{\boldsymbol{q}}) \leq 0, \tag{3.80}$$

where again $\sigma^*_{\xi,2} = -\min\big(g_{\xi,2}(\boldsymbol{x}),\, 0\big)$.

As a final example, consider the problem of limiting the aerodynamic loads on a vehicle during a powered descent maneuver. On ascent, aerodynamic loads are often limited by imposing what are known as $q-\alpha$ limits, where q refers to dynamic pressure, and α refers to

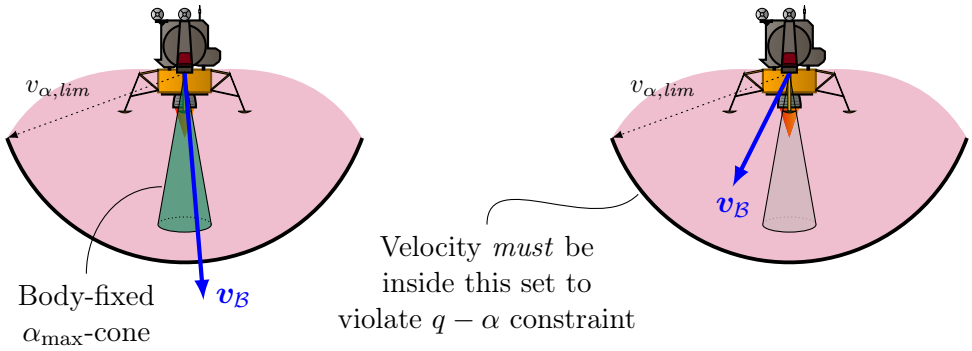


Figure 3.11: An illustration of the speed-triggered angle of attack constraint.

angle of attack. The angle of attack is kept small to ensure that the flight envelope remains in well-understood aerodynamic regimes and structural loads are not exceeded. Unlike an ascent trajectory, however, a powered descent maneuver can exhibit a wider range of angles of attack, possibly even exceeding 90 degrees. Because aerodynamic loads are a function of both dynamic pressure and angle of attack, a state-triggered constraint can be used to enforce a simplified $q-\alpha$ limit only during portions of the descent when q is high, see Figure 3.11 for an illustration. Stated another way, an angle of attack constraint can be enforced only when the vehicle is traveling at high speeds. Formally, the trigger function is

$$g_\alpha(\boldsymbol{x}) \coloneqq \begin{cases} v_{\alpha,lim} - \|E_v \tilde{\boldsymbol{\omega}}\|_2, & \text{Dual Quaternion,} \\ v_{\alpha,lim} - \|\boldsymbol{v}_{\mathcal{L}}\|_2, & \text{Cartesian,} \end{cases} \tag{3.81}$$

where $v_{\alpha,lim} \in \mathbb{R}_{++}$ denotes the speed-threshold above which the angle of attack is to be restricted. The angle of attack is defined to be the angular difference between the velocity and the vehicle's negative vertical axis in the body frame. In Cartesian coordinates, the constraint condition is

$$\boldsymbol{z}_{\mathcal{B}}^\top C_{\mathcal{BL}}(\boldsymbol{q}) \boldsymbol{v}_{\mathcal{L}} + \|\boldsymbol{v}_{\mathcal{L}}\|_2 \cos \alpha_{\max} \leq 0 \tag{3.82}$$

where $\alpha_{\max} \in \mathbb{R}_{++}$ denotes the maximum allowable angle of attack. The constraint (3.82) can be expressed as a function of the dual velocity in this case by noting that $\boldsymbol{v}_{\mathcal{B}} = C_{\mathcal{BL}}(\boldsymbol{q}) \boldsymbol{v}_{\mathcal{L}}$,

which is equivalent to $E_v\tilde{\boldsymbol{\omega}}$ as in (3.81). Hence the constraint condition can be written using dual quaternions as

$$c_\alpha(\boldsymbol{x}) \equiv c_\alpha(\tilde{\boldsymbol{\omega}}) \coloneqq \boldsymbol{z}_{\mathcal{B}}^\top E_v\tilde{\boldsymbol{\omega}} + \|E_v\tilde{\boldsymbol{\omega}}\|_2 \cos\alpha_{\max} \leq 0. \tag{3.83}$$

Per Remark 3.6, the definition of $\boldsymbol{v}_{\mathcal{B}}$ in the dual velocity results in the function (3.83) being a *convex function* of $\tilde{\boldsymbol{\omega}}$, as it is a second-order cone. This does not cause the resulting continuous STC formulation to be convex, but can provide a significant numerical advantage. The state-triggered constraint is obtained by using (3.81) and either of (3.82) or (3.83) to write

$$h_\alpha(\boldsymbol{x}) \coloneqq \sigma_\alpha^*(\boldsymbol{x}) c_\alpha(\boldsymbol{x}) \leq 0, \tag{3.84}$$

where $\sigma_\alpha^*(\boldsymbol{x}) = -\min(g_\alpha(\boldsymbol{x}), 0)$ uses the appropriate trigger function from (3.81).

Chapter 4

EXPLICIT TRAJECTORY OPTIMIZATION:
SEQUENTIAL CONVEX PROGRAMMING

The necessary number of iterations is one more than the number you have currently done.
This is true at any point in time.
– Akin's Third Law

This chapter focuses on the use of sequential convex programming, or successive convexification, to solve nonconvex optimal control problems. Solving nonconvex optimization problems is difficult. Alas, nonconvex problems are pervasive in guidance and control problems for aerospace systems, and more generally design problems in the broader fields of engineering. Problem 6 is one such example. In a taxonomy of techniques that are capable of solving nonconvex optimization problems, sequential convex programming (SCP) is one of several methodologies alongside nonlinear programming [8, 96], sum of squares optimization [97], genetic or evolutionary algorithms, dynamic programming, and augmented Lagrangian techniques, to name but a few.

This chapter outlines the design of an algorithm that emphasizes numerical robustness to parameter changes and the accuracy of approximations to the nonlinear dynamics and nonconvex constraints. The algorithm we develop generates *feedforward* trajectories that can be used as a reference for an appropriately designed feedback controller. As laid out in Chapter 1, optimality *is not* taken to be the driving requirement in the design of such algorithms. What is more important is that any trajectory computed: 1) adheres to the nonlinear dynamics of the problem, and 2) satisfies all desired state and control constraints. Colloquially, it is not worth a large increase in computational effort for a small reduction in fuel usage, but it is worth a large computational effort to avoid a crash landing. Indeed, it

is very likely that 6-DOF propellant-optimality can be well approximated for most powered descent scenarios by solving the 3-DOF problem and using an inner attitude tracking loop. However, a 3-DOF guidance method would require extensive analysis across the flight envelop to ensure that any computed trajectory will not violate a 6-DOF constraint (e.g., a line of sight constraint), and could subsequently lead to a reduction in the feasible trade space for guidance designs [34].

As the name suggests, the main idea of sequential convex programming is to solve a sequence of convex approximations, called subproblems, to the original nonconvex problem and update the approximation as new solutions are obtained. The solution to the nonconvex problem is then effectively a limit point of a sequence of solutions to convex optimization subproblems. This approach offers two main advantages. First, a wide range of numerical techniques exist to reliably solve each convex subproblem. Second, one can derive meaningful theoretical guarantees on algorithm performance and computational complexity. Taken together, these advantages have led to the development of very efficient algorithms with runtimes low enough to enable real-time deployment for aerospace applications.

Nonconvex optimization often has fundamental limitations that permit us to obtain either locally optimal solutions quickly or globally optimal solutions slowly. SCP algorithms are not immune to this trade-off, despite the fact that certain subclasses of convex optimization can be viewed as "easy" from a computational perspective due to the availability of interior point methods. All reported solutions of the 6-DOF powered descent guidance problem use the former strategy, and are only (theoretically) capable of approximating local optimality. In contrast, use cases such as truss structure design may favor global optimality over solution speed.

The description of the algorithm is best carried out by abstracting the specific details of powered descent (or any other specific application), and using the standard form optimal control problem described in Problem 1. Using the standard form problem as a starting point, this chapter is organized as follows. First, §4.1 provides a review of the pertinent literature on sequential convex programming and contextualizes the core algorithmic steps.

Next, §4.2 outlines the SCP algorithm in detail. This outline is split into several steps that each represent implementable functional units (i.e., one can create a functions in code that execute each step). Lastly, §4.3 provides two case studies of the SCP algorithm to the solution of Problem 6 with and without state-triggered constraints.

4.1 Review of Past Work

Tracing the origins of what we refer to as sequential convex programming is not a simple task. Since the field of nonlinear optimization gained traction as a popular discipline in the 1960s and 70s, many researchers have explored the solution of nonconvex optimization problems via convex approximations. The proceeding historical review is bound to omit some important works that have influenced, directly or indirectly, the current state of the art. The idea to solve a general (nonconvex) optimization problem by iteratively approximating it as a convex program was perhaps first developed using branch and bound techniques [98, 99, 100, 101]. Early results were of mainly academic interest, and computationally tractable methods remained a work in progress. One of the most important ideas that emerged from these early investigations appears to be that of McCormick relaxations [102]. These are a set of atomic rules for constructing convex/concave relaxations of a specific class of functions that everywhere under-/over-estimate the original functions. Algorithms based on McCormick relaxations continue to be developed with increasing computational capabilities [103, 104, 105, 106].

Perhaps the simplest class of SCP methods is that of Sequential Linear Programming (SLP). These algorithms linearize all nonlinear functions about a current reference solution so that each subproblem is a linear program. The linear programs are solved with a trust region to obtain a new reference, and the process is repeated. Early developments came from the petroleum industry and were intended to solve large problems [107, 108]. From a computational perspective, SLP was initially attractive due to the maturity of the simplex algorithm. Over time, however, solvers for more general classes of convex optimization problems have advanced to the point that restricting oneself to linear programs to save com-

putational resources has become unnecessary, except perhaps for extremely large problems.

Arguably the most mature class of SCP methods is that of Sequential Quadratic Programming (SQP). The works of Han [109, 110], Powell [111, 112, 113], Boggs and Tolle [114, 115, 116] and Fukushima [117] appear to have exerted significant influence on the early developments of SQP-type algorithms, and their impact remains evident today. An excellent survey was written by Boggs and Tolle [118]. Modern developments continue to address both theoretical and applied aspects [119, 120], and there have been several specific applications of SQP algorithms to solve problems in the domain of aerospace systems [25, 50, 121]. SQP methods approximate a nonconvex program with a quadratic program using some reference solution, and then use the solution to this quadratic program to update the approximation. The proliferation of SQP-type algorithms can be attributed to three main aspects: 1) their similarity with the familiar class of Newton methods, 2) the fact that the initial reference need not be feasible, and 3) the capability to quickly and reliably solve quadratic programs. In fact, the iterates obtained by SQP algorithms can be interpreted either as solutions to quadratic programs or as the application of Newton's method to the optimality conditions of the original problem [8]. That is not to say that SQP methods are without drawback. Gill and Wong nicely summarize the difficulties that can arise when using SQP methods [122], and we will only outline the basic ideas here. Most importantly – and this goes for any "second-order" method – it is difficult to accurately and reliably estimate the Hessian of the nonconvex problem's Lagrangian. Even if this is done, say, analytically, there is no guarantee that it will be positive semidefinite, and a non-positive semidefinite Hessian would result in an NP-hard nonconvex quadratic program. Hessian approximation techniques must therefore be used, such as keeping only the positive semidefinite part or using the BFGS update [123]. In the latter case, additional conditions must be met to ensure that the Hessian remains positive definite. These impose both theoretical and computational challenges that, if unaddressed, can both impede convergence and limit the real-time applicability. Fortunately, a great deal of effort has gone into making SQP algorithms highly practical [124].

One truly insurmountable drawback to SQP methods is that quadratic programs require

that all constraints are affine in the solution variable. As shown in §3.4, there are more general nonlinear constraints (both convex and nonconvex) that appear in the 6-DOF powered descent guidance problem alone. For example, each of the approach cone, tilt angle, gimbal angle, vehicle speed, and line of sight constraints fit into this description. The use of SQP algorithms in cases where second-order cone constraints are present may require more iterations to converge than solving a more general second-order cone program at each iteration instead.

We therefore focus on the class of SCP methods that solve a general convex program at each iteration, without a priori restriction to one of the previously mentioned classes of convex programs (e.g., LPs, QPs, SOCPs). More importantly, we use an algorithm that is of trust region type and uses slack variables to ensure that each convex subproblem is always feasible. This class of SCP methods has been developed largely over the last decade, and several state-of-the-art SCP algorithms for trajectory generation have emerged in the past few years. These are: Penalized Trust Region (PTR) [73, 78, 125], successive convexification (SCvx) [126, 127, 128], DESCENDO [129], TrajOpt [130], GuSTO [131, 132], and ALTRO [133]. This area is certainly the most active area of current development, and beyond these "generalized" solution methods, there have been several successful applications to a wide range of robotic and aerospace guidance problems [25, 71, 134, 135, 136, 137, 138].

The specific SCP algorithm described in this chapter is the PTR algorithm, and it is depicted in Figure 4.1. In the PTR algorithm, the trust region radius is a solution variable whose magnitude is penalized in the cost. This contrasts with algorithms such as SCvx and GuSTO that enforce hard trust regions with outer-loop update schemes.

4.2 PTR Algorithm Description

To describe the steps taken by the algorithm, we'll consider the general optimal control problem, Problem 1 from Chapter 2. Instead of using a set-based description of the state and control constraints, we assume instead that the desired set inclusion can be written in terms of n_{cvx} convex inequality constraints and n_{ncvx} nonconvex inequality constraints.

Problem 7 provides the formal problem statement.

Problem 7. *Find the piecewise control signal* $\boldsymbol{u}(\cdot) : \mathbb{R} \to \mathbb{R}^{n_u}$*, initial state* $\boldsymbol{x}(t_0) \in \mathbb{R}^{n_x}$ *and parameter vector* $\boldsymbol{p} \in \mathbb{R}^{n_p}$ *that solve the following problem:*

$$\min_{\boldsymbol{x}(t_0),\, \boldsymbol{u}(\cdot),\, \boldsymbol{p}} \quad J\big(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}\big) \tag{4.1a}$$

$$s.t. \quad \dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\big), \tag{4.1b}$$

$$g_i\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\big) \leq 0, \qquad i = 1, \ldots, n_{cvx}\,, \tag{4.1c}$$

$$h_j\big(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}\big) \leq 0, \qquad j = 1, \ldots, n_{ncvx}\,, \tag{4.1d}$$

$$S_0\big(\boldsymbol{x}(t_0), \boldsymbol{p}\big) = 0, \quad S_f\big(\boldsymbol{x}(t_f), \boldsymbol{p}\big) = 0. \tag{4.1e}$$

where each function in (4.1c)–(4.1e) *is assumed to be continuous everywhere and differentiable almost everywhere with respect to each argument. Assumption 2.1 is assumed to hold. The scalar-valued functions* g_i *are assumed to be (jointly) convex with respect their arguments, whereas the scalar-valued functions* h_j *are assumed to be nonconvex with respect to at least one of their arguments. It is assumed that the Jacobians* $\nabla S_0 \in \mathbb{R}^{n_0 \times (n_x + n_p)}$ *and* $\nabla S_f \in \mathbb{R}^{n_f \times (n_x + n_p)}$ *have full row rank.*

The cost function in (4.1a) is assumed to be in Bolza form, like (2.5), and is expressed as

$$J\big(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}\big) = M\big(\boldsymbol{x}(t_0), \boldsymbol{x}(t_f), \boldsymbol{p}\big) + \int_{t_0}^{t_f} L\big(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta), \boldsymbol{p}\big)\, \mathrm{d}\zeta, \tag{4.2}$$

where the cost functions M and L are assumed to be convex in each of their arguments.[1] Recall that the problem's dependence on the parameter vector $\boldsymbol{p}$ can be used, for example, to capture free-final-time problems or to capture the variable initial position and velocity by including the coast time introduced in Figure 3.7.

[1]This is without loss of generality because nonconvex functions can be equivalently replaced by the minimization of a parameter and an associated nonconvex inequality (for M) or a nonlinear differential equality constraint (for L).

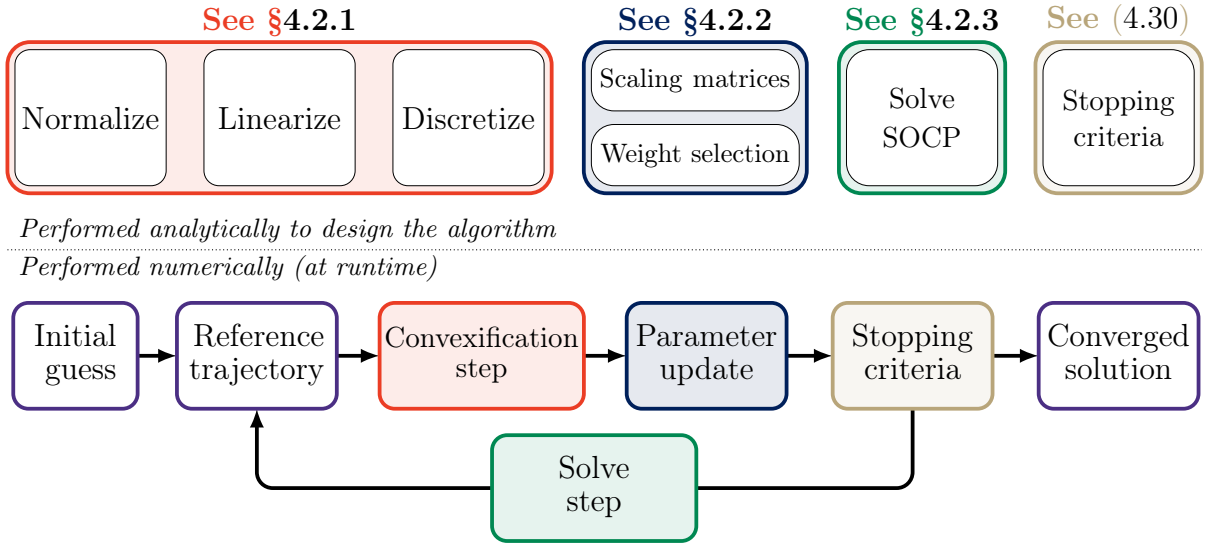


Figure 4.1: Overview of the sequential convex programming algorithm, referred to as the PTR algorithm, designed to solve problems in the class of Problem 7.

4.2.1 Convexification Step

SCP methods work by computing a sequence of local convex approximations to Problem 4.1, and we refer to the approximated problems as subproblems. To construct each subproblem, we will presume to have access to a reference solution that is used to approximate any nonconvexities. Henceforth, $\{\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{p}}\}$ will always play the role of "solution at the previous iteration" with which a new subproblem is derived.

Problem 7 has three possible sources of nonconvexity: the nonlinear equations of motion (4.1b); the nonconvex state and control constraints (4.1d); and the boundary conditions (4.1e). The equations of motion represent a differential relationship between the control and parameter vectors and the state vector, where the latter two sources of nonconvexity are algebraic functions of the solution variables at any time. The difference between these two types of nonconvexity are significant enough that we present the convexification method for each separately.

Propagation

We first consider the convexification of the nonlinear equations of motion (4.1b). Because the equations of motion are (almost always) stated as a set of *equality* constraints, in order for them to be convex functions they must be *affine* in the solution variables. Moreover, because the control function $\boldsymbol{u}(t)$ belongs to the infinite dimensional vector space of piecewise continuous functions, we must consider a finite-dimensional representation in order to numerically solve the problem on a digital computer. The procedure of approximating the space of admissible control functions by a subset of functions that are defined by a finite number of parameters is broadly referred to as *transcription* – an umbrella term that typically refers to one of temporal discretization or direct collocation [23]. In both cases, a set of N temporal nodes are selected by the user such that

$$t_0 = t_1 < t_2 < \ldots < t_{N-1} < t_N = t_f, \tag{4.3}$$

where the difference between adjacent temporal nodes does not need to be equal. We call N the temporal density. On one hand, temporal discretization refers to using the exact solution of the (or an approximated) differential equation to express the state vector at each temporal node as a function of the control and parameter vectors. On the other hand, direct collocation typically matches the derivatives of the differential equation and of a polynomial function at the temporal nodes. Both methods, when used properly in a sequential convex programming setting, are capable of satisfying the nonlinear equations of motion. There are important differences between the two techniques for real-time implementations (see [139]), but there are also different theoretical properties regarding the relationship to the optimum in the original infinite-dimensional space of functions [140].

In either case, a solution to a transcribed problem can only approximate the optimality and feasibility of the original problem because the control signal has fewer degrees of freedom than its continuous time counterpart (and because the state and control constraints are not enforced over the entire time horizon, as we shall see shortly). Coarse transcription techniques

can render a convex subproblem infeasible even if the original optimal control problem is feasible. In the absence of any state and control constraints, a solution to a transcribed problem will always be suboptimal with respect to its continuous time counterpart, though the suboptimality can be made to be arbitrarily small at the expense of problem size.

Likely the most popular direct collocation strategy are the so-called pseudospectral methods, introduced to the optimal control community by Vlassenbroeck [141, 142]. They approximate both the state and control using orthogonal Chebyshev or Legendre polynomials whose coefficients at the temporal nodes (4.3) are unknown [24, 143]. One can derive a large class of pseudospectral methods by different choices for the interpolating polynomial and the distribution of the temporal nodes. In contrast, temporal discretization methods approximate the control signal only, typically with a simpler representation than the interpolating polynomials used in direct collocation. This simplicity affords the ability to exactly solve the linearized differential equation between each temporal node (and the distribution of those temporal nodes is arbitrary).

We have found, after some experimentation, that the strategy of temporal discretization works well for the powered descent guidance problem (and many other aerospace problems) [139]. We approximate the control function $\boldsymbol{u}(t)$ as a piecewise affine function of time, whereby the values of the function at the temporal nodes (4.3) are solved for and the intermediate values are obtained by linear interpolation. We have found that this method provides a suitable balance between optimality, feasibility, and computational time in subsequent real-time implementations. Formally, we introduce the vectors $\{\boldsymbol{u}_k\}_{k=1}^N = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N\}$ and assume that

$$\begin{aligned}\boldsymbol{u}(t) &= \frac{t_{k+1} - t}{t_{k+1} - t_k}\boldsymbol{u}_k + \frac{t - t_k}{t_{k+1} - t_k}\boldsymbol{u}_{k+1}, \quad t \in [t_k, t_{k+1}] \\ &= \lambda_-(t)\boldsymbol{u}_k + \lambda_+(t)\boldsymbol{u}_{k+1},\end{aligned} \tag{4.4}$$

for each $k = 1, \ldots, N-1$.

The temporal discretization strategy works as follows. To solve problems with a free final

time, the dynamics are first rewritten in terms of a normalized time variable $\tau \in [0, 1]$ that satisfies $t = t_f \tau$. The normalized temporal nodes corresponding to (4.3) are therefore

$$0 = \tau_1 < \tau_2 < \ldots < \tau_{N-1} < \tau_N = 1. \tag{4.5}$$

Using the chain rule, we can write the derivative of the state with respect to the normalized time variable τ as

$$\boldsymbol{x}'(\tau) \coloneqq \frac{d\boldsymbol{x}}{d\tau} = \frac{dt}{d\tau}\frac{d}{dt}\boldsymbol{x}(\tau) = t_f\, f\big(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{p}\big). \tag{4.6}$$

By augmenting the parameter vector so that $\boldsymbol{p} \leftarrow \big(\boldsymbol{p},\, t_f\big)$, we can write (4.6) in the general form of

$$\boldsymbol{x}'(\tau) = F\big(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{p}\big). \tag{4.7}$$

Next, the normalized equations of motion (4.7) are linearized about the reference solution $\{\bar{\boldsymbol{x}},\, \bar{\boldsymbol{u}},\, \bar{\boldsymbol{p}}\}$ to obtain

$$\boldsymbol{x}'(\tau) \approx A(\tau)\boldsymbol{x}(\tau) + B(\tau)\boldsymbol{u}(\tau) + E(\tau)\boldsymbol{p} + \boldsymbol{r}(\tau), \tag{4.8}$$

where,

$$A(\tau) = \nabla_x F\big(\bar{\boldsymbol{x}}(\tau), \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big), \tag{4.9a}$$

$$B(\tau) = \nabla_u F\big(\bar{\boldsymbol{x}}(\tau), \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big), \tag{4.9b}$$

$$E(\tau) = \nabla_p F\big(\bar{\boldsymbol{x}}(\tau), \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big), \tag{4.9c}$$

$$\boldsymbol{r}(\tau) = F\big(\bar{\boldsymbol{x}}(\tau), \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big) - A(\tau)\bar{\boldsymbol{x}}(\tau) - B(\tau)\bar{\boldsymbol{u}}(\tau) - E(\tau)\bar{\boldsymbol{p}}. \tag{4.9d}$$

Now by using the piecewise affine parameterization of the control vector in (4.4), we can

write the linearized dynamics (4.8) for any $\tau \in [\tau_k, \tau_{k+1}]$ as

$$\boldsymbol{x}'(\tau) = A(\tau)\boldsymbol{x}(\tau) + B(\tau)\lambda_-(\tau)\boldsymbol{u}_k + B(\tau)\lambda_+(\tau)\boldsymbol{u}_{k+1} + E(\tau)\boldsymbol{p} + \boldsymbol{r}(\tau). \tag{4.10}$$

We now proceed to solve the differential equation (4.10) for the state vector $\boldsymbol{x}(\tau_{k+1})$. The zero-input state transition matrix associated with this linear time-varying system is

$$\Phi(\tau, \tau_k) = I_{n_x} + \int_{\tau_k}^{\tau} A(\zeta)\Phi(\zeta, \tau_k)\,\mathrm{d}\zeta. \tag{4.11}$$

Using the inverse and transitive properties of the state transition matrix [144], a linear discrete time equation that is equivalent to (4.10) is

$$\boldsymbol{x}_{k+1} = A_k\boldsymbol{x}_k + B_k^-\boldsymbol{u}_k + B_k^+\boldsymbol{u}_{k+1} + E_k\boldsymbol{p} + \boldsymbol{r}_k, \tag{4.12}$$

where,

$$A_k = \Phi(\tau_{k+1}, \tau_k) \tag{4.13a}$$

$$B_k^- = A_k \int_{\tau_k}^{\tau_{k+1}} \Phi^{-1}(\tau, \tau_k)B(\tau)\lambda_-(\tau)\,\mathrm{d}\tau \tag{4.13b}$$

$$B_k^+ = A_k \int_{\tau_k}^{\tau_{k+1}} \Phi^{-1}(\tau, \tau_k)B(\tau)\lambda_+(\tau)\,\mathrm{d}\tau \tag{4.13c}$$

$$E_k = A_k \int_{\tau_k}^{\tau_{k+1}} \Phi^{-1}(\tau, \tau_k)E(\tau)\,\mathrm{d}\tau \tag{4.13d}$$

$$\boldsymbol{r}_k = A_k \int_{\tau_k}^{\tau_{k+1}} \Phi^{-1}(\tau, \tau_k)\boldsymbol{r}(\tau)\,\mathrm{d}\tau \tag{4.13e}$$

In implementation, the integrands of (4.11) and (4.13) are numerically integrated along with the nonlinear state equations (4.7). The reference state vector used to evaluate each integrand during numerical integration is therefore computed for any $\tau \in [\tau_k, \tau_{k+1}]$ by using

$$\bar{\boldsymbol{x}}(\tau) = \bar{\boldsymbol{x}}_k + \int_{\tau_k}^{\tau} F\big(\boldsymbol{x}(\zeta), \boldsymbol{u}(\zeta), \boldsymbol{p}\big)\,\mathrm{d}\zeta \tag{4.14}$$

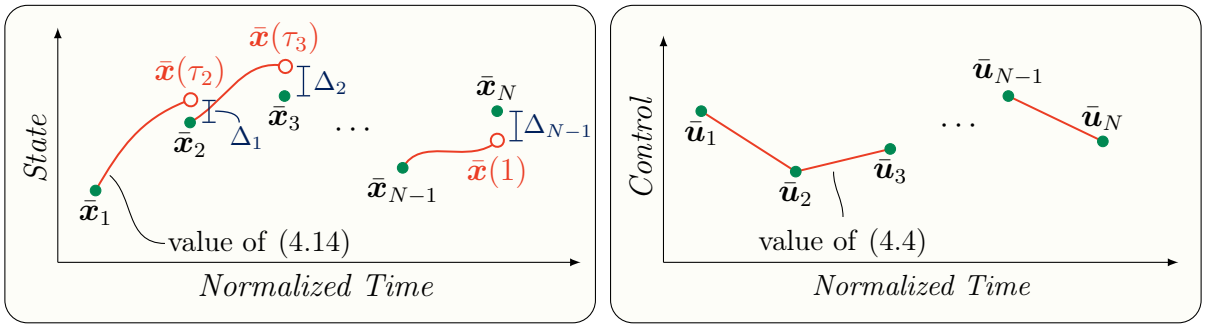


Figure 4.2: Illustration of the numerical integration procedure used to obtain the continuous-time reference state trajectory. The colors correspond to those in Figure 4.1 and are associated with the respective steps of the algorithm.

where $\boldsymbol{u}(\zeta)$ is obtained by using (4.4) and $\bar{\boldsymbol{x}}_k$ is the value of the reference solution at the k^{th} temporal node. Note that the expression (4.14) *resets* the reference state trajectory to evolve from $\bar{\boldsymbol{x}}_k$ over each interval $[\tau_k, \tau_{k+1}]$. The value from the preceding interval's integrated reference state is not used, and Figure 4.2 illustrates the difference. This strategy is analogous to a multiple shooting technique, and – similar to how that method exhibits improved convergence when compared to single shooting techniques – we have observed that (4.14) improves convergence (often significantly) by keeping $\bar{\boldsymbol{x}}(\tau)$ closer to the discrete points $\{\bar{\boldsymbol{x}}_k\}_{k=1}^{N}$ of the reference trajectory.

For later use, we define the *defect* associated with the k^{th} normalized time interval to be

$$\Delta_k := \|\bar{\boldsymbol{x}}(\tau_{k+1}) - \bar{\boldsymbol{x}}_{k+1}\|_2, \quad k = 1, \ldots, N-1. \tag{4.15}$$

The defects are also shown in Figure 4.2, and capture the miss distance between the final value of (4.14) and the predicted value $\bar{\boldsymbol{x}}_{k+1}$ from the (discrete) reference solution at the same temporal node. Importantly, if the reference state, control, and parameter vectors satisfy the nonlinear dynamics (4.7), then the defects will all be zero. The defects can therefore provide a necessary and sufficient measure of dynamic feasibility.

Constraint Approximation

The three-step procedure outlined in the previous section (normalize, linearize, discretize) is sufficient to render the nonlinear differential equality constraints convex in the solution variables $\{\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}\}_{k=1}^N$. In this section, we treat the second source of nonconvexity, the algebraic nonconvex inequality constraints (4.1d) and the equality constraints (4.1e). Because the former equations are assumed (without loss of generality) to be inequality constraints, there is more flexibility in how we can approximate them with convex functions.

By reusing the normalized time variable τ, we use a two-step procedure to convexify the nonconvex constraints. First, we approximate each inequality constraint h_j with a convex function of the solution variable

$$\bar{h}_j(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, \bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{p}}) \leq 0, \quad j = 1, \ldots, n_{ncvx} \tag{4.16}$$

There are several choices for the $\bar{h}_j$: first-order Taylor series, second-order Taylor series with (possibly approximated) positive semidefinite Hessian, inner convex approximations, or any other convex function that locally approximates the nonconvex h_j. First-order approximations are guaranteed to generate convex subproblems, and are computationally inexpensive to compute, relative to, at times, second-order approximations. Inner convex approximations can offer stronger theoretical guarantees to each subproblem (e.g., each subproblem's solution is guaranteed to be feasible with respect to the original nonconvex constraints) [145, 146]. Which approximation is used should be viewed as constraint-dependent.

The process of transcription for the algebraic nonconvex constraints is more straightforward than for their differential counterparts. The standard practice for the nonconvex inequality constraints is to enforce the convex approximations (4.16) at the temporal nodes (4.5), thereby *approximating* feasibility with respect to the original problem's constraints. The discrete-time constraints are therefore

$$\bar{h}_{jk} \coloneqq \bar{h}_j(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}, \bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}) \leq 0, \quad j = 1, \ldots, n_{ncvx}, \quad k = 1, \ldots, N. \tag{4.17}$$

Remark 4.1. *State-triggered constraints are handled slightly differently from traditional algebraic inequality constraints when being approximated. Because STCs are not differentiable at the point where the trigger function equals zero, due to* $\sigma^*(\cdot) = -\min\big(g(\cdot), 0\big)$*, we must be careful when computing any Jacobian-based approximations. To preserve the intended logical implication of the STC* (3.68) *and to ensure that the constraint condition is not enforced when* $g(\cdot) = 0$*, we define the approximation to be*

$$\bar{h}_{jk} = \begin{cases} \bar{h}_j(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}, \bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}), & \text{if } g(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}) < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4.18}$$

for each $k = 1, \ldots, N$*. When computing a Jacobian, this becomes equivalent to choosing the particular subgradient that is consistent with the logical implication of the STC.*

By also enforcing the convex functions g_i only at the temporal nodes, the only type of constraints that can be theoretically guaranteed to hold at all intermediate times are those functions g_i that depend only on the control vector – and this is a consequence of (4.4).

The nonconvex equality constraints given by the boundary conditions (4.1e) must again be made to be affine functions of the solution variable. We therefore approximate them with[2]

$$A_0\boldsymbol{x}_1 + E_0\boldsymbol{p} + \boldsymbol{r}_0 = 0, \quad \text{and} \quad A_N\boldsymbol{x}_N + E_N\boldsymbol{p} + \boldsymbol{r}_N = 0 \tag{4.19}$$

where,

$$A_0 = \nabla_x S_0\big(\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{p}}\big), \quad E_0 = \nabla_p S_0\big(\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{p}}\big), \quad \boldsymbol{r}_0 = S_0\big(\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{p}}\big) - A_0\bar{\boldsymbol{x}}_1 - E_0\bar{\boldsymbol{p}}, \tag{4.20a}$$

$$A_N = \nabla_x S_f\big(\bar{\boldsymbol{x}}_N, \bar{\boldsymbol{p}}\big), \quad E_N = \nabla_p S_f\big(\bar{\boldsymbol{x}}_N, \bar{\boldsymbol{p}}\big), \quad \boldsymbol{r}_N = S_f\big(\bar{\boldsymbol{x}}_N, \bar{\boldsymbol{p}}\big) - A_N\bar{\boldsymbol{x}}_N - E_N\bar{\boldsymbol{p}}. \tag{4.20b}$$

[2]The perhaps unsightly notation that associates A_0 with $\boldsymbol{x}_1$ is motivated by the fact that the first state, $\boldsymbol{x}_1$, corresponds to the linearization computed at normalized time $0 = \tau_1$. Moreover, A_1, E_1 and $\boldsymbol{r}_1$ are already used to represent the first set of discretization matrices from (4.13). The terminal boundary condition corresponds to the constraint at the N^{th} temporal node, and the subscripts match because there is no A_N, E_N or $\boldsymbol{r}_N$ computed by the discretization process, which runs only up to index $N-1$.

The boundary conditions are simply enforced at the appropriate normalized time.

As a final step, we can replace the continuous-time cost function $J\big(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}\big)$ in (4.2) with a discrete time approximation of the form

$$\bar{J}\big(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}\big) = M\big(\boldsymbol{x}(0), \boldsymbol{x}(1), \boldsymbol{p}\big) + \bar{L}\big(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}\big) \tag{4.21}$$

where $\bar{L}$ is a discrete-time representation of the integral cost in (4.2) obtained by a suitable quadrature rule.

By making the following replacements in Problem 7:

$$\text{(4.1a)} \leftarrow \text{(4.21)}, \quad \text{(4.1b)} \leftarrow \text{(4.12)}, \quad \text{(4.1d)} \leftarrow \text{(4.17)}, \quad \text{(4.1e)} \leftarrow \text{(4.19)}$$

the resulting problem is a parameter optimization problem that is convex with respect to each solution variable. In theory, one could solve this problem on a digital computer and use the solution to create an iterative algorithm. However, the transcription process can introduce two new problems, namely, artificial unboundedness and artificial infeasibility that may inhibit each subproblem from having non-empty feasible sets. We define and remedy these in the next section.

4.2.2 Parameter Update Step

Artificial Unboundedness: Trust Regions

Iterative algorithms that are based on the approximation of nonlinear or nonconvex terms are generally subject to a trust region constraint to ensure that the solution to each subproblem, or *iterate*, is chosen from a region where the approximation is valid. An independent issue is that the linearization of all nonconvex terms can lead to an unbounded solution of the subproblem. This phenomenon is referred to as *artificial unboundedness*.

Figure 4.3a shows a two-dimensional toy problem that exemplifies a single iteration of a generic SCP convergence process. In this example, the "original problem" consists of the

(nonconvex) parabolic equality constraint (shown in dark blue). If the nonconvex constraint is approximated with the dash-dot equality constraint, then we can see that without the trust region, an indefinite reduction in the cost function is possible. Moreover, if the new solution deviates too much from the reference $\bar{\boldsymbol{z}}$, then the dash-dot affine approximation of the parabola becomes quite poor.

To mitigate the issue of artificial unboundedness, we add a trust region constraint that limits how far each iterate, $\{\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}\}_{k=1}^N$, can deviate from the reference trajectory denoted by $\{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\}_{k=1}^N$. This is accomplished by adding the constraints

$$\alpha_x \|\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k\|_q + \alpha_u \|\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k\|_q \leq \eta_k, \quad k = 1, \ldots, N \tag{4.22a}$$

$$\alpha_p \|\boldsymbol{p} - \bar{\boldsymbol{p}}\|_q \leq \eta_p \tag{4.22b}$$

for some $q = 1, 2, 2^+, \infty$ and constants $\alpha_x, \alpha_u, \alpha_p \in \{0, 1\}$. We use $q = 2^+$ to denote the ability to impose the trust region as the quadratic two-norm squared. The trust region radii, $\boldsymbol{\eta} \in \mathbb{R}_+^N$ and $\eta_p \in \mathbb{R}_+$, are added as solution variables to each subproblem.

To encourage shrinking trust region radii, we augment the cost function (4.21) with

$$J_{tr} = \boldsymbol{w}_{tr}^\top \boldsymbol{\eta} + w_{tr,p} \eta_p \tag{4.23}$$

where $\boldsymbol{w}_{tr} \in \mathbb{R}_{++}^N$ and $w_{tr,p} \in \mathbb{R}_{++}$ are trust region weights. Note that in both cases, the added costs are effectively a weighted one-norm penalty on the trust region radii.

The trust region weights are typically fixed parameters that are defined by the user [78]. However, the weights can also be updated during the iterative convergence process to reflect the dynamic feasibility of the reference trajectory [73]. In particular, the defects computed

in (4.15) can be used to update the trust region weights from (4.22a) as

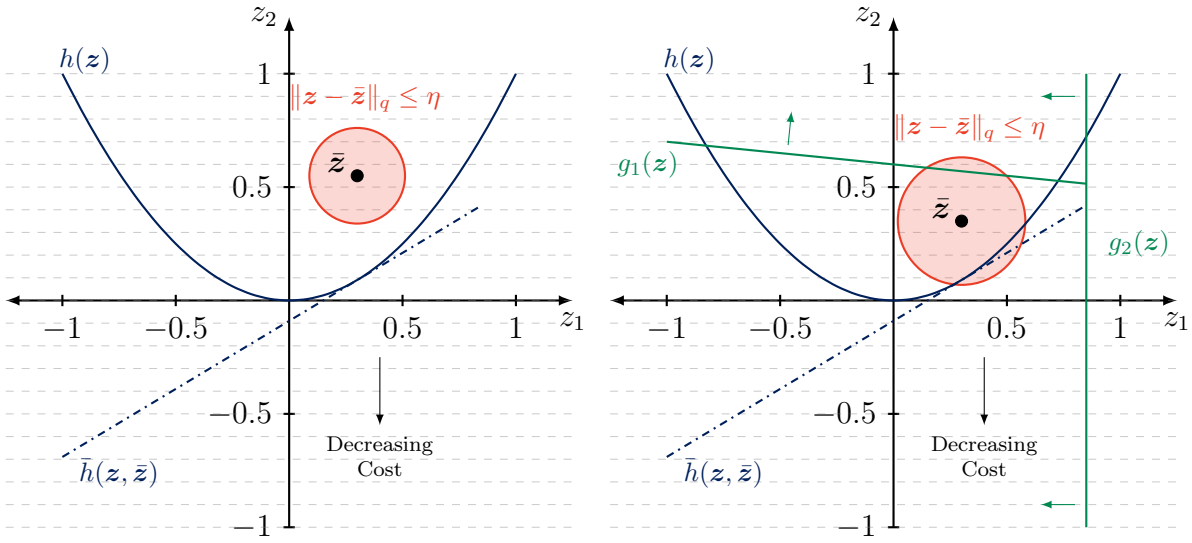
$$w_{tr,k} = \begin{cases} \frac{1}{|\Delta_k|}\,, & \texttt{update} \quad \& \quad |\Delta_k| \geq \Delta_{\min}, \\ \frac{1}{\Delta_{\min}}\,, & \texttt{update} \quad \& \quad |\Delta_k| < \Delta_{\min}, \\ w_{tr,k}\,, & \texttt{no update}, \end{cases} \tag{4.24}$$

for each $k = 1, \ldots, N$, where $\Delta_{\min}$ provides an upper bound on the weighting terms that helps to maintain proper scaling and avoid numerical issues when the defects become small. The trust region weight update induces the behavior that if the propagation step indicates a small defect at the k^{th} temporal node, then the cost of deviating from the reference trajectory's values at this node is increased. At the same time, a large defect implies that the algorithm is not yet close to convergence, and the trust region weights are lower to permit larger steps towards a feasible solution. This guides the solution process toward feasibility with respect to the nonlinear equations of motion while still permitting variations for the sake of constraint satisfaction and optimality. The weight $w_{tr,p}$ associated with the parameter trust region (4.22b) is not typically updated.

Artificial Infeasibility: Virtual Control

Even with the addition of a trust region, the resulting convex subproblem may not have a feasible solution. Formally, we refer to cases where subproblems become infeasible as *artificial infeasibility*, because it is a consequence of the approximations used to convexify the problem. That is, the original nonconvex problem has a non-empty feasible set, but the feasible set of the approximated problem is empty. Artificial infeasibility in sequential convex programming was recognized very early in the development of SCP algorithms [108, 111]. If artificial infeasibility is encountered at some iteration, the subproblem cannot be solved, and the optimization process would terminate without solution.

By allowing the trust region radii to be solution variables, there is only a single possible cause. (When the trust region is a fixed constant for each subproblem, then there can be a



(a) Artificial unboundedness.

(b) Artificial infeasibility.

Figure 4.3: An illustration of artificial unboundedness (by removal of the trust region) and of artificial infeasibility. Note that in (b), the "original problem" using the parabolic equality constraint has a non-empty feasible set, but the approximated problem using the dash-dot equality has an empty feasible set.

second independent cause, as shown in [147].) Mathematically, this is captured by:

$$\mathcal{F} = \big\{(\boldsymbol{x},\boldsymbol{u},\boldsymbol{p}) \,|\, (4.12),\ (4.16) \text{ and } (4.1c)\big\} = \emptyset. \tag{4.25}$$

In words, (4.25) says that the approximated constraints can be inconsistent with the original convex constraints, a scenario that is depicted in Figure 4.3b. Notice that the feasible set of the original problem is the portion of the parabola in the upper-right that satisfies both (green) convex inequalities. The approximated problem constructed from the dash-dot equality and two convex inequalities does not admit a feasible solution *regardless* of the trust region radius.

To counteract this issue of artificial infeasibility, we use so-called *virtual control* and add an unconstrained, but penalized, set of slack variables to the convexified constraints.

Specifically, (4.12), (4.16) and (4.19) are augmented to be[3]

$$\boldsymbol{x}_{k+1} = A_k\boldsymbol{x}_k + B_k^-\boldsymbol{u}_k + B_k^+\boldsymbol{u}_{k+1} + E_k\boldsymbol{p} + \boldsymbol{r}_k + \boldsymbol{\nu}_k, \qquad k = 1,\ldots,N-1 \quad (4.26a)$$

$$\bar{h}_{jk} - \nu'_k \leq 0, \quad \nu'_k \geq 0, \quad j = 1,\ldots,n_{ncvx}, \qquad k = 1,\ldots,N \quad (4.26b)$$

$$A_0\boldsymbol{x}_1 + E_0\boldsymbol{p} + \boldsymbol{r}_0 + \boldsymbol{\nu}_0 = 0, \quad \text{and} \quad A_N\boldsymbol{x}_N + E_N\boldsymbol{p} + \boldsymbol{r}_N + \boldsymbol{\nu}_N = 0, \quad (4.26c)$$

where $\{\boldsymbol{\nu}_k\}_{k=0}^N$ and $\boldsymbol{\nu}' \in \mathbb{R}^N$ are used to denote the virtual control terms that are added to each subproblem as solution variables.

To penalize the use of virtual control such that it is only used when necessary for constraint satisfaction, we augment the cost (4.21) with

$$J_{vc} = w_{vc}\left(\|\boldsymbol{\nu}'\|_1 + \sum_{k=0}^{N}\|\boldsymbol{\nu}_k\|_1\right) \quad (4.27)$$

where $w_{vc} \in \mathbb{R}_{++}$ is a large weighting term.

Because the virtual control is an artificial construct that we have added to the problem, it follows that the converged solution should have zero virtual control in order for it to be physically meaningful. Intuitively, a converged solution that uses non-zero virtual control is not a solution to the original optimal control problem.

4.2.3 Solve Step

We shall address the scaling of optimization variables in §5.1.1, and merely note here that for anything but fairly simple problems, proper scaling is necessary to obtain good solutions. This section summarizes the convex subproblem that is solved to full optimality during each solve step. Each subproblem is solved using an interior point algorithm [14, 15], but the specifics of the solver(s) are not discussed in this section.

---

[3]Note that the arguments of $\bar{h}_{jk} = \bar{h}_j(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}, \bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}})$ are suppressed here to keep the notation clean and focus on how the virtual control term enters into the expression.

For notational simplicity, we introduce the shorthands

$$\boldsymbol{X} \coloneqq \{\boldsymbol{x}_k\}_{k=1}^{N}, \quad \boldsymbol{U} \coloneqq \{\boldsymbol{u}_k\}_{k=1}^{N}, \quad \text{and} \quad \boldsymbol{V} \coloneqq \{\boldsymbol{\nu}_k\}_{k=0}^{N} \cup \{\boldsymbol{\nu}'\}.$$

The convex subproblem that is obtained after convexifying Problem 7 and introducing the trust region and virtual control modifications is summarized in Problem 8. In §4.3, we will specialize this generic problem statement to the particular case of 6-DOF powered descent.

Problem 8. *Find the vectors* $\{\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{p}, \boldsymbol{\eta}, \eta_p, \boldsymbol{V}\}$ *that solve the following parameter optimization problem:*

$$\min_{\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{p}, \boldsymbol{\eta}, \eta_p, \boldsymbol{V}} \quad \bar{J}\big(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{p}\big) + J_{tr} + J_{vc} \tag{4.28a}$$

$$s.t. \quad \boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k^- \boldsymbol{u}_k + B_k^+ \boldsymbol{u}_{k+1} + E_k \boldsymbol{p} + \boldsymbol{r}_k + \boldsymbol{\nu}_k, \qquad k \in \bar{\mathsf{N}} \tag{4.28b}$$

$$g_i\big(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}\big) \leq 0, \quad i = 1, \ldots, n_{cvx}\,, \qquad k \in \mathsf{N} \tag{4.28c}$$

$$\bar{h}_j(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}, \bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}) - \nu'_k \leq 0,\ \nu'_k \geq 0, \quad j = 1, \ldots, n_{ncvx}\,, \qquad k \in \mathsf{N} \tag{4.28d}$$

$$A_0 \boldsymbol{x}_1 + E_0 \boldsymbol{p} + \boldsymbol{r}_0 + \boldsymbol{\nu}_0 = 0, \quad A_N \boldsymbol{x}_N + E_N \boldsymbol{p} + \boldsymbol{r}_N + \boldsymbol{\nu}_N = 0, \tag{4.28e}$$

$$\alpha_x \|\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k\|_q + \alpha_u \|\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k\|_q \leq \eta_k, \qquad k \in \mathsf{N} \tag{4.28f}$$

$$\alpha_p \|\boldsymbol{p} - \bar{\boldsymbol{p}}\|_q \leq \eta_p, \tag{4.28g}$$

where $\mathsf{N} = \{1, \ldots, N\}$ *and* $\bar{\mathsf{N}} = \{1, \ldots, N-1\}$.

4.2.4 Initialization and Stopping Criteria

Sequential convex programming algorithms for trajectory optimization are good at obtaining locally optimal solutions regardless of the initial reference trajectory. This fact removes one of the more challenging issues with nonlinear programming: the need to find a good – and sometimes even feasible – initial guess of the solution. (Often for complex problems such as the 6-DOF powered descent guidance problem, finding a feasible solution can be just as hard as finding an optimal one.) Fortunately, we have found that the repeated solution of

the convex subproblem, Problem 8, is able to converge reliably using simple and infeasible initial guesses.

The simplest and most general strategy for computing an initial reference trajectory is *straight-line interpolation*. Using a user-specified guess for the initial and terminal states (if they are free variables), this strategy simply computes the initial values for $\bar{\boldsymbol{x}}_k$ by linearly interpolating between the boundary conditions. Formally, given some $\boldsymbol{x}_{ic}, \boldsymbol{x}_{fc} \in \mathbb{R}^{n_x}$ that can be assumed to be feasible with respect to (4.1e), we compute

$$\bar{\boldsymbol{x}}_k = \frac{N-k}{N-1}\boldsymbol{x}_{ic} + \frac{k-1}{N-1}\boldsymbol{x}_{fc}, \quad k \in \mathsf{N}. \tag{4.29}$$

Next, we select a similar set of control vectors. Whenever possible, we select these control vectors based on the physics of the problem. When the problem's structure does not admit an obvious choice for the initial control, then by choosing initial and final control vectors $\boldsymbol{u}_{ic}, \boldsymbol{u}_{fc} \in \mathbb{R}^{n_u}$ that are both feasible with respect to (4.1c) and (4.1d), the corresponding expression to (4.29) can be used to compute the initial reference control.

The guess of any parameters $\boldsymbol{p}$ can have a significant impact on the number of iterations required to obtain a solution. Because the parameters are inherently problem specific, it is unlikely that any rule of thumb will prove to be reliable.

For all but the simplest systems, the state, control, and parameter trajectories constructed by using the straight-line interpolation method are likely to be infeasible with respect to the equations of motion and/or the problem's constraints. Nonetheless, SCP algorithms are designed to permit infeasible reference trajectories through the inclusion of virtual control. This does not relieve the user entirely from carefully choosing the initial guess. A well chosen initial guess will influence both the time required to converge and possibly the global optimality of the solution that is reached. The latter is hard to check, but the former is measurable, often important, and indeed a driving objective behind much of the work in this dissertation.

Consequently, it should be understood that problem-specific initial guesses should be

used whenever possible. For example, the 6-DOF problem can be initialized using a solution of the 3-DOF problem [78]. The 3-DOF problem is solved to obtain trajectories for the mass, position and velocity states and the thrust (control) trajectory. An attitude trajectory can be computed such that the vertical axis of the vehicle is aligned with the thrust vector, and the angular velocity can be estimated by inverting the quaternion kinematics (3.48a). This procedure can provide a higher fidelity reference trajectory, but does not guarantee either dynamic feasibility or feasibility with respect to the constraints imposed in the 6-DOF problem.

Another important consideration for any iterative algorithm is the definition of "convergence" that is used to decide when to stop the iterations. Typically, this is achieved by comparing some notion of difference between the solutions of two subsequent iterates. By comparing the maximum deviation in the state solution between two iterates, we have found that an effective strategy is to terminate once the largest *scaled* difference is less than a prescribed tolerance. The differences must be scaled to ensure, for example, that we do not necessarily treat a 1 m deviation in position the same as a 1 rad/s change in angular velocity. Given some tolerance $\delta_{\text{tol}} \in \mathbb{R}_{++}$, the iterations are terminated whenever

$$\delta_{xp} \coloneqq \| S_p^{-1} \left(\boldsymbol{p} - \bar{\boldsymbol{p}} \right) \|_q + \max_{k \in \mathsf{N}} \| S_x^{-1} \left(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k \right) \|_q < \delta_{\text{tol}}, \tag{4.30}$$

where $q = 1, 2, 2^+, \infty$, and $S_x \in \mathbb{R}^{n_x \times n_x}$ and $S_p \in \mathbb{R}^{n_p \times n_p}$ are diagonal matrices that scale the state and parameter vectors appropriately – see §5.1.1 for details. We do not consider changes in the control vector because for typical space-vehicle problems, large changes in the thrust commands can cause negligible changes in the state trajectory. This would lead to an overly conservative definition of convergence and generally prohibit real-time implementations.

Remark 4.2. *The virtual control and trust region radii can also be an effective measure of convergence. Near-zero values for each of* $\|\boldsymbol{\eta}\|_1$, η_p, *and* J_{vc}/w_{vc} *can indicate that it is appropriate to terminate at the current iterate. However, different solvers have been observed to report different values for the trust region and virtual control terms for the same problem.*

Algorithm 1 The PTR algorithm designed to solve nonconvex optimal control problems.

Input: Initialize problem data and the algorithm parameters:
N, q, w_{vc}, $\boldsymbol{w}_{tr}$, $w_{tr,p}$, δ_{tol}, $\epsilon_{\text{feasible}}$ α_x, α_u, α_p, $\Delta_{\min}$, $\text{iter}_{\max}$

Compute initial guess: $\{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\}_{k=1}^{N}$
Compute scaling terms: $S_x, S_u, S_p, \boldsymbol{c}_x, \boldsymbol{c}_u, \boldsymbol{c}_p$
Convexify along initial guess ▷ convexification step
for $\text{iter} = 1, \ldots, \text{iter}_{\max}$ **do**
    Solve the convex subproblem, Problem 8 ▷ solve step
    Update reference trajectory $\{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\}_{k=1}^{N}$
    Convexify along current reference trajectory ▷ convexification step
        Determine feasibility of reference trajectory
    Update algorithm weights using (4.24) ▷ parameter update step
    Check stopping criteria (4.30)
end for

Output: If converged, solution satisfies the nonlinear continuous-time equations of motion to within a tolerance on the order of $\epsilon_{\text{feasible}}$, satisfies all algebraic constraints at each temporal node, and approximates (local) optimality of the original optimal control problem.

In addition, the tolerance(s) that define "near-zero" are more problem dependent than δ_{tol}. The state-based condition (4.30) *has been observed to be uniformly applicable across various solvers.*

The PTR algorithm is summarized in Algorithm 1. The arrangement of the convexification, parameter update, and solve steps in Algorithm 1 corresponds to an efficient implementation strategy, and should be compared to the arrangement Figure 4.1.

4.3 Case Studies in 6-DOF Powered Descent Guidance

This section provides two numerical case studies that highlight the capabilities of the SCP algorithm described in §4.2. First, a Monte Carlo analysis is performed to study the algorithm's ability to solve Problem 6 reliably across a variety of initial conditions. Second, we compare a solution of Problem 6 to a solution of the same problem but with a slant-range-triggered LOS constraint. We model this constraint as a compound state-triggered

Table 4.1: Parameters used for the two SCP case studies in 6-DOF powered descent guidance.

Parameter	Value	Units	Parameter	Value	Units
$\boldsymbol{g}_{\mathcal{L}}$	$-1.62\boldsymbol{z}_{\mathcal{L}}$	$\mathrm{m/s^2}$	J	$\mathbf{diag}\,\{13600, 13600, 19150\}$	$\mathrm{kg\,m^2}$
m_0	3250	kg	m_{dry}	2100	kg
$\boldsymbol{r}_{\mathcal{L},ic}$	$\big(250, 0, 433\big)$	m	$\boldsymbol{v}_{\mathcal{L},ic}$	$\big(-30, 0, -15\big)$	m/s
$\boldsymbol{r}_{\mathcal{L},f}$	$\big(0, 0, 30\big)$	m	$\boldsymbol{v}_{\mathcal{L},f}$	$\big(0, 0, -1\big)$	m/s
$\boldsymbol{r}_{u,\mathcal{B}}$	$-0.25 \cdot \boldsymbol{z}_{\mathcal{B}}$	m	$\boldsymbol{\omega}_{\mathcal{B},ic}$, $\boldsymbol{\omega}_{\mathcal{B},f}$	$\big(0, 0, 0\big)$	deg/s
I_{sp}	225.0	s	$\boldsymbol{q}_f$	$\big(0, 0, 0, 1\big)$	-
$\theta_{\max}$	80.0	deg	$\omega_{\max}$	28.6	deg/s
$\gamma_{\max}$	80.0	deg	$\delta_{\max}$	20.0	deg
$u_{\min}$	6000	N	$u_{\max}$	$22,500$	N
$\dot{\delta}_{\max}$	n/a	rad/s	$\dot{u}_{z,\max}$	n/a	N/s

constraint, and it is depicted in Figure 3.9.

The algorithm was implemented in C++ and was called using a MEX interface to Matlab®. We use the BSOCP solver to solve each convex subproblem [15]. For both case studies, we use the same set of nominal vehicle parameters outlined in Table 4.1. These parameters are derived from a lunar landing scenario, and therefore no atmospheric effects are considered. All examples presented in this section use the algorithm weights $w_{vc} = 10^4$, $\boldsymbol{w}_{tr} = 10^{-1} \cdot \mathbf{1}_N$, and $\Delta_{\min} = 10^{-3}$. In Chapter 5, we will study the sensitivities of the PTR algorithm to changes in these weights. To foreshadow those results, in general the computed trajectories are insensitive to changes in these weights within a relatively large range [125]. Lastly, the results in both case studies were obtained by using the straight-line interpolation method for the initial guess, with the identity quaternion and zero angular rates, as well as a thrust vector equal in magnitude to the product of the (interpolated) mass and gravity vector, projected into the feasible set.

4.3.1 Monte Carlo Analysis of Solutions to Problem 6

The objective of this first case study is to investigate the ability of the PTR algorithm to successfully compute trajectories given a wide range of initial conditions. At the same time,

we will introduce the idea of *real-time* solution capabilities that will become the focus of the next chapter.

The optimal trajectory – and hence algorithm performance – depends solely on the initial conditions of the problem for a given solver, transcription method, initial trajectory guess method, set of algorithm weights, terminal conditions, and temporal density N. We therefore investigate perturbations from the nominal initial conditions provided in Table 4.1 of sufficient size to test the algorithm's capabilities. The term *trial* is used to refer to the converged solution for a single initial condition (i.e., each run of the Monte Carlo study is one trial).

Because the attitude is optimized as part of the solution, we do not consider perturbations to $\boldsymbol{q}(t_0)$, and the initial guess for $\boldsymbol{q}(t_0)$ is always computed using the same method. The initial angular velocity is also kept constant for each trial because we can assume that it may always be controlled to near-zero during the preceding descent stage. The initial mass, position, and velocity are different for each trial. The mass is assumed to vary according to a uniform distribution between 90% and 110% of its nominal value:

$$m(t_0) = m_{ic}(1 + \delta m), \quad \text{where} \quad \delta m \sim \mathcal{U}[-0.1, 0.1]. \tag{4.31}$$

That is, δm is a uniform random variable that varies in the (inclusive) interval $[-0.1, 0.1]$. Next, we use the method described in [139] to compute the initial position and velocity in the landing frame. The velocity is chosen first according to

$$\boldsymbol{v}_{\mathcal{L}}(t_0) = \boldsymbol{v}_{\mathcal{L},ic} + \delta\boldsymbol{v}_{\mathcal{L}}, \tag{4.32}$$

where $\delta\boldsymbol{v}_{\mathcal{L}}$ is assumed to come from a zero-mean normal distribution with independent standard deviations of 7 m/s in the horizontal ($\boldsymbol{x}_{\mathcal{L}}$-$\boldsymbol{y}_{\mathcal{L}}$) directions and 4 m/s in the vertical ($\boldsymbol{z}_{\mathcal{L}}$) direction. The initial position $\boldsymbol{r}_{\mathcal{L}}(t_0)$ is then generated via hit-and-run sampling of the constrained controllability polytope generated for the 3-DOF problem [148, 149]. This

process is equivalent to sampling a random position from the feasible set of the 3-DOF problem (see Problem 2) constructed using the initial mass (4.31), the initial velocity (4.32), and the same problem data and thrust limits (3.62).

To discuss the results, we introduce a definition of *open-loop error* that will serve as a metric to discuss the dynamic feasibility of the computed trajectories. The open-loop error is the normed difference between the desired final state given in Table 4.1 and the value of the final state obtained after numerically integrating the computed control solution through the nonlinear equations of motion. When computing open-loop error, we *do not* reset the integration process as was done during the iteration process to compute the defects in (see (4.14)). Formally, we can express the open-loop error in position and velocity by

$$\varepsilon_r \coloneqq \|\boldsymbol{r}_{\mathcal{L}}(t_f) - \boldsymbol{r}_{\mathcal{L},f}\|_2 \quad \text{and} \quad \varepsilon_v \coloneqq \|\boldsymbol{v}_{\mathcal{L}}(t_f) - \boldsymbol{v}_{\mathcal{L},f}\|_2, \tag{4.33}$$

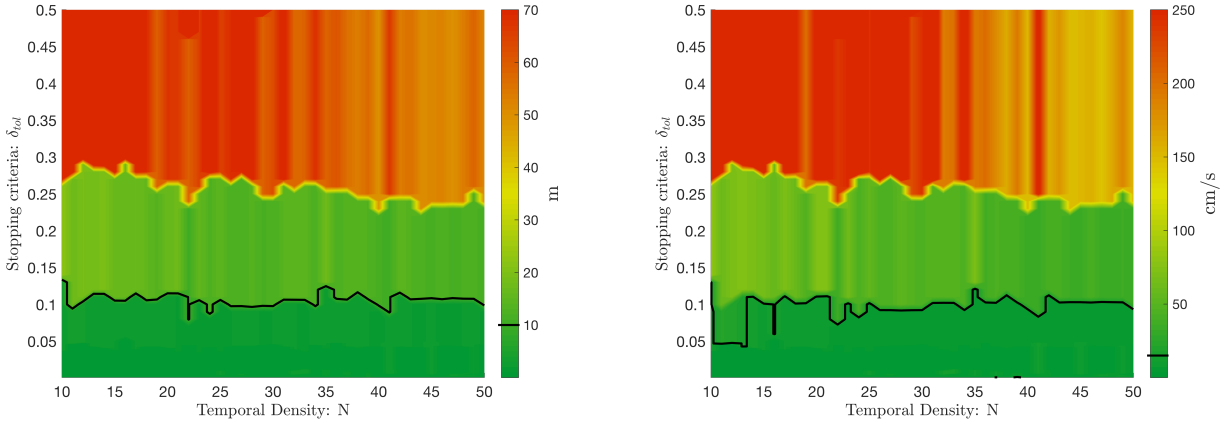
where $\boldsymbol{r}_{\mathcal{L}}(t_f)$ and $\boldsymbol{v}_{\mathcal{L}}(t_f)$ are the position and velocity components of the numerically integrated state vector.

We first investigate the relationship between convergence tolerance δ_{tol}, the temporal density N, and the open-loop errors (4.33). Of course, in the limit as $N \to \infty$ and $\delta_{\text{tol}} \to 0$, the open-loop error is expected to tend to zero at the expense of computation time. We are interested in how the open-loop error changes as a function of these two independent variables across ranges that may be suitable for subsequent real-time implementation. Given a desired set of open-loop error requirements for trajectory generation (allocated as part of a larger GNC error budget), we can use these relationships to choose appropriate values for N and δ_{tol}.

Suppose that the open-loop error tolerances in (4.33) are given values of

$$\varepsilon_r = 10 \text{ m}, \quad \text{and} \quad \varepsilon_v = 15 \text{ cm/s}. \tag{4.34}$$

Figure 4.4 shows the open-loop errors (4.33) as functions of N and δ_{tol} for Problem 6. These



(a) Open-loop position error

(b) Open-loop velocity error

Figure 4.4: Open-loop errors as functions of temporal density N and the stopping criteria δ_{tol} for Problem 6.

plots were generated for each N between 10 and 50, whereas δ_{tol} was varied in step sizes of 10^{-3} from 0.001 to 0.05, and then in step sizes of 10^{-2} from 0.06 to 0.5. The desired open-loop bounds are shown as the solid black lines, and can be seen to be achieved by a consistent choice of stopping criteria δ_{tol} across all temporal densities N, except for the velocity bound for small values of N that necessitate a lower δ_{tol} to achieve the desired open-loop error bound.

Because the size of each subproblem (4.28) depends strongly on N, it is prudent to select the smallest value of N that is able to achieve good results. While a precise choice requires more in-depth analysis of the problem at hand (and particularly of the constraints), the remainder of this Monte Carlo case study uses $N = 10$. To ensure open-loop errors that satisfy (4.34), we select $\delta_{\text{tol}} = 0.01$ as a conservative estimate based on Figure 4.4.

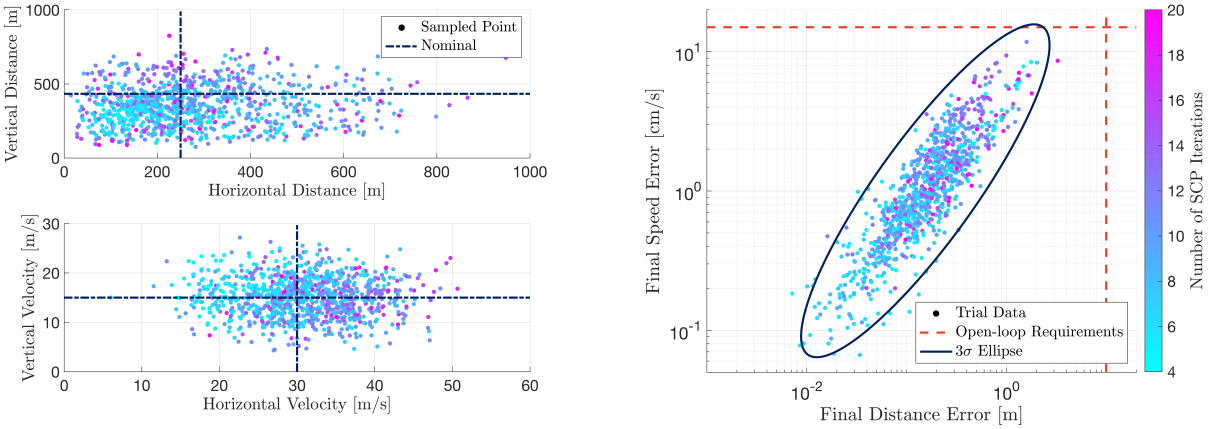
Dispersed Performance

A total of 1000 trials were run. If a trial met the open-loop requirements in (4.34), it was labeled as "success"; otherwise, it was labeled as "failed".

Of the 1000 trials, 971 were labeled as a success. Given that the straight-line interpolation method was used for the initial guess, it is not overly surprising that roughly 3% of initial conditions did not achieve the desired error bounds. We reiterate that a failure case does not mean the algorithm failed to converge, but merely that the trajectory to which it converged did not satisfy the desired open-loop error bounds (4.34). All trials resulted in a converged result in the sense of the stopping criteria δ_{tol}.[4] By taking the 29 failed trajectories and re-solving each of them using the 3-DOF initial guess method, success was achieved in 28 of these previously failed cases. This confirms that a higher fidelity initial guess can lead to more accurate results. A perfect success rate for the 3-DOF initialized problem was reported in [139], albeit for an implementation with a more robust solver, SDPT3, than the custom C++-based solver that was used for these trials.

Figure 4.5a shows the dispersed initial positions and velocities computed using our sampling method. This figure shows that the initial positions are selected from a set that is somewhat skewed towards the landing site but does include challenging initial conditions with nearly 1 km of initial downrange distance (nearly 4x the nominal downrange distance). Figure 4.5b shows the final open-loop error in position and velocity for each successful trial. Analysis of the data shows that the open-loop position and velocity errors follow a lognormal distribution, and therefore we compute statistics on the (natural) logarithm of the the open-loop errors and can use these to discuss confidence intervals. Figure 4.5b shows the three-standard-deviation (3σ) confidence ellipsoid compared to the open-loop error requirements (4.34). The figure reveals that the velocity and position errors are positively correlated, which conforms to intuition, and reveals a slight overlap with the velocity requirement in the upper right-hand corner. The trial data are color-coded based on the number of SCP iterations required to solve the problem, and we note that the iterations were stopped after a maximum of 20 for these trials. At least two trends are immediately evident: trials that end after a few SCP iterations tend to have lower open-loop errors, and lower initial horizontal

[4]The stopping criteria δ_{tol} required to achieve certain open-loop errors for a given temporal density will change for different nominal conditions. Figure 4.4 pertains only to the stated nominal conditions.



(a) Initial condition dispersions with respect to nominal (see colorbar to right).

(b) Computed open-loop errors.

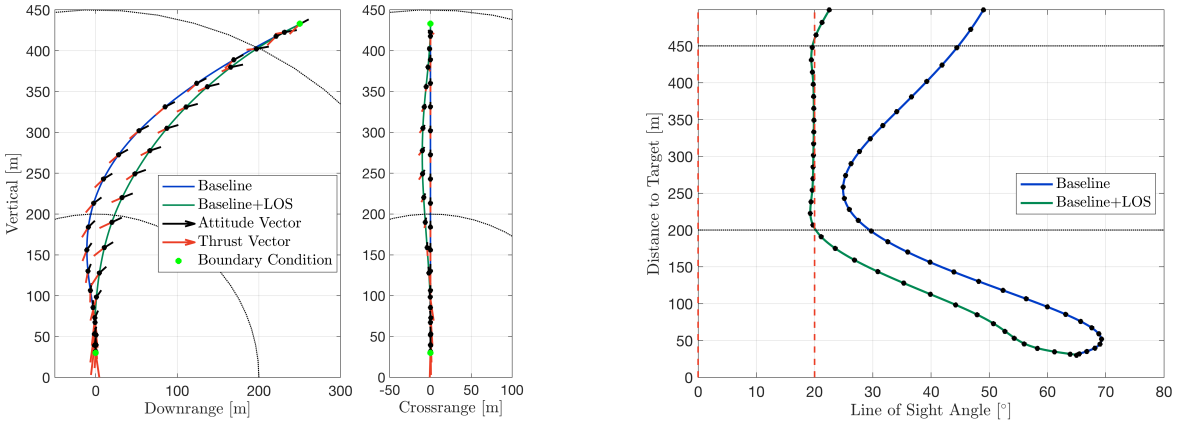
Figure 4.5: Position and velocity values at the beginning and end of each successful trial.

velocity components tend to lead to fewer SCP iterations.

4.3.2 Slant-Range-Triggered Line of Sight Case Study

The second case study examines how inclusion of the slant-range-triggered LOS constraint shown in Figure 3.9 affects the solution of the baseline problem. In addition to the parameters in Table 4.1, we assume in this section that an optical sensor has a boresight vector with coordinates $\boldsymbol{p}_{\mathcal{B}} = (0.91, 0, -0.42)$ in the body frame, and assume that $\boldsymbol{d}_{\mathcal{B}} = 0$ in (3.77). Given current sensor technology under development for future missions, we assume a field of view angle of $\xi_{\max} = 20$ deg [68]. We impose the LOS constraint whenever the vehicle's slant range lies between $\rho_{\min} = 200$ m and $\rho_{\max} = 450$ m.

In the figures, Problem 6 is referred to as the *Baseline* problem, whereas *Baseline+LOS* is used to denote the case where the slant-range-triggered LOS constraint is also imposed. In each figure, the black dots represent the discrete solution obtained from the final solve step, whereas the solid curves are the result of numerically integrating the control solution through the nonlinear equations of motion. Figure 4.6a shows the resulting trajectories using



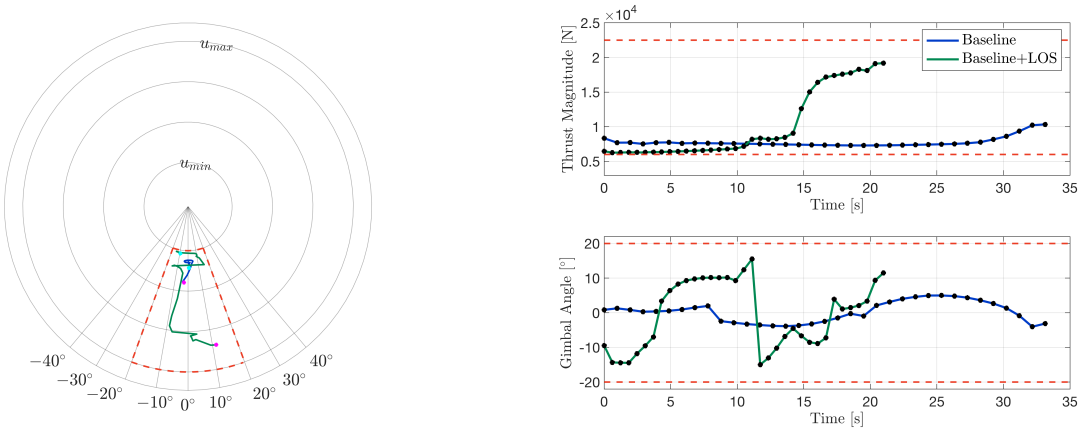
(a) Descent profiles in the landing frame

(b) LOS angle versus slant range

Figure 4.6: Converged trajectories for Problem (6) with and without the slant-range-triggered LOS constraint. Only half of the discrete points are shown on the left-hand axes for clarity.

a temporal density of $N = 35$. Both solid curves pass through each discrete point, indicating that the final convexification step has captured the nonlinear dynamics of the problem to sufficient accuracy. Figure 4.6b shows the LOS angle versus slant range, and reveals that the sensor cannot view the landing site throughout the entire baseline maneuver. In contrast, the inclusion of the STC leads to a trajectory that satisfies the LOS constraint between 450 m and 200 m as desired. A non-negligible deviation from the baseline trajectory is required to ensure feasibility with respect to this constraint. Note that the STC induces some crossrange motion that is not observed during the otherwise planar baseline maneuver.

The corresponding thrust curves can be seen in Figure 4.7. Figure 4.7a provides a polar plot of the thrust magnitude versus the gimbal angle, whereas Figure 4.7b shows the individual time histories of these two quantities. When the STC is imposed, the computed solution uses near-minimum thrust at the beginning of the maneuver while using the available gimbal to maintain a feasible attitude with respect to the LOS pointing constraint. As soon as the STC becomes inactive (i.e., the trigger condition no longer evaluates to true), there is a small divert to achieve a successful landing. This divert can be seen by the rapid change in gimbal



(a) Polar plot of thrust magnitude versus gimbal angle

(b) Time histories of thrust magnitude and gimbal angle

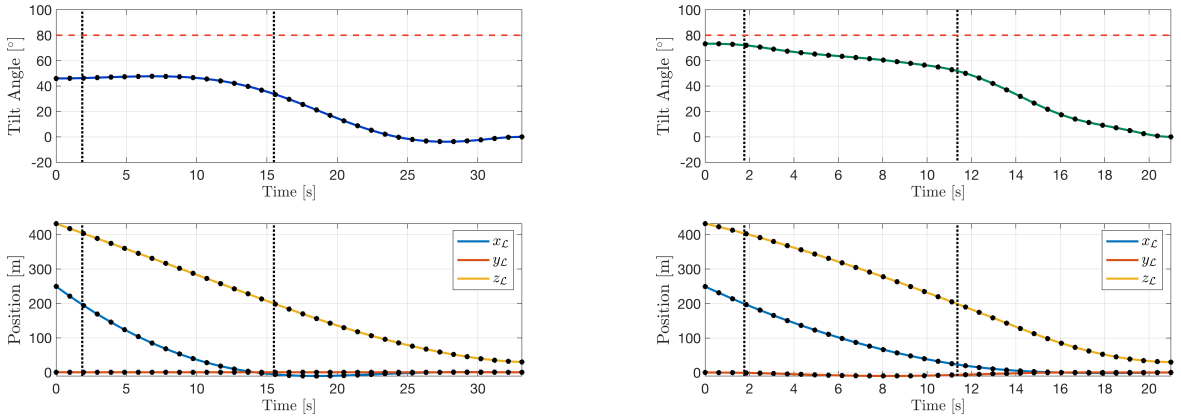
Figure 4.7: Control constraint trajectories for the converged solution to Problem 6 with and without the slant-range-triggered LOS constraint. On the polar plot, the cyan dot indicates the initial time, whereas the magenta dot represents the final time.

angle and throttle-up observed just after the 11 second mark.

Lastly, Figure 4.8 provides the time history of the vehicle's tilt angle and the landing frame position vector components. When the STC is imposed, the time spent further than 200 m from the landing site is reduced by several seconds, and the vehicle's tilt angle is generally larger. Interestingly, the total burn time for the baseline maneuver is much higher than for the same maneuver but with the slant-range-triggered LOS constraint imposed.

4.3.3 Prelude to Real-Time Implementations

In each of the preceding case studies, we have purposefully avoided discussion of the total computation time required to solve each problem instance. Chapter 5 deals with a standardized methodology for producing efficient implementations of SCP algorithms in an compiled programming language such as C or C++. Prior to having outlined those methods, it is not very informative to present detailed timing results, other than to say that there exists an implementation of the algorithm that achieves certain runtimes.



(a) Problem 6

(b) Problem 6 with slant-range-triggered LOS constraint

Figure 4.8: Vehicle tilt and landing-frame position time histories for the converged solution to Problem 6 with and without the slant-range-triggered LOS constraint.

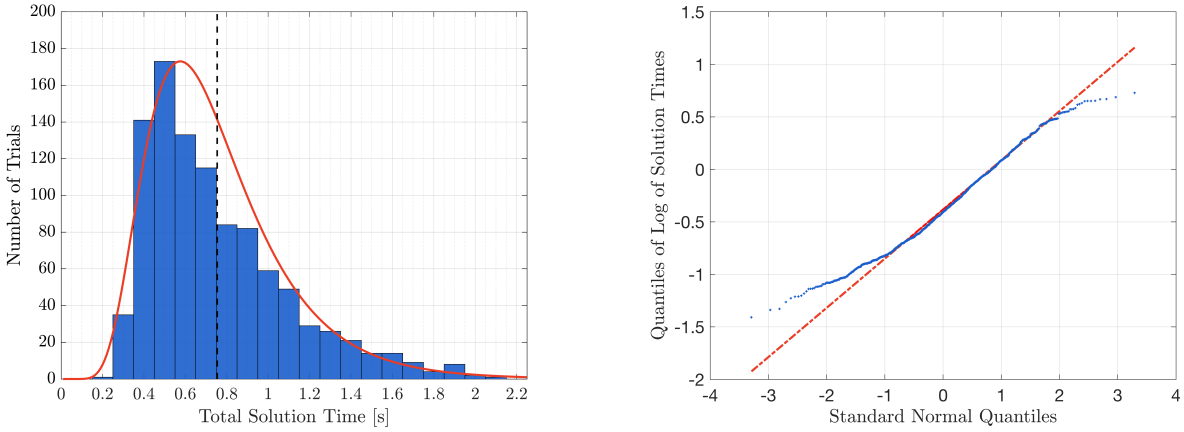
The runtime of a function can be loosely defined as the time elapsed between when the function is called and when it returns. The runtime depends on the processor being used (and also on what else is being done by that processor), as well as the problem data, programming implementation, and many other factors. Therefore, runtime results for algorithms that are intended to be deployed on space flight hardware are merely approximations (and often lower bounds) before they are computed on the target computational hardware using representative input data. Nevertheless, some basic trend analysis is still valid on non-spaceflight hardware, and we complete this chapter with a brief presentation that broadly depicts the characteristics of the runtimes observed during the Monte Carlo case study in §4.3.1.

Because all problem data was fixed except for the initial mass, position, and velocity during the Monte Carlo case study, we can think of a function that takes in this data and returns a solution to Problem 6. The runtime of this function is therefore also called the solution time. Instead of looking at the full picture of runtime, we focus only on the convexification and solve steps (see §4.2.1 and §4.2.3). The remaining portions of the algorithm have a negligible runtime, and for this example were directly implemented in Matlab®, not

a compiled language. The results were all generated on a 2015 iMac with a 3.2 GHz Intel Core i5 processor with 8 GB of RAM.

Likely the most important characteristics of the runtime for safety-critical spaceflight applications are the maximum value and the statistical distribution of all runtimes. Figure 4.9a shows the distribution of the total solution time across all successful Monte Carlo trials from §4.3.1. The solution time follows a lognormal distribution, and the solid red curve represents the fitted lognormal distribution. The black dashed line indicates the mean solution time computed, and has a value of roughly 0.75 s. The upper 3σ bound for this distribution was found to be roughly 2.2 s from a one-tailed analysis – which is also an upper bound on the observed data. To assess the validity of the lognormal distribution's fit to the data, the quantile-quantile (Q-Q) plot in Figure 4.9b was used. The S-shape of the Q-Q plot indicates *platykurtic* behavior – an indication that compared to a true (log)normal distribution, the data produces fewer and less extreme outliers. This behavior is desirable for spaceflight algorithms because it reduces the probability that excessively long solve times will be observed.

As a final comment, the correlation between total solve time and open-loop error is captured almost exactly by Figure 4.5. The solve time is dominated by the number of SCP iterations required to obtain a solution, and so the scale of the colorbar in Figure 4.5 could be equivalently replaced with the maximum solve time (magenta) and minimum solve time (cyan).



(a) Distribution of solution time across each Monte Carlo trial

(b) Q-Q plot of the log of the solution time

Figure 4.9: Trends in the SCP algorithm's runtime (solution time) for all successful trials from the Monte Carlo case study.

Chapter 5

EXPLICIT TRAJECTORY OPTIMIZATION:
REAL-TIME IMPLEMENTATIONS

If everything seems under control, you're not going fast enough
– Mario Andretti

The motivation underlying the development of algorithms that are capable of solving powered descent guidance problems is to enable future space missions that have a strong need for autonomous precision landing. Future robotic landers will have to navigate challenging environments such as hazardous, sloped terrain and jagged blades of ice in order to reach the most scientifically interesting locations [150, 151]. To support hazard avoidance and real-time intelligent re-targeting, algorithms must have predictable convergence behavior and be real-time capable on computationally constrained hardware. Moreover, the algorithms must be capable of handling a large suite of input and state constraints, and in particular may need to account for pointing requirements imposed by new sensing and navigation architectures. Consequently, trajectory planning must at least consider the 6-DOF rigid body rotation and translation of the vehicle. Lastly, human missions are likely to be preceded by cargo missions that will require landings to occur in close proximity [68]. Algorithms must therefore function fully autonomously. Even for currently-planned human missions to the Moon, the targeting of the lunar south pole creates extreme light-dark lighting conditions that may make piloted landings difficult and further necessitate autonomous precision landing capabilities [152, 153].

Chapters 3 and 4 were devoted primarily to developing techniques that address each of these requirements: modeling input and state constraints and using them to pose a 6-DOF powered descent guidance problem; developing an algorithm based on sequential convex programming; and demonstrating the algorithm's ability to solve the 6-DOF problem reliably.

In this chapter, we present further developments that are designed to facilitate computationally efficient implementations of said algorithm and explore its ability to solve problems fast enough to be practical for space missions.

Computational efficiency is a primary driver for optimization-based algorithm design for space systems – ultimately, algorithms need to be implementable on spaceflight hardware while adhering to the standards of safety-critical flight code.[1] This chapter presents a standardized approach to design efficient implementations of the SCP algorithm presented in Section 4.2. Because spaceflight hardware is often computationally limited relative to the computers used to design and prototype SCP algorithms, the memory usage (e.g., RAM) and total solve time become paramount – and can be the difference between an algorithm that is ready ascend the technology readiness level ladder and one that must wait for Moore's law to run its course.

Historical Context

Interestingly, the very early work of Meditch on the 1-DOF vertical descent problem did lead to approximately optimal solutions that could be computed "easily" (i.e., without the use of an onboard optimization solver) [41]. This was because the switching condition for the thrust magnitude could be approximated as a function of the vehicle's state, and subsequently used to throttle the main engine between a lower and upper bound. The same technique does not generalize to the 3-DOF problem, and even though the necessary conditions of optimality were understood in the 1960s [40], the computational resources required to solve the problem were not yet readily available. Using modern software, Topcu, Casoliva and Mease compared the attainable solution quality for the propellant-optimal 3-DOF problem to the necessary conditions of optimality derived by Lawden [49, 50]. Their focus was not to develop a real-time algorithm, as they used nonlinear programming methods that do not meet any the previously mentioned computational requirements. A real-time capable algorithm that solves

[1] See for example NASAs ongoing High Performance Spaceflight Computing project.

the 3-DOF necessary conditions using nonlinear root finding techniques has been developed by Lu [51, 54].

Over the past 40 years, research in numerical optimization has developed a family of interior point methods that can solve certain optimization problems to global optimality in polynomial time and with guaranteed convergence [8, 154]. These much needed properties for a real-time CGC algorithms stand in stark contrast to both shooting and nonlinear programming. The only limitation, roughly speaking, is that problems must be convex.

This is why the development of lossless convexification, originally for the 3-DOF landing problem, was an important step toward real-time capable powered descent algorithms. We have reviewed the literature and basic principle of lossless convexification for the 3-DOF problem in §3.2. In principle, all lossless convexification results are real-time implementable because they require solving one or a (small) bounded sequence of convex optimization problems, which are often second-order cone programs. Fast and efficient SOCP solvers that use only static memory allocation and adhere to the standards of spaceflight code have been developed [14, 15, 16].

Alas, a limitation inherent to any 3-DOF PDG algorithm (using lossless convexification or otherwise) is that the computed trajectory only respects translation dynamics and constraints related to either translation or the thrust vector. More importantly, 3-DOF methods cannot produce solutions that are guaranteed to respect sensing requirements such as LOS constraints for vehicle-mounted sensors. Even for missions where such constraints are "mild", an extensive simulation campaign is required to validate that the 3-DOF trajectories are feasible for a fundamentally 6-DOF lander system [34]. In the context of the aforementioned requirements for future missions where sensing and guidance are tightly coupled, the feasibility of using 3-DOF guidance solutions is called into question, and may require considerable backstage hand-tuning and edge-case handling efforts.

We argue that it is therefore well motivated to seek a real-time algorithm for 6-DOF landing problems that does not compromise the benefits of the current technology for 3-DOF landing problems (i.e., speed, guaranteed convergence, and feasibility), while simultaneously

providing the benefit of explicit feasibility with respect to the full 6-DOF dynamics and coupled translation-rotation (cf. guidance-navigation) constraints.

Sequential convex programming emerged as a very natural solution when transitioning from a fully convex 3-DOF problem to a 6-DOF problem in which some nonconvexity is present. The algorithm presented in §4.2 solves a sequence of convex problems, which are often SOCPs for aerospace applications. This permits the use of "heritage" SOCP solvers developed to solve the 3-DOF problem, and the balance of the algorithm design is used to ensure that the SCP iterations converge to a local optimum of the nonconvex 6-DOF problem. SCP algorithms can been used to handle nonconvexities such as aerodynamic lift and drag, thrust slew-rate constraints, free ignition time, state-triggered constraints, and the nonlinear 6-DOF equations of motion. Collectively, we may conclude that the constraint satisfaction requirements of future spaceflight missions can be handled by using SCP-based algorithms for the 6-DOF powered descent guidance problem. One notable omission is the theoretical guarantee of convergence to a feasible solution. No general purpose SCP algorithm can claim to guarantee a solution to nonconvex optimal control problems from an arbitrary initial guess – and the PTR algorithm is no exception. While convergence guarantees do exist for very similar algorithms, even these guarantees do not ensure (for example) that there will be no virtual control used in the converged solution [128, 131]. Practically speaking, however, SCP algorithms have been observed to provide very good convergence behavior across a wide range of problems.

What Is A Real-Time Implementation?

By real-time implementation, we mean code written in a compiled language (e.g., C, C++) that adheres to the standards of spaceflight code (see, for example, [155, 156]). For a general SCP algorithm, the implementation is assumed to encompass all of the steps described in §4.2. Typically, an implementation is formed by three components: a function that initializes required data and variables, a function that executes the algorithm using these data and variables, and a function that terminates the call to the algorithm and frees memory

as required. The modern paradigm is to use customized solvers, wherein problem structure is hard-coded into the solution process by an offline code generator in an attempt to remove the overhead required for a general purpose solver. Well-known examples include CVXGEN, FORCES [157, 158, 159, 160] and BSOCP [16, 161].

Ultimately, an SCP-based method requires the solution of a sequence of convex optimization problems which, in the case of most aerospace systems, are SOCPs. In assessing the real-time capability of our algorithm, it is therefore appropriate to consider existing work on the real-time solution of single-SOCP optimization problems. In the context of 3-DOF powered descent, a flight test campaign spanning seven flights over three years aboard the Masten Space Systems Xombie sounding rocket demonstrated that the onboard solution of an SOCP problem is feasible on spaceflight processors [67, 162]. The algorithm under test, G-FOLD, is the culmination of lossless convexification's application to the 3-DOF problem. G-FOLD was able to compute landing divert trajectories in 100 ms on a 1.4 GHz Intel Pentium M processor. Moreover, Dueri et al. have shown that the same algorithm can achieve runtimes of less than 700 ms on the radiation-hardened BAE RAD750 processor [15]. These values would be lower bounds for the runtimes of an SCP algorithm designed to solve the 6-DOF problem if the same hardware is used. It is anticipated, however, that the current NASA-led High Performance Spaceflight Computing project may provide a more powerful computing platform with which similar runtimes can be realized.

The real-time capability of the PTR algorithm for non-PDG applications has been tested on several occasions. Szmuk et al. have shown that the PTR algorithm is able to solve difficult quadrotor path planning problems (including obstacle avoidance and acrobatic flips) in less than 100 ms running on a 1.7 GHz Intel Atom processor of an embedded Intel Joule platform [163, 164]. More difficult tasks such as flying through hoops and cooperatively transporting a beam while avoiding obstacles were solved in less than one second [93]. These problems are fundamentally similar to the 3-DOF landing problem.

The majority of existing work on real-time optimization is found in the model predictive control (MPC) literature, where closed-loop feedback is achieved by (generally) calling an

optimizer as fast as the desired execution frequency of the controller. In this context, execution frequencies for linear and quadratic optimization problems in the kHz-MHz range are possible for non-spaceflight applications [165, 166]. MPC has also been used onboard autonomous race cars, achieving re-solve rates of 50 Hz for miniature cars on an embedded 1.7 GHz ARM A9 chip [167], and 20 Hz for a life-size car on a 2.1 GHz Intel i7-3612QE [168].

MPC strategies have been proposed for 6-DOF landing problems [71]. However, the MPC paradigm of constantly re-solving an optimization problem to provide feedback control decisions can pose constraint feasibility problems and is in conflict with the CGC philosophy that uses classical feedback control architectures instead. To provide the required 6-DOF powered descent capabilities while minimally impacting heritage guidance and control systems, we instead solve for a single, complete, trajectory from the spacecraft's current state all the way to a target final state. Given real-time computing capabilities, new trajectories can be found as-needed (e.g., during landing site re-targeting for hazard avoidance), but re-solving is not a fundamental component of a nominal maneuver. The objective of "real-time powered descent" is therefore that a *single solution* for a complete 6-DOF PDG trajectory is computable in a short enough time span on spaceflight hardware using an implementation that adheres to the standards of spaceflight code.

This chapter is organized as follows. A strategy for efficient real-time implementations is outlined in §5.1. We highlight the relationships between these implementation strategies and the algorithm design steps covered in Chapter 4. A case study using the planar powered descent guidance problem is then presented in §5.2 so that the real-time results can be compared against the known theoretical properties of optimal solutions.

5.1 Implementation Architecture

In Chapter 4, the original optimal control problem, Problem 7, was approximated by the convex subproblem, Problem 8, at each iteration of the PTR algorithm. Throughout this chapter, we make reference to "solver" and "PTR" iterations; the former are iterations performed internally by the solver, whereas the latter are the outer iterations around the PTR

algorithm's loop that is shown in Figure 4.1 and includes the convexification/parameter update/solve steps and stopping criteria check.

By using the output of the convexification step, we formulate an each subproblem as an SOCP in the general *conic form* of Problem 9.

Problem 9. *Find the vector* $\boldsymbol{z} \in \mathbb{R}^{n_z}$ *that solves the following second-order cone program:*

$$\min_{\boldsymbol{z}} \quad \boldsymbol{c}^\top \boldsymbol{z} \tag{5.1a}$$

$$s.t. \quad A\boldsymbol{z} = \boldsymbol{b} \tag{5.1b}$$

$$\boldsymbol{z} \in \mathcal{C}_L \times \mathcal{C}_{Q_1} \times \cdots \times \mathcal{C}_{Q_m} \tag{5.1c}$$

where $\mathcal{C}_L = \left\{\boldsymbol{w} \in \mathbb{R}^\ell \mid \boldsymbol{w} \geq 0\right\}$ *is a linear cone of dimension* ℓ*, and each*

$$\mathcal{C}_{Q_i} = \left\{(w_0, \boldsymbol{w}) \in \mathbb{R}^{d_i+1} \mid \|\boldsymbol{w}\|_2 \leq w_0\right\}$$

is a second-order cone of dimension $d_i + 1$*. The problem data* $A \in \mathbb{R}^{n_c \times n_z}$*,* $\boldsymbol{b} \in \mathbb{R}^{n_c}$ *and* $\boldsymbol{c} \in \mathbb{R}^{n_z}$ *are then passed to the solver in order to solve the subproblem.*

The process of translating Problem 8 into the form of Problem 9 is called *parsing*. The majority of all previous work on SCP algorithms has relied on "modeling" interfaces (often called parsers) that automate the parsing process, and allow a designer to rapidly prototype their code [19, 20]. However, these parsers, by their very nature, must introduce computational overhead, dynamically allocated memory, and sometimes redundant constraints in order to fulfill their intended purpose as general design tools. Each of these characteristics are undesirable for real-time implementations, and so we resort to "hand-parsing". Hand-parsing means to define explicitly the data $\{A, \boldsymbol{b}, \boldsymbol{c}, \ell, d_1, \ldots, d_m\}$ as functions of the problem's variables and constraints from Problem 8 (e.g., the specific indices of A that correspond to each constraint). Implementations that do not use some form of hand-parsing must construct the entire A, $\boldsymbol{b}$ and $\boldsymbol{c}$ matrices from scratch at each iteration and do not leverage the fact that the same optimization problem is being solved repeatedly.

5.1.1 Variable Initialization

This section details the initialization steps that are performed prior to executing the main PTR loop. The initialization step consists of three operations: generating the initial solution guess, computing the scaling matrices, and pre-parsing as much of the problem data as possible. If the initialization step is called at some time $t < t_0$, where t_0 is the time at which the guidance solution will begin to be executed by the vehicle, then the current vehicle state must be projected forward in time by an amount $t_0 - t$. The difference $t_0 - t$ must (at least) account for the time spent computing the guidance solution. For the aforementioned test flight campaign of the 3-DOF G-FOLD algorithm, this value was chosen to be one second [66].

As in the case study presented in §4.3.1, we use the straight-line interpolation initial guess solution to seed the real-time implementation. This initial guess method can be computed very quickly using only a small number of floating point operations. For the final time specifically, we have observed that initializing the parameter t_f to be higher than the expected flight time produces slightly better convergence results. This is similar to an observation made for the 3-DOF problem, where Dueri et al. found that as t_f approached the minimum feasible time of flight, more solver iterations were required to produce a solution [15]. As t_f was increased, the solver had an easier time find solutions, and we have observed a similar trend for planar and 6-DOF landing problems.

Scaling Matrices

In §4.2.3, we briefly mentioned that proper scaling of the solution variables of Problem 8 is critical to obtaining good solutions. The scaling of solution variables can be done in several ways, and there is no general consensus on the best way to scale optimal control problems to produce numerically well-conditioned parameter optimization problems. While some authors argue that scaling (and balancing) the continuous-time equations of motion is the most appropriate [51, 169], others argue that scaling the discrete-time optimization

variables is sufficient [21, 96]. We have found that the latter method is acceptable in our implementations, but do not claim superiority over the former method. To this end, we scale the solution variables of Problem 8 by defining the following affine transformations:

$$\boldsymbol{x}_k = S_x \hat{\boldsymbol{x}}_k + \boldsymbol{c}_x, \quad k \in \mathsf{N} \tag{5.2a}$$

$$\boldsymbol{u}_k = S_u \hat{\boldsymbol{u}}_k + \boldsymbol{c}_u, \quad k \in \mathsf{N} \tag{5.2b}$$

$$\boldsymbol{p} = S_p \hat{\boldsymbol{p}} + \boldsymbol{c}_p, \tag{5.2c}$$

where the S_x, S_u, S_p are diagonal matrices and $\boldsymbol{c}_x, \boldsymbol{c}_u, \boldsymbol{c}_p$ are vectors of commensurate dimension that scale and center the state, control, and parameter vectors respectively. Throughout this section, scaled quantities are referred to with the $\hat{\cdot}$ adornment. For the i^{th} component of the state, control, or parameter vectors, generically referred to using z_i, we can define two quantities: 1) the known range of the "true" component's value in physical units, $[z_{i,\min},\, z_{i,\max}]$, and 2) the interval that we wish to scale the component to, $[\hat{z}_{lb},\, \hat{z}_{ub}]$. Using this information, we use

$$S_{z,ii} = \frac{z_{i,\max} - z_{i,\min}}{\hat{z}_{ub} - \hat{z}_{lb}}, \quad \text{and} \quad \boldsymbol{c}_{z,i} = z_{i,\min} - S_{z,ii}\hat{z}_{lb}. \tag{5.3}$$

While the interval $[\hat{z}_{lb},\, \hat{z}_{ub}]$ is in theory arbitrary, a judicious choice with respect to parsing Problem 8 into the standard form of Problem 9 is to use $[\hat{z}_{lb},\, \hat{z}_{ub}] = [0, 1]$. This places the new (scaled) solution variables in the linear cone by construction, and eliminates the need to enforce the lower bound constraints $\hat{z}_{i,lb} = 0 \leq \hat{z}_i$ explicitly. The reduction in problem size, and corresponding runtime decrease, afforded by this choice can be significant and is quantified in Theorem 5.5.

Pre-Parsing

Pre-parsing consists of initializing the data $\{A, \boldsymbol{b}, \boldsymbol{c}, \ell, d_1, \ldots, d_m\}$ used in Problem 9 prior to entering the main PTR iterative loop. There is a significant amount of structure to the

nonconvex problems that are solved using any SCP method, and this structure is exploited to maximize the speed of the PTR algorithm by populating as many entries of the problem data as possible during the pre-parse step. For a fixed problem statement, the majority of the non-zero entries in these containers do not change across the PTR iterations, and the pre-parse step simply populates these constant non-zero entries.

For this step, an enumeration of the variables and constraints must be decided upon. A generic enumeration of the variable $\boldsymbol{z}$ and constraints is given in Figure 5.1. We define the block v_1 variables to be all those that appear explicitly in Problem 8 and/or contribute to the nonlinear equations of motion, block v_2 variables to be all linear slack variables added to write the problem in standard form, and block v_3 variables to be those used to form the trust region[2] in addition to any Second-Order Cone (SOC) slack variables added to write the problem in standard form. Similarly, we define block c_1 constraints to be those that represent the dynamics and boundary condition constraints, and define block c_2 constraints as all other constraints imposed.

These definitions permit a block row/column decomposition of the A, $\boldsymbol{b}$ and $\boldsymbol{c}$ matrices. For the PTR algorithm (and ones that are similar to it), the vector $\boldsymbol{c}$ can be fully populated during the pre-parse step, and only A and $\boldsymbol{b}$ will change across the PTR iterations. All block v_2 and v_3 slack variables added to write the convex approximation in the standard form of Problem 9 will have a corresponding entry in the A matrix of unit magnitude (± 1) that appears in the row corresponding to the constraint that the slack variable was added to. These are represented by the $\star_s$ (for linear slack variables) and the $\star_\chi^A$ (for SOC slack variables) blocks shown in Figure 5.1. Because all non-zero entries are ± 1, and occur in user-specified rows and columns, the entire $\star_s$ and $\star_\chi^A$ blocks can be populated during the pre-parse step and do not change throughout the iterations.

The $\star_{\dot{P}}$ blocks correspond to the equations of motion and boundary conditions, and can be partially populated during the pre-parsing step. The remaining portions can only be

[2]The trust region is often implemented as a second order cone, though this is not necessary. Hence block v_3 may contain both linear and second-order cone variables.

$$
\boldsymbol{z} = \begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{u}} \\ \hat{\boldsymbol{p}} \\ \boldsymbol{\nu}_+ \\ \boldsymbol{\nu}_- \end{bmatrix} \Big\} \text{ block } v_1 \quad \begin{bmatrix} \star_\phi \\ 0 \\ \star_\phi \\ 0 \\ 0 \end{bmatrix}
\begin{bmatrix} \boldsymbol{\nu}' \\ \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \vdots \\ \boldsymbol{s}_{m_l} \end{bmatrix} \Big\} \text{ block } v_2 \quad \begin{bmatrix} w_{vc}\,\mathbf{1} \\ w_{vc}\,\mathbf{1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}
\begin{bmatrix} \boldsymbol{\chi}_{\eta,p} \\ \boldsymbol{\chi}_\eta \\ \vdots \\ \boldsymbol{\chi}_{m_q} \end{bmatrix} \Big\} \text{ block } v_3 \quad \begin{bmatrix} w_{tr,p}\,\star_\chi^c \\ \boldsymbol{w}_{tr}\,\star_\chi^c \\ \vdots \\ 0 \end{bmatrix} = \boldsymbol{c}
$$

$$
A = \begin{array}{c} \begin{matrix} \text{block } v_1 & \text{block } v_2 & \text{block } v_3 \end{matrix} \\ \begin{bmatrix} \star_P^A & 0 & 0 \\ \star_C^A & \star_s & \star_\chi^A \end{bmatrix} \end{array} \begin{matrix} \Big\} \text{ block } c_1 \\ \Big\} \text{ block } c_2 \end{matrix} \quad \begin{matrix} \Big\{ \begin{bmatrix} \star_P^b \\ \star_C^b \end{bmatrix} = \boldsymbol{b} \end{matrix}
$$

$\star_P^{\cdot}$: computed during pre-parse & convexification
$\star_C^{\cdot}$: computed during pre-parse & convexification
$\star_s$: fully pre-parsed
$\star_\chi^{\cdot}$: fully pre-parsed
$\star_\phi$: fully pre-parsed

Figure 5.1: Generic enumeration of the variables and constraints for the standard form of a trajectory optimization problem. The constraint block c_1 corresponds to the dynamics and boundary conditions, while constraint block c_2 corresponds to all path constraints enforced as inequalities. Block v_1 variables are used either directly in Problem 8 or to impose the dynamics and boundary conditions. Block v_2 variables represent linear slack variables of arbitrary dimension, and there are m_l such variables. Block v_3 variables represent the trust region implementation and SOC variables of arbitrary dimension, and there are m_q such variables.

populated after the convexification step. The $\star_C^{\cdot}$ blocks correspond to the path constraints enforced as inequalities. These blocks can also only be partially populated during the pre-parse step, and the remaining portions may be added after the convexification step. The percentage of the entries in the $\star_P^{\cdot}$ and $\star_C^{\cdot}$ blocks that can be pre-populated is application dependent and will vary.

5.1.2 Convexification Step

The convexification step is responsible for approximating the nonconvex constraints that are part of the original optimal control problem, Problem 7. In implementation, this is

achieved by using the propagation and constraint approximation steps discussed in §4.2.1. To minimize the size of the matrices involved, and to avoid the use of 3D matrices to provide a temporal index, all matrix computations are performed using 1D "flattened" arrays. This also avoids passing around large 2D matrices that are mostly zeros [139]. The propagation step is summarized in Algorithm 2 and the constraint approximation step is summarized in Algorithm 3.

Propagation

The equations used to compute the discrete-time linear time varying approximation of the equations of motion are given by (4.13). In implementation, we must compute $N-1$ sets of these matrices, with each set representing the transition from one normalized temporal node to the next. Each set of matrices is computed by numerically integrating the integrands in (4.13) along with the state vector (4.14). We use a fixed-step Runge-Kutta-4 (RK4) integration scheme with N_{sub} points to do the numerical integration. The size of the vector being integrated is $n_x(n_x + 2n_u + n_p + 2)$. For each $k \in \bar{\mathsf{N}}$, we integrate the following differential equation over the interval $[\tau_k, \tau_{k+1}]$ using N_{sub} points:

$$\dot{P}(\tau) = \begin{bmatrix} F\big(P_x(\tau), \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big) \\ A(\tau)P_\Phi(\tau) \\ P_\Phi^{-1}(\tau)\lambda_-(\tau)B(\tau) \\ P_\Phi^{-1}(\tau)\lambda_+(\tau)B(\tau) \\ P_\Phi^{-1}(\tau)E(\tau) \\ P_\Phi^{-1}(\tau)\boldsymbol{r}(\tau) \end{bmatrix}, \quad P(\tau_k) = \begin{bmatrix} \bar{\boldsymbol{x}}_k \\ \text{flat}\,(I_{n_x}) \\ 0_{n_x n_u \times 1} \\ 0_{n_x n_u \times 1} \\ 0_{n_x n_p \times 1} \\ 0_{n_x \times 1} \end{bmatrix}, \quad \text{where} \quad P(\tau) = \begin{bmatrix} P_x(\tau) \\ P_\Phi(\tau) \\ P_{B^-}(\tau) \\ P_{B^+}(\tau) \\ P_E(\tau) \\ P_{\boldsymbol{r}}(\tau) \end{bmatrix}. \quad (5.4)$$

Here, the flat $(\cdot)$ operation maps a 2D array to a 1D array using a column major representation.[3] Note that the entire initial condition $P(\tau_k)$ is reset before each call to RK4, but only the entries corresponding to the state vector, $P_x(\tau_k)$, have a different value across the $N-1$

[3]In the code, no explicit conversions between 1D and 2D arrays are needed, and so the use of column major representation is not in conflict with the C/C++ row major array storage.

temporal intervals. This is the resetting strategy depicted in Figure 4.2 that is analogous to multiple-shooting.

The propagation step must be called a total of $N-1$ times per convexification step. For each $k \in \bar{\mathsf{N}}$, the propagation step can be performed in

$$\mathcal{O}\left(4N_{\text{sub}}\left(\frac{5}{3}n_x^3 + n_x^2(2n_u + n_p + 6)\right)\right)$$

floating point operations. Powered descent problems typically have $n_x \in [7, 15]$ and $n_u \in [2, 6]$ and $n_p \in [0, 2]$. As demonstrated in §5.2, the propagation step can be performed in a near-negligible amount of time if it is implemented properly.

Our results suggest that the PTR algorithm's convergence behavior and quality of the final solution is not strongly dependent on the choice of N_{sub}, and a value of $N_{\text{sub}} \in [5, 15]$ is typically sufficient for 6-DOF rigid body dynamics. To obtain the final values of the matrices in (4.13), we right-multiply each of $P_{B^-}(\tau_k)$ through $P_{\boldsymbol{r}}(\tau_k)$ by the value of $P_\Phi(\tau_k)$ for each $k \in \bar{\mathsf{N}}$. The final matrices in (4.13) can either be stored as flattened arrays and output in a data structure, or placed directly into the A and $\boldsymbol{b}$ matrices used to construct the standard form SOCP.

Remark 5.1. *One of the key steps that drives the computational complexity of the propagation step is the matrix inverse required to evaluate the derivative of $P_\Phi(\tau)$ in* (5.4)*. It is customary to use Gaussian elimination with partial pivoting to compute this inverse [170], a process that involves computing an LU decomposition and inverting the two factors. For powered descent problems that use Cartesian variables to represent the state vector and have the mass variable in the first position, we have observed that the entries below the diagonal in $P_\Phi(\tau)$ do not exceed* 1 *under the condition that the ratio $\Gamma(\tau)/m(\tau)^2 < 1$, where $\Gamma(\tau)$ is the thrust magnitude and $m(\tau)$ is the vehicle mass. Moreover, the equations of motion imply that the* $(1, 1)$ *entry in $P_\Phi(\tau)$ is constant and equal to* 1*. This means that partial pivoting may not be theoretically necessary to ensure numerical stability in this case. However, for powered descent problems stated using dual quaternions the same statement does not hold [73], and*

partial pivoting must be retained in this case. To maintain a general solution method between the two state representations, we use partial pivoting in our implementations.

One of the advantages of the propagation step is the ability to estimate the dynamic feasibility of the current reference solution $\{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\}_{k=1}^{N}$. Dynamic feasibility is estimated by using the defects introduced in (4.15). Using the notation of (5.4), the defect at the k^{th} temporal node is

$$\Delta_k = \|P_x(\tau_{k+1}) - \bar{\boldsymbol{x}}_{k+1}\|_2, \quad k \in \bar{\mathsf{N}}. \tag{5.5}$$

If each Δ_k is less than a prescribed tolerance $\epsilon_{\text{feasible}}$, then we say that the reference trajectory is dynamically feasible. Typically, choosing $\epsilon_{\text{feasible}} \leq 10^{-2}$ provides acceptable results, but this value is dependent on the original equations of motion. The propagation step is summarized in Algorithm 2.

Algorithm 2 The convexification step. All matrix operations are performed as 1D array operations.

Input: Reference trajectory $\{\bar{\boldsymbol{x}}_k,\ \bar{\boldsymbol{u}}_k,\ \bar{\boldsymbol{p}}\}$ for $k = 1, \ldots, N$ and feasibility tolerance $\epsilon_{\text{feasible}}$.
Output: Data structure, `output`, containing the discretized dynamics matrices, linearized constraint data, and feasibility indicator.

function CONVEXIFY
    Set `output.feasible = true`
    Call the PROPAGATE function
    Call the LINEARIZE function, see Algorithm 3
end function

function PROPAGATE
    $\hat{P}_0 \leftarrow \left[\,\text{flat}\,(I_{n_x})\ \ 0_{1\times n_x n_u}\ \ 0_{1\times n_x n_u}\ \ 0_{1\times n_x n_p}\ \ 0_{1\times n_x}\right]^\top$.
    for $k = 1, \ldots, N-1$ **do**
        $P \leftarrow \left[\bar{\boldsymbol{x}}_k^\top\ \ \hat{P}_0\right]^\top$ ▷ reset initial condition using reference state
        $h \leftarrow {}^{(\tau_{k+1}-\tau_k)}/_{N_{\text{sub}}-1}$ ▷ the `rk4` step size using N_{sub} steps
        Call `rk4` on the DERIVS function to update P
        Compute the defect Δ_k using (4.15)
        if $\Delta_k > \epsilon_{\text{feasible}}$ **then**
            `output.feasible = false`
        end if
        Set outputs:
            `output`.$A_d[:][k] \leftarrow P_\Phi$ ▷ [:] means "all rows"
            `output`.$B_{d,-}[:][k] \leftarrow P_\Phi P_{B^-}$
            `output`.$B_{d,+}[:][k] \leftarrow P_\Phi P_{B^+}$
            `output`.$E[:][k] \leftarrow P_\Phi P_E$
            `output`.$R[:][k] \leftarrow P_\Phi P_{\boldsymbol{r}}$
    end for
end function

function DERIVS(τ, P)
    Compute $\Psi \leftarrow P_\Phi^{-1}$ using an LU decomposition with partial pivoting
    Compute $\lambda_-(\tau)$ and $\lambda_+(\tau)$ and interpolate $\bar{\boldsymbol{u}}(\tau)$ using (4.4)
        $A \leftarrow \nabla_x F\big(P_x, \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big)$
        $B \leftarrow \nabla_u F\big(P_x, \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big)$
        $F \leftarrow \nabla_p F\big(P_x, \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big)$
        $\boldsymbol{r} \leftarrow F\big(P_x, \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big) - AP_x - B\bar{\boldsymbol{u}}(\tau) - E\bar{\boldsymbol{p}}$
    Set outputs:
        $\dot{P}_x \leftarrow F\big(P_x, \bar{\boldsymbol{u}}(\tau), \bar{\boldsymbol{p}}\big)$
        $\dot{P}_\Phi \leftarrow AP_\Phi, \quad \dot{P}_{B^-} \leftarrow \Psi\lambda_-(\tau)B, \quad \dot{P}_{B^+} \leftarrow \Psi\lambda_+(\tau)B, \quad \dot{P}_E \leftarrow \Psi E, \quad \dot{P}_{\boldsymbol{r}} \leftarrow \Psi\boldsymbol{r}$
end function

5.1.3 Constraint Approximation

The second part of the convexification step is to compute a convex approximation of all nonconvex algebraic inequality constraints. Because the standard form of the SOCP in Problem 9 requires the use of either affine or second-order cone constraints, we cannot use the full generality of the SCP methodology for real-time implementations using SOCP solvers (i.e., we're limited to sequential SOCPs).

Recall that the nonconvex constraints were assumed (without loss of generality) to be of the form

$$h_j\big(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}\big) \leq 0, \quad j = 1, \ldots, n_{ncvx}. \tag{5.6}$$

If $\nabla^2 h_j \succeq 0$, then it is possible to approximate (5.6) with a second-order cone constraint by computing a second-order Taylor series approximation about the reference solution. If computing these Hessians results in a sufficient increase in modeling fidelity relative to a first-order approximation, and the effort required to compute them is low, then it is advisable to use a second-order cone to approximate the nonconvex constraint. However, each of the quadratic constraints that were derived for the dual quaternion-based state constraints have *indefinite* Hessians (see §3.4.1).

As a result, we pursue affine approximations of each constraint.[4] Each nonconvex constraint is approximated by the N inequalities

$$h_j\big(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\big) + \nabla h_j\big(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\big) \begin{bmatrix} \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k \\ \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k \\ \boldsymbol{p} - \bar{\boldsymbol{p}} \end{bmatrix} \leq 0, \qquad j = 1, \ldots, n_{ncvx}, \quad k \in \mathsf{N} \tag{5.7a}$$

[4]There is another reason; problems with a large number of second-order cones can take longer to solve than similar problems with the same number of total constraints, but for which all of them are affine. Scharf et al. observed, anecdotally, "that one three-dimensional SOC constraint can be equivalent in terms of runtime to approximately 10 linear inequality constraints" [66].

$$\nabla \bar{h}_{jk} \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{u}_k \\ \boldsymbol{p} \end{bmatrix} \leq -\bar{h}_{jk}, \quad j = 1, \ldots, n_{ncvx}, \quad k \in \mathsf{N} \tag{5.7b}$$

where,

$$\nabla \bar{h}_{jk} \coloneqq \nabla h_j\big(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\big), \quad \bar{h}_{jk} \coloneqq h_j\big(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\big) - \nabla \bar{h}_{jk} \begin{bmatrix} \bar{\boldsymbol{x}}_k \\ \bar{\boldsymbol{u}}_k \\ \bar{\boldsymbol{p}} \end{bmatrix}. \tag{5.7c}$$

We can store the data for the approximation of the j^{th} nonconvex constraint in a single 2D array, where the first entry in the k^{th} row is the value of $\bar{h}_{jk}$ and the remaining columns are used to store the value of the gradient $\nabla \bar{h}_{jk}$. The first entries in each of these rows are used to populate the $\boldsymbol{b}$ vector, whereas the remaining entries are used to populate the A matrix in Problem 9 during the parsing step outlined in §5.1.4. Note that if a constraint h_j is a function of only a subset of the state, control, or parameter vectors, then the size of this 2D is shrunk accordingly.

As $h_j\big(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\big)$ is computed for each $j = 1, \ldots, n_{ncvx}$ and $k \in \mathsf{N}$, the feasibility with respect to each constraint at the reference nodes can be checked. If any constraints are found to be violated, then the current reference trajectory is marked as *infeasible*. If no constraints are violated, this does not necessarily imply that the trajectory will satisfy the constraints at times between the solution nodes, a phenomenon referred to as *constraint clipping*. Checking feasibility at the temporal nodes is therefore a necessary but not sufficient assessment of feasibility with respect to path constraints. In contrast, the assessment of dynamic feasibility based on the defects is both necessary and sufficient. Together, these measures of feasibility with respect to the nonconvex constraints provide a simple measure of the quality and feasibility of the solution at each PTR iteration.

The constraint linearization step is summarized in Algorithm 3.

Algorithm 3 The linearization step that forms part of the convexification step in Algorithm 2. All matrix operations are performed as 1D array operations.

```
function LINEARIZE
    for j = 1, ..., n_ncvx do
        for k = 1, ..., N do
            output.H_j[k][:2] ← ∇h_j(x̄_k, ū_k, p̄)
            output.H_j[k][1] ← h_j(x̄_k, ū_k, p̄) − ∇h_j(x̄_k, ū_k, p̄)[x̄_k^⊤, ū_k^⊤, p̄^⊤]^⊤
            if h_j(x̄_k, ū_k, p̄) > ε_feasible then
                output.feasible = false
            end if
        end for
    end for
end function                    ▷ [:2] means "all columns starting from 2"
```

5.1.4 Solve Step

The convexification step computes all of the data that is needed to populate the remaining non-zero entries of A, $\boldsymbol{b}$ and $\boldsymbol{c}$ that were not defined in the pre-parsing step. This process is referred to as parsing. Upon completion of the parsing step, we have fully defined the problem data for Problem 9, and the solver is then called. The PTR algorithm solves each convex approximation to full optimality, rather than settling for a sub-optimal solution and iterating again. Because calling the solver tends to dominate the algorithm runtime (even for a small number of solver iterations), reducing the number of calls to the solver is paramount to obtaining low runtimes.

Parsing

The parsing step adds the data from the convexification step to A and $\boldsymbol{b}$. These data form the remaining portions of $\star_P$ and $\star_C$ blocks shown in Figure 5.1. Because the standard form solution vector $\boldsymbol{z}$ contains the scaled state, control and parameters as 1D stacked vectors, the discretized dynamics (4.12) are written in block form as

$$\hat{R} = \hat{A}\hat{\boldsymbol{x}} + \hat{B}\hat{\boldsymbol{u}} + \hat{E}\hat{\boldsymbol{p}} + \boldsymbol{\nu}_+ - \boldsymbol{\nu}_-, \tag{5.8}$$

where $\boldsymbol{\nu} = \boldsymbol{\nu}_+ - \boldsymbol{\nu}_-$ is the virtual control term expressed using two variables in the linear cone [78, 126, 127, 128] and

$$\hat{A} = \begin{bmatrix} A_1 S_x & -S_x & 0 & \cdots & 0 \\ 0 & A_2 S_x & -S_x & \cdots & 0 \\ & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & A_{N-1} S_x & -S_x \end{bmatrix}, \tag{5.9a}$$

$$\hat{B} = \begin{bmatrix} B_1^- S_u & B_1^+ S_u & 0 & \cdots & 0 \\ 0 & B_2^- S_u & B_2^+ S_u & \cdots & 0 \\ & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & B_{N-1}^- S_u & B_{N-1}^+ S_u \end{bmatrix}, \tag{5.9b}$$

$$\hat{E} = \begin{bmatrix} E_1 S_p \\ E_2 S_p \\ \vdots \\ E_{N-1} S_p \end{bmatrix}, \tag{5.9c}$$

$$\hat{R} = \begin{bmatrix} \boldsymbol{c}_x - \boldsymbol{r}_1 - A_1 \boldsymbol{c}_x - B_1^- \boldsymbol{c}_u - B_1^+ \boldsymbol{c}_u - E_1 \boldsymbol{c}_p \\ \vdots \\ \boldsymbol{c}_x - \boldsymbol{r}_{N-1} - A_{N-1} \boldsymbol{c}_x - B_{N-1}^- \boldsymbol{c}_u - B_{N-1}^+ \boldsymbol{c}_u - E_{N-1} \boldsymbol{c}_p \end{bmatrix}. \tag{5.9d}$$

By the same steps, the boundary conditions (4.28e) can be written as

$$-\boldsymbol{r}_0 - A_0 \boldsymbol{c}_x - E_0 \boldsymbol{c}_p = A_0 S_x \hat{\boldsymbol{x}}_1 + E_0 S_p \hat{\boldsymbol{p}} + \boldsymbol{\nu}_{0,+} - \boldsymbol{\nu}_{0,-}, \tag{5.10a}$$

$$-\boldsymbol{r}_N - A_N \boldsymbol{c}_x - E_N \boldsymbol{c}_p = A_N S_x \hat{\boldsymbol{x}}_N + E_0 S_p \hat{\boldsymbol{p}} + \boldsymbol{\nu}_{N,+} - \boldsymbol{\nu}_{N,-}. \tag{5.10b}$$

Using (5.9) and (5.10), the $\star_P^{\cdot}$ blocks can be filled in as

$$\star_P^A = \begin{bmatrix} \begin{bmatrix} A_0 S_x & 0_{n_0 \times n_x(N-1)} \end{bmatrix} & 0_{n_0 \times n_u N} & E_0 S_p & I_{\nu,11} & -I_{\nu,11} \\ \hat{A} & & \hat{B} & \hat{E} & I_{\nu,22} & -I_{\nu,22} \\ \begin{bmatrix} 0_{n_f \times n_x(N-1)} & A_N S_x \end{bmatrix} & 0_{n_f \times n_u N} & E_N S_p & I_{\nu,33} & -I_{\nu,33} \end{bmatrix}, \tag{5.11a}$$

$$\star_P^B = \begin{bmatrix} -\boldsymbol{r}_0 - A_0 \boldsymbol{c}_x - E_0 \boldsymbol{c}_p \\ \hat{R} \\ -\boldsymbol{r}_N - A_N \boldsymbol{c}_x - E_N \boldsymbol{c}_p \end{bmatrix} \tag{5.11b}$$

where,

$$I_{\nu,11} = \begin{bmatrix} I_{n_0} & 0_{n_0 \times n_x(N-1)+n_f} \end{bmatrix}, \tag{5.12a}$$

$$I_{\nu,22} = \begin{bmatrix} 0_{n_x(N-1) \times n_0} & I_{n_x(N-1)} & 0_{n_x(N-1) \times n_f} \end{bmatrix}, \tag{5.12b}$$

$$I_{\nu,33} = \begin{bmatrix} 0_{n_f \times n_0} & 0_{n_f \times n_x(N-1)} & I_{n_f} \end{bmatrix}. \tag{5.12c}$$

The entries in (5.12) are independent of the matrices computed during the propagation step and would be populated during the pre-parse step.

Remark 5.2. *The notation adopted in some previous work on this block representation of the dynamics has used an identity block in the upper left corner of the $\hat{A}$ matrix given in* (5.9a) *[139]. Including this block in addition to the initial condition block* (5.10a) *can create an A matrix that does not have full row rank and potentially lead to numerical issues in the solver.*

Remark 5.3. *If the boundary conditions are imposed originally as affine constraints, then the additional virtual control term $\boldsymbol{\nu}_0$ is not required, and we set $I_{\nu,11} = 0$. An example of this is provided in the case study in §5.2.*

While every trajectory optimization problem will have the same basic structure for the $\star_P^{\cdot}$ blocks, the $\star_C^{\cdot}$ blocks are heavily application dependent. A trajectory optimization problem

stated with two different sets of constraints will have the same $\star_P$ blocks, but will have different $\star_C$ blocks. As such, it is only really informative to provide the block-expressions for the two classes of path constraints that can be included in Problem 9: convex affine or second-order cone constraints. Recall that general nonconvex constraints are assumed to be linearized, and so they are treated the same as affine constraints for the purposes of this section.

Affine Constraints There are several affine constraints that are common to trajectory optimization problems solved by sequential convex programming. For example, the use of one-norm penalty functions for the virtual control and box-type constraints on the state, control, and parameter vectors each produce affine constraints. As mentioned, affine constraints also arise from the linearization of nonconvex constraints. To illustrate how these are parsed, consider an affine state constraint of the form of (4.28c), rewritten here:[5]

$$\nabla \bar{g}_k \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{u}_k \\ \boldsymbol{p} \end{bmatrix} \leq -\bar{g}_k, \quad k \in \mathsf{N}. \tag{5.13}$$

This constraint is written in standard form by introducing the slack variable $\boldsymbol{s} \in \mathbb{R}^N_+$ and writing

$$G_x \hat{\boldsymbol{x}} + G_u \hat{\boldsymbol{u}} + G_p \hat{\boldsymbol{p}} + \boldsymbol{s} = G_b, \quad \boldsymbol{s} \geq 0, \tag{5.14}$$

where,

$$G_x = \begin{bmatrix} \nabla_x \bar{g}_1 \, S_x & & 0 \\ & \ddots & \\ 0 & & \nabla_x \bar{g}_N \, S_x \end{bmatrix}, \quad G_u = \begin{bmatrix} \nabla_u \bar{g}_1 \, S_u & & 0 \\ & \ddots & \\ 0 & & \nabla_u \bar{g}_N \, S_u \end{bmatrix}, \tag{5.15a}$$

[5]The notation for $\nabla \bar{g}_k$ and $\bar{g}_k$ represents the same thing as shown in (5.7c).

$$G_p = \begin{bmatrix} \nabla_p \bar{g}_1 \, S_p & & 0 \\ & \ddots & \\ 0 & & \nabla_p \bar{g}_N \, S_p \end{bmatrix}, \quad G_b = \begin{bmatrix} -\bar{g}_1 - \nabla_x \bar{g}_1 \boldsymbol{c}_x - \nabla_u \bar{g}_1 \boldsymbol{c}_u - \nabla_p \bar{g}_1 \boldsymbol{c}_p \\ \vdots \\ -\bar{g}_N - \nabla_x \bar{g}_N \boldsymbol{c}_x - \nabla_u \bar{g}_N \boldsymbol{c}_u - \nabla_p \bar{g}_N \boldsymbol{c}_p \end{bmatrix}. \quad \text{(5.15b)}$$

The matrices G_x, G_u and G_p are then added to $\star_C^A$ in the columns that correspond to the state, control, and parameters respectively, and in the N rows of the block c_2 constraints that correspond to the affine constraint at hand. Similarly, the vector G_b is added to $\star_C^b$ in the same N rows of the c_2 block of constraints. An identity block for the slack variable $\boldsymbol{s}$ can be pre-parsed into A in these same rows and the appropriate columns based on the definition of the block v_2 variables (see Figure 5.1).

If the affine constraint comes from the linearization of a nonconvex path constraint, then G_x, G_u, G_p and G_b will in general be functions of the reference trajectory $\{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\}_{k=1}^N$, and must be re-evaluated during each convexification step. The same can be true even for constraints that are affine in their original form in Problem 8. As an example of the latter, consider a trust region imposed using $q = \infty$, which is equivalent to two linear constraints on each of the state, control, and parameter vectors. In this case, the vector G_b will need to be re-evaluated during the parsing step as the reference solution changes.

Remark 5.4. *In addition, the nonconvex constraints* (5.7) *have the additional virtual control term ν'_k present in the expression. For these constraints, a constant matrix equal to $-I_N$ is pre-parsed into to the columns of A that correspond to the block v_2 variable $\boldsymbol{\nu}'$, and in the rows of block c_2 that correspond to the particular nonconvex constraint being approximated.*

Theorem 5.5 summarizes the increase in the SOCP problem size required to impose an affine path constraint in each convex subproblem solved during a PTR iteration. Note that the first nonconvex constraint incurs an addition one-time cost of N additional variables (the vector $\boldsymbol{\nu}'$) but no additional rows of A or $\boldsymbol{b}$. Subsequent nonconvex constraints may reuse the same vector $\boldsymbol{\nu}'$.

Theorem 5.5. *To impose an affine path constraint on the state, control, or parameter in*

Problem 8 requires adding at least N and at most $2N$ block v_2 variables to Problem 9 and adding N block c_2 rows to both A and $\boldsymbol{b}$.

As mentioned when the scaling method was presented in §5.1.1, the state, control, and parameter vectors can be scaled so that the solution variables $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{p}}$ are already in the linear cone $\mathbb{R}_+$. This means that no explicit affine constraints are needed to enforce the scaled variable's lower bound by adding rows to A and $\boldsymbol{b}$. Rather, the constraints are enforced implicitly by the solver. Theorem 5.5 tells us that scaling the solution variables in this way saves us $(n_x + n_u + n_p)N$ slack variables and as many rows in the matrices A and $\boldsymbol{b}$. Even for the simplest problems in powered descent guidance, this represents a savings on the order of 100 variables and rows of A and $\boldsymbol{b}$ for typical implementations.

Second-Order Cone Constraints Not all trajectory optimization problems will make use of second-order cone constraints. For problems in the aerospace domain specifically, however, there are several constraints that are naturally expressed in this form, such as an upper bound on thrust magnitude, thrust pointing constraints, approach cone constraints, and more (see §3.4.1 and §3.4.2 for examples). Moreover, we have found that two-norm-based trust regions, using $q = 2$ or 2^+, are an efficient method to guide the convergence process for the PTR algorithm.

To illustrate how these constraints are handled during the parsing step, consider a generic second-order cone path constraint imposed on the state vector only, to ease the notational burden of the presentation. The constraint is of the form

$$\|A_q \boldsymbol{x}_k + \boldsymbol{c}_q\|_2 \leq \boldsymbol{g}_q^\top \boldsymbol{x}_k + d_q, \quad k \in \mathsf{N} \tag{5.16}$$

for some $A_q \in \mathbb{R}^{d \times n_x}$, $\boldsymbol{c}_q \in \mathbb{R}^d$, $\boldsymbol{g}_q \in \mathbb{R}^{n_x}$ and $d_q \in \mathbb{R}$. Note that for the trajectory optimization problems considered here, it is rare for the data describing a second-order cone constraint to vary in time (i.e., to change with k) – and we have assumed that this is the case in (5.16).

First, the constraint (5.16) is rewritten in terms of the scaled state as

$$\|\hat{A}_q \hat{\boldsymbol{x}}_k + \hat{\boldsymbol{c}}_q\|_2 \leq \hat{\boldsymbol{g}}_q^\top \hat{\boldsymbol{x}}_k + \hat{d}_q, \quad k \in \mathsf{N} \tag{5.17}$$

where,

$$\hat{A}_q = A_q S_x, \quad \hat{\boldsymbol{c}} = A_q \boldsymbol{c}_x + \boldsymbol{c}_q, \quad \hat{\boldsymbol{g}}_q = S_x \boldsymbol{g}_q, \quad \hat{d}_q = d_q + \boldsymbol{g}_q^\top \boldsymbol{c}_x.$$

The constraint (5.17) is rewritten in standard form by introducing three slack variables for each $k \in \mathsf{N}$: $\boldsymbol{\mu}_k \in \mathbb{R}^d$, $\sigma_k \in \mathbb{R}_+$ and $s_k \in \mathbb{R}_+$. Using these slack variables, write

$$\boldsymbol{\mu}_k = \hat{A}\hat{\boldsymbol{x}}_k + \hat{\boldsymbol{c}}_q, \quad \sigma_k + s_k = \hat{\boldsymbol{g}}_q^\top \hat{\boldsymbol{x}}_k + d_q \quad \Rightarrow \quad \begin{bmatrix} \sigma_k \\ \boldsymbol{\mu}_k \end{bmatrix} \in \mathcal{C}_{Q_{d+1}}. \tag{5.18}$$

The expressions in (5.18) now form two affine constraints. Next, define a vector

$$\boldsymbol{\chi} \coloneqq (\sigma_1,\ \boldsymbol{\mu}_1,\ \sigma_1,\ \boldsymbol{\mu}_1,\ \cdots\ \sigma_N,\ \boldsymbol{\mu}_N) \in \mathbb{R}^{N(d+1)}. \tag{5.19}$$

Then, the following constraints are equivalent to (5.18):

$$-\left(I_N \otimes_K \hat{A}_q\right) \hat{\boldsymbol{x}} + \left(I_N \otimes_K \begin{bmatrix} 0 & I_d \end{bmatrix}\right) \boldsymbol{\chi} = \mathbf{1}_N \otimes_K \hat{\boldsymbol{c}}_q, \tag{5.20a}$$

$$-\left(I_N \otimes_K \hat{\boldsymbol{g}}_q\right) \hat{\boldsymbol{x}} + \left(I_N \otimes_K \begin{bmatrix} 1 & 0_{1\times d} \end{bmatrix}\right) \boldsymbol{\chi} + I_N \boldsymbol{s} = \hat{d}_q \mathbf{1}_N, \tag{5.20b}$$

$$(\sigma_k,\ \boldsymbol{\mu}_k) \in \mathcal{C}_{Q_{d+1}}, \quad k \in \mathsf{N}. \tag{5.20c}$$

where $\otimes_K$ represents the Kronecker product.

Remark 5.6. *The representation of the SOC constraint* (5.20) *can be entirely added to the matrices A and $\boldsymbol{b}$ during the pre-parse step, unless the constraint represents a trust region constraint. For a trust region constraint stated as an SOC constraint, the matrices $\hat{A}_q$, $\hat{\boldsymbol{g}}_q$, $\hat{d}_q$ are constant and may be pre-parsed, while $\hat{\boldsymbol{c}}_q$ changes across the iterations – the vector $\boldsymbol{c}_q = -\bar{\boldsymbol{x}}_k$ in this case – and forms part of the $\star_C^b$ block.*

Theorem 5.7 summarizes the *maximum* increase in the SOCP problem size required to impose a second-order cone path constraint in each convex subproblem solved during a PTR iteration.

Theorem 5.7. *To impose a $d+1$ dimensional second-order cone path constraint in Problem 4.28 requires adding at least $N(d+1)$ and at most $N(d+2)$ block v_3 variables to Problem 9 and an additional $N(d+1)$ block c_2 rows to both A and $\boldsymbol{b}$.*

The *at least* part in Theorem 5.7 is due to the structure of variable-width trust regions in the PTR algorithm, where the right hand side of the SOC path constraint is already a solution variable and an additional slack variable, $\boldsymbol{s}$, does not need to be added to the problem. Contrasting Theorem 5.7 with Theorem 5.5 allows us to quantify the possible difference in problem size that is available by replacing second-order cone variables with linear variables and vice versa.

Calling the Solver

After the parsing step is complete, the data $\{A, \boldsymbol{b}, \boldsymbol{c}, \ell, d_1, \ldots, d_m\}$ are passed to the solver that in turn provides an optimal solution. The solve step is summarized in Algorithm 4. Recall that ℓ is the dimension of the linear cone, the number of variables in $\boldsymbol{z}$ that are constrained to be in the nonnegative orthant. Similarly, the d_i are the dimensions of each second-order cone variable of the form of (5.20c). Each of these problem dimensions is known at compile time and do not need to be updated.

The solver used for the numerical case studies in this chapter is the BSOCP solver [15]. The BSOCP solver uses an interior point method and was designed with the express goal of solving powered descent guidance algorithms for real-time flight implementations. All technical details relating to the solver are summarized in [161] and we do not attempt to cover them here. The solver's capabilities were demonstrated during the aforementioned G-FOLD flight test campaign as part of the Autonomous Ascent and Descent Powered-Flight Testbed (ADAPT) project led by NASA's Jet Propulsion Laboratory in collaboration with

Algorithm 4 The solve step.

Input: Matrices A, $\boldsymbol{b}$, $\boldsymbol{c}$ and the `output` data structure of the convexification step.
Output: The data structure `output` with the reference trajectory overwritten.

function SOLVE_SOCP
    Call PARSE to update A, $\boldsymbol{b}$, $\boldsymbol{c}$
    $\boldsymbol{z}^* = \texttt{bsocp}(A, \boldsymbol{b}, \boldsymbol{c})$ ▷ Call solver BSOCP with pre-defined tolerance values
    Compute `output.delta_xp` using (4.30)
    Set outputs: ▷ $\hat{\boldsymbol{x}}^*$, $\hat{\boldsymbol{u}}^*$, $\hat{\boldsymbol{p}}^*$ are retrieved from $\boldsymbol{z}^*$.
        $\texttt{output.x} = (I_N \otimes_K S_x)\hat{\boldsymbol{x}}^* + (\mathbf{1}_N \otimes_K \boldsymbol{c}_x)$
        $\texttt{output.u} = (I_N \otimes_K S_u)\hat{\boldsymbol{u}}^* + (\mathbf{1}_N \otimes_K \boldsymbol{c}_x)$
        $\texttt{output.p} = S_p\hat{\boldsymbol{p}}^* + \boldsymbol{c}_p$
end function

Masten Space Systems [66, 67].

The parameters used to define numerical tolerances and convergence are listed in [161, Table A.1], and must be tailored to the specific problem instance being implemented. If $\boldsymbol{z}^*$ is the solution returned by the solver, we note that not all of $\boldsymbol{z}^*$ needs to be retained. The new reference trajectory $\{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\}_{k=1}^N$ can be extracted by unscaling the entries of $\boldsymbol{z}^*$ that correspond to $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{p}}$ – which are the first three variables in block v_1.

Prior to terminating the solve step, we leverage the fact that the scaled state and parameter vectors $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{p}}$ are already available on the stack and compute the value of (4.30) for later use in determining convergence.

5.1.5 Stopping Criteria

The final component of the algorithm is the stopping criteria. The details were covered in 4.2.4, and Table 5.1 shows the possible exit conditions of the algorithm. Once the criteria (4.30) is satisfied, the iterations are terminated provided that `output.feasible` is true. Note that a trajectory cannot be dynamically feasible unless the virtual control norm is small enough to allow this, and checking `output.feasible` is a more robust exit criterion than checking the norm of the virtual control.

Table 5.1: Possible exit conditions for the PTR algorithm based on the criterion (4.30). Feasibility is assessed on line 13 of Algorithm 2 and line 6 of Algorithm 3 during the convexification step.

Converged	Feasible	Description
✓	✓	Converged and feasible.
✗	✓	Reached maximum PTR iterations before $\delta_{xp} < \delta_{\text{tol}}$. Sub-optimal.
✓	✗	Converged but not feasible. Do not use.
✗	✗	Not converged and not feasible. Do not use.

5.2 Case Study: Planar Landing

To illustrate the design methodology for real-time implementable algorithms, we examine the propellant-optimal planar landing problem that was introduced in §3.3 of Chapter 3. The reason that we use this problem is so that we may compare our results to the optimal solution structure proven in Theorem 3.4 and Lemma 3.5.

The planar landing scenario is depicted in Figure 3.2. This problem has nonlinear dynamics that serve as the sole source of nonconvexity. While nonconvex constraints can be added and the design process remains unchanged, we do not do so for this case study in order to preserve the theoretical properties of the solution. Planar landing problems have $n_x = 7$ states, $n_u = 2$ controls and $n_p = 1$ parameter (the final time) for which to solve.

The state vector, $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$, is comprised of the mass, $m(t) \in \mathbb{R}_{++}$, the landing-frame position $\boldsymbol{r}_{\mathcal{L}}(t) \in \mathbb{R}^2$, the landing-frame velocity, $\boldsymbol{v}_{\mathcal{L}}(t) \in \mathbb{R}^2$, a single attitude variable $\theta(t) \in \mathbb{R}$ and the angular velocity $\omega(t) \in \mathbb{R}$. The control, $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$, is assumed to come from a body-fixed rocket engine capable of generating variable thrust $\Gamma(t) \in \mathbb{R}_{++}$ and a separate actuation mechanism capable of independently providing a torque $\tau(t) \in \mathbb{R}$ (e.g., an RCS system).

For this problem, because the control inputs are scalar and decoupled, we can formulate bounds on their magnitude as the box constraints (3.27) – which are affine constraints on

the input. For numerical solutions, we bound the state variable using

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{\max}, \tag{5.21}$$

for some $\boldsymbol{x}_{\min}$, $\boldsymbol{x}_{\max} \in \mathbb{R}^{n_x}$. In this case study, we choose the components of $\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\max}$ to be just large enough so that none of the bounds are active during the descent, thereby using their numerical values for scaling purposes only.

The minimum propellant cost function can be expressed in Mayer form by using the final mass of the vehicle. Maximizing the final mass is equivalent to minimizing the fuel consumed, and so we take $M\big(\boldsymbol{x}(t_0), \boldsymbol{x}(t_f), t_f\big) = -\boldsymbol{e}_1^\top \boldsymbol{x}(t_f) = -m(t_f)$ as the cost. The continuous time optimal control problem that we wish to solve is given by

$$\begin{aligned} \min_{\theta(t_0),\, \boldsymbol{u}(\cdot),\, t_f} \quad & -\boldsymbol{e}_1^\top \boldsymbol{x}(t_f) \\ \text{s.t.} \quad & (3.26),\ (3.27),\ (3.28),\ (5.21). \end{aligned} \tag{5.22}$$

Note that (5.22) is equivalent to Problem 5 with the addition of the (affine) state constraint (5.21).

The initial and final boundary conditions from (3.28) can be written as

$$A_0 \boldsymbol{x}(t_0) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{x}(t_0) = \boldsymbol{x}_{ic}, \quad \boldsymbol{x}_{ic} = \begin{bmatrix} m_{ic} \\ \boldsymbol{r}_{\mathcal{L},ic} \\ \boldsymbol{v}_{\mathcal{L},ic} \\ \boldsymbol{\omega}_{\mathcal{B},ic} \end{bmatrix}, \tag{5.23a}$$

$$A_N \boldsymbol{x}(t_f) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{x}(t_f) = \boldsymbol{x}_{fc}, \quad \boldsymbol{x}_{fc} = \begin{bmatrix} \boldsymbol{r}_{\mathcal{L},f} \\ \boldsymbol{v}_{\mathcal{L},f} \\ \theta_f \\ \boldsymbol{\omega}_{\mathcal{B},f} \end{bmatrix}, \tag{5.23b}$$

where $\boldsymbol{x}_{ic} \in \mathbb{R}^{n_0}$ and $\boldsymbol{x}_{fc} \in \mathbb{R}^{n_f}$ with $n_0 = n_f = 6$.

Following the procedures discussed in §4.2, for each PTR iteration we solve the convex subproblem given by Problem 10, posed for clarity using unscaled variables. This case study uses N equally spaced temporal nodes during the convexification step. Note that we elect to use a two-norm trust region in order to demonstrate the use of SOC variables. Use of a different norm, such as the infinity norm, would allow the trust region to be written using only linear constraints. However, Theorems 5.5 and 5.7 can be used to show that an infinity norm trust region used in place of a two-norm trust region would in fact require $N(n_x + n_u - 1)$ *additional* variables and $N(n_x + n_u)$ *additional* rows in the A and $\boldsymbol{b}$ matrices. Empirical results for this particular problem reveal that this increase in problem size has a significant impact on the resulting solve times, even though the use of linear trust regions leads to a linear program. Over 100 trials using both methods on the exact same problem, the two-norm trust region provided an average decrease of 11 ms in solver time *per PTR iteration* compared to the infinity norm trust region. While this speed-up may be specific to this particular problem, we nonetheless proceed by using the two-norm trust region on both the state and control.

Problem 10. *Find the vectors* $\{\boldsymbol{X}, \boldsymbol{U}, p, \boldsymbol{\eta}, \eta_p, \boldsymbol{V}\}$ *that solve the following parameter optimization problem:*

$$\min_{\boldsymbol{X}, \boldsymbol{U}, p, \boldsymbol{\eta}, \eta_p, \boldsymbol{V}} \quad -\boldsymbol{e}_1^\top \boldsymbol{x}_N + w_{vc}\|\boldsymbol{\nu}\|_1 + \boldsymbol{w}_{tr}\mathbf{1}_N^\top \boldsymbol{\eta} + w_{tr,p}\eta_p \tag{5.24a}$$

$$s.t. \quad \boldsymbol{x}_{k+1} = A_k x_k + B_k^- \boldsymbol{u}_k + B_k^+ \boldsymbol{u}_{k+1} + E_k p + \boldsymbol{r}_k + \boldsymbol{\nu}_k, \quad k \in \bar{\mathsf{N}} \tag{5.24b}$$

$$A_0\boldsymbol{x}_0 = \boldsymbol{x}_{ic}, \quad A_N\boldsymbol{x}_N = \boldsymbol{x}_{fc}, \tag{5.24c}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_k \leq \boldsymbol{x}_{\max}, \qquad k \in \mathsf{N} \tag{5.24d}$$

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{\max}, \qquad k \in \mathsf{N} \tag{5.24e}$$

$$p_{\min} \leq p \leq p_{\max}, \tag{5.24f}$$

$$\|\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k\|_2 + \|\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k\|_2 \leq \eta_k, \qquad k \in \mathsf{N} \tag{5.24g}$$

$$|p - \bar{p}| \leq \eta_p, \tag{5.24h}$$

where $\boldsymbol{X} = \{\boldsymbol{x}_k\}_{k=1}^N$, $\boldsymbol{U} = \{\boldsymbol{u}_k\}_{k=1}^N$ *and* $\boldsymbol{V} = \{\boldsymbol{\nu}_k\}_{k=1}^{N-1}$.

We then scale the solution variables in Problem 10 by substituting $\boldsymbol{x}_k \leftarrow S_x\hat{\boldsymbol{x}}_k + \boldsymbol{c}_x$, $\boldsymbol{u}_k \leftarrow S_u\hat{\boldsymbol{u}}_k + \boldsymbol{c}_u$ and $p \leftarrow S_p\hat{p} + c_p$ wherever they appear, *except* in the trust regions. Following [72, Remark IV.1], we scale the trust region constraints (5.24g) and (5.24h) using

$$\|\hat{\boldsymbol{x}}_k - S_x^{-1}(\bar{\boldsymbol{x}}_k - \boldsymbol{c}_x)\|_2 + \|\hat{\boldsymbol{u}}_k - S_u^{-1}(\bar{\boldsymbol{u}}_k - \boldsymbol{c}_u)\|_2 \leq \eta_k, \quad k \in \mathsf{N} \tag{5.25a}$$

$$|\hat{p} - S_p^{-1}(\bar{p} - c_p)| \leq \eta_p. \tag{5.25b}$$

5.2.1 Standard SOCP Form

Next, we introduce slack variables, the block v_2 variables, so that each of the affine inequality constraints (5.24d)-(5.24f) are expressed as equalities. This follows the general procedure outlined in (5.14). Slack variables are also introduced in order to express the cost function as a linear function of the solution variables by using equivalent problem transformations [10]. Each of these steps are straightforward. The implementation of the trust region is less straightforward, and so it is highlighted here. The state and control trust region is implemented by introducing a total of $2N$ variables, $(\eta_{x,k}, \boldsymbol{\mu}_{x,k}) \in \mathcal{C}_{Q_{n_x+1}}$ and $(\eta_{u,k}, \boldsymbol{\mu}_{u,k}) \in \mathcal{C}_{Q_{n_u+1}}$ for each $k \in \mathsf{N}$, such that

$$\boldsymbol{\mu}_{x,k} = \hat{\boldsymbol{x}}_k - S_x^{-1}\left(\bar{\boldsymbol{x}}_k - \boldsymbol{c}_x\right) \quad \text{and} \quad \boldsymbol{\mu}_{u,k} = \hat{\boldsymbol{u}}_k - S_u^{-1}\left(\bar{\boldsymbol{u}}_k - \boldsymbol{c}_x\right), \tag{5.26}$$

whereafter $\|\boldsymbol{\mu}_{x,k}\|_2 \leq \eta_{x,k}$ and $\|\boldsymbol{\mu}_{u,k}\|_2 \leq \eta_{u,k}$ are now equivalent to (5.25a). We can reconstruct the original trust region, η_k, if needed from $\eta_k = \eta_{x,k} + \eta_{u,k}$. Note that because $\eta_{x,k}$ and $\eta_{u,k}$ are already solution variables, we do not need to add another slack variable (the σ_k in (5.18)) to express the trust region constraint in standard form. This illustrates the *at least* qualification in the statement of Theorem 5.7.

Next, define the variables

$$\boldsymbol{\chi}_x \coloneqq \begin{bmatrix} \eta_{x,1} \\ \boldsymbol{\mu}_{x,1} \\ \vdots \\ \eta_{x,N} \\ \boldsymbol{\mu}_{x,N} \end{bmatrix} \in \mathbb{R}^{N(n_x+1)}, \quad \boldsymbol{\chi}_u \coloneqq \begin{bmatrix} \eta_{u,1} \\ \boldsymbol{\mu}_{u,1} \\ \vdots \\ \eta_{u,N} \\ \boldsymbol{\mu}_{u,N} \end{bmatrix} \in \mathbb{R}^{N(n_u+1)}, \quad \chi_{p,1}, \chi_{p,2} \in \mathbb{R}_+. \tag{5.27}$$

The trust region constraints (5.25) can then be expressed as

$$\hat{\boldsymbol{x}} + H_{\mu,x}\boldsymbol{\chi}_x = \boldsymbol{b}_x, \quad H_{\mu,x} \coloneqq I_N \otimes_K \begin{bmatrix} 0_{n_x \times 1} & -I_{n_x} \end{bmatrix}, \quad k \in \mathsf{N} \tag{5.28a}$$

$$\hat{\boldsymbol{u}} + H_{\mu,u}\boldsymbol{\chi}_u = \boldsymbol{b}_u, \quad H_{\mu,x} \coloneqq I_N \otimes_K \begin{bmatrix} 0_{n_u \times 1} & -I_{n_u} \end{bmatrix}, \quad k \in \mathsf{N} \tag{5.28b}$$

$$\hat{p} - \eta_p + \chi_{p,1} = b_p, \quad \hat{p} + \eta_p - \chi_{p,2} = b_p, \tag{5.28c}$$

where,

$$\boldsymbol{b}_x = \begin{bmatrix} S_x^{-1}\left(\bar{\boldsymbol{x}}_1 - \boldsymbol{c}_x\right) \\ \vdots \\ S_x^{-1}\left(\bar{\boldsymbol{x}}_N - \boldsymbol{c}_x\right) \end{bmatrix}, \quad \boldsymbol{b}_u = \begin{bmatrix} S_u^{-1}\left(\bar{\boldsymbol{u}}_1 - \boldsymbol{c}_u\right) \\ \vdots \\ S_u^{-1}\left(\bar{\boldsymbol{u}}_N - \boldsymbol{c}_u\right) \end{bmatrix}, \quad b_p = S_p^{-1}\left(\bar{p} - c_p\right). \tag{5.29}$$

Note that because there is only a single parameter in this problem, the trust region (5.25b) is a linear constraint; and each of $\hat{p}$, η_p, $\chi_{p,1}$ and $\chi_{p,2}$ reside in the linear cone.

Figure 5.2 shows the resulting structure of the matrices $A, \boldsymbol{b}$ and $\boldsymbol{c}$ for this problem, with the dotted gray lines indicating the block structure illustrated in Figure 5.1. Recall that

the block v_1 variables are those that are used to write the equations of motion, the block v_2 variables contain the linear slack variables added to write the problem in standard form, and the block v_3 variables contain the variables related to the trust region implementations and all SOC variables. Note that in Figure 5.2 we have defined

$$H_{\eta,x} \coloneqq \mathbf{1}_N^\top \otimes_K \begin{bmatrix} 1 & 0_{1\times n_x} \end{bmatrix} \in \mathbb{R}^{1\times N(n_x+1)} \quad \text{and} \quad H_{\eta,u} \coloneqq \mathbf{1}_N^\top \otimes_K \begin{bmatrix} 1 & 0_{1\times n_u} \end{bmatrix} \in \mathbb{R}^{1\times N(n_u+1)}.$$

The problem dimensions can be computed as a function of the base units N, n_x and n_u (by using $n_p = 1$ and $n_0 = n_f = n_x - 1$) to be:

$$\begin{aligned} \text{Variables:} \quad & N(8n_x + 3n_u + 2) + 5(1 - n_x), \\ \text{Equality constraints:} \quad & N(5n_x + 2n_u) + 1 - n_x, \\ \text{Linear cones:} \quad & N(7n_x + 2n_u) + 5(1 - n_x), \\ \text{Second order cones:} \quad & 2N. \end{aligned}$$

The first N SOCs are each of dimension $n_x + 1$, whereas the last N SOCs are each of dimension $n_u + 1$.

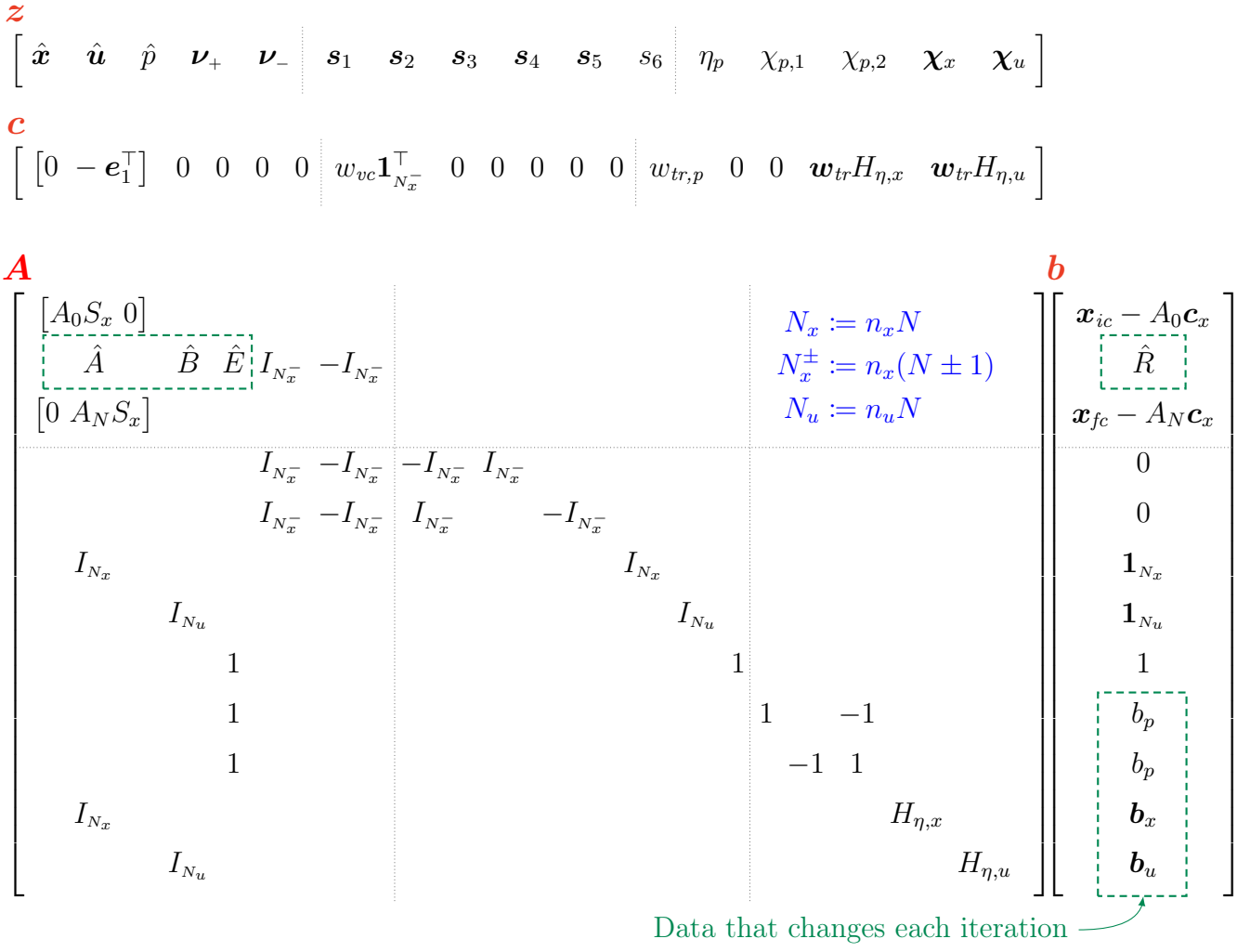


Figure 5.2: The A, $\boldsymbol{b}$ and $\boldsymbol{c}$ matrices for the planar landing case study. All empty blocks are zero, and the data that is not pre-parsed are highlighted by green boxes.

Table 5.2: Percentage of the data defined during the pre-parse step for the planar landing problem using a temporal density of $N = 20$. Note that A is 99.7% sparse.

	Total Size	**Non-Zero Entries**	**Defined at Pre-parse**	**Pct.** %
A	789×1285	2919	2410	73.3
b	789	506	193	38.1
c	1285	182	182	100

5.2.2 Initialization Step

The initial guess was computed by using the straight-line interpolation method between the vectors $\left(m_{ic},\, \boldsymbol{r}_{\mathcal{L},ic},\, \boldsymbol{v}_{\mathcal{L},ic},\, 0,\, \boldsymbol{\omega}_{\mathcal{B},ic}\right)$ and the vector $\left(m_f,\, \boldsymbol{r}_{\mathcal{L},f},\, \boldsymbol{v}_{\mathcal{L},f},\, \theta_f,\, \boldsymbol{\omega}_{\mathcal{B},f}\right)$ where $m_f = \frac{m_{ic}+m_{\min}}{2}$. Given that the optimal thrust solution is known to be always on the boundary of the interval (3.27), we initialize the thrust input channel using

$$\Gamma_k = \begin{cases} \Gamma_{\min}, & k = 1, \ldots, n_m \\ \Gamma_{\max}, & k = n_m + 1, \ldots, N. \end{cases} \tag{5.30}$$

The effect of the parameter n_m on the computed solution for a fixed N is studied in Figure 5.6. The optimal torque input does not necessarily activate either of its constraints (3.27b), and so we initialize the torque to be zero at each temporal node.

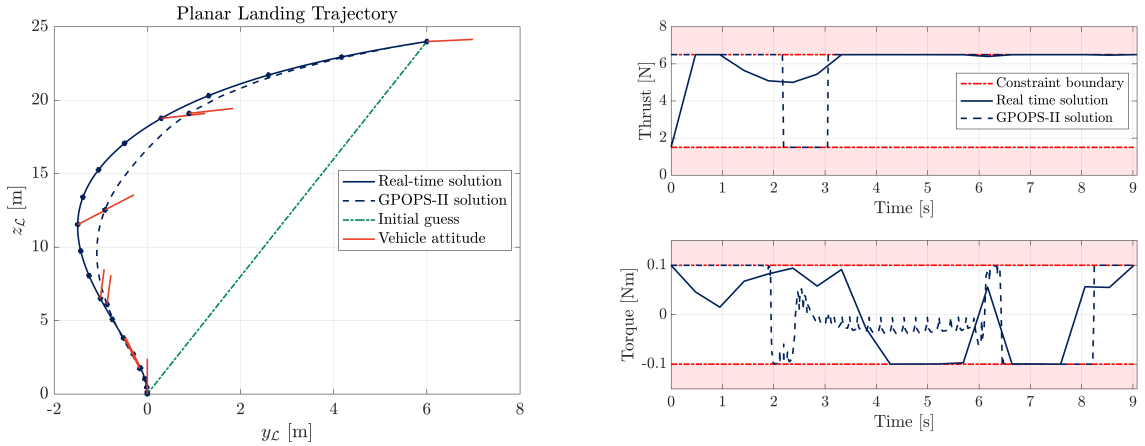
The pre-parse step was implemented after forming the matrices in Figure 5.2. Each non-zero entry that is not encompassed by a green box was set during the pre-parse step, along with a subset of the data within the green boxes. For the planar landing problem, Table 5.2 indicates the number of non-zero entries that can be populated during pre-parsing. Decoupling the pre-parse and parse steps allows significant computational savings, because only about 800 non-zero values need to be updated during each PTR iteration.

Table 5.3: Nominal parameters used in the planar landing case study.

Parameter	Value	Units	Parameter	Value	Units
$t_{f,\text{guess}}$	8.0	s	$t_{f,\max\,/\,\min}$	4.0/12.0	s
m_{ic}	5.0	kg	$m_{\min}$	2.0	kg
$\boldsymbol{r}_{\mathcal{L},ic}$	(6, 24)	m	$\boldsymbol{r}_{\mathcal{L},f}$	(0, 0)	m
$\boldsymbol{v}_{\mathcal{L},ic}$	$(-4, -2)$	m/s	$\boldsymbol{v}_{\mathcal{L},f}$	(0, 0)	m/s
ω_{ic}	0	rad/s	ω_f	0	rad/s
θ_0	$\in [-\pi, \pi]$	rad	θ_f	0	rad
$\Gamma_{\min}$	1.5	N	$\Gamma_{\max}$	6.5	N
$\tau_{\max}$	0.1	Nm	J	0.5	$\text{kg}\,\text{m}^2$
I_{sp}	3	s^{-1}	g	-1	m/s^2
w_{vc}	10^2	–	$\boldsymbol{w}_{tr}$	5^{-2}	–
$w_{tr,p}$	10^{-2}	–	N_{sub}	15	–
$\epsilon_{\text{feasible}}$	10^{-2}	–	δ_{tol}	10^{-3}	–

5.2.3 Computational Results

Prior to discussing the computational performance, we briefly compare the computed trajectory to one that is assumed to represent a locally optimal trajectory. The problem parameters used for this comparison are given in Table 5.3. We use GPOPS-II to compute the locally optimal solution for comparison [171]. Figure 5.3 shows the resulting trajectories in the $\boldsymbol{y}_{\mathcal{L}}$-$\boldsymbol{z}_{\mathcal{L}}$ plane, alongside the thrust and torque trajectories. For the real-time solution, the circles represent the last reference solution $\{\bar{\boldsymbol{x}}_k\}_{k=1}^N$ obtained by the final solve step prior to exiting the main PTR loop, whereas the solid line corresponds to the continuous state trajectory obtained by integrating the solution $\{\bar{\boldsymbol{u}}_k, \bar{\boldsymbol{p}}\}_{k=1}^N$ through the nonlinear dynamics using the affine interpolation (4.4). The attitude of the vehicle is depicted by the red lines in Figure 5.3a at identical temporal points in each trajectory, and it can be seen to match well at each point. Both the state and control trajectories show some key differences; namely the real-time control solution does not have the minimum thrust arc seen in the solution computed by GPOPS-II and the torque solution follows a different albeit smoother trajectory.



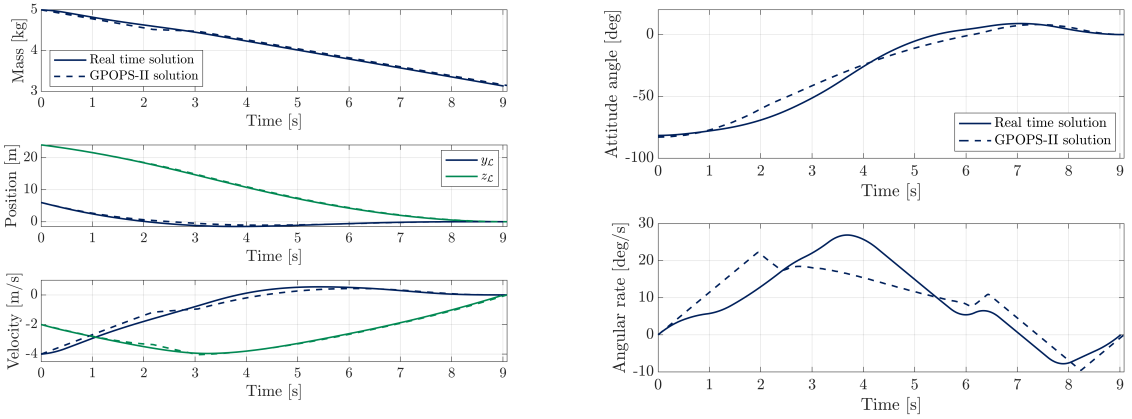
(a) The computed inertial position trajectories

(b) The computed control trajectories

Figure 5.3: The real-time solution using $N = 20$ compared to the (locally) optimal planar landing solution. In (a), the circles represent solution nodes at each $k \in \mathsf{N}$ and the vehicle attitude is shown for both solutions at the same time instances.

The torque jitter in the GPOPS-II solution is likely an artifact of that software's known difficulty in finding singular solutions when encompassed by non-singular arcs [171]. These control differences do not greatly impact the PTR algorithm's ability to find an acceptable sub-optimal solution, as the cost is not significantly affected by the presence of the minimum thrust arc. As seen in Figure 5.4, the final mass is very close between the two trajectories. In fact, the real-time solution finds a solution that is roughly 0.6% sub-optimal and with a final time that is within 1.3% of the optimal solution. This sub-optimality is traded-off for the ability to compute the solutions in real-time in an architecture that is appropriate for flight code.

We now study the computational performance obtained by the real-time PTR algorithm in addition to the sensitivity of key outputs to user-defined algorithm parameters. The solution method was implemented in C++ and run using a single 3.2 GHz Intel i5 processor with 8GB of RAM. All code was compiled using the `-O3` flag. Figure 5.5 shows the computational times obtained as a function of the temporal density N. Each data point represents the mean



(a) Translational states

(b) Rotational states

Figure 5.4: Individual state trajectories for the planar landing case study. The real-time results use $N = 20$.

value of 100 trials using the specified value of N and all other parameters as in Table 5.3.[6] The error bars that are shown indicate one standard deviation. The contributions of the pre-parse, convexification, parse and solve steps are shown individually, in addition to the total solution time. The top plot in Figure 5.5 shows that the total solution time and the solver time per PTR iteration follow the same trend as N increases. It is expected that the solver time is cubic in N because the BSOCP solver performs a Cholesky factorization each time it is called to obtain the Newton search direction [16], a function whose operation count scales with the cube of the matrix dimension [170]. The cubic polynomial fit to the total time (shown by the red line) supports the hypothesis that the total solution time is dominated by the internal operations of the BSOCP solver, and in particular the Cholesky factorization. The time spent outside the solver consumes on average 2% of the total solution time. The desirable conclusion is that the PTR algorithm adds little computational overhead to what

[6]Because the results shown are on a per iteration basis, we hypothesize that varying the initial conditions in a more traditional Monte Carlo test would not change the results as presented. There are no additional constraints to activate and no additional non-zero entries in A or b to make the problem more challenging to solve.

is needed by the solver. However, caution should be used when extrapolating this result to other problem instances, as the ratio of time spent in the solver to time spent in other functions can vary between different problems and implementations.

The bottom four plots in Figure 5.5 show that the percent sub-optimality is roughly constant across all tested values of N, and that the rumtimes of the pre-parse, convexification and parse steps scale linearly with N. For the convexification step, this is expected based on the computational complexity discussion presented in §5.1.2. Both the pre-parse and parsing steps can be done very quickly, and do not impact the overall solution time of the PTR algorithm appreciably.

Lastly, Figure 5.6 shows the variation in the final time, the final mass (i.e., the cost) and the total solution time versus several parameters that are selected by the user. For each plot, all nominal values given in Table 5.3 were held constant (with $N = 15$) and only the variable shown on the horizontal axis was changed. The final time guess is seen to strongly factor into the total solution time, and a minimum is observed around the optimal value of ≈ 9s, as expected. As mentioned during the case study in Chapter 4, final time guesses that are much shorter than the optimal final time often lead to higher runtimes compared to guesses that are longer than the optimal final time. The values chosen for the virtual control and trust region weights can affect the dependent variables tested if they are chosen improperly; the results suggest there is a rather large range for each parameter where the computed solution remains nearly the same. The solution using $\boldsymbol{w}_{tr} = 10^{-3}$ produced results that did not converge and are marked with a red $\times$ (the total solve time is off the chart). This unsurprisingly indicates that if state and control deviations are not penalized enough, then convergence suffers accordingly. The variations observed for the solution that uses $w_{vc} = 10^3$ can be attributed to scaling issues that start to arise when this parameter is too large.

The fourth row of Figure 5.6 shows that the values of the inertia and specific impulse can affect the total solution time, but do not strongly affect the final time or mass for this case study. This implies that algorithm parameters tuned for nominal values do not need to be changed if the actual values change by up to $^{+10}/_{-20}\%$. Beyond this range, the algorithm

parameters may need to be changed slightly to obtain the best computational performance. The increased solution time for large I_{sp} is caused by the increasing optimal final time that eventually requires an additional PTR iteration to converge to. Caution must be exercised when extrapolating these latter two results to other problem instances, because different dynamics, boundary conditions and/or constraints can change the relationship between J, I_{sp} and the dependent variables studied here. A similar sensitivity analysis should be carried out for each new implementation. Lastly, the fifth row of Figure 5.6 shows the effect of varying n_m in (5.30). While the final mass and final time are largely unaffected, it can be seen that lower values of n_m produce faster convergence. The optimal choice in this case is $n_m = 1$, the nominal value that is used.

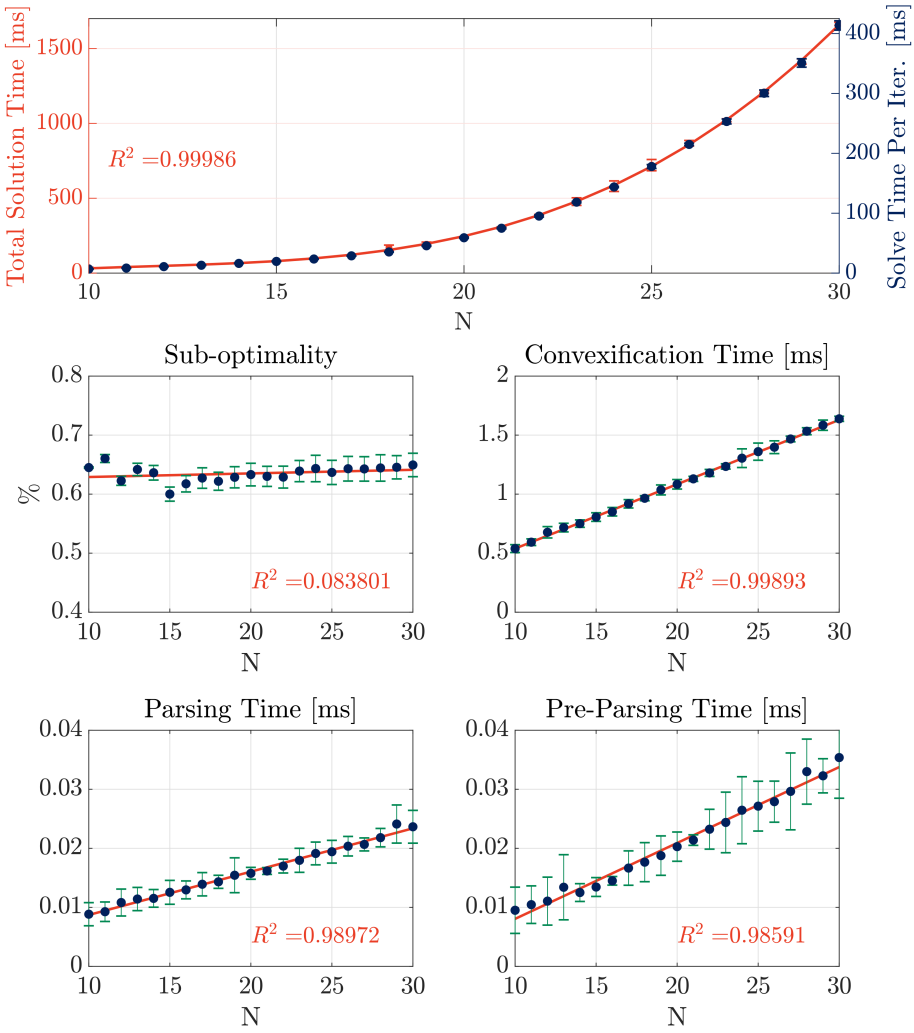


Figure 5.5: The computational times and percent sub-optimality obtained for each primary step in the algorithm versus the discretization density N for the planar landing case study.

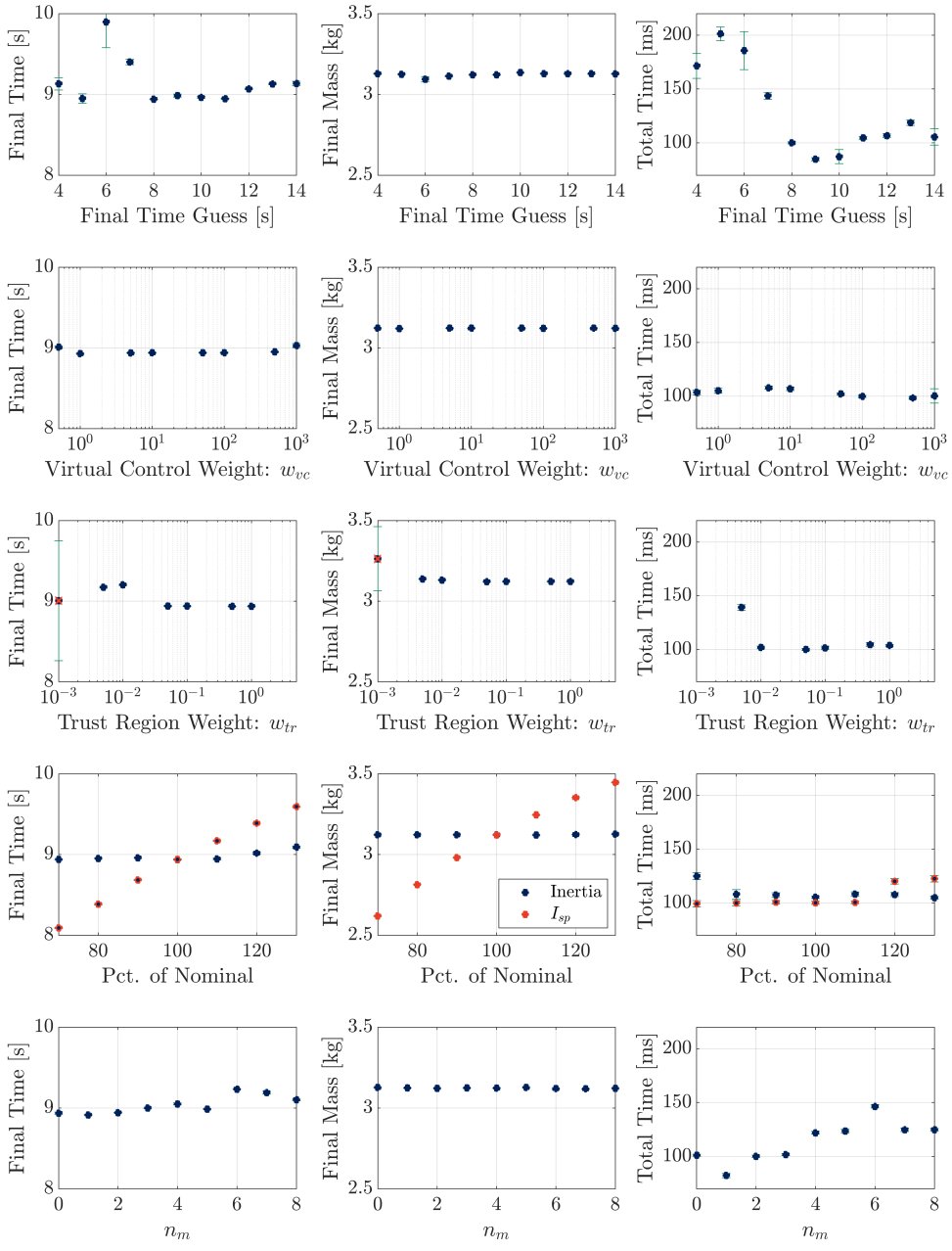


Figure 5.6: Variations in the final time t_f, the final mass m_f and the total solve time versus several key algorithm parameters. All tests were run using $N = 15$. The variable n_m is defined in (5.30).

Chapter 6

IMPLICIT TRAJECTORY OPTIMIZATION:
FUNNEL SYNTHESIS

No one shall expel us from the paradise that Cantor has created for us.
– David Hilbert

This chapter builds on the explicit trajectory optimization methods that have been introduced so far in order to develop what we call implicit trajectory optimization methods. Implicit trajectory optimization provides one (or more) functions that implicitly define a set of trajectories in both state and control space. Instead of a single trajectory that connects two boundary conditions, we obtain a group of functions that connect two sets of initial and terminal boundary conditions.The particular implicit trajectory optimization method that is developed in this chapter is called *funnel synthesis*.

Explicit trajectory generation suffers from two fundamental drawbacks.[1] First, if a trajectory is computed by an explicit method and any problem data is subsequently changed (e.g., an initial condition), then this trajectory is no longer strictly feasible. An attempt to follow the originally computed trajectory by executing the control commands would place an undue burden on the downstream tracking controllers that would be responsible for cleaning up the dispersions, and the satisfaction of state and control constraints may not be guaranteed. Explicit trajectory generation is inherently specific to the given problem data, and any changes necessitate a full re-solve of the optimal control problem.

Second, there is a lack of formal convergence guarantees for the solution of nonconvex optimal control problems. No known algorithm can be formally guaranteed to solve a nonconvex optimal control problem from an arbitrary initial guess. There is therefore no

[1]Trajectory generation and trajectory optimization are used synonymously in this chapter.

theoretical reason that we should expect to always be able to solve a nonconvex trajectory optimization problem in real-time – though certainly there is an empirical reason to expect reliable convergence. It is natural, in response, to study alternative methods for which the ability to provide a feasible trajectory in real-time can be theoretically established.

Consider the following guiding question: if problem data changes *after* an explicit method is used to obtain a trajectory, can this trajectory be augmented to provide a new trajectory without re-solving the original problem, and can we guarantee that these trajectories will be both dynamically feasible and feasible with respect to all state and control constraints?

This question is intimately related to *neighboring optimal control*, a concept that was initially developed by the trajectory optimization community [172, 173]. The main idea is to use a first-order model of a nonlinear system and a second-order model of the cost function around some trajectory, and study the necessary conditions for optimality given by the maximum principle. A state feedback law that is obtained by a solution of these necessary conditions can provide a near-optimal controller and one that is rather robust to parameter variations. It is interesting to note that several of the authors whose works were cited in earlier chapters on powered descent are the same authors who developed the earliest ideas of neighboring optimal control. Since early work in the 1960s, neighboring optimal control has been applied to several aerospace and robotics problems, including launch vehicle ascent [174, 175], aircraft wind gust alleviation [176], and robot trajectory tracking [177], among others. The key difference between neighboring optimal control and the funnel synthesis methods presented in this chapter are the use of a Lyapunov function (as opposed to the maximum principle-based techniques) and by extension the degree to which state and control constraints can be handled. We can do more than separate the topics based on their technical approaches; neighboring optimal control is designed to seek out nearby *optimal* solutions, whereas our funnel synthesis methods are designed to seek out nearby *feasible* trajectories.

The differences are somewhat analogous to the differences between solving optimal control problems by using the maximum principle and by using sequential convex programming.

The implicit trajectory optimization methods discussed here also have roots in the field of

robotics. Burridge et al. appear to be the first to have made the explicit connection between Lyapunov functions and a "funnel" [178]. The authors discuss the sequential composition of atomic funnels that are each designed for a single control objective; with the net result of funnel composition being that a robot can achieve a broader control objective (called preimage backchaining [179]). These ideas were advanced significantly in the recent decade by Julius and Pappas, and Tedrake et al. [180, 181, 182, 183]. These works, for the most part, use Sums-of-Squares (SOS) to compute polynomial Lyapunov functions for systems whose dynamics can be represented as polynomial functions of the state and control vectors. SOS optimization connects the search for a polynomial Lyapunov function with semidefinite programming, providing a practical connection between funnel synthesis and convex optimization [184].

Prior to reviewing additional relevant literature (funnel synthesis fuses elements from numerous different fields and has many interpretations in their contexts), it will be useful to provide a formal definition of a funnel.

Definition 2 (Funnel)**.** *A funnel, denoted by $\mathcal{F}(t)$, is a time-varying set in state and control space that is both invariant and lies entirely inside a feasible region.*

The term *funnel synthesis* refers to the algorithmic procedure designed to compute a funnel. The invariance property of a funnel means that if a particular initial condition is inside the entry of the funnel (at some initial time t_0), then the entire subsequent trajectory remains inside the funnel as well [185]. Stated mathematically, if $\big(\boldsymbol{x}(t_0), \boldsymbol{u}(t_0)\big) \in \mathcal{F}(t_0)$ then $\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) \in \mathcal{F}(t)$ for all $t \geq t_0$. Relative to existing definitions of a funnel, e.g., [181], Definition 2 adds the second clause that requires the funnel to lie inside a feasible region defined by a set of state and control constraints.

Based on the guiding question posed at the beginning of this chapter, it should be no surprise that we seek the *largest* possible funnel size. This allows us to implicitly define a large family of trajectories by using a set of functions that will define the funnel – providing the ability to guarantee the availability of a feasible trajectory over a larger region of parameter

variations.

A connection between funnel synthesis and convex optimization is also made in this work by defining a subclass of funnels, called *quadratic funnels*. Instead of using SOS and polynomial dynamics, we use a first-order approximation of the dynamics and derive a differential matrix inequality (DMI) that is similar to a differential Riccati equation. In fact, the algorithms that we propose are reminiscent of the Kleinman iteration [186], or of the D-K iteration [187], but with additional requirements placed on the solution to ensure constraint satisfaction and to maximize the funnel's size.

The state-space portion of a funnel can also be viewed as an outer-approximation of the forward reachable set or an inner-approximation of the backward reachable set; each defined using the control-space portion of the funnel. Ideally, the funnel is as close as possible to the backward reachable set. There has been a great deal of work on reachability analysis for both linear and nonlinear systems subject to bounded control inputs [188]. The reachable set of a nonlinear system is, in theory, obtained by a solution to the Hamilton-Jacobi-Bellman equation [189, 190] – but for the systems we are interested in, the curse of dimensionality precludes us from considering this as a practical solution method. Conservative approximations of nonlinear reachable sets abound, see, e.g., [191], and our methods fit into this category.

The last comparison that we will discuss here is with robust control methods, such as $\mathcal{H}_\infty$ control, robust MPC, and LQG. We stress, however, that the algorithms presented in this chapter are not designed to reject exogenous disturbances – though the formalism can be extended to consider this in a relatively straightforward way – we merely use many of the same mathematical frameworks and have borrowed ideas here and there. Many of the matrix inequalities bear resemblance to differential Riccati equations found in $\mathcal{H}_\infty$ synthesis, and some of the results can be interpreted in the frequency domain by using the KYP lemma [192] under the assumption of time-invariant system matrices. We define a type of structured nonlinearity that (intentionally) has the same form as the structured uncertainty defined by Doyle [193]. We also account for input and output constraints using similar

strategies to robust MPC (RMPC) methods. Time-domain discussions of $\mathcal{H}_\infty$ control also use the exact same general control architecture (and Lyapunov functions) as we do in order to pose the funnel synthesis problem [188, 194, 195] – but the objective of the synthesis steps are fundamentally distinct.

Setting aside the differences in control architecture between RMPC and our open-loop trajectory optimization methods for a moment, the RMPC literature contains many of the same matrix inequalities that we will use. Kothare, Balakrishnan, and Morari were perhaps the first to pose a problem that is similar to what we shall derive for funnel synthesis in [196]. Similar results as those of Kothare, Balakrishnan, and Morari have appeared in subsequent RMPC work [197, 198, 199, 200]. Tube-based MPC, a type of RMPC, is also based on the basic idea of an invariant set that lies strictly inside a feasible region. These methods are typically set-based [201, 202], and many of the minimax results in the literature in fact depend on the use of a (quadratic) Lyapunov function, as well as the definition of quadratic stability that we shall define and use shortly.

Philosophically, a portion of what differentiates this work from others is the intent to *maximize* a controlled-invariant set for a nonlinear dynamical system, as opposed to *minimizing* an invariant set in the presence of disturbances or parametric uncertainty. Funnel synthesis aims to simultaneously approximate the largest invariant set around a nominal trajectory of a nonlinear system while choosing the control policy that maximizes this same invariant set. Funnel synthesis, as it is espoused here, can be viewed as a new combination of old techniques in order to compute feasible solutions for nonconvex optimal control problems in real-time.

This chapter provides three contributions on top of what seems to be known in the broad literature just discussed. First, we obtain three different methods to systematically obtain models of structured nonlinear systems around a nominal trajectory that are conducive to funnel synthesis. So-called "black-box" approaches are derived that are not problem-specific per se, and whenever possible allow an optimization routine to select the least conservative characterization of the nonlinear terms that is consistent with the objective of funnel max-

imization. Second, the temporal parameterization techniques used to derive a set of linear matrix inequalities (LMIs) from a DMI appear to be new. Third, it can be regarded as a contribution of this work that we have shown that funnel synthesis (and by extension, RMPC and reachable set approximation) can be a *practical* tool for high dimensional systems. Having solved numerous problems with ≥ 12 states and ≥ 6 controls, our methods scale well and are able to solve a majority of the problems of interest in the aerospace domain. A preliminary, and necessarily abridged, version of these ideas was presented in [203].

Before presenting the detailed construction of funnel synthesis for nonlinear systems, we highlight the key ideas using a simpler linear time-varying system. This will allow us to define the appropriate terms and outline the main steps necessary to derive LMI-based funnel synthesis algorithms.

6.1 Linear Time-Varying Systems

Suppose that we have the linear time-varying system

$$\dot{\boldsymbol{x}}(t) = A(t)\boldsymbol{x}(t) + B(t)\boldsymbol{u}(t), \quad t \in [t_0, t_f], \tag{6.1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$, $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$. We shall always assume that $A(t)$ and $B(t)$ are stabilizable at each instant in time $t \in [t_0, t_f]$.

Let $\boldsymbol{u}(t) = K(t)\boldsymbol{x}(t)$ for some matrix-valued function of time $K(t) \in \mathbb{R}^{n_u \times n_x}$ so that the closed-loop system becomes

$$\dot{\boldsymbol{x}}(t) = A_{cl}(t)\boldsymbol{x}(t), \tag{6.2}$$

with $A_{cl}(t) \coloneqq A(t) + B(t)K(t)$. A funnel $\mathcal{F}(t)$ must be both invariant and feasible, and so we require that any computed $K(t)$ renders the closed-loop system (6.2) stable. The funnel synthesis techniques that we develop are based on the notion of *quadratic stability* as defined in [188] and [200, 204, 205], the latter of which offer necessary and sufficient conditions for stability based on quadratic Lyapunov functions. Lyapunov stability theory is the most general approach when studying the stability of nonlinear systems [206]. But they

also provide a very practical tool for designing matrices $K(t)$ to achieve certain performance objectives, and invariance of a particular set is one of them. To this end, consider the scalar-valued function $V : \mathbb{R} \times \mathbb{R}^{n_x} \to \mathbb{R}$ defined by

$$V\big(t, \boldsymbol{x}(t)\big) = \boldsymbol{x}(t)^\top Q(t)^{-1} \boldsymbol{x}(t), \tag{6.3}$$

where $Q(t) \in \mathbb{S}^{n_x}_{++}$ is a matrix-valued function of time whose range space lies in the set of positive definite matrices. As a result, we have $V\big(t, \boldsymbol{x}(t)\big) > 0$ for all $t \in [t_0, t_f]$ whenever $\boldsymbol{x}(t) \neq 0$.

We seek a pair $\{Q(t), K(t)\}_{t=t_0}^{t_f}$ that together ensure the function V in (6.3) decays sufficiently fast along trajectories of (6.2). Formally, we can ensure a *decay rate* of at least $\alpha/2$ by choosing $\{Q(t), K(t)\}_{t=t_0}^{t_f}$ so that

$$\dot{V}\big(t, \boldsymbol{x}(t)\big) \leq -\alpha V\big(t, \boldsymbol{x}(t)\big), \quad t \in [t_0, t_f]. \tag{6.4}$$

Having introduced each time-varying term, we henceforth drop the argument of time, t, whenever possible, with the understanding that the preceding discussion can be referenced to specify which terms are time-varying.

The condition (6.4) can be rewritten for system (6.2) as

$$\boldsymbol{x}^\top \left({A_{cl}}^\top Q^{-1} + Q^{-1} A_{cl} - Q^{-1} \dot{Q} Q^{-1} + \alpha Q^{-1} \right) \boldsymbol{x} \leq 0, \tag{6.5}$$

which holds for any $\boldsymbol{x} \in \mathbb{R}^{n_x}$ if and only if

$${A_{cl}}^\top Q^{-1} + Q^{-1} A_{cl} - Q^{-1} \dot{Q} Q^{-1} + \alpha Q^{-1} \preceq 0. \tag{6.6}$$

Multiplying on either side by the positive definite matrix Q leads to the equivalent condition

$$Q {A_{cl}}^\top + A_{cl} Q - \dot{Q} + \alpha Q \preceq 0. \tag{6.7}$$

Expanding A_{cl} and performing the variable substitution $Y = KQ$ then gives

$$QA^\top + AQ + BY + Y^\top B^\top - \dot{Q} + \alpha Q \preceq 0. \tag{6.8}$$

Equation (6.8) is a linear DMI that must be satisfied for all times $t \in [t_0, t_f]$ in order to establish quadratic stability. It is linear in the variables Q and Y.

One of the main ideas underlying funnel synthesis is the use of the matrix decomposition result in Lemma C.2 to rewrite the DMI (6.8) as a set of LMIs. To do so, we require the following assumption regarding the time-varying structure of the problem's matrix components.

Assumption 6.1. *The matrices $Q(t)$, $Y(t)$, $A(t)$ and $B(t)$ can be expressed as*

$$Q(t) = \sum_{i=1}^{n_M} \sigma_i(t) Q_i, \quad Y(t) = \sum_{i=1}^{n_M} \sigma_i(t) Y_i, \quad A(t) = \sum_{i=1}^{n_M} \sigma_i(t) A_i, \quad B(t) = \sum_{i=1}^{n_M} \sigma_i(t) B_i,$$

for some integer $n_M > 1$ and some interpolating functions $\sigma_i(t) \geq 0$ such that $\sum_{i=1}^{n_M} \sigma_i(t) = 1$.

Note that Assumption 6.1 implies that (6.1) can equivalently be written as the linear differential inclusion $\dot{\boldsymbol{x}} \in \mathbf{co}\{A_i \boldsymbol{x} + B_i \boldsymbol{u}\}$, where $\mathbf{co}\{\cdot\}$ denotes the convex hull. This particular linear differential inclusion has appeared often in the literature, and we pause to clarify why we have not obtained a control policy using those existing methods. Typically, one either computes a common matrix Q that stabilizes each pair $\{A_i, B_i\}$ as in [188], or uses spatially-varying piecewise quadratic Lyapunov functions [207, 208, 209]. The key difference here is that we will know a priori the trajectory of the parameter that interpolates the matrices $\{A_i, B_i\}$ – whereas these existing methods do not have this information, and must therefore proceed in a different direction. Interestingly, because we are effectively solving for a single case of the more general system $\dot{\boldsymbol{x}} \in \mathbf{co}\{A_i \boldsymbol{x} + B_i \boldsymbol{u}\}$, the resulting funnel is larger, a consequence of having accounted for fewer possible transitions between system representations.

The main result of this section can be stated as follows.

Theorem 6.2. *Suppose that Assumption 6.1 holds. If there exists a set of positive definite matrices $Q_i \succ 0$ and matrices Y_i for $i = 1, \ldots, n_M$ such that the following matrix inequalities hold:*

$$F_{i,i} - \dot{Q} \preceq 0, \quad i = 1, \ldots, n_M \tag{6.9a}$$

$$F_{i,j} + F_{j,i} - 2\dot{Q} \preceq 0, \quad i = 1, \ldots, n_M - 1, \ j = i+1, \ldots, n_M \tag{6.9b}$$

where,

$$F_{i,j} \coloneqq Q_i A_j^\top + A_j Q_i + B_j Y_i + Y_i^\top B_j^\top + \alpha Q_i, \tag{6.10}$$

then condition (6.8) *holds for all times $t \in [t_0, t_f]$. In this case, the function* (6.3) *is a Lyapunov function that proves quadratic stability of the time-varying system* (6.1) *using the controller $K = YQ^{-1}$.*

Proof. Notice that

$$QA^\top + AQ + BY + Y^\top B^\top + \alpha Q = MN^\top + NM^\top$$

where $M = \begin{bmatrix} A & B & \frac{\alpha}{2} I \end{bmatrix}$ and $N = \begin{bmatrix} Q & Y^\top & Q \end{bmatrix}$. By invoking Lemma C.2 once on the product $MN^\top$ and once on $NM^\top$, we can establish that (6.9) are sufficient for (6.8) to hold. This implies that (6.4) holds, where $V(t, \boldsymbol{x})$ is defined as in (6.3). The closed-loop system under the action of $K = YQ^{-1}$ is therefore quadratically stable and hence stable. □

An important corollary of Theorem 6.2 that ensures both boundedness and sufficient decay of the Lyapunov function and state vector is established in Corollary 6.3. Versions of these same inequalities can be found in [188].

Corollary 6.3. *The Lyapunov function* (6.3) *and the state vector satisfy the following inequalities:*

$$\lambda_{\max}^{-1} \|\boldsymbol{x}\|_2^2 \leq V\big(t, \boldsymbol{x}\big) \leq \lambda_{\min}^{-1} \|\boldsymbol{x}\|_2^2 \tag{6.11a}$$

$$\|\boldsymbol{x}\|_2 \leq \kappa_Q^{1/2} e^{-\frac{\alpha}{2}(t-t_0)} \|\boldsymbol{x}(t_0)\|_2 \tag{6.11b}$$

where $\lambda_{\min}$ *and* $\lambda_{\max}$ *are the minimum/maximum eigenvalues of* Q*, and* $\kappa_Q = {}^{\lambda_{\max}}/_{\lambda_{\min}}$ *is the two-norm condition number of the matrix* Q*.*

Several important remarks can now be made regarding Theorem 6.2. Because we have

$$\dot{Q} = \sum_{i=1}^{n_M} \dot{\sigma}_i(t) Q_i, \tag{6.12}$$

the equations (6.9) are linear in the variables Q_i, Y_i for $i = 1, \ldots, n_M$. The precise formula for $\dot{Q}$ is dependent on the specific choice of interpolating functions $\sigma_i(t)$.

Hu et al. have proposed a similar looking Lyapunov function in [210, 211]. They write $Q(\gamma) = \sum_{i=1}^{n_M} \gamma_i Q_i$ for some parameter $\gamma \in \Sigma^{n_M - 1}$, and then use the Lyapunov function

$$V(t, \boldsymbol{x}) = \min_{\gamma \in \Sigma^{n_M - 1}} \boldsymbol{x}^\top Q(\gamma)^{-1} \boldsymbol{x}. \tag{6.13}$$

The essential difference here is that we *pre-suppose* a *time-varying* function $\sigma(t) \in \Sigma^{n_M - 1}$, and commit to the Lyapunov function that results from this selection. We do not allow γ to be defined by the value of $\boldsymbol{x}$ via the minimization problem in (6.13). In effect, our approach is equivalent to using

$$V(t, \boldsymbol{x}) = \min_{\gamma \in \Sigma^{n_M - 1}, \gamma = \sigma(t)} \boldsymbol{x}^\top Q(\gamma)^{-1} \boldsymbol{x}.$$

The difference is that in our case, Q can be viewed as a *temporal* convex combination, whereas that of Hu et al. results in a *spatial* convex combination. That is, the parameterization of the matrix Q used in our formulation is *not state-dependent*. Presupposing the time-varying function $\sigma(t)$ in this way is beneficial (and indeed crucial) because: 1) we will need to know the value of Q at any given time $t \in [t_0, t_f]$ to enforce constraints during the funnel synthesis procedure, and 2) it will be most beneficial to allow temporal variations in Q to account for temporal variations in the matrices A and B, in particular for trajectory generation where the distance to a constraint boundary may change temporally. If the value of Q is tied to

the value of $\boldsymbol{x}$ through the minimization (6.13), then one would need to constrain *each* Q_i according to the minimum distance to a constraint boundary to ensure feasibility of the funnel $\mathcal{F}(t)$. A temporal parameterization, by contrast, would constrain *only* the Q_i that corresponds to the temporal point where the nominal trajectory is closest to a constraint boundary. This can offer a significant improvement in the computable funnel volume.

As a final remark on Theorem 6.2, we note that its conditions are not necessary for (6.8) to hold. This is primarily due to the fact that any necessary conditions must be dependent on the choice of interpolating functions $\sigma(t)$, whereas the sufficient conditions of Theorem 6.2 are agnostic to this choice. We can also provide a simple counterexample for why they are not necessary. If we have $n_M = 2$, $t \in [0, 1]$, $\sigma_1(t) = 1 - t$, and $\sigma_2(t) = t$, then $\dot{Q} = Q_2 - Q_1$ and evaluating (6.8) at $t = \left\{0, \frac{1}{2}, 1\right\}$ shows that it is necessary for

$$F_{i,i} - (Q_2 - Q_1) \preceq 0 \quad i = 1, 2 \tag{6.14a}$$

$$F_{1,2} + F_{2,1} - 2(Q_2 - Q_1) \preceq 2(Q_2 - Q_1) - F_{1,1} - F_{2,2}. \tag{6.14b}$$

The right-hand side of (6.14b) is positive semidefinite due to (6.14a). Hence for this example, it would be more restrictive than necessary to require that $F_{1,2} + F_{2,1} - 2(Q_2 - Q_1) \preceq 0$ as called for by Theorem 6.2. It is straightforward to find numerical examples that support this claim.

6.1.1 Quadratic Funnels

Using the preliminary notation introduced in this section, we formally define a quadratic funnel – the specific class of funnels that is obtained by using quadratic stability.

First, we note that level sets of the Lyapunov function (6.3) are *invariant sets*, in the sense that if $V(t_0, \boldsymbol{x}(t_0)) \leq \rho$ then $V(t, \boldsymbol{x}(t)) \leq \rho$ for all $t \geq t_0$ for the closed-loop system [188]. The 1-level set of $V(t, \boldsymbol{x})$ is the set of states that satisfy the quadratic inequality $\boldsymbol{x}^\top Q^{-1} \boldsymbol{x} \leq 1$, which is also the equation of a non-degenerate n_x-dimensional ellipsoid. We denote the

ellipsoid defined by the positive definite matrix Q and centered at the origin as

$$\mathcal{E}_Q = \left\{ \boldsymbol{x} \in \mathbb{R}^{n_x} \,|\, \boldsymbol{x}^\top Q^{-1} \boldsymbol{x} \leq 1 \right\} = \left\{ Q^{1/2} \boldsymbol{w} \,|\, \|\boldsymbol{w}\|_2 \leq 1 \right\}. \tag{6.15}$$

If $\boldsymbol{x} \in \mathcal{E}_Q$, then $C\boldsymbol{x} \in \mathcal{E}_{CQC^\top}$, a fact that can be proved easily via Schur complements when C is full row-rank.[2] The assumption that $\boldsymbol{u} = K\boldsymbol{x}$ used to form the closed-loop system (6.2) thus results in the following implication:

$$\boldsymbol{x} \in \mathcal{E}_Q \quad \Rightarrow \quad \boldsymbol{u} \in \mathcal{E}_{KQK^\top}. \tag{6.16}$$

Suppose that $\mathcal{X} \subset \mathbb{R}^{n_x}$ and $\mathcal{U} \subset \mathbb{R}^{n_u}$ are the (possibly nonconvex) sets of feasible state and control vectors, as they were for the general optimal control setting in Problem 1. Using these feasible sets, we can formally define a quadratic funnel.

Definition 3 (Quadratic Funnel)**.** *A quadratic funnel, $\mathcal{F}(t)$, is a set in state and control space that is parameterized by a time-varying positive definite matrix $Q(t) \in \mathbb{S}^{n_x}_{++}$ and a time-varying matrix $K(t) \in \mathbb{R}^{n_u \times n_x}$. Specifically, we have*

$$\mathcal{F} = \mathcal{E}_Q \times \mathcal{E}_{KQK^\top} \quad \textit{and} \quad \mathcal{E}_Q \subseteq \mathcal{X}, \quad \mathcal{E}_{KQK^\top} \subseteq \mathcal{U}. \tag{6.17}$$

We call $K(t)$ the correction law associated with the quadratic funnel.

The invariance of a quadratic funnel is a consequence of the invariance of the 1-level set of the Lyapunov function $V(t, \boldsymbol{x})$. The feasibility of a quadratic funnel is a consequence of the set inclusions on the right half of (6.17). The funnel synthesis techniques presented in the remainder of this chapter are designed to compute quadratic funnels for nonlinear systems. Definition 2 (funnels) and Definition 3 (quadratic funnels) are intended to mirror the definitions of stability and quadratic stability. Figure 6.1 provides an illustration of the quadratic funnel concept.

---

[2]When C is not full row-rank, the ellipsoid $\mathcal{E}_{CQC^\top}$ is a degenerate ellipsoid.

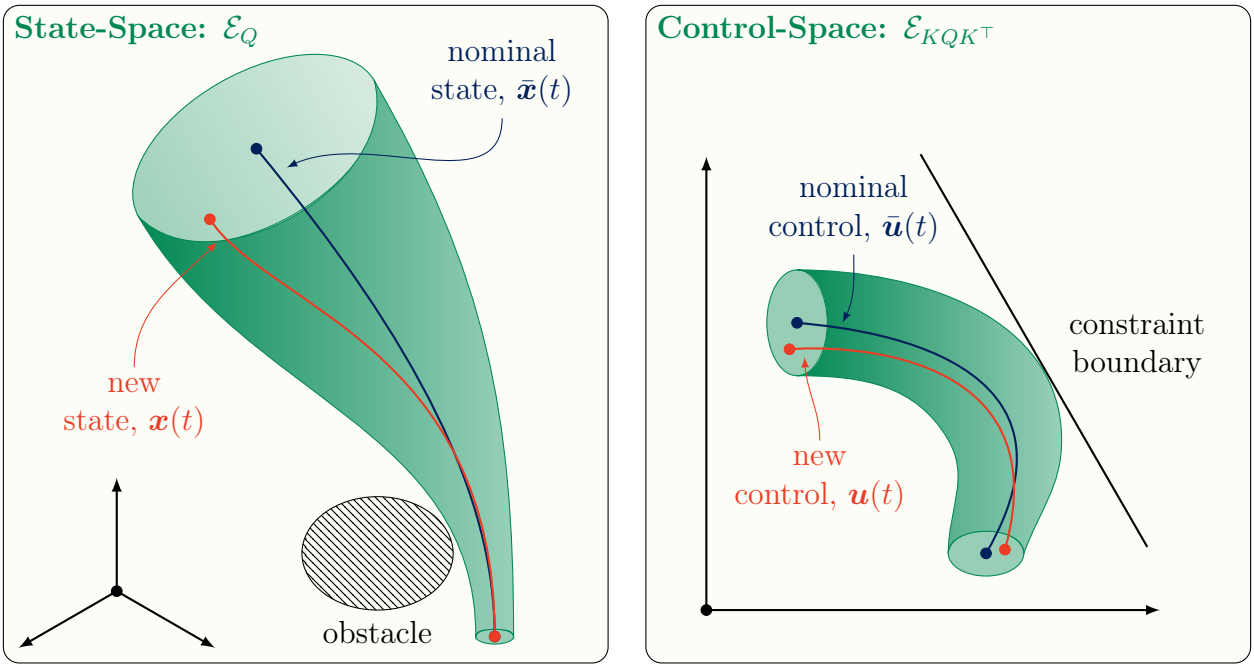


Figure 6.1: A depiction of a quadratic funnel in both state- and control-spaces. At each instant of time, the quadratic funnel is an ellipsoid (green) centered around the nominal trajectory (blue) that lies inside the feasible region. Any new trajectory (red) that starts in the funnel will remain in the funnel.

As mentioned before, part of what differentiates this work from others is the intent to maximize the (quadratic) funnel, as opposed to minimize an invariant set in the presence of disturbances or uncertainty. We note that the work of Polyak et al. [212, 213] tackles the latter strategy using similar invariant ellipsoid methods.

6.2 Nonlinear System Models

Having defined the basic terminology, we now proceed to develop a quadratic funnel synthesis technique for nonlinear systems. We begin by deriving the nonlinear model used to synthesize

a funnel around a nominal trajectory.[3] Consider the autonomous nonlinear dynamics

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}), \quad t \in [t_0, t_f], \tag{6.18}$$

where $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and $\boldsymbol{u} \in \mathbb{R}^{n_u}$ represent the state and control signals. We assume that f is at least once differentiable. We assume that t_f is fixed throughout this chapter.

Because f is differentiable, we can write

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + g(\boldsymbol{x}, \boldsymbol{u}) \tag{6.19}$$

for some nonlinear function g. Suppose then that we have a trajectory $\{\bar{\boldsymbol{x}}(t), \bar{\boldsymbol{u}}(t)\}_{t=t_0}^{t_f}$ that may or may not satisfy (6.18) exactly. Define

$$\boldsymbol{\eta}(t) \coloneqq \boldsymbol{x}(t) - \bar{\boldsymbol{x}}(t) \quad \text{and} \quad \boldsymbol{\xi}(t) \coloneqq \boldsymbol{u}(t) - \bar{\boldsymbol{u}}(t). \tag{6.20}$$

Then using (6.19) we have

$$\dot{\boldsymbol{\eta}} = A\boldsymbol{\eta} + B\boldsymbol{\xi} + g(\boldsymbol{x}, \boldsymbol{u}) - g(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}). \tag{6.21}$$

It is true that for any nonlinear function $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$, there exists another (nonlinear) function $\phi : \mathbb{R}^{n_q} \to \mathbb{R}^{n_p}$, where $n_q \leq (n_x + n_u)$ and $n_p \leq n_x$ such that

$$g(\boldsymbol{x}, \boldsymbol{u}) = E\phi\big(C\boldsymbol{x} + D\boldsymbol{u}\big) \tag{6.22}$$

for constant matrices $E \in \mathbb{R}^{n_x \times n_p}$, $C \in \mathbb{R}^{n_q \times n_x}$ and $D \in \mathbb{R}^{n_q \times n_u}$. Using this fact, we can write (6.21) as

$$\dot{\boldsymbol{\eta}} = A\boldsymbol{\eta} + B\boldsymbol{\xi} + E\left(\phi\big(C\boldsymbol{x} + D\boldsymbol{u}\big) - \phi\big(C\bar{\boldsymbol{x}} + D\bar{\boldsymbol{u}}\big)\right). \tag{6.23}$$

---

[3] The nominal trajectory in the preceding linear time-varying system analysis can be understood as simply the origin. This is generalized to arbitrary time-varying trajectories for the study of nonlinear systems.

Define now

$$\boldsymbol{q}_x \coloneqq C\boldsymbol{x} + D\boldsymbol{u}, \quad \bar{\boldsymbol{q}} \coloneqq C\bar{\boldsymbol{x}} + D\bar{\boldsymbol{u}} \quad \Rightarrow \quad \boldsymbol{q} = \boldsymbol{q}_x - \bar{\boldsymbol{q}} = C\boldsymbol{\eta} + D\boldsymbol{\xi} \tag{6.24a}$$

$$\boldsymbol{p}_x \coloneqq \phi(\boldsymbol{q}_x), \qquad \bar{\boldsymbol{p}} \coloneqq \phi(\bar{\boldsymbol{q}}) \qquad \Rightarrow \quad \boldsymbol{p} = \boldsymbol{p}_x - \bar{\boldsymbol{p}} = \phi(\boldsymbol{q}_x) - \phi(\bar{\boldsymbol{q}}) \tag{6.24b}$$

We now make two assumptions on the nature of the nominal trajectory that will lead to the final model used by the funnel synthesis procedures.

Assumption 6.4. *The trajectory* $\{\bar{\boldsymbol{x}}(t), \bar{\boldsymbol{u}}(t)\}_{t=t_0}^{t_f}$ *satisfies the differential equation* (6.18) *exactly for some initial condition* $\boldsymbol{x}_{ic}$ *such that* $\bar{\boldsymbol{x}}(t_0) = \boldsymbol{x}_{ic}$.

This assumption is not explicitly necessary, but is satisfied nonetheless by trajectories obtained using the SCP-based explicit trajectory optimization methods described in Chapter 4 and it permits an overall cleaner exposition. All of the subsequent steps can proceed without this assumptions. The important consequence of Assumption 6.4 is that $g(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) = 0$ when A and B are chosen as the partial derivatives of f evaluated along the nominal trajectory. To simplify the presentation, let us abuse notation slightly and write the following:

$$\boldsymbol{p} = \phi(\boldsymbol{q}) \equiv \phi(\bar{\boldsymbol{q}} + \boldsymbol{q}) - \phi(\bar{\boldsymbol{q}}).$$

Assumption 6.4 then implies that $E\phi(\bar{\boldsymbol{q}}) = 0$ and hence $E\boldsymbol{p} = E\phi(\bar{\boldsymbol{q}} + \boldsymbol{q})$. Consequently, the dynamical system (6.18) can be expressed *equivalently* as the following system:

$$\begin{aligned} \dot{\boldsymbol{\eta}}(t) &= A(t)\boldsymbol{\eta}(t) + B(t)\boldsymbol{\xi}(t) + E\boldsymbol{p}(t) \\ \boldsymbol{q}(t) &= C\boldsymbol{\eta}(t) + D\boldsymbol{\xi}(t) \\ \boldsymbol{p}(t) &= \phi\big(\boldsymbol{q}(t)\big) \end{aligned} \tag{6.25}$$

where we have explicitly added the argument of time to emphasize which terms are time-varying and which are not.

6.2.1 Structured Nonlinearity

We can think of the model (6.25) as being composed of a single *nonlinear channel* that takes $\boldsymbol{q}$ as an input and provides $\boldsymbol{p}$ as an output. By grouping all of the nonlinear effects into a single channel, we are effectively assuming that the nonlinear term $g(\boldsymbol{x}, \boldsymbol{u})$ is *unstructured*, or that we have no further information about how the nonlinear effects are connected to the chosen state and control vectors and underlying dynamical system. In practice, when the dynamical system is derived in part from first principles physical laws, we often have a deeper understanding of the system that we can use to our benefit. While unstructured nonlinearity offers simplicity, it is often better, in the context of funnel synthesis, to decompose the nonlinear terms described by $g(\boldsymbol{x}, \boldsymbol{u})$ into several nonlinear channels.

As a simple example, consider a system whose dynamics are $\dot{x}_1 = x_2$ and $\dot{x}_2 = x_1 \sin x_2$. Only the second state has nonlinear components, but it is composed of two nonlinear channels: one from x_1 to x_2 and another from x_2 to x_2. Suppose that we have a nominal trajectory given by $\bar{x}_1$ and $\bar{x}_2$. We can illustrate the benefit of considering these channels separately by examining the Lipschitz constants of the nonlinear functions $g_1(x_1) = x_1 \sin \bar{x}_2$ and $g_2(x_2) = \bar{x}_1 \sin x_2$ compared to that of $g(\boldsymbol{x}) = x_1 \sin x_2$. On the bounded interval $-\pi \leq \boldsymbol{x} \leq \pi$, the function g_1 has a local Lipschitz constant of $|\sin \bar{x}_2| \leq 1$, and g_2 has a local Lipschitz constant of $|\bar{x}_1| \leq \pi$. On this same bounded set, the Lipschitz constant of the function g is *equal to* π. Therefore if we look at each nonlinear channel separately, the nominal trajectory can be leveraged to reduce the Lipschitz constant. Colloquially, these Lipschitz constants can be understood as proxies for the "severity" of the nonlinearity in the neighborhood of the nominal trajectory. For funnel generation, a reduction in the Lipschitz constant(s) obtained by exploiting the particular structure of the nonlinearities can lead to significant improvements in terms of computable funnel size, especially as we move to systems with complex dynamics and many states.[4]

[4] A caveat to this motivating example: we are not so much interested in the Lipschitz constant of f in the dynamics $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ as we allude to here, but in local Lipschitz constants for the second- and higher-order components of f. The example merely provides the correct intuition without being overly complicated.

Rather than simply stating the model (6.25) expanded to include several additional terms $\boldsymbol{q}_i$ and $\boldsymbol{p}_i$, it is beneficial to provide a more formal derivation. This leads to an understanding of how one can select and construct the appropriate nonlinear channels. For notational simplicity, let us consider the nonlinear function $g : \mathbb{R}^{n_z} \to \mathbb{R}^{n_x}$, where $\mathbb{R}^{n_z} = \mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$. Let $\boldsymbol{z}, \bar{\boldsymbol{z}} \in \mathbb{R}^{n_z}$ and decompose g into K vector-valued functions so that

$$g(\boldsymbol{z}) - g(\bar{\boldsymbol{z}}) = \begin{bmatrix} g_1(\boldsymbol{z}) - g_1(\bar{\boldsymbol{z}}) \\ \vdots \\ g_K(\boldsymbol{z}) - g_K(\bar{\boldsymbol{z}}) \end{bmatrix} = \sum_{k=1}^{K} \tilde{E}_k \left(g_k(\boldsymbol{z}) - g_k(\bar{\boldsymbol{z}}) \right) \tag{6.26}$$

where each $g_k : \mathbb{R}^{n_z} \to \mathbb{R}^{m_k}$ and $\tilde{E}_k \in \mathbb{R}^{n_x \times m_k}$ are constant matrices. Note that $\sum_{k=1}^{K} m_k = n_x$. Focusing on the k^{th} entry for a moment, let $\mathcal{I}_j \subset \{1, \ldots, n_z\}$ be some subset of indices, and let $\boldsymbol{z}_{\mathcal{I}_j}$ be the entries of the vector $\boldsymbol{z}$ that correspond to the indices in $\mathcal{I}_j$. Then it is true that we can write

$$g_k(\boldsymbol{z}) - g_k(\bar{\boldsymbol{z}}) = g_k\left(\boldsymbol{z}_{\mathcal{I}_1}, \ldots, \boldsymbol{z}_{\mathcal{I}_{J_k}}\right) - g_k\left(\bar{\boldsymbol{z}}_{\mathcal{I}_1}, \ldots, \bar{\boldsymbol{z}}_{\mathcal{I}_{J_k}}\right) \tag{6.27}$$

where the index sets $\{\mathcal{I}_j\}_{j=1}^{J_k}$ are mutually distinct but need not cover all of $\{1, \ldots, n_z\}$. This simply means that g_k does not necessarily depend on all entries of the vector $\boldsymbol{z}$, and we can collect the entries that it does depend on into independent groupings. By cleverly adding and subtracting new terms, we can rewrite (6.27) as

$$g_k(\boldsymbol{z}) - g_k(\bar{\boldsymbol{z}}) = \sum_{j=1}^{J_k} g_k\left(\bar{\boldsymbol{z}}_{\mathcal{I}_1}, \ldots, \boldsymbol{z}_{\mathcal{I}j}, \ldots, \boldsymbol{z}_{\mathcal{I}_{J_k}}\right) - g_k\left(\bar{\boldsymbol{z}}_{\mathcal{I}_1}, \ldots, \bar{\boldsymbol{z}}_{\mathcal{I}_j}, \ldots, \bar{\boldsymbol{z}}_{\mathcal{I}_{J_k}}\right), \tag{6.28}$$

where each summand only has one set of (grouped) indices that are different. Now, invoking (6.22) on each term in the summand we have

$$g_k(\boldsymbol{z}) - g_k(\bar{\boldsymbol{z}}) = \sum_{j=1}^{J_k} \tilde{E}_{kj} \left(\phi_{kj}\left(C_{kj}\boldsymbol{z}\right) - \phi_{kj}\left(C_{kj}\bar{\boldsymbol{z}}\right) \right), \tag{6.29}$$

where $\phi_{kj} : \mathbb{R}^{n_{p_j}} \to \mathbb{R}^{n_{q_j}}$ is a nonlinear function and $\tilde{E}_{kj} \in \mathbb{R}^{m_k \times n_{p_j}}$ and $C_{kj} \in \mathbb{R}^{n_{q_j} \times n_z}$ are constant matrices. Note that we've used n_{q_j} to represent the number of indices contained in $\mathcal{I}_j$ (i.e., its cardinality). Consequently, we see that C_{kj} is simply a matrix that extracts the desired entries from $\boldsymbol{z}$ – and so its entries can be (without loss of generality) assumed to be either zero or one. The same is true of $\tilde{E}_{kj}$ and $\tilde{E}_k$. Lastly, we can substitute (6.29) into (6.26) and simplify to get

$$\begin{aligned} g(\boldsymbol{z}) - g(\bar{\boldsymbol{z}}) &= \sum_{k=1}^{K} \tilde{E}_k \left(\sum_{j=1}^{J_k} \tilde{E}_{kj} \left(\phi_{kj} \left(C_{kj} \boldsymbol{z} \right) - \phi_{kj} \left(C_{kj} \bar{\boldsymbol{z}} \right) \right) \right) \\ &= \sum_{k=1}^{K} \sum_{j=1}^{J_k} E_{kj} \left(\phi_{kj} \left(C_{kj} \boldsymbol{z} \right) - \phi_{kj} \left(C_{kj} \bar{\boldsymbol{z}} \right) \right) \end{aligned} \tag{6.30}$$

where $E_{kj} = \tilde{E}_k \tilde{E}_{kj} \in \mathbb{R}^{n_x \times n_{p_j}}$. By relabeling terms using the linear index $i \leftarrow \left(\sum_{l=1}^{k} J_l \right) + j$, the sum in (6.30) is equivalent to

$$g(\boldsymbol{z}) - g(\bar{\boldsymbol{z}}) = \sum_{i=1}^{N_p} E_i \left(\phi_i(C_i \boldsymbol{z}) - \phi_i(C_i \bar{\boldsymbol{z}}) \right) \tag{6.31}$$

with $N_p = \sum_{k=1}^{K} J_k$. (The notation N_p is used to differentiate between the *number* of nonlinear channels, and their individual *dimensions*, which are n_{p_j}.) Equation (6.31) is a structured decomposition of the nonlinear function g, where the matrices E_i and C_i can be selected to reflect the particular system at hand. This so-called *structured nonlinearity* (see [193]) affords greater modeling fidelity and has a profound impact on our ability to synthesize funnels for dynamical systems with complex nonlinear equations of motion.

Recall that we substituted $\boldsymbol{z}$ for the stacked state and control vectors and so $\boldsymbol{z}$ composed of $\boldsymbol{x}$ and $\boldsymbol{u}$. Overloading the matrix C_i, we therefore have $C_i \boldsymbol{z} = C_i \boldsymbol{x} + D_i \boldsymbol{u}$ – the former (left-hand side) definition was used solely for the derivation and only the latter (right-hand side) will be used in the analysis that follows to avoid any confusion. Let us therefore make

the analogous definitions to (6.24) as

$$\boldsymbol{q}_{x,i} \coloneqq C_i\boldsymbol{x} + D_i\boldsymbol{u}, \quad \bar{\boldsymbol{q}}_i \coloneqq C_i\bar{\boldsymbol{x}} + D_i\bar{\boldsymbol{u}} \quad \Rightarrow \quad \boldsymbol{q}_i = \boldsymbol{q}_{x,i} - \bar{\boldsymbol{q}}_i = C_i\boldsymbol{\eta} + D_i\boldsymbol{\xi} \tag{6.32a}$$

$$\boldsymbol{p}_{x,i} \coloneqq \phi(\boldsymbol{q}_{x,i}), \qquad \bar{\boldsymbol{p}}_i \coloneqq \phi(\bar{\boldsymbol{q}}_i) \qquad \Rightarrow \quad \boldsymbol{p}_i = \boldsymbol{p}_{x,i} - \bar{\boldsymbol{p}}_i = \phi(\boldsymbol{q}_{x,i}) - \phi(\bar{\boldsymbol{q}}_i) \tag{6.32b}$$

Invoking Assumption 6.4 and working through the same consequences as done for the single nonlinear channel case, we can express the dynamical system (6.18) equivalently using structured nonlinearity as

$$\begin{aligned} \dot{\boldsymbol{\eta}}(t) &= A(t)\boldsymbol{\eta}(t) + B(t)\boldsymbol{\xi}(t) + \textstyle\sum_{i=1}^{N_p} E_i\boldsymbol{p}_i(t) \\ \boldsymbol{q}_i(t) &= C_i\boldsymbol{\eta}(t) + D_i\boldsymbol{\xi}(t), && i = 1, \dots, N_p \\ \boldsymbol{p}_i(t) &= \phi_i\big(\boldsymbol{q}_i(t)\big), && i = 1, \dots, N_p \end{aligned} \tag{6.33}$$

where we have explicitly added the argument of time to re-emphasize which terms are time-varying and which are not. Lastly, we can write the system (6.33) in a more compact form by making the definitions

$$\boldsymbol{p} = \begin{bmatrix} \boldsymbol{p}_1 \\ \vdots \\ \boldsymbol{p}_{N_p} \end{bmatrix}, \quad \boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_1 \\ \vdots \\ \boldsymbol{q}_{N_p} \end{bmatrix}, \quad C = \begin{bmatrix} C_1 \\ \vdots \\ C_{N_p} \end{bmatrix}, \quad D = \begin{bmatrix} D_1 \\ \vdots \\ D_{N_p} \end{bmatrix}, \quad E = \begin{bmatrix} E_1 & \cdots & E_{N_p} \end{bmatrix}. \tag{6.34}$$

Here, $\boldsymbol{p} \in \mathbb{R}^{n_p}$ and $\boldsymbol{q} \in \mathbb{R}^{n_q}$, where $n_p = \sum_{j=1}^{N_p} n_{p_j}$ and $n_q = \sum_{j=1}^{N_p} n_{q_j}$, and hence $C \in \mathbb{R}^{n_q \times n_x}$, $D \in \mathbb{R}^{n_q \times n_u}$ and $E \in \mathbb{R}^{n_x \times n_p}$. By constructing an appropriate function $\phi : \mathbb{R}^{n_q} \to \mathbb{R}^{n_p}$ by stacking each of the ϕ_i, we can write (6.33) as

$$\begin{aligned} \dot{\boldsymbol{\eta}}(t) &= A(t)\boldsymbol{\eta}(t) + B(t)\boldsymbol{\xi}(t) + E\boldsymbol{p}(t) \\ \boldsymbol{q}(t) &= C\boldsymbol{\eta}(t) + D\boldsymbol{\xi}(t) \\ \boldsymbol{p}(t) &= \phi\big(\boldsymbol{q}(t)\big) \end{aligned} \tag{6.35}$$

Because this system has precisely the same form as (6.25), we proceed to develop all of the theory using this standard form model. Whenever necessary, we shall point out any special considerations that must be taken into account when using structured nonlinearity in place of its unstructured counterpart.

6.3 The Quadratic Funnel Synthesis Problem

System (6.35) can be thought of as the LTV system (6.1) augmented with N_p nonlinear feedback terms whose inputs are the $\boldsymbol{q}_i$ and whose outputs are the $\boldsymbol{p}_i$. Because of Assumption 6.4, the quadratic stability analysis presented in §6.1 is a necessary condition for the system (6.35) to be quadratically stable.

Consider the quadratic function (6.3) again with $Q(t) \succ 0$ defined to be a matrix-valued function of time. Our objective is to obtain a quadratic funnel parameterized by a pair $\{Q(t), K(t)\}_{t=t_0}^{t_f}$, where $K(t)$ is the correction law associated with the quadratic funnel. Applying the correction law to the system (6.35) results in the closed-loop system

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= A_{cl}\boldsymbol{\eta} + E\boldsymbol{p} \\ \boldsymbol{q} &= C_{cl}\boldsymbol{\eta} \\ \boldsymbol{p} &= \phi(\boldsymbol{q}) \end{aligned} \tag{6.36}$$

where $A_{cl} \coloneqq A + BK$ and $C_{cl} \coloneqq C + DK$. Using the system model (6.36), we can split the discussion of the invariance and feasibility properties of a quadratic funnel into two parts. We begin with invariance.

6.3.1 Invariance of a Quadratic Funnel

For the system model (6.36), the condition that must be met to achieve quadratic stability is

$$\dot{V}(t, \boldsymbol{x}) \leq -\alpha V(t, \boldsymbol{x}), \quad \forall\, t \in [t_0, t_f], \quad \forall\, \boldsymbol{q} \in \mathcal{E}_{C_{cl}QC_{cl}^\top},\ \boldsymbol{p} = \phi(\boldsymbol{q}) \tag{6.37}$$

The invariance property of a quadratic funnel, achieved as a consequence of (6.37), leads to the set inclusion $\boldsymbol{\eta} \in \mathcal{E}_Q$. This implies that the vector $\boldsymbol{q}$, which is the input to the nonlinear terms in system (6.36), satisfies the set inclusion

$$\boldsymbol{q} \in \mathcal{E}_{C_{cl} Q C_{cl}^\top}. \tag{6.38}$$

To use the condition (6.37) to synthesize a quadratic funnel, we need to express the condition "$\forall\, \boldsymbol{q} \in \mathcal{E}_{C_{cl} Q C_{cl}^\top},\ \boldsymbol{p} = \phi(\boldsymbol{q})$" in such a way that it is consistent with the quadratic Lyapunov function $V(t, \boldsymbol{x})$. To this end, we use *multiplier matrices* and the $\mathcal{S}$-procedure.

Multiplier matrices and quadratic Lyapunov functions as we are using them have been an object of study at least since Yakubovich in the 1960s [214, 215]. Much of the nomenclature (and notation) that we use, however, is based on the work of Corless and Açıkmeşe [204].

Multiplier matrices as defined in [204] account for the range space of ϕ, but require (6.39) to hold for all $\boldsymbol{q} \in \mathbb{R}^{n_q}$. Because the funnel synthesis procedure results in a bounded set of possible *inputs* to the nonlinear term, via (6.38), we are able to refine the definition of multiplier matrices to a (new) "local" version.

Definition 4 (Local Multiplier Matrix)**.** *Let $\phi : \mathbb{R}^{n_q} \to \mathbb{R}^{n_p}$ be a nonlinear map such that $\boldsymbol{q} \mapsto \phi(\boldsymbol{q})$. A symmetric matrix $M \in \mathbb{S}^{(n_q+n_p)}$ is a local multiplier matrix for ϕ over the set Ω if and only if*

$$\mu(\boldsymbol{q}) \coloneqq \begin{bmatrix} \boldsymbol{q} \\ \phi(\boldsymbol{q}) \end{bmatrix}^\top M \begin{bmatrix} \boldsymbol{q} \\ \phi(\boldsymbol{q}) \end{bmatrix} \geq 0, \quad \textit{for all } \boldsymbol{q} \in \Omega. \tag{6.39}$$

Denote the set of local multiplier matrices for ϕ over the set Ω by

$$\mathcal{M}_{\phi,\Omega} = \left\{ M \in \mathbb{S}^{(n_q+n_p)} \;\middle|\; \mu(\boldsymbol{q}) \geq 0 \ \textit{for all } \boldsymbol{q} \in \Omega \right\}. \tag{6.40}$$

Note that a multiplier matrix is a local multiplier matrix, but the converse is not necessarily true. The set $\mathcal{M}_{\phi,\Omega}$ is a convex cone. Importantly, however, matrices $M \in \mathcal{M}_{\phi,\Omega}$ are not necessarily positive semidefinite (the set of which is also a convex cone). This is

because we only require that (6.39) holds *over the subset* of $\mathbb{R}^{(n_q+n_p)}$ defined by the set Ω and the range space of the function ϕ. A positive semidefinite matrix M would of course immediately satisfy the definition (6.39), but would enforce the inequality over all of $\mathbb{R}^{(n_q+n_p)}$ and provide no characterization of the nonlinearity. As a result, using a positive semidefinite matrix would be overly conservative (and as we will see, would lead to an infeasible funnel synthesis problem).

Using local multiplier matrices, we can rewrite the quadratic stability condition (6.37) as

$$\dot{V}(t,\boldsymbol{x}) \leq -\alpha V(t,\boldsymbol{x}), \quad \forall\, t \in [t_0, t_f], \quad \forall \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix}^\top M \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{bmatrix} \geq 0, \tag{6.41}$$

for some $M \in \mathcal{M}_{\phi,\mathcal{E}_{C_{cl}QC_{cl}{}^\top}}$. Expanding this condition in terms of the closed-loop system (6.36) leads to

$$\begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{p} \end{bmatrix}^\top \begin{bmatrix} A_{cl}{}^\top Q^{-1} + Q^{-1}A_{cl} - Q^{-1}\dot{Q}Q^{-1} + \alpha Q^{-1} & Q^{-1}E \\ E^\top Q^{-1} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{p} \end{bmatrix} \leq 0,$$
$$\text{for all} \quad \begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{p} \end{bmatrix}^\top \begin{bmatrix} C_{cl}{}^\top & 0 \\ 0 & I \end{bmatrix} M \begin{bmatrix} C_{cl} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{p} \end{bmatrix} \geq 0 \tag{6.42}$$

Using the $\mathcal{S}$-procedure, the condition (6.42) holds if and only if there exists a scalar $\lambda \geq 0$ such that

$$\begin{bmatrix} A_{cl}{}^\top Q^{-1} + Q^{-1}A_{cl} - Q^{-1}\dot{Q}Q^{-1} + \alpha Q^{-1} + \lambda C_{cl}{}^\top M_{11} C_{cl} & Q^{-1}E + \lambda C_{cl}{}^\top M_{12} \\ E^\top Q^{-1} + \lambda M_{12}^\top C_{cl} & \lambda M_{22} \end{bmatrix} \preceq 0, \tag{6.43}$$

where we have used the block decomposition

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{12}^\top & M_{22} \end{bmatrix} \in \mathcal{M}_{\phi,\mathcal{E}_{C_{cl}QC_{cl}{}^\top}}. \tag{6.44}$$

Remark 6.5. *In the case of multiple nonlinearities, $N_p > 1$, the block decomposition shown in* (6.44) *has a banded structure. Each matrix M_{11}, M_{12} and M_{22} is itself a block-diagonal matrix, with the size of each block determined by the dimension of the respective nonlinear channel, denoted by n_{q_j} and n_{p_j} in §6.2.1. This follows by constructing an independent characterization of each nonlinear channel by using a suitable multiplier matrix, and the definitions in* (6.34)*.*

For synthesis using convex optimization, we must pose the matrix inequality (6.43) as an LMI in the variable Q, not its inverse [188]. Pre- and post-multiplying (6.43) by $\mathbf{diag}\{Q, I\}$ yields the equivalent condition

$$\begin{bmatrix} QA_{cl}^{\top} + A_{cl}Q - \dot{Q} + \alpha Q + \lambda QC_{cl}^{\top}M_{11}C_{cl}Q & E + \lambda QC_{cl}^{\top}M_{12} \\ E^{\top} + \lambda M_{12}^{\top}C_{cl}Q & \lambda M_{22} \end{bmatrix} \preceq 0. \tag{6.45}$$

Immediately we can see that to satisfy (6.45) requires $M_{22} \preceq 0$.

Remark 6.6. *Because $\mathcal{M}_{\phi,\Omega}$ is a convex cone, it is true that if $M \in \mathcal{M}_{\phi,\Omega}$ then $\lambda M \in \mathcal{M}_{\phi,\Omega}$ for $\lambda \geq 0$. It is therefore possible to exclude the parameter λ in* (6.45)*, as can be seen in most works [204, 205, 216]. However, for numerical solution purposes, we keep the parameter. Empirical evidence across a variety of examples supports that numerical solutions can suffer numerical problems when λ is omitted.*

The matrix inequality (6.45) is a nonlinear DMI in four variables: the matrices Q, K, M and the scalar λ. For later use, we reformulate (6.45) into another nonlinear DMI by expanding the closed-loop system matrices and using the same change of variables $Y = KQ$ as done to derive (6.8). In particular, we have that (6.45) can be rewritten equivalently as

$$\begin{bmatrix} F - \dot{Q} + \lambda G^{\top}M_{11}G & E + \lambda G^{\top}M_{12} \\ E^{\top} + \lambda M_{12}^{\top}G & \lambda M_{22} \end{bmatrix} \preceq 0,\ Q \succ 0,\ \lambda \geq 0, \tag{6.46}$$

for some $M \in \mathcal{M}_{\phi,\mathcal{E}_{GQ^{-1}G^\top}}$ where,

$$F = QA^\top + AQ + BY + Y^\top B^\top + \alpha Q, \tag{6.47a}$$

$$G = CQ + DY. \tag{6.47b}$$

Satisfaction of the nonlinear DMI (6.45) or (6.46) is the condition that provides the invariance property of a quadratic funnel.

6.3.2 Feasibility of a Quadratic Funnel

We now discuss the conditions that must be met in order for a quadratic funnel to be contained in a given feasible region. Suppose again that $\mathcal{X} \subset \mathbb{R}^{n_x}$ and $\mathcal{U} \subset \mathbb{R}^{n_u}$ are the (possibly nonconvex) sets of feasible state and control vectors. We assume that the nominal trajectory satisfies $\bar{\boldsymbol{x}} \in \operatorname{int} \mathcal{X}$ and $\bar{\boldsymbol{u}} \in \operatorname{int} \mathcal{U}$, or, in words, the nominal trajectory lies in the strict interior of the two respective sets. This assumption is necessary to avoid degenerate solutions during funnel synthesis. Note that this assumption does not preclude nominal trajectories that activate a constraint; it merely suggests that the constraint sets used for funnel synthesis are slightly more relaxed than those imposed during nominal trajectory synthesis, if necessary.

We also assume that $\mathcal{X}_f \subset \mathbb{R}^{n_x}$ is a set that determines the maximum funnel size at the final time $t = t_f$ (i.e., the acceptable set of state deviations at the final time).

Because quadratic funnels are constructed using ellipsoids, it is natural to compute the largest ellipsoids that are able to fit in the feasible space defined by $\mathcal{X}, \mathcal{U}$ and $\mathcal{X}_f$. This is equivalent to a "maximum quadratic funnel" denoted by $\mathcal{F}_{\max}$ that satisfies

$$\mathcal{F}_{\max} = \mathcal{E}_{Q_{\max}} \times \mathcal{E}_{R_{\max}} \quad \text{and} \quad \mathcal{E}_{Q_{\max}} \subseteq \mathcal{X}, \quad \mathcal{E}_{R_{\max}} \subseteq \mathcal{U}$$

where $Q_{\max} \in \mathbb{S}^{n_x}_{++}$ and $R_{\max} \in \mathbb{S}^{n_u}_{++}$. Inclusion in the terminal $\mathcal{X}_f$ can be guaranteed by ensuring that $\mathcal{E}_{Q_{\max}} \subseteq \mathcal{X}_f$ when $t = t_f$.

To compute the matrix $Q_{\max}$ at any time $t < t_f$, we assume that the set $\mathcal{X}$ can be expressed as

$$\mathcal{X} = \{\boldsymbol{x} \,|\, h_i(\boldsymbol{x}) \leq 0,\ i = 1, \ldots, m_x,\ \|\boldsymbol{x}\|_2 \leq x_{\max}\}\,, \tag{6.48}$$

where each constraint function $h_i : \mathbb{R}^{n_x} \to \mathbb{R}$ is a scalar-valued nonlinear function of the state. Note that *equality* constraints must be excluded from the definition (6.48) because of the assumption that the nominal trajectory lies in the *strict* interior of the set $\mathcal{X}$. The best quasi-polytopic approximation of $\mathcal{X}$ in the vicinity of the nominal trajectory is

$$\mathcal{P}_{\bar{\boldsymbol{x}}} = \left\{\boldsymbol{x} \,|\, \boldsymbol{a}_i^\top \boldsymbol{x} \leq b_i,\ i = 1, \ldots, m_x\right\} \cap \{\boldsymbol{x} \,|\, \|\boldsymbol{x}\|_2 \leq x_{\max}\} \tag{6.49}$$

where the data $\{\boldsymbol{a}_i, b_i\}_{i=1}^{m_x}$ are first-order approximations to the functions h_i computed using either direct linearization along the nominal trajectory, or by using a project-and-linearize technique if any h_i is a concave function [127].[5] Note that if the state constraints h_i are all affine then $\mathcal{X} = \mathcal{P}_{\bar{\boldsymbol{x}}}$.

The matrix $Q_{\max}$ is computed as the maximum volume ellipsoid so that

$$\bar{\boldsymbol{x}} \oplus \mathcal{E}_{Q_{\max}} \subset \mathcal{P}_{\bar{\boldsymbol{x}}} \subseteq \mathcal{X}, \tag{6.50}$$

where $\oplus$ denotes the Minkowski sum of a vector and set. The term $\bar{\boldsymbol{x}} \oplus \mathcal{E}_{Q_{\max}}$ is equivalent to the ellipsoid defined by $Q_{\max}$ centered at the vector $\bar{\boldsymbol{x}}$. The assumption that $\bar{\boldsymbol{x}}$ lies in the strict interior of $\mathcal{X}$ ensures that $Q_{\max}$ describes a full n_x-dimensional ellipsoid.

Using techniques described in [10], we can write the condition (6.50) as the following

[5] Common constraints that fit this description include ellipsoidal collision avoidance constraints, or in the control space, a minimum two-norm constraint. The project-and-linearize technique enlarges the feasible domain defined by the approximation and can lead to larger funnels.

optimization problem:

$$\begin{aligned} Q_{\max}^{1/2}(t) = \arg\max_{Z} \quad & \log\det Z \\ \text{s.t.} \quad & \|Z\boldsymbol{a}_i(t)\|_2 + \boldsymbol{a}_i(t)^\top \bar{\boldsymbol{x}}(t) \leq b_i(t), \quad i = 1, \ldots, m_x \\ & 0 \preceq Z \preceq x_{\max} I_{n_x} \end{aligned} \tag{6.51}$$

When $t = t_f$, a set of constraints can be added to constrain $\mathcal{E}_{Q_{\max}} \subseteq \mathcal{X}_f$. By assuming that the set $\mathcal{X}_f$ can described in the same manner as $\mathcal{X}$, this set of constraints is identical to those that are shown in (6.51), albeit calculated using the data corresponding to $\mathcal{X}_f$.

The matrices $R_{\max}$ can be computed in precisely the same manner. Starting with the analogous description of $\mathcal{U}$ as

$$\mathcal{U} = \{\boldsymbol{u} \,|\, h_j(\boldsymbol{u}) \leq 0,\ j = 1, \ldots, m_u,\ \|\boldsymbol{u}\|_2 \leq u_{\max}\}, \tag{6.52}$$

we form the approximate quasi-polytopic constraint set $\mathcal{P}_{\bar{\boldsymbol{u}}}$ in the vicinity of the nominal trajectory as

$$\mathcal{P}_{\bar{\boldsymbol{u}}} = \left\{\boldsymbol{u} \,|\, \boldsymbol{a}_j^\top \boldsymbol{u} \leq b_j,\ u = 1, \ldots, m_u\right\} \cap \{\boldsymbol{u} \,|\, \|\boldsymbol{u}\|_2 \leq u_{\max}\} \tag{6.53}$$

and then solve the following optimization problem for any time $t \in [t_0, t_f]$:

$$\begin{aligned} R_{\max}^{1/2}(t) = \arg\max_{Z} \quad & \log\det Z \\ \text{s.t.} \quad & \|Z\boldsymbol{a}_j(t)\|_2 + \boldsymbol{a}_j(t)^\top \bar{\boldsymbol{u}}(t) \leq b_j(t), \quad j = 1, \ldots, m_u \\ & 0 \preceq Z \preceq u_{\max} I_{n_u} \end{aligned} \tag{6.54}$$

Remark 6.7. *If the problem at hand is posed with mixed state and control constraints, then we cannot separate the computation of $Q_{\max}$ and $R_{\max}$ into two independent problems. In this case, one can define a combined set $\mathcal{P}_{\bar{\boldsymbol{x}},\bar{\boldsymbol{u}}}$ that contains the linearization of all state and control constraints, including those that are only functions of one variable type. Next, a*

matrix

$$Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{12}^\top & Z_{22} \end{bmatrix} \in \mathbb{S}_+^{(n_x+n_u)} \tag{6.55}$$

can be computed using the procedure outlined above, where $Z_{12} \in \mathbb{R}^{n_x \times n_u}$*. Then, the required matrices can be computed by projecting the* $(n_x + n_u)$*-dimensional ellipsoid defined by* Z *onto its first* n_x *components for* $Q_{\max}^{1/2}$ *and its last* n_u *components for* $R_{\max}^{1/2}$*.*

Problems (6.51) and (6.54) can be solved to obtain matrices that approximate the maximum ellipsoidal regions fully contained within the feasible state and control domains. Note that if $\mathcal{X} = \mathcal{P}_{\bar{\boldsymbol{x}}}$, then this is not an approximation (similarly for $\mathcal{U} = \mathcal{P}_{\bar{\boldsymbol{u}}}$).

Recall from the definition that a quadratic funnel is defined by the ellipsoids $\mathcal{E}_Q$ and $\mathcal{E}_{KQK^\top}$. A sufficient condition to ensure that a quadratic funnel is fully contained within the original feasible domain defined by $\mathcal{X}$ and $\mathcal{U}$ is

$$\mathcal{E}_Q \subseteq \mathcal{E}_{Q_{\max}} \quad \text{and} \quad \mathcal{E}_{KQK^\top} \subseteq \mathcal{E}_{R_{\max}}.$$

By using the definition of an ellipsoid, one can show that these conditions are equivalent to

$$Q \preceq Q_{\max} \quad \text{and} \quad KQK^\top \preceq R_{\max}. \tag{6.56}$$

The former is affine in the variable Q, whereas the latter is not jointly convex in the variables Q and K. However, a brief reformulation provides the necessary remedy. Using a Schur complement, we can rewrite the second matrix inequality in (6.56) to be

$$R_{\max} - KQK^\top \succeq 0 \quad \iff \quad \begin{bmatrix} Q^{-1} & K^\top \\ K & R_{\max} \end{bmatrix} \succeq 0.$$

Pre- and post-multiplying by $\mathbf{diag}\{Q, I_{n_u}\}$ then yields the equivalent condition that

$$\begin{bmatrix} Q & Y^\top \\ Y & R_{\max} \end{bmatrix} \succeq 0, \tag{6.57}$$

where we've reused the variable substitution $Y = KQ$. Together, the constraints $Q \preceq Q_{\max}$ and (6.57) ensure that a quadratic funnel is contained in the feasible region defined by $\mathcal{X}$ and $\mathcal{U}$. Both are linear matrix inequalities and convex in the variables Q and Y.

Formal Problem Statement

We have now formulated the two properties of a quadratic funnel separately as a set of three matrix inequalities. In this section, we combine these ideas and pose a single optimization problem that can be solved in order to obtain a quadratic funnel.

After some experimentation, a good cost function to maximize the "size" of the funnel was found to be the volume of its entry (the volume at time $t = t_0$). The continuous-time quadratic funnel synthesis problem for the system (6.25) is summarized in Problem 11.

Problem 11 (Quadratic Funnel Synthesis)**.** *Given a nominal trajectory $\{\bar{\boldsymbol{x}}(t), \bar{\boldsymbol{u}}(t)\}_{t=t_0}^{t_f}$ that satisfies Assumption 6.4, an appropriate definition of system* (6.25) *and $\alpha \geq 0$, find the matrix-valued functions of time $Q(t)$, $Y(t)$ and $M(t)$ and scalar $\lambda(t)$ that solve the following optimization problem.*

$$\max_{Q(\cdot), Y(\cdot), M(\cdot), \lambda(\cdot)} \quad \log\det Q(t_0) \tag{6.58a}$$

$$s.t. \quad 0 \preceq Q \preceq Q_{\max},\ 0 \leq \lambda,\ M \in \mathcal{M}_{\phi, \mathcal{E}_{GQ^{-1}G^\top}}, \qquad t \in [t_0, t_f], \tag{6.58b}$$

$$\begin{bmatrix} F - \dot{Q} + \lambda G^\top M_{11} G & E + \lambda G^\top M_{12} \\ E^\top + \lambda M_{12}^\top G & \lambda M_{22} \end{bmatrix} \preceq 0, \quad t \in [t_0, t_f], \tag{6.58c}$$

$$\begin{bmatrix} Q & Y^\top \\ Y & R_{\max} \end{bmatrix} \succeq 0, \qquad t \in [t_0, t_f]. \tag{6.58d}$$

The matrices F and G are given by (6.47).

Problem 11 is a nonlinear, nonconvex optimization problem with matrix variables. The solution variable Q defines a (quadratic) Lyapunov function through (6.3) for the closed-loop system that satisfies Corollary 6.3. The main difficulty in solving Problem 11 lies in the fact that it is not possible (to the best of our knowledge) to express (6.58c) as a linear DMI in the problem's variables. Moreover, the inclusion $M \in \mathcal{M}_{\phi,GQ^{-1}G^\top}$ can at best be approximated – a point we will discuss in more detail shortly.

A quadratic funnel obtained by solving Problem 11 can be used to generate a feasible trajectory in the following manner. Let $\bar{\boldsymbol{x}}(t_0)$ be the initial condition of the reference state trajectory for which Assumption 6.4 holds. For any initial condition $\boldsymbol{x}(t_0)$ such that $\boldsymbol{x}(t_0) - \bar{\boldsymbol{x}}(t_0) \in \mathcal{E}_{Q(t_0)}$, the state and control vectors

$$\boldsymbol{u}(t) = \bar{\boldsymbol{u}}(t) + K(t)\big(\boldsymbol{x}(t) - \bar{\boldsymbol{x}}(t)\big), \tag{6.59a}$$

$$\boldsymbol{x}(t) = \boldsymbol{x}(t_0) + \int_{t_0}^{t} f\big(\boldsymbol{x}(\zeta),\, \boldsymbol{u}(\zeta)\big)\,\mathrm{d}\zeta, \tag{6.59b}$$

satisfy:

$$\boldsymbol{\eta} = \boldsymbol{x} - \bar{\boldsymbol{x}} \in \mathcal{E}_Q, \quad \text{and} \quad \boldsymbol{\xi} = \boldsymbol{u} - \bar{\boldsymbol{u}} = K\boldsymbol{\eta} \in \mathcal{E}_{KQK^\top}$$

for all $t \in [t_0, t_f]$. Feasibility with respect to the constraint sets $\mathcal{X}$ and $\mathcal{U}$ follows from the fact that $\mathcal{E}_Q \subseteq \mathcal{E}_{Q_{\max}} \subseteq \mathcal{X}$ and $\mathcal{E}_{KQK^\top} \subseteq \mathcal{E}_{R_{\max}} \subseteq \mathcal{U}$. Notice that dynamic feasibility follows from (6.59) by construction, and the control $\boldsymbol{u}$ is guaranteed to be stabilizing for the original nonlinear dynamics.

Because a direct solution of Problem 11 is in general not possible, we now present two different iterative methods. Each method has a different set of assumptions and a different iterative structure, and consequently have different numerical properties.

6.4 Solution Method 1: QM-Iteration

The first methodology to solve Problem 11 uses the following observation: if we fix the triple (Q, Y, λ), then Problem 11 is convex in M provided we have an appropriate description of the convex cone $\mathcal{M}_{\phi, GQ^{-1}G^\top}$; conversely, for a fixed M, Problem 11 is convex in (Q, Y, λ).

A natural and common technique is to alternate between fixing one set of variables and solving for the other, and then swapping roles. This is the premise of the QM-iteration, and has been used in similar contexts for funnel generation [181], as well as the Kleinman iteration [186] and the D-K iteration [187].

6.4.1 The Q-Problem

We begin with a description of the subproblem obtained by fixing the multiplier matrix M in Problem 11. We call this subproblem the Q-problem.

Consider the differential matrix inequality (6.58c). Let us make the assumption that $M_{11} \succeq 0$. In this case, there exists a real matrix N_{11} that satisfies

$$M_{11} = N_{11}^\top N_{11}.$$

The matrix inequality (6.58c) can therefore be written

$$\begin{bmatrix} F - \dot{Q} & E + \lambda G^\top M_{12} \\ E^\top + \lambda M_{12}^\top G & \lambda M_{22} \end{bmatrix} + \begin{bmatrix} G^\top N_{11}^\top \\ 0 \end{bmatrix} (\lambda I) \begin{bmatrix} N_{11} G & 0 \end{bmatrix} \preceq 0$$

If $E \neq 0$ then we must have $\lambda > 0$. Hence we can use a Schur complement to arrive at the equivalent condition

$$\begin{bmatrix} F - \dot{Q} & E + \lambda G^\top M_{12} & G^\top N_{11}^\top \\ E^\top + \lambda M_{12}^\top G & \lambda M_{22} & 0 \\ N_{11} G & 0 & -\lambda^{-1} I \end{bmatrix} \preceq 0.$$

Pre- and post-multiplying by $\mathbf{diag}\{I, \lambda^{-1}I, I\}$ gives the equivalent condition that

$$\begin{bmatrix} F - \dot{Q} & \nu E + G^\top M_{12} & G^\top N_{11}^\top \\ \nu E^\top + M_{12}^\top G & \nu M_{22} & 0 \\ N_{11} G & 0 & -\nu I \end{bmatrix} \preceq 0, \quad \nu \coloneqq \lambda^{-1}. \tag{6.60}$$

The matrix inequality (6.60) is now a *linear* differential matrix inequality in the variables Q, Y and ν. By using the temporal parameterization methods outlined in Section 6.1 we can transform this into a set of LMIs.

The continuous time Q-problem is summarized below in Problem 12. It is assumed that a nominal trajectory, system definition, and decay rate $\alpha \geq 0$ have all been given. The minimum funnel $Q_{\min}$, referenced as an input to the Q-problem, is present for matters of convergence and will be addressed in §6.4.3.

Problem 12 (Q-Problem)**.** *Given a minimum funnel $Q_{\min}$ and a local multiplier matrix $M \in \mathcal{M}_{\phi, \mathcal{E}_{GQ^{-1}G^\top}}$, extract the three constituent matrices as shown in* (6.44)*. Compute N_{11} such that $M_{11} = N_{11}^\top N_{11}$. Find the matrix-valued functions of time $Q(t)$, $Y(t)$ and scalar $\nu(t)$ that solve the following optimization problem.*

$$\max_{Q(\cdot), Y(\cdot), \nu(\cdot)} \quad \log\det Q(t_0) \tag{6.61a}$$

$$s.t. \quad Q_{\min} \preceq Q \preceq Q_{\max},\ 0 \leq \nu, \qquad t \in [t_0, t_f], \tag{6.61b}$$

$$\begin{bmatrix} F - \dot{Q} & \nu E + G^\top M_{12} & G^\top N_{11}^\top \\ \nu E^\top + M_{12}^\top G & \nu M_{22} & 0 \\ N_{11} G & 0 & -\nu I \end{bmatrix} \preceq 0, \quad t \in [t_0, t_f], \tag{6.61c}$$

$$\begin{bmatrix} Q & Y^\top \\ Y & R_{\max} \end{bmatrix} \succeq 0, \qquad t \in [t_0, t_f]. \tag{6.61d}$$

The matrices F and G are given by (6.47)*.*

6.4.2 The M-Problem

We now describe the subproblem obtained by fixing the triple (Q, Y, λ) in Problem 11. We call this subproblem the M-problem.

Consider again the matrix inequality (6.58c). This constraint is already linear in the multiplier matrix M. The main question that arises is what constitutes a *good* multiplier matrix, in terms of being able to synthesize a large funnel as measured by the cost function of the Q-problem.

Note that because of the assumption that $M_{11} \succeq 0$, used to derive the Q-problem, we must enforce this as a constraint in the M-problem. Because of this constraint, the $\lambda G^\top M_{11} G$ term in the (1,1) block of (6.58c) is positive semidefinite. We therefore want this term to be as small as possible, because the negative-definiteness of the (1,1) block is a necessary condition for the overall matrix to be negative semidefinite. It is relatively straightforward to see that M_{22} being large and negative definite is beneficial as well, because it alone occupies the $(2, 2)$ block of (6.58c). Lastly, we would like to encourage M_{12} to be as small as possible while still ensuring that M is a valid multiplier matrix.

Let us introduce three scalar variables that we use to embed these qualitative observations into an optimization problem. Let $\rho_1, \rho_2, \rho_3 \in \mathbb{R}_+$, and consider the constraints

$$0 \preceq M_{11} \preceq \rho_1 I \quad \text{and} \quad \|M_{12}\| \leq \rho_2 \quad \text{and} \quad M_{22} \preceq -\rho_3 I. \tag{6.62}$$

While the constraints (6.62) help to capture the qualitative description of a good multiplier matrix, we also introduce a solution variable $\rho_0 \in \mathbb{R}_+$ and augment (6.58c) to be

$$\begin{bmatrix} F - \dot{Q} + \lambda G^\top M_{11} G & E + \lambda G^\top M_{12} \\ E^\top + \lambda M_{12}^\top G & \lambda M_{22} \end{bmatrix} \preceq -\rho_0 I, \tag{6.63}$$

in order to select M so that the matrix inequality (6.58c) (and therefore (6.61c)) is not close to being infeasible when the Q-problem is subsequently solved.

Lastly, we must ensure that M is actually a valid multiplier matrix and satisfies the set inclusion $M \in \mathcal{M}_{\phi,\mathcal{E}_{GQ^{-1}G^\top}}$. We know that if γ is a local Lipschitz constant for the nonlinear function ϕ over the set $\mathcal{E}_{GQ^{-1}G^\top}$, then

$$M_\gamma \coloneqq \begin{bmatrix} \gamma^2 I & 0 \\ 0 & -I \end{bmatrix} \tag{6.64}$$

is a valid local multiplier matrix [204, 205]. We will discuss two strategies to compute γ for an arbitrary nonlinear function and set shortly in §6.5. One of these strategies is a sampling based approach, where a finite number of samples are used to estimate the value of γ and approximate $M_\gamma \in \mathcal{M}_{\phi,\mathcal{E}_{GQ^{-1}G^\top}}$. We adopt a similar strategy for the M-problem to compute the more general multiplier matrix M.

Recall the definition of $\mu(\boldsymbol{q})$ in (6.39), repeated here:

$$\mu(\boldsymbol{q}) \coloneqq \begin{bmatrix} \boldsymbol{q} \\ \phi(\boldsymbol{q}) \end{bmatrix}^\top M \begin{bmatrix} \boldsymbol{q} \\ \phi(\boldsymbol{q}) \end{bmatrix} \geq 0, \quad \text{for all } \boldsymbol{q} \in \mathcal{E}_{GQ^{-1}G^\top} = \mathcal{E}_{C_{cl}QC_{cl}^\top}$$

Essentially, we sample a number of vectors $\boldsymbol{\eta}_s \in \mathcal{E}_Q$, transform these to the corresponding vectors $\boldsymbol{q}_s = C_{cl}\boldsymbol{\eta}_s$, and then enforce $\mu(\boldsymbol{q}_s) \geq 0$ for each sample. Given a large enough number of samples, we have found that this strategy reliably computes local valid multiplier matrices.

There is one central issue: beyond very simple dynamical systems, we do not have direct access to $\phi(\boldsymbol{q})$ when using the model (6.36). We can only compute $E\phi(\boldsymbol{q})$ for a known $E \in \mathbb{R}^{n_x \times n_p}$. Observe that for a sampled point $\boldsymbol{\eta}_s \in \mathcal{E}_Q$, we have $\boldsymbol{q}_s = C_{cl}\boldsymbol{\eta}_s$ and

$$\boldsymbol{x}_s = \bar{\boldsymbol{x}} + \boldsymbol{\eta}_s, \quad \boldsymbol{\xi}_s = K\boldsymbol{\eta}_s, \quad \boldsymbol{u}_s = \bar{\boldsymbol{u}} + \boldsymbol{\xi}_s \tag{6.65a}$$

$$E\phi(\boldsymbol{q}_s) = f(\boldsymbol{x}_s, \boldsymbol{u}_s) - f(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) - A\boldsymbol{\eta}_s - B\boldsymbol{\xi}_s. \tag{6.65b}$$

When using structured nonlinearities, it is very often the case that $n_p > n_x$ and that

$E^\top E \in \mathbb{R}^{n_p \times n_p}$ is not invertible.[6] For example, consider a one-dimensional mass-spring-damper system with a nonlinear spring and a nonlinear damper. After expressing the second-order dynamics as two first-order differential equations, both nonlinearities can be seen to affect the second state only. If the nonlinearities are considered independently, then $E = \begin{bmatrix} \boldsymbol{e}_2 & \boldsymbol{e}_2 \end{bmatrix}$, and $E^\top E$ is not invertible. In general, this phenomenon is a consequence of using structured nonlinearity.

The following result helps to resolve this issue.

Lemma 6.8. *If $\bar{M} \in \mathbb{S}^{(n_q+n_x)}$ is a multiplier matrix (local or otherwise) for the nonlinear function $\psi(\boldsymbol{q}) = E\phi(\boldsymbol{q})$, then*

$$M = \begin{bmatrix} I & 0 \\ 0 & E^\top \end{bmatrix} \bar{M} \begin{bmatrix} I & 0 \\ 0 & E \end{bmatrix} = \begin{bmatrix} \bar{M}_{11} & \bar{M}_{12}E \\ E^\top \bar{M}_{12} & E^\top \bar{M}_{22} E \end{bmatrix} \tag{6.66}$$

is a multiplier matrix (local or otherwise) for the nonlinear function ϕ. For a local multiplier matrix, the set Ω in Definition 4 is consistent for both M and $\bar{M}$.

Proof. Let $\bar{M}$ be a multiplier matrix for the nonlinear function $\psi : \mathbb{R}^{n_q} \to \mathbb{R}^{n_x}$. We give the proof for the more general case because this implies that it holds for the more specific class of local multiplier matrices as well. We have that

$$\begin{bmatrix} \boldsymbol{q} \\ \psi(\boldsymbol{q}) \end{bmatrix}^\top \bar{M} \begin{bmatrix} \boldsymbol{q} \\ \psi(\boldsymbol{q}) \end{bmatrix} \geq 0,$$

$$\begin{bmatrix} \boldsymbol{q} \\ E\phi(\boldsymbol{q}) \end{bmatrix}^\top \bar{M} \begin{bmatrix} \boldsymbol{q} \\ E\phi(\boldsymbol{q}) \end{bmatrix} \geq 0,$$

$$\begin{bmatrix} \boldsymbol{q} \\ \phi(\boldsymbol{q}) \end{bmatrix}^\top \begin{bmatrix} I & 0 \\ 0 & E^\top \end{bmatrix} \bar{M} \begin{bmatrix} I & 0 \\ 0 & E \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \phi(\boldsymbol{q}) \end{bmatrix} \geq 0.$$

[6] Of course, if $E^\top E$ is indeed invertible, then one can compute the term $\boldsymbol{p} = \phi(\boldsymbol{q})$ in (6.36) directly, and subsequently solve the M-problem with M as a variable.

Hence the matrix M as defined in (6.66) is a multiplier matrix for the function ϕ. $\square$

The variable $\bar{M}$ becomes the solution variable for the M-problem, which is formally stated below in Problem 13.

Problem 13 (M-Problem)**.** *Given a triple (Q, Y, ν) obtained by solving Problem 12 (the Q-problem), set $\lambda = \nu^{-1}$. Collect a set of N_s samples from the set $\mathcal{E}_Q$, and use these to compute the values of $\boldsymbol{q}_s$ and $E\phi(\boldsymbol{q}_s)$ for each $s = 1, \ldots, N_s$ using* (6.65)*. Using user-selected weights $w_1, w_2, w_3 \in \mathbb{R}_+$, find the matrices $\bar{M}_{11} \in \mathbb{S}^{n_q}$, $\bar{M}_{12} \in \mathbb{R}^{n_q \times n_x}$ and $\bar{M}_{22} \in \mathbb{S}^{n_x}$ and slack variables $\rho_0, \rho_1, \rho_2, \rho_3 \in \mathbb{R}_+$ that solve the following optimization problem.*

$$\min_{\substack{\bar{M}_{11}, \bar{M}_{12}, \bar{M}_{22} \\ \rho_0, \rho_1, \rho_2, \rho_3}} \quad \rho_0 + w_1\rho_1 + w_2\rho_2 - w_3\rho_3 \tag{6.67a}$$

$$\begin{aligned}
s.t. \quad & 0 \preceq \bar{M}_{11} \preceq \rho_1 I, && 0 \le \rho_1 \le \gamma^2 && \text{(6.67b)}\\
& \|\bar{M}_{12}E\| \le \rho_2, && 0 \le \rho_2 && \text{(6.67c)}\\
& E^\top \bar{M}_{22} E \preceq -\rho_3 I, && 0 \le \rho_3 \le 1 && \text{(6.67d)}\\
& \begin{bmatrix} F - \dot{Q} + \lambda G^\top \bar{M}_{11} G & \left(I + \lambda G^\top \bar{M}_{12}\right) E \\ E^\top \left(I + \lambda \bar{M}_{12}^\top G\right) & \lambda E^\top \bar{M}_{22} E \end{bmatrix} \preceq -\rho_0 I, && 0 \le \rho_0 \le 1 && \text{(6.67e)}\\
& \mu(\boldsymbol{q}_s) \ge 0, && s = 1, \ldots, N_s && \text{(6.67f)}
\end{aligned}$$

6.4.3 Algorithm Summary and Convergence

Having stated the Q- and M-problems individually, we can now summarize how the QM-iteration is constructed. A fair amount of input data is required to run the algorithm; a feasible nominal trajectory $\{\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}\}_{t=t_0}^{t_f}$, matrices A and B that satisfy (6.19), and (constant) matrices C, D and E that complete the definition of the model (6.35). In addition, definitions of the constraint sets $\mathcal{X}$, $\mathcal{U}$ and $\mathcal{X}_f$ are required. The decay rate $\alpha \ge 0$ should be specified,[7]

[7]It is possible to have this be a free parameter and chosen as part of the M-problem. However, the objective of maximizing the funnel volume implies that α will always be selected as the minimum allowed value – and so the user may as well specify this value and treat α as a parameter.

Algorithm 5 The QM-iteration designed to solve the quadratic funnel synthesis problem.

Input: A nominal trajectory $\{\bar{\boldsymbol{x}}(t), \bar{\boldsymbol{u}}(t)\}_{t=t_0}^{t_f}$, the system matrices for (6.35), decay rate $\alpha \geq 0$, constraint sets $\mathcal{X}$,$\mathcal{U}$ and $\mathcal{X}_f$, integers N_s and n_M, weights $w_1, w_2, w_3 \in \mathbb{R}_+$, and a tolerance $\epsilon_{\text{tol}} > 0$.

solve Problems (6.51) and (6.54) for $Q_{\max}$ and $R_{\max}$
set $Q_{\min} \leftarrow \varepsilon I_{n_x}$ and $M \leftarrow 0$ ▷ ε denotes a "small number".
solve the Q-problem to get $\{Q^{(0)}, Y^{(0)}, \nu^{(0)}\}$ ▷ Problem 12
update $Q_{\max} \leftarrow Q^{(0)}$
for $k = 1, \ldots$ **do**
    solve the M-problem to get $M^{(k)}$ using $\{Q^{(k-1)}, Y^{(k-1)}, \nu^{(k-1)}\}$ ▷ Problem 13
    solve the Q-problem to get $\{Q^{(k)}, Y^{(k)}, \nu^{(k)}\}$ using $M^{(k)}$ ▷ Problem 12
    if k = 1 **then**
        $Q_{\min} \leftarrow Q^{(k)}$
    else
        $Q_{\min}(t_0) \leftarrow Q^{(k)}(t_0)$
        if $|\kappa^{(k)}(t_0) - \kappa^{(k-1)}(t_0)| \leq \epsilon_{\text{tol}}$ **then** ▷ see (6.68)
            break
        end if
    end if
end for

Output: Time-varying matrices $Q(t)$ and $K(t)$ that define a quadratic funnel in the sense of Definition 3.

along with the number of sample points N_s, M-problem weights w_1, w_2 and w_3, and the number of discrete temporal points at which to solve for the funnel, n_M.

The QM-iteration is presented in Algorithm 5. Note that the value of a solution variable Z at the k^{th} iteration is denoted using $Z^{(k)}$. The process begins by solving the Q-problem with $M = 0$, which (after temporal parameterization) is equivalent to solving Theorem 6.2 with the additional feasibility constraints (6.56) and (6.57). (With "no nonlinearities", systems (6.25) and (6.35) are equivalent to the LTV model (6.1).) The core steps of the algorithm then simply alternate between the solution of the M-problem and of the Q-problem. The latter steps of Algorithm 5 are designed to facilitate convergence, which is formally stated in Theorem 6.9.

Theorem 6.9. *For any tolerance $\epsilon_{\text{tol}} > 0$, the QM-iteration in Algorithm 5 converges to a solution of Problem 11 if each individual Q- and M-problem are feasible.*

Proof. If any Q- or M-problem is infeasible, then the iterations terminate, and convergence is not achieved. Therefore, assume that each Q- and M-problem is feasible. For each iteration $k > 1$, the initial funnel volume, $Q^{(k)}(t_0)$, must increase, because of the constraint

$$Q^{(k-1)}(t_0) = Q_{\min}(t_0) \preceq Q^{(k)}(t_0)$$

imposed by the Q-problem (see (6.61b)). The *fill ratio* computed by using

$$\kappa = \min_{i=1,\ldots,n_x} \left(\frac{\text{proj}_i \, \mathcal{E}_Q}{\text{proj}_i \, \mathcal{E}_{Q_{\max}}} \right)^{1/2} \tag{6.68}$$

must therefore increase monotonically at the initial time t_0 because $Q_{\max}$ does not change throughout the iterations. The notation $\text{proj}_i \, \mathcal{E}_Q$ denotes the projection of the ellipsoid $\mathcal{E}_Q$ onto the dimension i, and is equivalent to the maximum distance that $\mathcal{E}_Q$ extends along the i^{th} axis in n_x-dimensional space.

The fill ratio is bounded above by 1 due to the constraint $Q \preceq Q_{\max}$. The sequence $\{\kappa^{(k)}(t_0)\}_{k=2}^{\infty}$ is therefore a bounded and monotonically increasing sequence, and we can conclude that for any $\epsilon_{\text{tol}} > 0$, there must exist some finite integer $K \geq 2$ for which

$$|\kappa^{(K)}(t_0) - \kappa^{(K-1)}(t_0)| \leq \epsilon_{\text{tol}}.$$

□

6.4.4 Case Study: Satellite Attitude Control

This section provides a numerical example of the QM-iteration applied to a simple satellite attitude control problem. For this problem, we consider the state vector $\boldsymbol{x}$ to be composed

of a set of 3-2-1 Euler angles and angular velocity with the following nonlinear dynamics

$$\boldsymbol{x} = \begin{bmatrix} \Theta \\ \boldsymbol{\omega}_{\mathcal{B}} \end{bmatrix} \in \mathbb{R}^6, \quad \text{and} \quad \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} T(\Theta)\boldsymbol{\omega}_{\mathcal{B}} \\ J^{-1}\left(\boldsymbol{u} - \boldsymbol{\omega}_{\mathcal{B}}^{\times} J \boldsymbol{\omega}_{\mathcal{B}}\right) \end{bmatrix}. \tag{6.69}$$

Here, $J \in \mathbb{R}^{3\times 3}$ is the inertia matrix, $\boldsymbol{u} \in \mathbb{R}^3$ is the control input (a torque), and

$$\Theta = \begin{bmatrix} \varphi \\ \theta \\ \psi \end{bmatrix} \quad \Rightarrow \quad T(\Theta) = \begin{bmatrix} 1 & \sin\varphi\tan\theta & \cos\varphi\tan\theta \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi\sec\theta & \cos\varphi\sec\theta \end{bmatrix}.$$

A nominal trajectory was computed by using the PTR algorithm (see Algorithm 1) using the boundary conditions $\boldsymbol{x}(t_0) = 0_{6\times 1}$ and $\boldsymbol{x}(t_f) = \left(\frac{\pi}{4}, \frac{\pi}{8}, \frac{\pi}{4}, 0, 0, 0\right)$ and an optimized time span of $t_f = 35$ seconds. The constraints from the sets $\mathcal{X}$ and $\mathcal{U}$ as defined in (6.70) were also enforced.

The matrices A and B are the partial derivatives of f along the reference trajectory. For this example, the parameters C, D and E can be constructed by using structured nonlinearity to be

$$C = \begin{bmatrix} I_3 & 0_{3\times 3} \\ 0_{3\times 3} & I_3 \end{bmatrix}, \quad D = 0_{6\times 3}, \quad E = \begin{bmatrix} I_3 & I_3 \\ 0_{3\times 3} & 0_{3\times 3} \end{bmatrix}, \qquad \text{if } J = cI_3,\ c > 0,$$

$$C = \begin{bmatrix} I_3 & 0_{3\times 3} \\ 0_{3\times 3} & I_3 \\ 0_{3\times 3} & I_3 \end{bmatrix}, \quad D = 0_{9\times 3}, \quad E = \begin{bmatrix} I_3 & I_3 & 0_{3\times 3} \\ 0_{3\times 3} & 0_{3\times 3} & I_3 \end{bmatrix}, \qquad \text{else.}$$

For this example, we use the following constraint sets:

$$\mathcal{X} = \{\boldsymbol{x} \,|\, \boldsymbol{x}_{lb} \leq \boldsymbol{x} \leq \boldsymbol{x}_{ub},\ \|\boldsymbol{x}\|_2 \leq \infty\}, \tag{6.70a}$$

$$\mathcal{U} = \{\boldsymbol{u} \,|\, \boldsymbol{u}_{lb} \leq \boldsymbol{u} \leq \boldsymbol{u}_{ub},\ \|\boldsymbol{u}\|_2 \leq u_{\max}\}, \tag{6.70b}$$

$$\mathcal{X}_f = \mathcal{E}_{Q_{\max,f}}, \tag{6.70c}$$

Table 6.1: The parameters for the satellite attitude control example used to demonstrate the QM-iteration.

Parameter	Value	Parameter	Value
J	I_3 kg m^2	α	0.1 s
$\boldsymbol{x}_{lb}$	$-\big(\pi,\, \frac{\pi}{2},\, \pi,\, 0.15,\, 0.15,\, 0.15\big)$	$\boldsymbol{x}_{ub}$	$\big(\pi,\, \frac{\pi}{2},\, \pi,\, 0.15,\, 0.15,\, 0.15\big)$
$\boldsymbol{u}_{lb}$	$-8.8 \times 10^{-3} \cdot \mathbf{1}_3$	$\boldsymbol{u}_{ub}$	$8.8 \times 10^{-3} \cdot \mathbf{1}_3$
$u_{\max}$	1.52×10^{-2} Nm	$N_s,\, n_M$	100, 30
ϵ_{tol}	10^{-3}	(w_1, w_2, w_3)	$(1, 1, 1)$

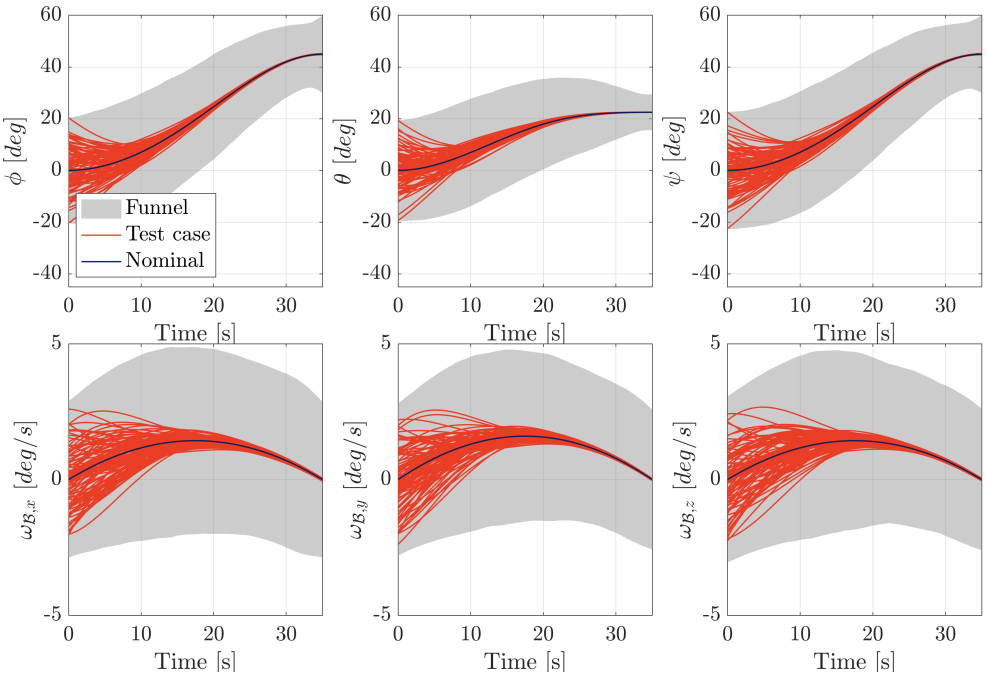
the parameters for which are given in Table 6.1. For this example, we have $\mathcal{X} = \mathcal{P}_{\bar{\boldsymbol{x}}}$ and $\mathcal{U} = \mathcal{P}_{\bar{\boldsymbol{u}}}$.

To solve both the Q- and M-problems, a temporal discretization of the problem's variables is required. To this end, we use Assumption 6.1 for the matrices Q, Y, A and B. We extend this assumption to include $Q_{\max}$ and $R_{\max}$, both of which are parameterized as linear functions of time. We assume that M and ν are piecewise constant over each discrete interval (only the latter is without loss of generality).

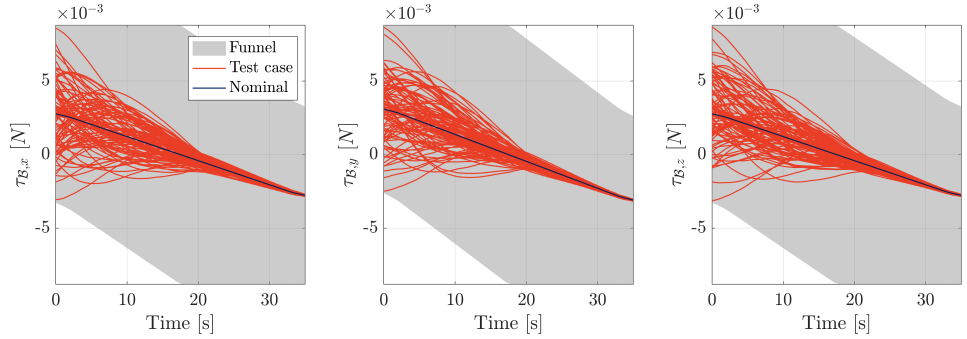
The QM-iteration, as given in Algorithm 5, converged in 16 iterations for this problem setup. Figure 6.2 provides a depiction of the quadratic funnel synthesized by the procedure, along with a set of test cases obtained by sampling a random initial condition from the synthesized funnel entry and numerically integrating the closed-loop system. As expected, the correction law maintains both invariance of the funnel and ensures that each test case (shown as a red trajectory) is feasible with respect to the constraints (6.70).

Figure 6.3 provides an illustration of how the fill ratio progresses across each iteration, as well as the value of the $V(t)$ for each test case. One can see that the fill ratio at the initial time is monotonically increasing, as expected, and that no test case leaves the funnel – hence invariance is achieved.

For this problem, the QM-iteration achieves a funnel entry that spans (plus or minus) roughly 20 deg for each Euler angle state, and roughly 3 deg/s in each angular velocity state. This means that for any initial condition within these bounds from the nominal trajectory,



(a) In state space, $\mathcal{E}_Q(t)$.



(b) In control space, $\mathcal{E}_{R_{\max}}$.

Figure 6.2: The quadratic funnel computed by using the QM-iteration for the satellite attitude control problem. A set of test cases are also shown whose initial conditions were randomly sampled from the funnel entry.

we can guarantee the availability of a feasible trajectory by numerically integrating the dynamics (6.69) using the control input $\boldsymbol{u} = \bar{\boldsymbol{u}} + K\boldsymbol{\eta}$. This can be done very quickly on nearly any modern processor using standard fixed-step size numerical integration strategies.

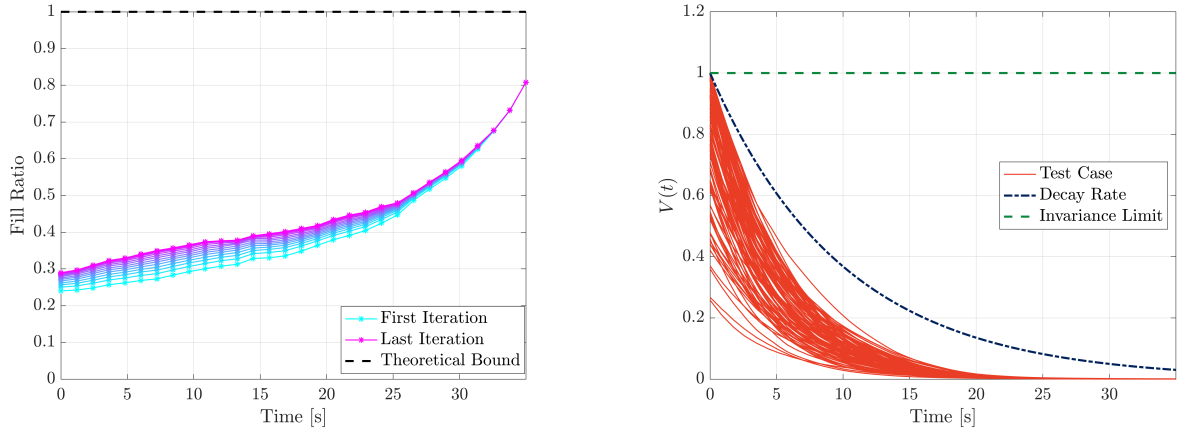


Figure 6.3: The fill ratio across each iteration of the QM-iteration and the value of the Lyapunov function $V(t)$ for each test case for the satellite attitude control problem.

6.5 Solution Method 2: γ-Iteration

The QM-iteration is a convergent algorithm designed to compute a quadratic funnel by decomposing the nonconvex Problem 11 into two convex subproblems and alternating between them. However, an issue has been observed for some more complex problems. The value of ν obtained by solving an instance of the Q-problem (Problem 12) can be quite small, at times on the order of 10^{-6} or less. Once inverted and used to solve the subsequent M-problem, numerical difficulties can arise that result in infeasibility and cause the algorithm to terminate without solution.

The γ-iteration is an alternative solution method that also solves Problem 11, but is not susceptible to the same numerical challenges. The trade-off is that some simplifying assumptions are needed, and we effectively limit our search for solutions to the quadratic funnel synthesis problem (Problem 11) to a subset of the feasible space to gain better numerical conditioning.

6.5.1 Simplification of the M-Problem

The γ-iteration primarily simplifies the M-problem that was introduced in §6.4.2. Here, we assume a specific structure for the local multiplier matrix M, and restrict our search to the one-parameter family of matrices given by

$$M_\gamma = \begin{bmatrix} \gamma^2 I & 0 \\ 0 & -I \end{bmatrix}. \tag{6.71}$$

Note that $M_\gamma \in \mathcal{M}_{\phi,\Omega}$ if and only if γ is a local Lipschitz constant for the function ϕ over the set Ω. By computing a suitable value for γ, we proceed to solve the Q-problem using M_γ as the input.

The quadratic inequality (6.39) under the assumption that $M = M_\gamma$ becomes

$$\boldsymbol{p}^\top \boldsymbol{p} \leq \gamma^2 \boldsymbol{q}^\top \boldsymbol{q} \quad \iff \quad \|\boldsymbol{p}\|_2 \leq \gamma \|\boldsymbol{q}\|_2. \tag{6.72}$$

Introducing a matrix $\Delta \in \mathbb{R}^{n_p \times n_q}$, we can then rewrite the model (6.25) as

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= A\boldsymbol{\eta} + B\boldsymbol{\xi} + E\boldsymbol{p}, \\ \boldsymbol{q} &= C\boldsymbol{\eta} + D\boldsymbol{\xi}, \\ \boldsymbol{p} &= \Delta \boldsymbol{q}, \quad \|\Delta\|_2 \leq \gamma. \end{aligned} \tag{6.73}$$

By using the closed-loop matrices $A_{cl} = A + BK$ and $C_{cl} = C + DK$, we can rewrite (6.73) as

$$\dot{\boldsymbol{\eta}} = (A_{cl} + E\Delta C_{cl})\,\boldsymbol{\eta}, \quad \|\Delta\|_2 \leq \gamma. \tag{6.74}$$

The question becomes: How to compute, or estimate, the value of γ? We have two procedures for doing so; one method that is based on a nonlinear optimization problem, and one method that is based on sampling. It is worth pointing out that Wood and Zhang in [217] separate Lipschitz constant estimation methods into two categories: *white box* and

black box. In their context, a white box method poses an optimization problem for which the analytical expression of the cost and its Jacobian are known. A black box method assumes only to have the ability to query a function at any particular point. This terminology also captures the spirit of the two methods that we now present.

Computing γ via Nonlinear Optimization

We begin with the white box approach, wherein an optimization problem is constructed to estimate γ, and analytical expressions for the cost function and all necessary Jacobians are derived.

To compute the local multiplier matrix M_γ, we would like to find the point $\boldsymbol{\eta} \in \mathcal{E}_Q$ that corresponds to the *largest* value of $\|\Delta\|_2$. That is, we need to solve[8]

$$\begin{aligned} \Gamma^* = \max_{\boldsymbol{\eta}} \quad & \frac{1}{2}\delta^*(\boldsymbol{\eta})^2 \\ \text{s.t.} \quad & \boldsymbol{\eta} \in \mathcal{E}_Q \iff \boldsymbol{\eta}^\top Q^{-1} \boldsymbol{\eta} \leq 1, \end{aligned} \tag{6.75}$$

where,

$$\begin{aligned} \delta^*(\boldsymbol{\eta}) = \min_{\Delta} \quad & \|\Delta\|_2 \\ \text{s.t.} \quad & \dot{\boldsymbol{\eta}} - A_{cl}\boldsymbol{\eta} = E\Delta C_{cl}\boldsymbol{\eta}. \end{aligned} \tag{6.76}$$

The *inner* optimization problem (6.76) finds the smallest matrix (in the spectral norm) that satisfies the nonlinear equations of the dynamic model at a particular point $\boldsymbol{\eta}$. This inner problem is the key component that allows us to be as non-conservative as possible in our search for a local Lipschitz constant. The *outer* optimization problem (6.75) then finds the point $\boldsymbol{\eta}$ that maximizes the value of δ^* inside the funnel. The desired Lipschitz constant can

[8]The cost function has been squared to ensure differentiability and to facilitate numerical solutions. The factor of 1/2 is added to facilitate Jacobian calculations. Neither modification changes the argmax that is being sought.

then be computed by using

$$\gamma^* = \sqrt{2\Gamma^*}. \tag{6.77}$$

To facilitate an analytical solution to the inner problem (6.76), we reformulate it so that the problem is expressed in terms of $\boldsymbol{\delta} \coloneqq \operatorname{vec} \Delta \in \mathbb{R}^{n_q n_p}$. First, note that the inner problem's constraint can be rewritten as

$$\dot{\boldsymbol{\eta}} - A_{cl}\boldsymbol{\eta} = E\Delta C_{cl}\boldsymbol{\eta} \quad \Rightarrow \quad y(\boldsymbol{\eta}) = H(\boldsymbol{\eta})\boldsymbol{\delta} \tag{6.78}$$

where, $y(\boldsymbol{\eta}) = \dot{\boldsymbol{\eta}} - A_{cl}\boldsymbol{\eta}$ and

$$H(\boldsymbol{\eta}) = \begin{bmatrix} E\left(\boldsymbol{e}_1^\top C_{cl}\boldsymbol{\eta}\right) & \cdots & E\left(\boldsymbol{e}_{n_q}^\top C_{cl}\boldsymbol{\eta}\right) \end{bmatrix} \in \mathbb{R}^{n_x \times n_q n_p}. \tag{6.79}$$

Using the fact that $\|\Delta\|_2 \leq \|\Delta\|_F = \|\boldsymbol{\delta}\|_2$, the optimal cost of the inner problem (6.76) can be *upper bounded* by the optimal cost of the optimization problem

$$\begin{aligned} \delta^*(\boldsymbol{\eta}) = \min_{\boldsymbol{\delta}} \quad & \|\boldsymbol{\delta}\|_2 \\ \text{s.t.} \quad & y(\boldsymbol{\eta}) = H(\boldsymbol{\eta})\boldsymbol{\delta}. \end{aligned} \tag{6.80}$$

The optimization problem (6.80) is a minimum-norm least squares problem, whose solution can be computed via

$$\boldsymbol{\delta}^* = H^\dagger(\boldsymbol{\eta})y(\boldsymbol{\eta}) \quad \Rightarrow \quad \delta^*(\boldsymbol{\eta}) = \|\boldsymbol{\delta}^*\|_2. \tag{6.81}$$

Plugging this solution into the outer problem's cost function, results in a single (nonconvex) optimization problem:

$$\begin{aligned} \Gamma^* = \max_{\boldsymbol{\eta}} \quad & \frac{1}{2} y(\boldsymbol{\eta})^\top {H^\dagger}^\top(\boldsymbol{\eta}) H^\dagger(\boldsymbol{\eta}) y(\boldsymbol{\eta}) \\ \text{s.t.} \quad & \boldsymbol{\eta}^\top Q^{-1} \boldsymbol{\eta} \leq 1. \end{aligned} \tag{6.82}$$

By solving (6.82), we can obtain an estimate of the local Lipschitz constant. Be-

cause (6.82) is nonconvex program, in general we can only ensure local optimality; the value of γ^* will be an underestimator of the true local Lipschitz constant. To minimize the likelihood of arriving at a local minima, we solve Problem (6.82) using several different initial guesses from within the feasible region (the ellipsoid $\mathcal{E}_Q$).

Lastly, note that analytical Jacobians can be computed for both the cost function and the constraint in (6.82). These are provided in Appendix B.

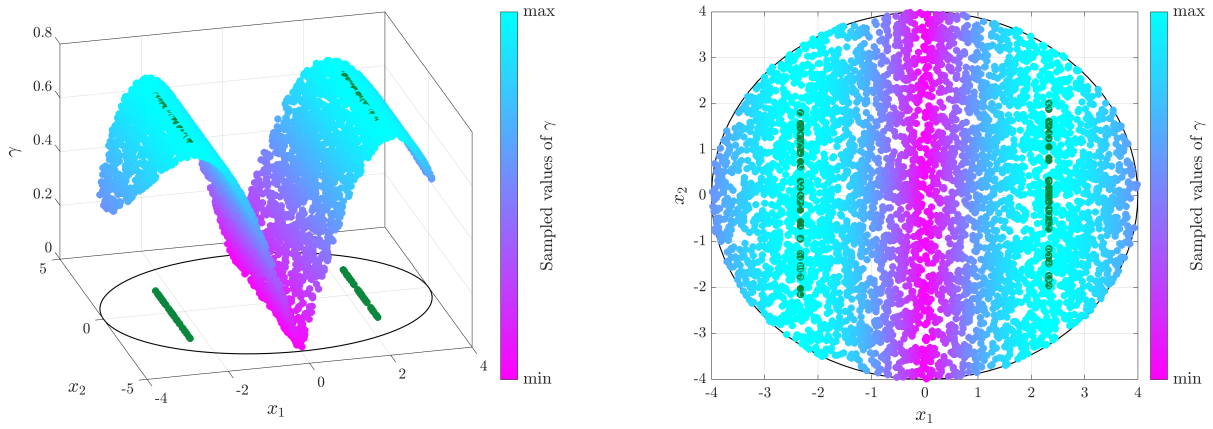
Example 6.10. *To demonstrate how the solution of* (6.82) *works in practice, consider the simple nonlinear oscillator with dynamics* $\dot{x}_1 = x_2$ *and* $\dot{x}_2 = \cos x_1$. *The matrices used to describe the model* (6.25) *are*

$$A(\boldsymbol{x}) = \begin{bmatrix} 0 & 1 \\ -\sin x_1 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad E = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

and we use a nominal trajectory of $\bar{\boldsymbol{x}} = 0$. *We assign a value to the funnel by selecting* $Q = 16 \cdot I_2$. *For this example, we have*

$$y(\boldsymbol{\eta}) = f(\boldsymbol{x}) - f(\bar{\boldsymbol{x}}) - A(\bar{\boldsymbol{x}})\boldsymbol{\eta}, \quad \textit{and} \quad H(\boldsymbol{\eta}) = EC\boldsymbol{\eta},$$

from which we can construct the cost function of (6.82). *The gradient is computed by using the expressions in* (B.16). *For demonstration purposes, the optimization problem* (6.82) *was solved from 100 initial guesses randomly chosen from* $\mathcal{E}_Q$. *It was found that Matlab's*® `fmincon`*, with well-selected parameters, can be quite effective for solving these problems [218], even compared to commercial solvers like SNOPT [124]. The results are shown in Figure 6.4, where the green circles are constructed by using the* $\arg\max$ *and the optimal cost of* (6.82) *for each trial. The cyan-magenta points are pre-sampled points used to illustrated the curvature of how* γ *varies as a function of* x_1 *and* x_2 *so that we may clearly see that a (local) maxima was achieved for each test of* (6.82).



(a) 3D view (b) Top view

Figure 6.4: The results of solving the optimization problem (6.82) from 100 random initial condition chosen inside the black circle (which is $\mathcal{E}_Q$). The z-axis value is the optimal cost of (6.82), and we can see that a maximum is achieved in each trial.

Computing γ via Sampling

The second method is the black box approach, where we only assume to have the ability to query the desired function value at a particular point. The function alluded to here is the cost function of the outer optimization problem (6.75) – which is the inner optimization problem (6.76). Rather than use a nonlinear programming solver to maximize its value over the funnel, we instead sample the function's value at several points inside the funnel and use these to compute an estimate for γ^*.

Because of the assumption that the local multiplier matrix is of the form (6.71), we will not run into the same technical issues as were found during the QM-iteration and resolved by Lemma 6.8.

By sampling a set of states $\{\boldsymbol{\eta}_s\}_{s=1}^{N_s}$ from the ellipsoid $\mathcal{E}_Q$, *each at different temporal points*, and using the steps in (6.65a) N_s times, we can compute at each time the data:

$$A_{cl} = A + BK, \quad C_{cl} = C + DK, \quad \dot{\boldsymbol{\eta}}_s = f\big(\boldsymbol{x}_s, \boldsymbol{u}_s\big) - f\big(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}\big).$$

These are collected and the following optimization problem is solved to obtain again a *lower bound* of the true local Lipschitz constant:

$$\gamma^* = \max_{s=1,\ldots,N_s} \quad \delta^*(\boldsymbol{\eta}_s) \tag{6.83}$$

where δ^* is given by the solution of (6.76). Note that no solver is used, we simply select the sample that produces the maximum value across the N_s samples.

This spatiotemporal sampling procedure has been observed to be accurate enough for practical problems with a reasonable number of points N_s (typically on the order of 10^3 or 10^4 for an entire trajectory). Note that because it is possible to solve N_s independent optimization problems, each of which is quite small, this sampling procedure can be done very quickly and is parallelizable.

6.5.2 Algorithm Summary and Convergence

The Q-problem remains unchanged, save for the simplifications afforded by the assumed structure of the local multiplier matrix M_γ in (6.71). Moreover, note that $N_{11} = \gamma I_{n_q}$ can be computed analytically. The M-problem is replaced by either a solution of (6.82) or of (6.83). Consequently, the weights (w_1, w_2, w_3) are no longer required as part of the algorithm design.

While it is perfectly valid to use the same iterative structure as the QM-iteration, we present an alternative method here. The premise is the following: The QM-iteration increases the funnel's lower bound, $Q_{\min}$, until it cannot be increased further and solutions of the Q-problem return $Q \approx Q_{\min}$; but we can also *decrease* the upper bound, $Q_{\max}$, until we are able to synthesize a funnel $Q \approx Q_{\max}$. The γ-iteration will be constructed using the latter strategy.

The only steps from Algorithm 5 that need to change to facilitate this alternative approach are steps $8-15$. These are replaced with a *contraction step* that shrinks $Q_{\max}$ by an amount proportional to the fill ratio (6.68). The key idea that leads to a convergent algorithm for which *every* iteration's funnel is a valid quadratic funnel, is to use $Q_{\max}$ and the *initial*

step's correction law to compute the value of M_γ at each iteration. We therefore discard the intermediate solutions of the Q-problem, and only make use of the final iteration's value – this is the basis for the omission of "Q" in the name of this method (i.e., as opposed to calling it the $Q\gamma$-iteration).

The contraction step is simple, we maintain the shape of the ellipsoid $\mathcal{E}_{Q_{\max}}$ and shrink its size by multiplying by a *contraction factor* $\varsigma \in [\varsigma_{\min}, 1)$. The fill ratio determines the value of the contraction factor in this interval through the use of a sigmoid function

$$\varsigma = \varsigma_{\min} + (1 - \varsigma_{\min}) \frac{1}{1 + e^{h(0.5-\kappa)}} \quad \Rightarrow \quad Q_{\max} \leftarrow \varsigma Q_{\max}, \tag{6.84}$$

where h is a width parameter and κ is given by (6.68). Equation (6.84) is designed so that when the fill ratio κ is 0.5, the contraction factor is chosen as the midpoint of the interval $[\varsigma_{\min}, 1)$. An example set of contraction factor curves versus the fill ratio is provided in Figure 6.5 for an array of width parameters h.

Remark 6.11. *It is also possible to use a spatiotemporal contraction factor, as opposed to the solely temporal one given by* (6.84)*. For this method, one would compute a diagonal matrix* $S \in \mathbb{R}^{n_x \times n_x}$ *where the* $(i,i)^{th}$ *entry is the minimand of* (6.68)*. One then would notionally compute* $Q_{\max} \leftarrow SQ_{\max}S$*, but there is no guarantee that* $\mathcal{E}_{SQ_{\max}S} \subseteq \mathcal{E}_{Q_{\max}}$*, because the scaling can rotate the ellipsoid. To remedy this, one simply solves for the maximum volume ellipsoid that can be inscribed in the intersection of* $\mathcal{E}_{Q_{\max}}$ *and* $\mathcal{E}_{SQ_{\max}S}$ *– see [188].*

The γ-iteration is detailed in Algorithm 6. Note that the solution variable ν from Problem 12 is no longer required to compute M_γ, and so it is simply omitted. The convergence criteria is not the same as that of Algorithm 5, but is instead defined in terms of a minimum fill ratio. Theorem 6.12 establishes the convergence properties of Algorithm 6.

Theorem 6.12. *Suppose that there exists a* $\epsilon > 0$ *such that* $\epsilon I_{n_x} \preceq Q^{(1)}$*. If Assumption 6.4 holds, then for any tolerance* $0 < \kappa_{tol} < 1$*, the* γ*-iteration in Algorithm 6 converges to a solution of Problem 11 in a finite number of iterations.*

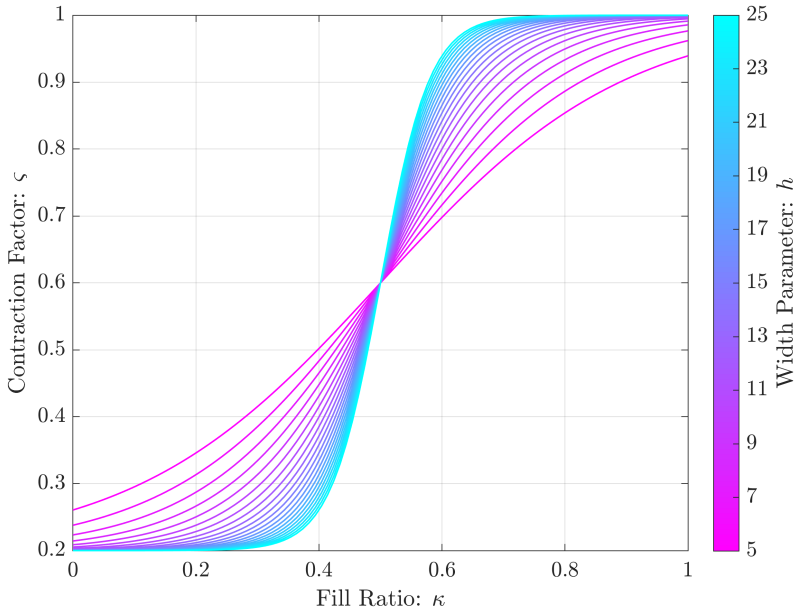


Figure 6.5: Example range of contraction factors as a function of the width parameter h using $\varsigma_{\min} = 0.2$.

Proof. Note that for each instance of Problem 12, the solution $Q = 0$ and $Y = 0$ is always feasible, because $Q_{\min} = 0$ at each iteration. Because the strict inclusions $\bar{\boldsymbol{x}} \in \operatorname{int} \mathcal{X}$ and $\bar{\boldsymbol{u}} \in \operatorname{int} \mathcal{U}$ hold, the initial solution to Problem 12 is feasible and produces a non-zero and positive definite solution ${Q_{\max}}^{(0)} \preceq {\epsilon_1}^{(0)} I_{n_x}$ for some ${\epsilon_1}^{(0)} > 0$.

For any $Q_{\max} \succ 0$, the local multiplier matrix obtained by either (6.82) or (6.83) will be non-zero. Because $Q_{\max}$ is bounded from above and the dynamics (6.18) are assumed to be differentiable, we know that at any iteration, there must exist some $\delta > 0$ such that $\|M_\gamma\|_2 \leq \delta$.[9] Let $\delta^{(1)}$ be this bound at the first iteration. Due to the first assumption in the statement of the theorem, there exists some $\epsilon > 0$ such that there is a feasible solution to the Q-problem at the first iteration that satisfies $\epsilon I_{n_x} \preceq Q^{(1)}$. This implies that matrices that satisfy $Q \preceq \epsilon I_{n_x}$ are also feasible solutions to the Q-problem.

[9] If $\gamma < 1$, then this bound is $\delta = 1$. Otherwise, it is γ^2.

Algorithm 6 The γ-iteration designed to solve the quadratic funnel synthesis problem. Recall that a variable z at the k^{th} iteration is denoted using $z^{(k)}$.

Input: A nominal trajectory $\{\bar{\boldsymbol{x}}(t), \bar{\boldsymbol{u}}(t)\}_{t=t_0}^{t_f}$, the system matrices for (6.33), a decay rate $\alpha \geq 0$, constraint sets $\mathcal{X}$,$\mathcal{U}$ and $\mathcal{X}_f$, integers N_s and n_M, and a tolerance $\kappa_{\text{tol}} \in (0, 1)$.

1 solve Problems (6.51) and (6.54) for $Q_{\max}$ and $R_{\max}$
2 set $Q_{\min} \leftarrow 0$ and $M_\gamma \leftarrow 0$
3 solve the Q-problem to get $\{Q^{(0)}, Y^{(0)}\}$ ▷ Problem 12
4 update ${Q_{\max}}^{(0)} \leftarrow Q^{(0)}$ and ${Y_{\max}}^{(0)} \leftarrow Y^{(0)}$
5 **for** $k = 1, \ldots$ **do**
6 solve either (6.82) or (6.83) to get ${M_\gamma}^{(k)}$ using $\{{Q_{\max}}^{(k-1)}, {Y_{\max}}^{(k-1)}\}$
7 solve the Q-problem to get $\{Q^{(k)}, Y^{(k)}\}$ using ${M_\gamma}^{(k)}$ ▷ Problem 12
8 **if** $\kappa^{(k)}(t_0) \geq \kappa_{\text{tol}}$ **then** ▷ see (6.68)
9 Converged.
10 **break**
11 **else**
12 ${Q_{\max}}^{(k)} \leftarrow \varsigma {Q_{\max}}^{(k-1)}$ and ${Y_{\max}}^{(k)} \leftarrow \varsigma {Y_{\max}}^{(k-1)}$ ▷ contraction step, see (6.84)
13 **end if**
14 **end for**

Output: Time-varying matrices $Q(t)$ and $K(t)$ that define a quadratic funnel in the sense of Definition 3.

Suppose that the algorithm has not converged by iteration $k-1$, and that ${Q_{\max}}^{(k-1)} \preceq {\epsilon_1}^{(k-1)} I_{n_x}$ for some ${\epsilon_1}^{(k-1)} > 0$. The contraction step implies that during the next iteration, we have ${Q_{\max}}^{(k)} \preceq {\epsilon_1}^{(k)} I_{n_x}$, where ${\epsilon_1}^{(k)} = \varsigma {\epsilon_1}^{(k-1)}$ due to the contraction factor $0 < \varsigma < 1$. Because we hold $K_{\max}$ constant through the iterations via the update rules in step 12, we must have $\|{M_\gamma}^{(k)}\|_2 \leq \delta^{(k)}$ for some $\delta^{(k)} \leq \delta^{(k-1)}$. Therefore, there must still exist a feasible solution to the Q-problem at the k^{th} iteration for which $\epsilon I_{n_x} \preceq Q^{(k)}$. By induction on k, each instance of the Q-problem has a feasible solution that satisfies $\epsilon I_{n_x} \preceq Q^{(k)}$.

Because the contraction factor satisfies

$$\varsigma < \varsigma_{\min} + (1 - \varsigma_{\min}) \frac{1}{1 + e^{h(0.5 - \kappa_{\text{tol}})}} < 1,$$

at each iteration prior to convergence, there must exist some finite integer K for which

the value of ${\epsilon_1}^{(K)} < \epsilon$. In this case, we have ${Q_{\max}}^{(K)} \preceq \epsilon I_{n_x}$, at hence ${Q_{\max}}^{(K)}$ is a feasible solution and results in a fill ratio of exactly one. Because the cost function (6.61a) maximizes the volume of the fill ratio at the initial time, the optimal solution at the K^{th} iteration will be ${Q_{\max}}^{(K)}$ and the algorithm will terminate. □

6.5.3 Case Study: 6-DOF Powered Descent Guidance

This section provides a numerical example of the γ-iteration applied to a 6-DOF powered descent and landing problem. This example is intended to be more complex than the satellite attitude problem presented in §6.4.4 in order to showcase the broader capabilities of quadratic funnel synthesis and the γ-iteration in particular.

Because the definition of $\boldsymbol{\eta}$ in (6.20) requires that the difference between the nominal state trajectory, $\bar{\boldsymbol{x}}$, and the actual state trajectory, $\boldsymbol{x}$, be additive (and not multiplicative), we cannot use either unit quaternions or dual quaternions to represent the state of the vehicle. We therefore use the Cartesian variable dynamics, see §3.4.1, and parameterize the attitude using the same 3-2-1 Euler angles as in the example of §6.4.4.

If we use solely the thrust vector from a single (gimbaled) main engine as the control input, then we will find that the linearized system is not stabilizable (in the linear systems sense of the term). As a result, any Lyapunov-type inequality that we must solve in order to synthesize a funnel will be infeasible. While stabilizability in this sense was not an issue for sequential convex programming in Chapter 4, it is a fundamental limitation of funnel synthesis as we have presented it. To alleviate this issue, we must add another actuation mechanism to the problem formulation; differential thrust from multiple rocket engines or a reaction control system will suffice. We shall use the latter for this example.

The state, control, and nonlinear dynamics for this problem are taken to be

$$\boldsymbol{x} = \begin{bmatrix} m \\ \boldsymbol{r}_{\mathcal{L}} \\ \boldsymbol{v}_{\mathcal{L}} \\ \Theta \\ \boldsymbol{\omega}_{\mathcal{B}} \end{bmatrix}, \quad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{F}_{\mathcal{B}} \\ \boldsymbol{\tau}_{\mathcal{B}} \end{bmatrix}, \quad \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} -\alpha \|\boldsymbol{F}_{\mathcal{B}}\|_2 \\ \boldsymbol{v}_{\mathcal{L}} \\ \frac{1}{m}\left(C_{\mathcal{LB}}(\Theta)\boldsymbol{F}_{\mathcal{B}}\right) + \boldsymbol{g}_{\mathcal{L}} \\ T(\Theta)\boldsymbol{\omega}_{\mathcal{B}} \\ J^{-1}\left(\boldsymbol{\tau}_{\mathcal{B}} + \boldsymbol{r}_{u,\mathcal{B}}^{\times}\boldsymbol{F}_{\mathcal{B}} - \boldsymbol{\omega}_{\mathcal{B}}^{\times} J \boldsymbol{\omega}_{\mathcal{B}}\right) \end{bmatrix}, \tag{6.85}$$

where the variables are defined as in §3.4.1 and in §6.4.4. Note that $n_x = 13$ and $n_u = 6$. The nominal trajectory was computed by using the PTR algorithm (see Algorithm 1) with the following boundary conditions and an optimized final time of $t_f = 29.7$ s (cf. the case studies in §4.3):

$$\begin{aligned} \text{At } t_0: \quad & \bar{m} = 3250 \text{ kg}, \quad \bar{\boldsymbol{r}}_{\mathcal{L}} = (250, 0, 433) \text{ m}, \quad \bar{\boldsymbol{v}}_{\mathcal{L}} = (-35.7, 0, -11.8) \text{ m/s}, \\ & \Theta = (0, 59.8, 0) \text{ deg}, \quad \boldsymbol{\omega}_{\mathcal{B}} = 0_{3\times 1} \\ \text{At } t_f: \quad & \bar{m} = 3130.3 \text{ kg}, \quad \bar{\boldsymbol{r}}_{\mathcal{L}} = (0, 0, 30) \text{ m}, \quad \bar{\boldsymbol{v}}_{\mathcal{L}} = (0, 0, -1) \text{ m/s}, \\ & \Theta = 0_{3\times 1}, \quad \boldsymbol{\omega}_{\mathcal{B}} = 0_{3\times 1}. \end{aligned}$$

For this example, we enforce two nonlinear control constraints that serve to demonstrate a case where $\mathcal{U} \neq \mathcal{P}_{\bar{\boldsymbol{u}}}$. In particular, the nominal trajectory is computed using both upper and lower thrust bounds (see (3.62)) and a gimbal angle constraint (see (3.63)). Using the thrust vector $\boldsymbol{F}_{\mathcal{B}}$, these constraints are

$$F_{\min} \leq \|\boldsymbol{F}_{\mathcal{B}}\|_2 \leq F_{\max} \quad \text{and} \quad \|\boldsymbol{F}_{\mathcal{B}}\|_2 \leq \sec \delta_{\max} \boldsymbol{z}_{\mathcal{B}}^{\top} \boldsymbol{F}_{\mathcal{B}}. \tag{6.86}$$

The constraint sets $\mathcal{X}, \mathcal{U}$ and $\mathcal{X}_f$ are taken to be

$$\mathcal{X} = \left\{ \boldsymbol{x} \,|\, \boldsymbol{x}_{lb} \leq \boldsymbol{x} \leq \boldsymbol{x}_{ub},\ \delta\boldsymbol{x}_{lb} \leq \boldsymbol{x} - \bar{\boldsymbol{x}} \leq \delta\boldsymbol{x}_{ub},\ \|\boldsymbol{x}\|_2 \leq \infty \right\}, \tag{6.87a}$$

$$\mathcal{U} = \left\{ \boldsymbol{u} \,|\, \boldsymbol{u}_{lb} \leq \boldsymbol{u} \leq \boldsymbol{u}_{ub},\ \ (6.86) \right\}, \tag{6.87b}$$

$$\mathcal{X}_f = \mathcal{E}_{Q_{\max,f}}, \quad Q_{\max,f}^{1/2} = \mathbf{diag}\left\{1637, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{\pi}{60}, \tfrac{\pi}{60}, \tfrac{\pi}{60}, \tfrac{\pi}{60}, \tfrac{\pi}{60}, \tfrac{\pi}{60}\right\}. \tag{6.87c}$$

Note that $\mathcal{X}$ enforces both an absolute bound on the state vector, as well as a bound on the deviation from the nominal. The terminal constraints imposed by $\mathcal{X}_f$ ensures that any trajectory does not deviate from the nominal path by more than 0.5 m in position, 0.25 m/s in velocity, or 3 deg and 3 deg/s in attitude and angular rate in any direction. The sets $\mathcal{X}$ and $\mathcal{U}$ are described by

$$\begin{aligned}
\boldsymbol{x}_{lb} &= -\left(-2100, 150, 150, 0, 40, 40, 30, \pi, \tfrac{\pi}{2}, \pi, 0.5, 0.5, 0.5\right), \\
\boldsymbol{x}_{ub} &= +\left(3737.7, 350, 300, 500, 30, 30, 5, \pi, \tfrac{\pi}{2}, \pi, 0.5, 0.5, 0.5\right), \\
\delta\boldsymbol{x}_{lb} &= -\left(\infty, 100, 100, 100, \infty, \infty, \infty, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi\right), \\
\delta\boldsymbol{x}_{ub} &= +\left(\infty, 100, 100, 100, \infty, \infty, \infty, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi, \tfrac{2}{9}\pi\right), \\
\boldsymbol{u}_{lb} &= -\left(7695.5,\ 7695.5,\ -5400,\ 150,\ 150,\ 150\right), \\
\boldsymbol{u}_{ub} &= +\left(7695.5,\ 7695.5,\ \ 24750,\ 150,\ 150,\ 150\right).
\end{aligned}$$

The remaining data for this problem are provided in Table 6.2, and are (loosely) based on an Apollo-class lander.

The matrices A and B are the partial derivatives of f along the reference trajectory, and the parameters C, D and E are constructed using a total of six ($N_p = 6$) nonlinear channels to be

$$C = \begin{bmatrix}
I_1 & 0_{1\times3} & 0_{1\times3} & 0_{1\times3} & 0_{1\times3} \\
0_{3\times1} & 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} \\
0_{3\times1} & 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} \\
0_{3\times1} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 \\
0_{3\times1} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 \\
0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3}
\end{bmatrix}, \quad D = \begin{bmatrix}
0_{1\times3} & 0_{1\times3} \\
0_{3\times3} & 0_{3\times3} \\
0_{3\times3} & 0_{3\times3} \\
0_{3\times3} & 0_{3\times3} \\
0_{3\times3} & 0_{3\times3} \\
I_3 & 0_{3\times3}
\end{bmatrix},$$

Table 6.2: The parameters for the powered descent case study used to demonstrate the γ-iteration.

Parameter	**Value**	**Parameter**	**Value**
J	$\mathbf{diag}\left\{13600, 13600, 19150\right\}$ kg m^2	I_{sp}	225 s/m
$\boldsymbol{r}_{F,\mathcal{B}}$	$\left(0, 0, -0.25\right)$ m	$\boldsymbol{g}_{\mathcal{I}}$	$\left(0, 0, -1.62\right)$ m/s^2
$F_{\min}$	5400 N	$F_{\max}$	24750 N
$\delta_{\max}$	25 deg	N_s, n_M	100, 25
κ_{tol}	0.5	α	0.1 s

$$E = \begin{bmatrix} 0_{1\times3} & 0_{1\times3} & 0_{1\times3} & 0_{1\times3} & 0_{1\times3} & I_1 \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{1\times1} \\ I_3 & I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{1\times1} \\ 0_{3\times3} & 0_{3\times3} & I_3 & I_3 & 0_{3\times3} & 0_{1\times1} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 & 0_{1\times1} \end{bmatrix},$$

and therefore we have $n_p = 16$ and $n_q = 16$ for this problem.

Again, we employ Assumption 6.1 in order to temporally discretize each problem variable for the Q-problem, with the exception of ν and γ, which are each be taken to be piecewise constant. Only the former is without loss of generality, the latter can introduce some additional conservatism.

The γ-iteration, as given in Algorithm 6, converges after 6 iterations for this problem setup. In the state space, the synthesized quadratic funnel has the following projections onto each of the n_x state dimensions at the initial time t_0:

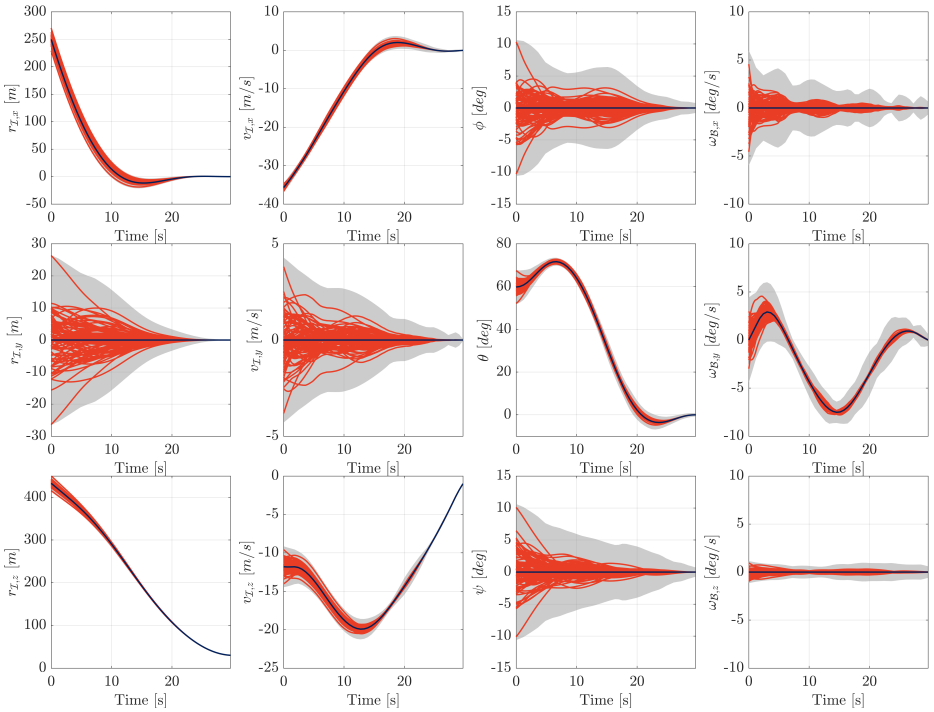
$$m : 3.0 \text{ kg}, \quad \boldsymbol{r}_{\mathcal{L}} : \begin{bmatrix} 20.6 \\ 26.5 \\ 17.5 \end{bmatrix} \text{ m}, \quad \boldsymbol{v}_{\mathcal{L}} : \begin{bmatrix} 1.2 \\ 4.3 \\ 2.7 \end{bmatrix} \text{ m/s}, \quad \Theta : \begin{bmatrix} 10.6 \\ 8.0 \\ 10.6 \end{bmatrix} \text{ deg}, \quad \boldsymbol{\omega}_{\mathcal{B}} : \begin{bmatrix} 6.0 \\ 4.4 \\ 1.1 \end{bmatrix} \text{ deg/s}.$$

Figure 6.6 shows the computed quadratic funnel. In Figure 6.6a, the ellipsoid $\mathcal{E}_Q$ was projected onto each state dimension (mass is omitted) and is depicted as the shaded gray area.

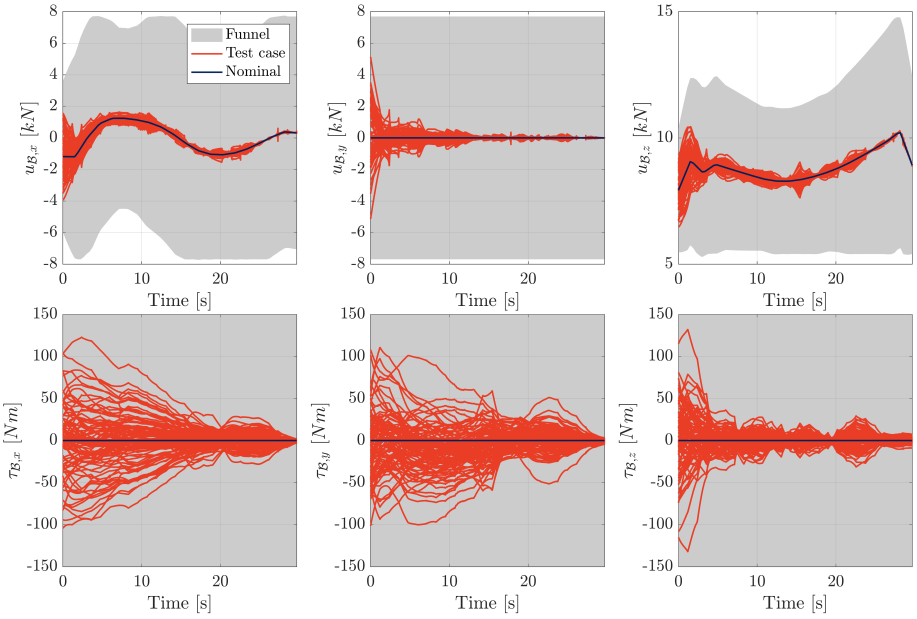
The red trajectories correspond to test cases for which an initial condition was randomly (uniformly) selected from the ellipsoid $\mathcal{E}_{Q(t_0)}$, and the nominal control and correction law were used to numerically integrate the equations of motion according to (6.59). Figure 6.6b shows the ellipsoid $\mathcal{E}_{R_{\max}}$ projected into each control dimension along with the corresponding control trajectories from each test case. Figure 6.7 further displays the three-dimensional thrust space over the entire maneuver. The portion of the feasible set $\mathcal{U}$ that corresponds to the thrust constraints, constructed from the (convex) thrust upper bound, (nonconvex) thrust lower bound and (convex) gimbal angle constraints, is shown in green in order to visually confirm the feasibility of the quadratic funnel in the thrust-space.

Figure 6.8a provides the fill ratio versus the iteration number. To give a better sense of how the convergence process looks, the fill ratios shown are computed by using the three-dimensional ellipsoids that result from projecting both $\mathcal{E}_Q$ and $\mathcal{E}_{Q_{\max}}$ into each of the position, velocity, attitude, and angular rate dimensions. It can be seen that by the last iteration, each of these projections has surpassed the desired fill ratio, with the attitude and angular rate ratio's being nearly at the theoretical limit of one.

These results are quite promising – the powered descent problem is a challenging problem with nonlinear dynamics, some nonlinear and nonconvex constraints, and relatively large state and control dimensions. What these results show is that we are able to generate feasible trajectories (both dynamically and with respect to the constraints that were considered) for *any initial condition* in a set that stretches more than 35 m in every position direction, 2 m/s in each velocity direction, 16 deg in each Euler angle, and 2 deg/s in each angular velocity direction, all by using a single quadratic funnel.



(a) In state space, $\mathcal{E}_Q$.



(b) In control space, $\mathcal{E}_{R_{\max}}$.

Figure 6.6: The quadratic funnel computed by the γ-iteration for the 6-DOF powered descent problem. The initial condition of each test case was randomly sampled from the funnel entry.

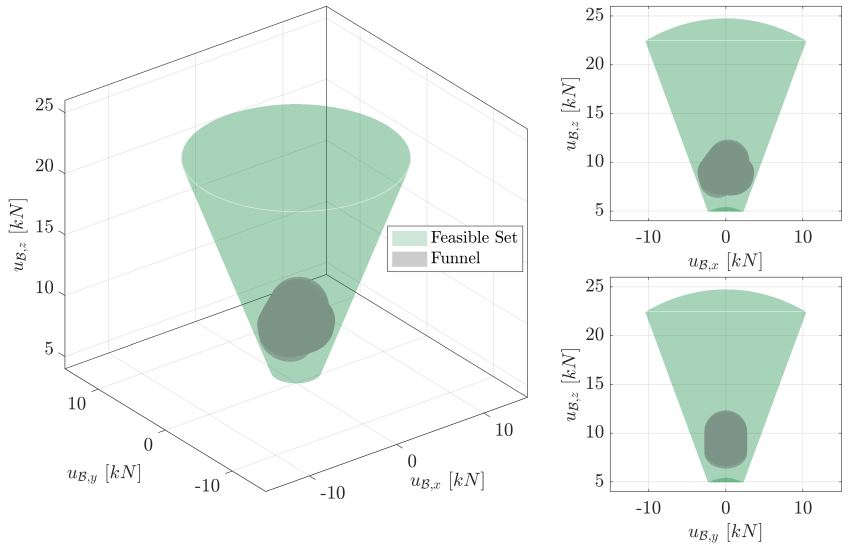


Figure 6.7: The ellipsoid $\mathcal{E}_{R_{\max}}$ projected into thrust space (gray) compared to the thrust portions of the original feasible set, $\mathcal{U}$, (green). All thrust trajectories must lie inside the gray set.

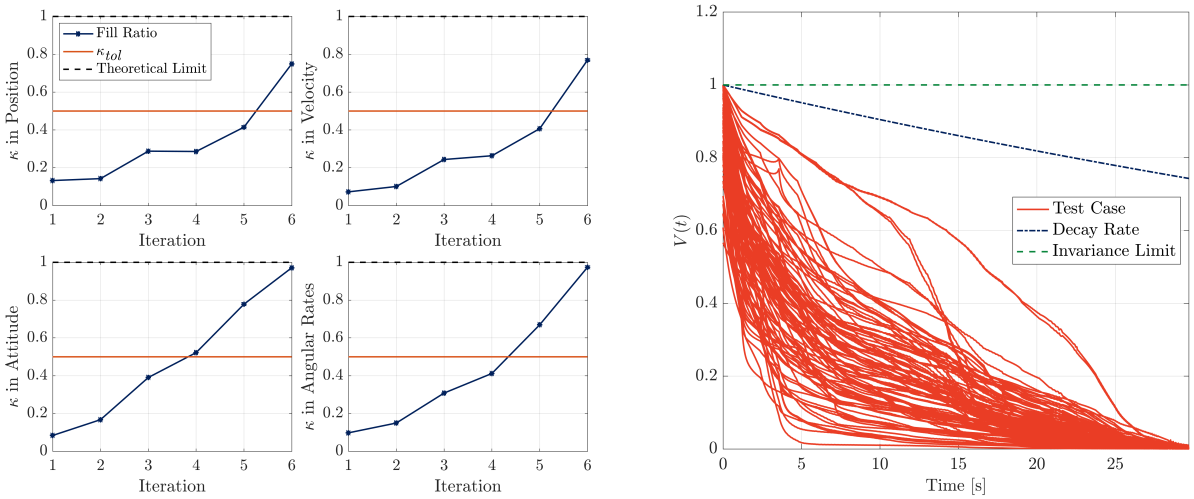


Figure 6.8: The fill ratio of $\mathcal{E}_{Q(t_0)}$ projected into four states across each iteration of the γ-iteration, and the value of the Lyapunov function $V(t)$ for each test case for the powered descent problem.

Chapter 7
CONCLUDING REMARKS

Each day it came a little nearer,
and each day he made its coming a little easier,
for was not his work...a process of progressive failing?
– Doctor Copernicus, John Banville

The theme of this dissertation is computational guidance and control. The primary objective was to develop new techniques for real-time trajectory optimization that can be used on resource-constrained aerospace systems. Two different but related approaches were detailed, namely explicit and implicit trajectory optimization. Throughout the text, examples in powered descent and satellite attitude control were used to highlight the capabilities of both approaches.

The main contribution of this work is an advancement in the state of the art in trajectory generation. Explicit trajectory optimization using sequential convex programming is not new, per se, but the maturity of the algorithm and its tailoring to real-time implementation have advanced the readiness level of the technology to the point that it will have flown in space on board two different space vehicles as a result of this work. The general strategy to relate high-level algorithm development with real-time implementations is essential in order to establish computational guidance and control as a viable element of a flight software system.

For powered descent specifically, Chapter 3 likely represents the most comprehensive overview of the various problem formulations available in the literature. The definition of and solution to the planar landing problem are new, and the scope of the 6-DOF landing problem is the widest that has been presented thus far. State-triggered constraints are

a new way to represent logical relationships between optimization variables and problem constraints without resorting to mixed-integer formulations. When it comes to subsequent real-time algorithm implementation using mature (and flight proven) convex optimization solvers, state-triggered constraints are an enabling design tool that allow the intended logical implication to be properly represented, without requiring new solvers or sacrificing runtime performance.

Often, simply finding a feasible solution to an optimal control problem with several state and control constraints is challenging. As an offline design tool, SCP-based algorithms offer a way to obtain feasible trajectories that can serve as reference points for subsequent mission design and analysis. The ability to directly consider constraints can also enable an easier study of new and ambitious mission architectures, such as landing near geological features on Mars by using Hazard Relative Navigation, or by extending the operating life of a satellite without changing the existing hardware by using more exotic maneuvers that save propellant. The fortuitous bi-product of near-optimality also implies that these trajectories can offer a good upper-bound on the achievable cost (e.g., propellant mass). Reduced conservatism in the estimation of fuel consumption across a space vehicle's concept of operations could lead to a very valuable reduction in the required launch mass.

The order of Chapters 3-5 is a reflection of how I believe trajectory optimization should be approached. First, an understanding of the (most general) equations of motion and problem constraints is required. These equations can be simplified as appropriate to develop intuition and understanding of special-case scenarios. While it is often difficult (and rather unlikely) to be able to fully solve a complex and highly constrained optimal control problem using the maximum principle, these properly simplified versions can provide invaluable insight. The development of a numerical algorithm should not be undertaken with a blind eye to the tools of optimal control. In reality, a parallel development of theoretical understanding and numerical algorithms can be carried out. Often times, details observed during numerical trials will help reveal where to begin theoretical analysis, or what to attempt to prove or disprove. Conversely, parameters for numerical trials can be contrived to demonstrate

theoretical properties of optimal solutions. Preliminary algorithm development should be done using a high-level programming language that prioritizes ease of development over computational resource utilization. The final step is to develop a real-time implementation using a lower-level compiled programming language that reverses the priorities. Because flight hardware is almost always resource constrained, minimizing the runtime and memory footprint of the resulting implementation while adhering to the standards of safety-critical code are of paramount importance. During this step, the specifics of the problem being solved should be exploited to the maximum extent possible; specificity often leads to more efficient algorithms.

The funnel-based implicit trajectory optimization methods in Chapter 6 represent a new approach to funnel synthesis and feasible trajectory design. The definitions, algorithms, and results have not yet been presented elsewhere. Implicit trajectory optimization represents a paradigm shift from the more standard optimal-control-based explicit trajectory generation methods, in so far as the state and control vectors are given implicitly, rather than explicitly, as functions of time. The funnel synthesis algorithms are able to provide a very important feature that is not available from SCP-based explicit trajectory optimization, namely that the real-time computation of a feasible trajectory can be *theoretically guaranteed* from a certain set of initial conditions, the funnel entry. If all problem and algorithm parameters are held fixed, then SCP-based methods cannot be guaranteed *in general* to have a 100 percent success rate across a fixed set of initial conditions, whereas funnel synthesis can. Moreover, the number of floating point operations required to compute the implicit trajectory can be easily computed, offering a computational complexity bound on the steps that must be executed onboard the vehicle. The same statement cannot be made for SCP-based algorithms used to solve nonconvex optimal control problems. As a result, implicit trajectory optimization significantly reduces the amount of computations required in real-time, though perhaps at the expense of additional memory storage requirements.

Flight Implementations of the PTR Algorithm

Ultimately, all of the algorithms discussed in this work are designed to be capable of being integrated into the flight software of an aerospace vehicle. Those who decide whether or not such algorithms warrant integration are not always interested in the mathematical elegance of the solution method. Rather, flight tests and rigorous numerical experimentation that is backed with a solid understanding of the underlying theory is the best (and perhaps only) way to convince those who remain skeptical about real-time optimization-based guidance and control. And this skepticism is not unwarranted, primarily because there is limited flight heritage for this class of algorithms. At the time of writing, I am proud to say that an implementation of the PTR algorithm has been flown onboard Blue Origin's New Shepard vehicle to solve a 6-DOF powered descent problem in an open-loop fashion (i.e., the algorithm received flight telemetry, computed solutions, and stored the results, but did not control the rocket's engines). To the best of our knowledge, this was the first time that sequential convex programming was used in space, and represents a significant advancement in the field of computational guidance and control. Subsequent flight tests are being planned, and there is mounting evidence that real-time optimization-based guidance can and will play an important role in the design of future precision landing missions.

A separate implementation of the PTR algorithm will be flown on the SOC-i satellite that is planned for launch in 2022-23. This algorithm is designed to solve constrained attitude control problems with attitude keep-out and keep-in zones [219]. To the best of our knowledge, this will be the first *closed-loop* demonstration of sequential convex programming on any orbiting spacecraft. This will provide significant flight heritage, and the ability to demonstrate the capabilities of these types of algorithms repeatedly over a six month mission.

Considerations for Future Work

Objectively, these two demonstrations advance the technology readiness level of the PTR algorithm, and lower the psychological barrier to pursuing computational guidance methods

in future applications. It is worth stressing at the same time that there remains a significant amount of work to be done in this area.

Foremost are algorithmic changes to improve the runtime characteristics of real-time implementations. As mentioned in Chapter 5, the dominating factor that determines the algorithm's runtime is the number of PTR iterations. At the same time, SCP-based methods can be susceptible to a *crawling phenomenon* that increases the number of PTR iterations without greatly improving the quality of the solution [147]. Of course, faster solvers will always lead to faster SCP algorithms, but the relative maturity of interior point methods likely implies that runtime improvements due to faster solvers will be incremental.[1] Changes to the algorithm's basic structure that allow each iteration to make more progress towards a solution and to avoid the crawling phenomenon should be explored. Techniques analogous to second-order corrections, acceleration methods, and other augmentations designed to achieve faster and more robust algorithms have been studied in the classical optimization literature for decades, but have not been explored in sufficient detail, somewhat paradoxically, for SCP-based trajectory optimization. Any progress made in this direction will increase the likelihood that the algorithm will be real-time capable on resource-constrained spaceflight hardware.

With regards to funnel synthesis, Chapter 6 devised two provably-convergent algorithms that are each capable of synthesizing quadratic funnels for problems of interest in the aerospace domain. But funnel synthesis and nominal trajectory generation are coupled; the nominal trajectory affects the computable funnel. It follows that in order to truly maximize the size of the funnel, the nominal trajectory should be chosen simultaneously. It is in fact possible to do so. The nonconvex quadratic funnel synthesis problem, Problem 11, can be augmented to be a "trajectory" optimization problem in the same spirit as a canonical optimal control problem. This is done by reformulating the differential matrix inequality into a set of ordinary differential equations and algebraic constraints. By combining these

[1] Specifically the class SOCP-based solvers.

two ideas, we would obtain a rather fascinating design tool with which a nominal trajectory and quadratic funnel are selected simultaneously.

Sensitivity of the computable funnel size with respect to vehicle parameters can also provide insight into how to design the vehicle in the first place. A large funnel, loosely speaking, implies a dynamical system that is easily controllable even in the presence of a given set of state and control constraints. Assessing the relationship between vehicle parameters, such as mass distribution (inertia) or actuator placement, and controllability in a nonlinear and constrained sense is a fundamental challenge in vehicle design, especially early in the prototype phase before any detailed control analysis and simulations have been carried out. Funnel synthesis can offer an indication as to whether or not a particular set of vehicle parameters improve or degrade the ease with which the vehicle may be controlled during different operating conditions.

Lastly, we have not thoroughly studied the composition of several funnels to cover a given set. Consider a scenario where a landing vehicle is given a 3σ ellipsoid of possible initial conditions at the beginning of a powered descent phase. Suppose that a funnel is generated about a nominal path, and the entry to this funnel is smaller than the given 3σ ellipsoid. To ensure the availability of a feasible trajectory from anywhere inside the 3σ ellipsoid, we would need to then compute enough trajectory-funnel pairs so that the union of all of the funnel entries overbounds the 3σ ellipsoid. Obviously we would like to find the *minimum* number of such pairs. How to do so is not obvious, and merits further study.

BIBLIOGRAPHY

[1] P. Lu, "Introducing Computational Guidance and Control," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 193–193, 2017.

[2] P. Tsiotras and M. Mesbahi, "Toward an Algorithmic Control Theory," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 194–196, 2017.

[3] L. D. Berkovitz, *Optimal Control Theory*. New York, NY: Springer-Verlag, 1974.

[4] D. Liberzon, *Calculus of Variations and Optimal Control Theory*. Princeton, NJ: Princeton University Press, 2007.

[5] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*. Montreux, Switzerland: Gordon and Breach Science Publishers, 1986.

[6] L. D. Berkovitz and N. G. Medhin, *Nonlinear Optimal Control Theory*. Boca Raton, FL: CRC Press, 2013.

[7] G. Dantzig, *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press, 2016.

[8] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY: Springer Science & Business Media, 2006.

[9] R. T. Rockafellar, *Convex Analysis*. Princeton, NJ: Princeton University Press, 1970.

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

[11] E. D. Andersen and K. D. Andersen, "The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm," in *High Performance Optimization* (H. Frenk, K. Roos, T. Terlaky, and S. Zhang, eds.), vol. 33, pp. 197–232, Boston, MA: Springer, 2000.

[12] M. Andersen, J. Dahl, Z. Liu, and L. Vandenberghe, "Interior-Point Methods for Large-Scale Cone Programming," in *Optimization for Machine Learning* (S. Sra, S. Nowozin, and S. J. Wright, eds.), pp. 55–83, MIT Press, 2012.

[13] M. Andersen, J. Dahl, and L. Vandenberghe, "CVXOPT: A Python Package for Convex Optimization." online, 2019.

[14] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP Solver for Embedded Systems," in *European Control Conference*, (Zurich, Switzerland), pp. 3071–3076, 2013.

[15] D. Dueri, B. Açıkmeşe, D. P. Scharf, and M. W. Harris, "Customized Real-Time Interior-Point Methods for Onboard Powered-Descent Guidance," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 197–212, 2016.

[16] D. Dueri, J. Zhang, and B. Açıkmeşe, "Automated Custom Code Generation for Embedded, Real-time Second Order Cone Programming," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 1605–1612, 2014.

[17] J. F. Sturm, "Using SeDuMi 1.02, a MATLAB Toolbox for Optimization over Symmetric Cones," *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 625–653, 1999.

[18] K.-C. Toh, M. J. Todd, and R. H. Tütüncü, "SDPT3: A Matlab Software Package for Semidefinite Programming, Version 1.3," *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 545–581, 1999.

[19] M. Grant and S. Boyd, "CVX: Matlab Software for Disciplined Convex Programming, version 2.1." online, 2014.

[20] J. Lofberg, "YALMIP: A Toolbox for Modeling and Optimization in Matlab," in *Proceedings of the CACSD Conference*, (Taipei, Taiwan), 2004.

[21] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*. Philadelphia, PA: SIAM, 2001.

[22] B. Conway, ed., *Spacecraft Trajectory Optimization*. New York, NY: Cambridge University Press, 2010.

[23] J. T. Betts, "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[24] D. Garg, M. Patterson, W. W. Hager, A. V. Rao, D. A. Benson, and G. T. Huntington, "A Unified Framework for the Numerical Solution of Optimal Control Problems using Pseudospectral Methods," *Automatica*, vol. 46, no. 11, pp. 1843–1851, 2010.

[25] X. Liu and P. Lu, "Solving Nonconvex Optimal Control Problems by Convex Optimization," *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 3, pp. 750–765, 2014.

[26] X. Liu, P. Lu, and B. Pan, "Survey of Convex Optimization for Aerospace Applications," *Astrodynamics*, vol. 1, no. 1, pp. 1–23, 2017.

[27] J. Strizzi, I. M. Ross, and F. Fahroo, "Towards Real-Time Computation of Optimal Controls for Nonlinear Systems," in *AIAA Guidance, Navigation, and Control Conference*, (Monterey, CA), 2002.

[28] A. H. J. de Ruiter, C. J. Damaren, and J. R. Forbes, *Spacecraft Dynamics and Control*. Chichester, United Kingdom: John Wiley & Sons Ltd., 2013.

[29] P. C. Hughes, *Spacecraft Attitude Dynamics*. Toronto, Canada: John Wiley & Sons, 1986.

[30] W. R. Hamilton, "On Quaternions, or On A New System of Imaginaries in Algebra," *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, vol. xxv, 1844.

[31] J. M. McCarthy, *Introduction to Theoretical Kinematics*. Cambridge, MA: The MIT Press, 1990.

[32] U. Lee and M. Mesbahi, "Optimal Powered Descent Guidance with 6-DoF Line of Sight Constraints via Unit Dual Quaternions," in *AIAA Guidance, Navigation, and Control Conference*, (Kissimmee, FL), 2015.

[33] L. Blackmore, "Autonomous Precision Landing of Space Rockets," *National Academy of Engineering: 'The Bridge on Frontiers of Engineering'*, vol. 4, no. 46, pp. 15–20, 2016.

[34] J. M. Carson III, M. M. Munk, R. R. Sostaric, J. N. Estes, F. Amzajerdian, J. B. Blair, D. K. Rutishauser, C. I. Restrepo, A. Dwyer-Cianciolo, G. T. Chen, and T. Tse, "The SPLICE Project: Continuing NASA Development of GN&C Technologies for Safe and Precise Landing," in *AIAA SciTech Forum*, (San Diego, CA), 2019.

[35] A. R. Klumpp, "Apollo Lunar-Descent Guidance," Tech. Rep. R-695, Charles Stark Draper Laboratory, 1971.

[36] G. F. Mendeck and L. E. Craig, "Entry Guidance for the 2011 Mars Science Laboratory Mission," in *AIAA Atmospheric Flight Mechanics Conference*, (Portland, OR), 2011.

[37] G. Singh, A. M. San Martin, and E. C. Wong, "Guidance and Control Design for Powered Descent and Landing on Mars," in *IEEE Aerospace Conference*, (Big Sky, MT), 2007.

[38] R. R. Sostaric and J. Rea, "Powered Descent Guidance Methods For The Moon and Mars," in *AIAA Guidance, Navigation, and Control Conference*, (San Francisco, CA), 2005.

[39] T. N. Edelbaum, "Optimal Space Trajectories," Tech. Rep. TR-AD0701112, Jericho, NY, 1969.

[40] D. F. Lawden, *Optimal Trajectories for Space Navigation*. London, United Kingdom: Butterworths, 1963.

[41] J. S. Meditch, "On the Problem of Optimal Thrust Programming For a Lunar Soft Landing," *IEEE Transactions on Automatic Control*, vol. 9, no. 4, pp. 477–484, 1964.

[42] R. McHenry, A. Long, B. Cockrell, J. Thibodeau III, and T. Brand, "Space Shuttle Ascent Guidance, Navigation, and Control," *Journal of the Astronautical Sciences*, vol. 27, pp. 1–38, 1979.

[43] T. J. Brand, D. W. Brown, and J. P. Higgins, "Unified Powered Flight Guidance," Tech. Rep. CR-134149, NASA, 1973.

[44] A. G. Azizov and N. A. Korshunova, "On An Analytical Solution of the Optimum Trajectory Problem in a Gravitational Field," *Celestial Mechanics*, vol. 38, no. 4, pp. 297–306, 1986.

[45] J. V. Breakwell and J. F. Dixon, "Minimum-Fuel Rocket Trajectories Involving Intermediate-Thrust Arcs," *Journal of Optimization Theory and Applications*, vol. 17, no. 5, pp. 465–479, 1975.

[46] A. L. Kornhauser and P. M. Lion, "Optimal Deterministic Guidance for Bounded-Thrust Spacecrafts," *Celestial Mechanics*, vol. 5, no. 3, pp. 261–281, 1972.

[47] J.-P. Marec, *Optimal Space Trajectories*. Amsterdam, Netherlands: Elsevier Scientific Publishing Company, 1979.

[48] C. N. D'Souza, "An Optimal Guidance Law for Planetary Landing," in *AIAA Guidance, Navigation, and Control Conference*, (New Orleans, LA), 1997.

[49] U. Topcu, J. Casoliva, and K. D. Mease, "Fuel Efficient Powered Descent Guidance for Mars Landing," in *AIAA Guidance, Navigation, and Control Conference*, (San Francisco, CA), 2005.

[50] U. Topcu, J. Casoliva, and K. D. Mease, "Minimum-Fuel Powered Descent for Mars Pinpoint Landing," *Journal of Spacecraft and Rockets*, vol. 44, no. 2, pp. 324–331, 2007.

[51] P. Lu, "Propellant-Optimal Powered Descent Guidance," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 4, pp. 813–826, 2018.

[52] F. Najson and K. Mease, "A Computationally Non-Expensive Guidance Algorithm for Fuel Efficient Soft Landing," in *AIAA Guidance, Navigation, and Control Conference*, (San Francisco, CA), 2005.

[53] J. R. Rea and R. Bishop, "Analytical Dimensional Reduction of a Fuel Optimal Powered Descent Subproblem," in *AIAA Guidance, Navigation, and Control Conference*, (Toronto, Canada), 2010.

[54] P. Lu, S. Forbes, and M. Baldwin, "A Versatile Powered Guidance Algorithm," in *AIAA Guidance, Navigation ,and Control Conference*, (Minneapolis, MN), 2012.

[55] B. Açıkmeşe and S. Ploen, "A Powered Descent Guidance Algorithm for Mars Pinpoint Landing," in *AIAA Guidance, Navigation, and Control Conference*, (San Francisco, CA), 2005.

[56] B. Açıkmeşe and S. Ploen, "Convex Programming Approach to Powered Descent Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, pp. 1353–1366, 2007.

[57] S. Ploen, B. Açıkmeşe, and A. Wolf, "A Comparison of Powered Descent Guidance Laws for Mars Pinpoint Landing," in *AIAA/AAS Astrodynamics Specialist Conference*, (Keystone, CO), 2006.

[58] B. Açıkmeşe, J. M. Carson III, and L. Blackmore, "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2104–2113, 2013.

[59] J. M. Carson III, B. Açıkmeşe, L. Blackmore, and A. Wolf, "Capabilities of Convex Powered-Descent Guidance Algorithms for Pinpoint and Precision Landing," in *IEEE Aerospace Conference*, (Big Sky, MT), 2011.

[60] J. M. Carson III, B. Açıkmeşe, and L. Blackmore, "Lossless Convexification of Powered-Descent Guidance with Non-Convex Thrust Bound and Pointing Constraints," in *IEEE American Control Conference*, (San Francisco, CA), 2011.

[61] L. Blackmore, B. Açıkmeşe, and D. P. Scharf, "Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 4, pp. 1161–1171, 2010.

[62] B. Açıkmeşe and L. Blackmore, "Lossless Convexification of a Class of Optimal Control Problems with Non-Convex Control Constraints," *Automatica*, vol. 47, no. 2, pp. 341–347, 2011.

[63] L. Blackmore, B. Açıkmeşe, and J. M. Carson III, "Lossless Convexification of Control Constraints for a Class of Nonlinear Optimal Control Problems," *Systems and Control Letters*, vol. 61, no. 8, pp. 863–870, 2012.

[64] M. W. Harris and B. Açıkmeşe, "Lossless Convexification of Non-Convex Optimal Control Problems for State Constrained Linear Systems," *Automatica*, vol. 50, no. 9, pp. 2304–2311, 2014.

[65] D. Malyuta, M. Szmuk, and B. Açıkmeşe, "Lossless Convexification of Non-Convex Optimal Control Problems with Disjoint Semi-Continuous Inputs," *arXiv e-prints*, 2019. arXiv ID:1902.02726.

[66] D. P. Scharf, B. Açıkmeşe, D. Dueri, J. Benito, and J. Casoliva, "Implementation and Experimental Demonstration of Onboard Powered-Descent Guidance," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 213–229, 2017.

[67] D. P. Scharf, M. W. Regehr, G. M. Vaughan, J. Benito, H. Ansari, M. Aung, A. Johnson, J. Casoliva, S. Mohan, D. Dueri, B. Açıkmeşe, D. Masten, and S. Nietfeld, "ADAPT Demonstrations of Onboard Large-Divert Guidance with a VTVL Rocket," in *IEEE Aerospace Conference*, (Big Sky, MT), 2014.

[68] A. M. Dwyer-Cianciolo, S. Striepe, J. M. Carson III, R. R. Sostaric, D. Woffinden, C. D. Karlgaard, R. A. Lugo, R. Powell, and J. Tynis, "Defining Navigation Requirements for Future Missions," in *AIAA SciTech Forum*, (San Diego, CA), 2019.

[69] T. P. Reynolds and M. Mesbahi, "Optimal Planar Powered Descent with Independent Thrust and Torque," *Journal of Guidance, Control, and Dynamics*, vol. 47, no. 3, pp. 1225–1231, 2020.

[70] U. Lee and M. Mesbahi, "Dual Quaternions, Rigid Body Mechanics, and Powered-Descent Guidance," in *IEEE Conference on Decision and Control*, (Maui, HI), 2012.

[71] U. Lee and M. Mesbahi, "Constrained Autonomous Precision Landing via Dual Quaternions and Model Predictive Control," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 292–308, 2017.

[72] T. P. Reynolds, M. Szmuk, D. Malyuta, M. Mesbahi, B. Açıkmeşe, and J. M. Carson III, "A State-Triggered Line of Sight Constraint for 6-DoF Powered Descent Guidance Problems," in *AIAA SciTech Forum*, (San Diego, CA), 2019.

[73] T. P. Reynolds, M. Szmuk, D. Malyuta, M. Mesbahi, B. Açıkmeşe, and J. M. Carson III, "Dual Quaternion-Based Powered Descent Guidance with State-Triggered Constraints," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 9, pp. 1584–1599, 2020.

[74] M. Szmuk and B. Açıkmeşe, "Successive Convexification for 6-DoF Mars Rocket Powered Landing with Free-Final-Time," in *AIAA Guidance, Navigation, and Control Conference*, (Orlando, FL), 2018.

[75] M. Szmuk, B. Açıkmeşe, and A. W. Berning, "Successive Convexification for Fuel-Optimal Powered Landing with Aerodynamic Drag and Non-Convex Constraints," in *AIAA Guidance, Navigation, and Control Conference*, (San Diego, CA), 2016.

[76] M. Szmuk, U. Eren, and B. Açıkmeşe, "Successive Convexification for Mars 6-DoF Powered Descent Landing Guidance," in *AIAA Guidance, Navigation, and Control Conference*, (Grapevine, TX), 2017.

[77] M. Szmuk, T. P. Reynolds, B. Açıkmeşe, M. Mesbahi, and J. M. Carson III, "Successive Convexification for Real-Time Rocket Landing Guidance with Compound State-Triggered Constraints," in *AIAA SciTech Forum*, (San Diego, CA), 2019.

[78] M. Szmuk, T. P. Reynolds, and B. Açıkmeşe, "Successive Convexification for Real-Time 6-DoF Powered Descent Guidance with State-Triggered Constraints," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 8, pp. 1399–1413, 2020.

[79] D. J. Jezewski, "An Optimal, Analytic Solution to the Linear-Gravity, Constant-Thrust Trajectory Problem," *Journal of Spacecraft and Rockets*, vol. 8, no. 7, pp. 793–796, 1971.

[80] J. R. Rea, *An Investigation of Fuel Optimal Terminal Descent.* Ph.D. Dissertation, University of Texas at Austin, 2009.

[81] M. W. Harris, *Lossless Convexification of Optimal Control Problems.* Ph.D. Dissertation, University of Texas at Austin, 2014.

[82] M. Szmuk, *Successive Convexification & High Performance Feedback Control for Agile Flight.* Ph.D. Dissertation, University of Washington, 2019.

[83] W. T. Thomson, *Introduction to Space Dynamics*. Toronto, Canada: Dover Publications, Inc., 1986.

[84] P. C. Garcia, L. E. M. Hernandez, and P. G. Gil, *Indoor Navigation Strategies for Aerial Autonomous Systems*. Cambridge, MA: Butterworth-Heinemann, 2017.

[85] N. Filipe and P. Tsiotras, "Adaptive Position and Attitude-Tracking Controller for Satellite Proximity Operations Using Dual Quaternions," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 4, pp. 566–577, 2015.

[86] A. Valverde and P. Tsiotras, "Dual Quaternion Framework for Modeling of Spacecraft-Mounted Multibody Robotic Systems," *Frontiers in Robotics and AI*, vol. 5, p. 128, 2018.

[87] T. P. Reynolds and M. Mesbahi, "Coupled 6-DOF Control for Distributed Aerospace Systems," in *IEEE Conference on Decision and Control*, (Miami Beach, FL), 2018.

[88] R. Hartl, S. Sethi, and R. Vickson, "A Survey of the Maximum Principles for Optimal Control Problems With State Constraints," *SIAM Review*, vol. 37, no. 2, pp. 181–218, 1995.

[89] R. R. Sostaric, "Powered Descent Trajectory Guidance and Some Considerations for Human Lunar Landing," in *AAS Guidance and Control Conference*, (Breckenridge, CO), 2007.

[90] L. T. Biegler, "Recent Advances in Chemical Process Optimization," *Chemie-Ingenieur-Technik*, vol. 86, no. 7, pp. 943–952, 2014.

[91] A. Richards and J. How, "Mixed-integer Programming for Control," in *IEEE American Control Conference*, (Portland, OR), 2005.

[92] R. W. Cottle, J.-S. Pang, and R. E. Stone, *Linear Complementarity Problem*. San Diego, CA: Academic Press Limited, 1992.

[93] M. Szmuk, D. Malyuta, T. P. Reynolds, M. S. Mceowen, and B. Açıkmeşe, "Real-Time Quad-Rotor Path Planning Using Convex Optimization and Compound State-Triggered Constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Macau, China), 2019.

[94] A. Boyali and S. Thompson, "Autonomous Parking by Successive Convexification and Compound State Triggers," in *IEEE International Conference on Intelligent Transportation Systems*, (Rhodes, Greece), 2020.

[95] D. Malyuta, T. P. Reynolds, M. Szmuk, B. Açıkmeşe, and M. Mesbahi, "Fast Trajectory Optimization via Successive Convexification for Spacecraft Rendezvous with Integer Constraints," in *AIAA SciTech Forum*, (Orlando, FL), 2020.

[96] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. Philadelphia, PA: Academic Press, Inc., 1981.

[97] G. Blekherman, P. A. Parrilo, and R. R. Thomas, eds., *Semidefinite Optimization and Convex Algebraic Geometry*. Philadelphia, PA: SIAM, 2012.

[98] J. E. Falk and R. M. Soland, "An Algorithm for Separable Nonconvex Programming Problems," *Management Science*, vol. 15, no. 9, pp. 550–569, 1969.

[99] R. M. Soland, "An Algorithm for Separable Nonconvex Programming Problems II: Nonconvex Constraints," *Management Science*, vol. 17, no. 11, pp. 759–773, 1971.

[100] R. Horst, "An Algorithm for Nonconvex Programming Problems," *Mathematical Programming*, vol. 10, pp. 312–321, 1976.

[101] R. Horst, "On the Convexification of Nonlinear Programming Problems: An Applications-Oriented Survey," *European Journal of Operational Research*, vol. 15, no. 3, pp. 382–392, 1984.

[102] G. P. McCormick, "Computability of Global Solutions to Factorable Nonconvex Programs: Part I - Convex Underestimating Problems," *Mathematical Programming*, vol. 10, pp. 147–175, 1976.

[103] A. Mitsos, B. Chachuat, and P. I. Barton, "McCormick-Based Relaxations of Algorithms," *SIAM Journal on Optimization*, vol. 20, no. 2, pp. 573–601, 2009.

[104] A. Tsoukalas and A. Mitsos, "Multivariate McCormick relaxations," *Journal of Global Optimization*, vol. 59, pp. 633–662, 2014.

[105] A. B. Singer and P. I. Barton, "Global Solution of Optimization Problems with Parameter-Embedded Linear Dynamic Systems," *Journal of Optimization Theory and Applications*, vol. 121, pp. 613–646, 2004.

[106] A. B. Singer and P. I. Barton, "Bounding the Solutions of Parameter Dependent Nonlinear Ordinary Differential Equations," *SIAM Journal on Scientific Computing*, vol. 27, no. 6, pp. 2167–2182, 2006.

[107] F. Palacios-Gomez, L. Lasdon, and M. Engquist, "Nonlinear Optimization by Successive Linear Programming," *Management Science*, vol. 28, no. 10, pp. 1106–1120, 1982.

[108] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz, "An Algorithm for Nonlinear Optimization Using Linear Programming and Equality Constrained Subproblems," *Mathematical Programing, Series B*, vol. 100, pp. 27–48, 2003.

[109] S. P. Han, "A Globally Convergent Method for Nonlinear Programming," *Journal of Optimization Theory and Applications*, vol. 22, no. 3, pp. 297–309, 1977.

[110] S. P. Han and O. L. Mangasarian, "Exact Penalty Functions in Nonlinear Programming," *Mathematical Programming*, vol. 17, pp. 251–269, 1979.

[111] M. J. D. Powell, "A Fast Algorithm for Nonlinearly Constrained Optimization Calculations," in *Numerical Analysis* (G. A. Watson, ed.), ch. Lecture Notes in Mathematics, pp. 144–157, Springer, Berlin, Heidelberg, 1978.

[112] M. J. D. Powell, "Algorithms for Nonlinear Constraints That Use Lagrangian Functions," *Mathematical Programming*, vol. 14, pp. 224–248, 1978.

[113] M. J. D. Powell and Y. Yuan, "A Recursive Quadratic Programming Algorithm That Uses Differentiable Exact Penalty Functions," *Mathematical Programming*, vol. 35, pp. 265–278, 1986.

[114] P. T. Boggs, J. W. Tolle, and P. Wang, "On the Local Convergence of Quasi-Newton Methods for Constrained Optimization," *SIAM Journal of Control and Optimization*, vol. 20, no. 2, pp. 161–171, 1982.

[115] P. T. Boggs and J. W. Tolle, "A Family of Descent Functions for Constrained Optimization," *SIAM Journal on Numerical Analysis*, vol. 21, no. 6, pp. 1146–1161, 1984.

[116] P. T. Boggs and W. J. Tolle, "A Strategy for Global Convergence in a Sequential Quadratic Programming Algorithm," *SIAM Journal of Numerical Analysis*, vol. 26, no. 3, pp. 600–623, 1989.

[117] M. Fukushima, "A Successive Quadratic Programming Algorithm with Global and Superlinear Convergence Properties," *Mathematical Programming*, vol. 35, pp. 253–264, 1986.

[118] P. T. Boggs and W. J. Tolle, "Sequential Quadratic Programming," *Acta Numerica*, vol. 4, pp. 1–52, 1995.

[119] J. T. Betts and W. P. Huffman, "Path-Constrained Trajectory Optimization Using Sparse Sequential Quadratic Programming," *Journal of Guidance, Control, and Dynamics*, vol. 16, no. 1, pp. 59–68, 1993.

[120] C. T. Lawrence and A. L. Tits, "A Computationally Efficient Feasible Sequential Quadratic Programming Algorithm," *SIAM Journal on Optimization*, vol. 11, no. 4, pp. 1092–1118, 2001.

[121] Y.-Z. Luo, G.-J. Tang, Z.-G. Wang, and H.-Y. Li, "Optimization of Perturbed and Constrained Fuel-Optimal Impulsive Rendezvous Using a Hybrid Approach," *Engineering Optimization*, vol. 38, no. 8, pp. 959–973, 2006.

[122] P. E. Gill and E. Wong, "Sequential Quadratic Programming Methods," in *Mixed Integer Nonlinear Programming* (J. Lee and S. Leyffer, eds.), pp. 147–224, New York, NY: Springer, 2012.

[123] R. Fletcher, *Practical Methods of Optimization*. Chichester, United Kingdom: John Wiley & Sons, 2013.

[124] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.

[125] T. P. Reynolds, D. Malyuta, M. Mesbahi, B. Açıkmeşe, and J. M. Carson III, "A Real-Time Algorithm for Non-Convex Powered Descent Guidance," in *AIAA SciTech Forum*, (Orlando, FL), 2020.

[126] Y. Mao, M. Szmuk, and B. Açıkmeşe, "Successive Convexification of Non-Convex Optimal Control Problems and its Convergence Properties," in *IEEE Conference on Decision and Control*, pp. 3636–3641, 2016.

[127] Y. Mao, D. Dueri, M. Szmuk, and B. Açıkmeşe, "Successive Convexification of Non-Convex Optimal Control Problems with State Constraints," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4063–4069, 2017.

[128] Y. Mao, M. Szmuk, and B. Açıkmeşe, "Successive Convexification: A Superlinearly Convergent Algorithm for Non-convex Optimal Control Problems," *arXiv e-prints*, 2018. arXiv ID:1804.06539.

[129] P. Simplício, A. Marcos, and S. Bennani, "Guidance of Reusable Launchers: Improving Descent and Landing Performance," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 10, pp. 2206–2219, 2019.

[130] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion Planning with Sequential Convex Optimization and Convex Collision Checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[131] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed Sequential Trajectory Optimization via Sequential Convex Programming," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[132] R. Bonalli, A. Bylard, A. Cauligi, T. Lew, and M. Pavone, "Trajectory Optimization on Manifolds: A Theoretically-Guaranteed Embedded Sequential Convex Programming Approach," in *Robotics: Science and Systems XV*, (Freiburg im Breisgau, Germany), 2019.

[133] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A Fast Solver for Constrained Trajectory Optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Macau, China), pp. 7674–7679, 2019.

[134] X. Liu, Z. Shen, and P. Lu, "Solving the Maximum-Crossrange Problem via Successive Second-Order Cone Programming with a Line Search," *Aerospace Science and Technology*, vol. 47, pp. 10–20, 2015.

[135] X. Liu, Z. Shen, and P. Lu, "Entry Trajectory Optimization by Second-Order Cone Programming," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 2, pp. 227–241, 2016.

[136] M. Sagliano, "Pseudospectral Convex Optimization for Powered Descent and Landing," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 2, pp. 320–334, 2018.

[137] Z. Wang and M. J. Grant, "Constrained Trajectory Optimization for Planetary Entry via Sequential Convex Programming," in *AIAA Atmospheric Flight Mechanics Conference*, (San Diego, CA), 2016.

[138] Z. Wang and M. J. Grant, "Improved Sequential Convex Programming Algorithms for Entry Trajectory Optimization," in *AIAA Scitech Forum*, (San Diego, CA), 2019.

[139] D. Malyuta, T. P. Reynolds, M. Szmuk, M. Mesbahi, B. Açıkmeşe, and J. M. Carson III, "Discretization Performance and Accuracy Analysis for the Powered Descent Guidance Problem," in *AIAA SciTech Forum*, (San Diego, CA), 2019.

[140] L. N. Trefethen, *Spectral Methods in MATLAB*. Phildelphia, PA: Society for Industrial and Applied Mathematics, 2000.

[141] J. Vlassenbroeck, "A Chebyshev Polynomial Method for Optimal Control with State Constraints," *Automatica*, vol. 24, no. 4, pp. 499–506, 1988.

[142] J. Vlassenbroeck and R. Van Dooren, "A Chebyshev Technique for Solving Nonlinear Optimal Control Problems," *IEEE Transactions on Automatic Control*, vol. 33, no. 4, pp. 333–340, 1988.

[143] F. Fahroo and I. M. Ross, "Direct Trajectory Optimization by a Chebyshev Pseudospectral Method," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 160–166, 2002.

[144] P. J. Antsaklis and A. N. Michel, *A Linear Systems Primer*. Boston, MA: Birkhauser, 2007.

[145] B. R. Marks and G. P. Wright, "A General Inner Approximation Algorithm for Nonconvex Mathematical Programs," *Operations Research*, vol. 26, no. 4, pp. 681–683, 1978.

[146] J. Virgili-Llop and M. Romano, "A Recursively Feasible and Convergent Sequential Convex Programming Procedure to Solve Non-Convex Problems with Linear Equality Constraints," *arXiv e-prints*, 2018. arXiv ID: 1810.10439v1.

[147] T. P. Reynolds and M. Mesbahi, "The Crawling Phenomenon in Sequential Convex Programming," in *IEEE American Control Conference*, (Denver, CO), pp. 3613–3618, 2020.

[148] D. Dueri, B. Açıkmeşe, M. Baldwin, and R. S. Erwin, "Finite-Horizon Controllability and Reachability for Deterministic and Stochastic Linear Control Systems with Convex Constraints," in *IEEE American Control Conference*, (Portland, OR), 2014.

[149] U. Eren, D. Dueri, and B. Açıkmeşe, "Constrained Reachability and Controllability Sets for Planetary Precision Landing via Convex Optimization," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 11, pp. 2067–2083, 2015.

[150] E. A. Robertson, "Synopsis of Precision Landing and Hazard Avoidance (PL&HA) Capabilities for Space Exploration," in *AIAA Guidance, Navigation, and Control Conference*, (Grapevine, TX), 2017.

[151] Europa Study Team, "Europa Study 2012 Report: Europa Lander Mission," Tech. Rep. NMO711062 Outer Planets Flagship Mission, Jet Propulsion Laboratory, 2012.

[152] NASA Science, "Moon's South Pole in NASA's Landing Sites." online, 2019. Accessed: 11/26/2019.

[153] M. Robinson, "Lunar Reconnaissance Orbiter Camera (LROC)." online, 2018. Accessed: 11/26/2019.

[154] A. Forsgren, P. E. Gill, and M. H. Wright, "Interior Methods for Nonlinear Optimization," *SIAM Review*, vol. 44, no. 4, pp. 525–597, 2002.

[155] G. J. Holzmann, *The Power of Ten – Rules for Developing Safety Critical Code*, pp. 188–201. John Wiley & Sons, Ltd, 2018.

[156] S. Hughes, L. Jun, and W. Shoan, "C++ Coding Standards and Style Guide," Tech. Rep. 20080039927, NASA, 2005.

[157] J. Mattingley and S. Boyd, "CVXGEN: A Code Generator for Embedded Convex Optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2011.

[158] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "FORCES NLP: An Efficient Implementation of Interior-Point Methods for Multistage Nonlinear Nonconvex Programs," *International Journal of Control*, vol. 93, no. 1, pp. 13–29, 2017.

[159] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control," in *IEEE Conference on Decision and Control*, (Maui, HI), 2012.

[160] A. Domahidi, *Methods and Tools for Embedded Optimization and Control*. Ph.D. Dissertation, ETH Zurich, 2013.

[161] D. Dueri, *Real-time Optimization in Aerospace Systems*. Ph.D. Dissertation, University of Washington, 2018.

[162] B. Açıkmeşe, M. Aung, J. Casoliva, S. Mohan, A. Johnson, D. Scharf, D. Masten, J. Scotkin, A. A. Wolf, and M. Regehr, "Flight Testing of Trajectories Computed by G-FOLD: Fuel Optimal Large Divert Guidance Algorithm for Planetary Landing," in *AAS/AIAA Space Flight Mechanics Meeting*, (Kauai, HI), 2013.

[163] M. Szmuk, C. A. Pascucci, D. Dueri, and B. Açıkmeşe, "Convexification and Real-Time On-board Optimization for Agile Quad-Rotor Maneuvering and Obstacle Avoidance," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Vancouver, Canada), 2017.

[164] M. Szmuk, C. A. Pascucci, and B. Açıkmeşe, "Real-Time Quad-Rotor Path Planning for Mobile Obstacle Avoidance Using Convex Optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Madrid, Spain), 2018.

[165] S. Mariethoz, A. Domahidi, and M. Morari, "High-Bandwidth Explicit Model Predictive Control of Electrical Drives," *IEEE Transactions on Industry Applications*, vol. 48, no. 6, pp. 1980–1992, 2012.

[166] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded Predictive Control on an FPGA using the Fast Gradient Method," in *European Control Conference*, (Zurich, Switzerland), 2013.

[167] A. Liniger, A. Domahidi, and M. Morari, "Optimization-Based Autonomous Racing of 1:43 Scale RC Cars," *arXiv e-prints*, 2017. arXiv ID:1711.07300.

[168] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-Based Model Predictive Control for Autonomous Racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[169] I. M. Ross, Q. Gong, M. Karpenko, and R. J. Proulx, "Scaling and Balancing for High-Performance Computation of Optimal Controls," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 10, pp. 2086–2097, 2018.

[170] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. Philadelphia, PA: SIAM, 1997.

[171] M. A. Patterson and A. V. Rao, "GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming," *ACM Transactions on Mathematical Software*, vol. 41, no. 1, pp. 1–37, 2014.

[172] J. V. Breakwell, J. L. Speyer, and A. E. Bryson, "Optimization and Control of Nonlinear Systems Using the Second Variation," *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, vol. 1, no. 2, pp. 193–223, 1963.

[173] H. J. Kelley, "An Optimal Guidance Approximation Theory," *IEEE Transactions on Automatic Control*, vol. 9, no. 4, pp. 375–380, 1964.

[174] H. Seywald and E. M. Cliff, "Neighboring Optimal Control Based Feedback Law for the Advanced Launch System," *Journal of Guidance, Control, and Dynamics*, vol. 17, no. 6, pp. 1154–1162, 1994.

[175] H. Yan, F. Fahroo, and I. M. Ross, "Real-Time Computation of Neighboring Optimal Control Laws," in *AIAA Guidance, Navigation, and Control Conference*, (Monterey, CA), 2002.

[176] M. R. Jardin and A. E. Bryson, "Neighboring Optimal Aircraft Guidance in Winds," *Journal of Guidance, Control, and Dynamics*, vol. 24, no. 4, pp. 710–715, 2001.

[177] M. S. Miah and W. Gueaieb, "RFID-Based Mobile Robot Trajectory Tracking and Point Stabilization Through On-Line Neighboring Optimal Control," *Journal of Intelligent & Robotic Systems*, vol. 78, no. 3-4, pp. 377–399, 2015.

[178] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential Composition of Dynamically Dexterous Robot Behaviors," *International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.

[179] T. Lozano-Perez, M. T. Mason, and R. H. Taylor, "Automatic Synthesis of Fine-Motion Strategies for Robots," *The International Journal of Robotics Research*, vol. 3, no. 1, pp. 3–24, 1984.

[180] A. A. Julius and G. J. Pappas, "Trajectory Based Verification Using Local Finite-Time Invariance," in *Hybrid Systems: Computation and Control*, (Berlin, Germany), pp. 223–236, 2009.

[181] A. Majumdar and R. Tedrake, "Funnel Libraries for Real-Time Robust Feedback Motion Planning," *International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017. arXiv ID:1601.04037v3.

[182] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-Trees: Feedback Motion Planning via Sums-of-Squares Verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[183] M. M. Tobenkin, I. R. Manchester, and R. Tedrake, "Invariant Funnels Around Trajectories using Sum-of-Squares Programming," in *IFAC-PapersOnline*, (Milan, Italy), pp. 9218–9223, 2011.

[184] P. A. Parrilo, "Semidefinite Programming Relaxations for Semialgebraic Problems," *Mathematical Programming, Series B*, vol. 96, pp. 293–320, 2003.

[185] F. Blanchini, "Set Invariance in Control," *Automatica*, vol. 35, no. 11, pp. 1747–1767, 1999.

[186] D. L. Kleinman, "On an Iterative Technique for Riccati Equation Computations," *IEEE Transactions on Automatic Control*, vol. 13, no. 1, pp. 114–115, 1968.

[187] J. C. Doyle, "Synthesis of Robust Controllers and Filters," in *IEEE Conference on Decision and Control*, (Las Vegas, NV), pp. 109–114, 1983.

[188] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Studies in Applied Mathematics, Philadelphia, PA: SIAM, 1994.

[189] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances," in *IEEE Conference on Decision and Control*, (Melbourne, Australia), pp. 2242–2253, 2017.

[190] J. K. Scott and P. I. Barton, "Bounds on the Reachable Sets of Nonlinear Control Systems," *Automatica*, vol. 49, no. 1, pp. 93 – 100, 2013.

[191] M. Althoff, O. Stursberg, and M. Buss, "Reachability Analysis of Nonlinear Systems with Uncertain Parameters Using Conservative Linearization," in *IEEE Conference on Decision and Control*, (Cancun, Mexico), pp. 4042–4048, 2008.

[192] A. Rantzer, "On the Kalman-Yakubovich-Popov Lemma," *Systems & Control Letters*, vol. 28, no. 1, pp. 7–10, 1996.

[193] J. C. Doyle, "Analysis of Control Systems with Structured Uncertainties," *IEE Proc., Part D, Control Theory and Applications*, vol. 129, no. 6, pp. 242–250, 1982.

[194] P. P. Khargonekar, I. R. Peterson, and K. Zhou, "Robust Stabilization of Uncertain Linear Systems: Quadratic Stabilizability and $\mathcal{H}_\infty$ Control Theory," *IEEE Transactions on Automatic Control*, vol. 35, no. 3, pp. 356–361, 1990.

[195] M. A. Rotea and P. P. Khargonekar, "Stabilization of Uncertain Systems with Norm Bounded Uncertainty – A Control Lyapunov Function Approach," *SIAM Journal of Control and Optimization*, vol. 27, no. 6, pp. 1462–1476, 1989.

[196] M. V. Kothare, V. Balakrishnan, and M. Morari, "Robust Constrained Model Predictive Control Using Linear Matrix Inequalities," *Automatica*, vol. 32, no. 10, pp. 1361–1379, 1996.

[197] J. Löfberg, *Minimax Approaches to Robust Model Predictive Control*. Ph.D. Dissertation, Linköping University, 2003.

[198] F. A. Cuzzola, J. C. Geromel, and M. Morari, "An Improved Approach for Constrained Robust Model Predictive Control," *Automatica*, vol. 38, no. 7, pp. 1183–1189, 2002.

[199] J. Oravec and M. Bakošová, "Alternative LMI-based Robust MPC Design Approaches," *IFAC-PapersOnLine*, vol. 28, no. 14, pp. 180–185, 2015.

[200] B. Açıkmeşe, J. M. Carson III, and D. S. Bayard, "A Robust Model Predictive Control Algorithm for Incrementally Conic Uncertain/Nonlinear Systems," *International Journal of Robust and Nonlinear Control*, vol. 21, no. 5, pp. 563–590, 2011.

[201] S. V. Raković, "Invention of Prediction Structures and Categorization of Robust MPC Syntheses," *IFAC-PapersOnline*, vol. 4, no. 17, pp. 245–273, 2012.

[202] D. Q. Mayne, "Model Predictive Control: Recent Developments and Future Promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[203] T. P. Reynolds, D. Malyuta, M. Mesbahi, and B. Açıkmeşe, "Temporally-Interpolated Funnel Synthesis for Nonlinear Systems," in *2nd RSS Workshop on Robust Autonomy*, (Virtual), 2020.

[204] B. Açıkmeşe and M. Corless, "Stability Analysis with Quadratic Lyapunov Functions: Some Necessary and Sufficient Multiplier Conditions," *Systems & Control Letters*, vol. 57, no. 1, pp. 78–94, 2008.

[205] L. P. D'Alto, *Incremental Quadratic Stability.* M.S. Thesis, Purdue University, 2004.

[206] A. M. Lyapunov, *Stability of Motion.* New York, NY: Academic Press, 1966.

[207] A. Hassibi and S. Boyd, "Quadratic Stabilization and Control of Piecewise-Linear Systems," in *IEEE American Control Conference*, (Philadelphia, PA), pp. 3659–3664, 1998.

[208] M. C. Johnson, "A Parameterized Approach to the Design of Lunar Lander Attitude Controllers," in *AIAA Guidance, Navigation, and Control Conference*, (Keystone, CO), 2006.

[209] D. Liberzon and A. S. Morse, "Basic Problems in Stability and Design of Switched Systems," *Control Systems Magazine*, vol. 19, no. 5, pp. 59–70, 1999.

[210] T. Hu, "Nonlinear Control Design for Linear Differential Inclusions via Convex Hull of Quadratics," *Automatica*, vol. 43, no. 4, pp. 685–692, 2007.

[211] T. Hu and Z. Lin, "Composite Quadratic Lyapunov Functions for Constrained Control Systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 3, pp. 440–450, 2003.

[212] M. V. Khlebnikov, B. T. Polyak, and V. M. Kuntsevich, "Optimization of Linear Systems Subject to Bounded Exogenous Disturbances: The Invariant Ellipsoid Technique," *Automation and Remote Control*, vol. 72, no. 11, pp. 2227–2275, 2011.

[213] B. T. Polyak, A. V. Nazin, M. V. Topunov, and S. A. Nazin, "Rejection of Bounded Disturbances via Invariant Ellipsoids Technique," in *IEEE Conference on Decision and Control*, (San Diego, CA), pp. 1429–1434, 2006.

[214] V. A. Yakubovich, "The Solution of Some Matrix Inequalities Encountered in Automatic Control Theory," *Dokl. Akad. Nauk SSSR*, vol. 143, no. 6, pp. 1304–1307, 1962.

[215] V. A. Yakubovich, "The Method of Matrix Inequalities in the Stability Theory of Nonlinear Control Systems I, II, III," *Automation and Remote Control*, vol. 25-26, no. 4, pp. 905–917, 577–592, 753–763, 1967.

[216] A. Megretski and A. Rantzer, "System Analysis via Integral Quadratic Constraints," *IEEE Transactions on Automatic Control*, vol. 42, no. 6, pp. 819–830, 1997.

[217] G. R. Wood and B. P. Zhang, "Estimation of the Lipschitz Constant of a Function," *Journal of Global Optimization*, vol. 8, no. 1, pp. 91–103, 1996.

[218] R. H. Byrd, M. E. Hribar, and J. Nocedal, "An Interior Point Algorithm for Large-Scale Nonlinear Programming," *SIAM Journal on Optimization*, vol. 9, no. 4, pp. 877–900, 1999.

[219] T. P. Reynolds, C. L. Kelly, C. Morgan, A. Grebovic, J. Erickson, H. Brown, W. C. Pope, J. Casamayor, K. Kearsley, G. Caylak, K. E. Fisher, C. Wutzke, K. Ashton, J. Rosenthal, D. Tormey, E. Freneau, G. Giddings, H. E. Horata, A. Dighde, S. Parakh, J. Zhen, J. C. Purpura, D. B. Pratt, and A. Hunt, "SOC-i: A CubeSat Demonstration of Optimization-Based Real-Time Constrained Attitude Control," in *IEEE Aerospace Conference*, (Virtual), 2020.

[220] C. Chevalley, *The Algebraic Theory of Spinors and Clifford Algebras: Collected Works*, vol. 2. Berlin, Germany: Springer-Verlag, 1997.

[221] J. H. M. Wedderburn, "On Hypercomplex Numbers," *Proceedings of the London Mathematical Society*, vol. 2-6, no. 1, pp. 77–118, 1908.

[222] A. P. Murray, F. Pierrot, P. Dauchez, and J. M. McCarthy, "A Planar Quaternion Approach to the Kinematic Synthesis of a Parallel Manipulator," *Robotica*, vol. 15, pp. 361–365, 1997.

[223] G. H. Golub and V. Pereyra, "The Differentiation of Pseudo-Inverses and Nonlinear Least Squares Problems Whose Variables Separate," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 413–432, 1973.

Appendix A
HYPERCOMPLEX NUMBERS

One must be able to say at all times
instead of points, straight lines, and planes
tables, chairs, and beer mugs.
– David Hilbert

By the 19^{th} Century, the utility of the complex unit i, where $i^2 = -1$, was already well-established. The success of complex analysis and the two-dimensional geometry that could be elegantly described by complex numbers offered a powerful driving force for the desire to expand the ideas to beyond two dimensions. Hypercomplex numbers can be thought of as a generalization of complex numbers to higher dimensional spaces. The role of this appendix is to point out a common "generator" for the hypercomplex numbers used in this dissertation, and to comment on the potential for new hypercomplex numbers to be used in the modeling of dynamical systems for trajectory generation. Quaternions and dual quaternions have much to offer in terms of algebraic elegance, computational efficiency, and in the regularity and convexity of various dynamic and constraint functions. But there is nothing special, per se, about quaternions and dual quaternions[1] in this regard; it is entirely possible that similarly generated hypercomplex number systems may have similarly attractive properties. I believe that we have only scratched the surface of how these parameters can be used for modeling dynamical systems in the trajectory optimization literature. Some relevant understanding may exist in the field of theoretical physics, where hypercomplex numbers are more often studied and have found more applications, but this understanding has not permeated into the field of computational guidance and control.

[1]Dual quaternions are not the same as the octonions mentioned in Hurwitz's theorem [220].

Hamilton pondered the question of how to generalize complex numbers to three dimensions, and it is worth spending a moment to reflect on his original (incorrect) ideas. Recall that for a regular complex number $x+iy \in \mathbb{C}$, the product $(x+iy)(x-iy) = x^2+y^2$ returns the length of the vector in the two-dimensional plane. In three dimensions, it would seem natural then to simply use a second complex unit, j, and express the new complex number as $x + iy + jz$. Using the same multiplication rules as for regular complex numbers then leads to

$$(x + iy + jz)(x - iy - jz) = x^2 + y^2 + z^2 - yz(ij - ji). \tag{A.1}$$

Therefore, we should define $ij = -ji$ in (A.1) so that the meaning of length is preserved. This is fine, but the issues start to appear when we consider the more general product of two distinct complex numbers:

$$(x + iy + jz)(a + ib + jc) = (ax - by - cz) + i(ay + bx) + j(az + cx) + ij(bz - cy). \tag{A.2}$$

Even with $ij = -ji$, there is no way to remove the fourth term from (A.2). This implies that the generalized algebra does not close; meaning that the product of two elements in the space does not return an element of the same space. The resolution of this issue is what Hamilton carved on the Brougham Bridge in October 16, 1843. By introducing a *third* complex unit, k, the algebra would close. The carving in the bridge reads:

$$i^2 = j^2 = k^2 = ijk = -1, \tag{A.3a}$$

$$ij = -ji, \quad jk = -kj, \quad ki = -ik. \tag{A.3b}$$

So the idea of a *quaternion* was born using the four basis elements $\{1, i, j, k\}$ and the multiplicative rules that result from (A.3).

Why can we use two-parameter complex numbers to represent two dimension, but need four-parameter quaternions to represent three dimensions? What will happen if we want to study things in four dimensions? Or five? What is the common foundation that complex

numbers and quaternions share? The useful tool of Clifford algebras offers an interesting viewpoint on these questions, and can be interpreted as a sort of "generator" for the complex numbers, the quaternions, and their higher-dimensional analogues.

A.1 Clifford Algebras

Consider a general n–dimensional vector space $\mathcal{V}$ over the field of real numbers $\mathbb{R}$ endowed with a (symmetric) scalar product, $\cdot : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$. A Clifford algebra, $C(\mathcal{V})$, is constructed by using a product operation, $\otimes : \mathcal{V} \times \mathcal{V} \to C(\mathcal{V})$ that satisfies the following properties: If $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathcal{V}$ and $\alpha, \beta \in \mathbb{R}$ then

$$\boldsymbol{x} \otimes (\boldsymbol{y} + \boldsymbol{z}) = \boldsymbol{x} \otimes \boldsymbol{y} + \boldsymbol{x} \otimes \boldsymbol{z}, \tag{A.4a}$$

$$(\alpha \boldsymbol{x}) \otimes (\beta \boldsymbol{y}) = (\alpha\beta) \boldsymbol{x} \otimes \boldsymbol{y}, \tag{A.4b}$$

$$(\boldsymbol{x} \otimes \boldsymbol{y}) \otimes \boldsymbol{z} = \boldsymbol{x} \otimes (\boldsymbol{y} \otimes \boldsymbol{z}), \tag{A.4c}$$

$$\boldsymbol{x} \otimes \boldsymbol{y} + \boldsymbol{y} \otimes \boldsymbol{x} = -2(\boldsymbol{x} \cdot \boldsymbol{y})\, 1, \tag{A.4d}$$

where 1 is the multiplicative identity of $C(\mathcal{V})$. Properties (A.4a) and (A.4b) imply that the product is a linear map, while (A.4c) makes it associative. The term *algebra* is indeed meant as a shorthand for the term *linear associative algebra* [221]. The less familiar (A.4d) in turn provides an anti-commutative property that is reminiscent of the properties in (A.3b).

Notice that (A.4d) is a rather counter intuitive expression, in that the left-hand side is an element of the Clifford algebra, and the right-hand side is a scalar from the field of real numbers times the multiplicative identity. Using conventional algebra, it would appear that we are equating a vector to a scalar. But this is not conventional algebra, and in fact both sides of the equation are elements of the Clifford Algebra because $1 \in C(\mathcal{V})$. The scalar product on the right hand side of (A.4d) shall provide crucial flexibility with which to properly expand from complex numbers and quaternions to dual quaternions and beyond – and its main role is to define the pairwise product of two (eventually orthonormal) elements from a basis of $\mathcal{V}$. The equation (A.4d) says that, under the chosen product operation,

the vector product of two basis vectors anti-commutes if and only if the basis vectors are orthogonal.

Elements of the Clifford algebra, denoted by $C(\mathcal{V})$ for the vector space $\mathcal{V}$, are generated using all possible products between the basis vectors of $\mathcal{V}$ plus the multiplicative identity, 1. For a vector space of dimension n, the corresponding Clifford algebra has dimension 2^n, with $\binom{n}{k}$ basis elements of "rank" k for each $k = 0, \ldots, n$. We say a basis element (of the Clifford algebra) has rank k if it is the product of k basis vectors from $\mathcal{V}$. The product of zero basis vectors is defined to be the multiplicative identity. This implies that the vector space $\mathcal{V}$ appears as a linear subspace in the Clifford algebra. The notion of "closedness" that gave Hamilton fits is related to the parity of the rank of these basis elements. *Only the subalgebras generated by the basis vectors of even rank form closed algebras.* The even Clifford subalgebra is denoted by $C^+(V)$ and has dimension 2^{n-1}.

A.1.1 Complex Numbers

To demonstrate how Clifford algebras can be used to generate hypercomplex numbers, consider the vector space $\mathcal{V} = \mathbb{R}^2$ under the usual Euclidean scalar product. The Clifford algebra is denoted by $C(\mathbb{R}^2)$ and has dimension $2^2 = 4$. The basis elements for $C(\mathbb{R}^2)$ are

$$\{1,\, \boldsymbol{e}_1,\, \boldsymbol{e}_2,\, \boldsymbol{e}_1 \otimes \boldsymbol{e}_2\} \tag{A.5}$$

where $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are the standard basis vectors for $\mathbb{R}^2$. The elements of the basis (A.5) that have even rank are $\{1,\, \boldsymbol{e}_1 \otimes \boldsymbol{e}_2\}$. Hence the even Clifford subalgebra, $C^+(\mathbb{R}^2)$, is the subspace spanned by these two vectors, and has dimension $2^1 = 2$. Observe that (A.4d) implies that

$$\boldsymbol{e}_i \otimes \boldsymbol{e}_j = \begin{cases} -\boldsymbol{e}_j \otimes \boldsymbol{e}_i, & i \neq j \\ -1, & i = j \end{cases}. \tag{A.6}$$

The associative property of the product $\otimes$ in (A.4c) and the pairwise relationship (A.6) leads to

$$\begin{aligned}(\boldsymbol{e}_1 \otimes \boldsymbol{e}_2)^2 &= (\boldsymbol{e}_1 \otimes \boldsymbol{e}_2) \otimes (\boldsymbol{e}_1 \otimes \boldsymbol{e}_2) \\ &= \boldsymbol{e}_1 \otimes (\boldsymbol{e}_2 \otimes \boldsymbol{e}_1) \otimes \boldsymbol{e}_2 \\ &= -\boldsymbol{e}_1 \otimes (\boldsymbol{e}_1 \otimes \boldsymbol{e}_2) \otimes \boldsymbol{e}_2 \\ &= -(\boldsymbol{e}_1 \otimes \boldsymbol{e}_1) \otimes (\boldsymbol{e}_2 \otimes \boldsymbol{e}_2) \\ &= -1.\end{aligned}$$

Therefore the basis element $\boldsymbol{e}_1 \otimes \boldsymbol{e}_2$ becomes -1 when "squared". Making the convenient association $i \equiv \boldsymbol{e}_1 \otimes \boldsymbol{e}_2$, an element $\boldsymbol{x} \in C^+(\mathbb{R}^2)$ can be expressed as

$$\boldsymbol{x} = x + iy, \tag{A.7}$$

for $x, y \in \mathbb{R}$, which is a complex number. Hence the even Clifford subalgebra for the vector space $\mathbb{R}^2$ with the usual Euclidean inner product is isomorphic to the complex numbers, and this is denoted by $C^+(\mathbb{R}^2) \cong \mathbb{C}$.

A.2 Quaternions

The utility of quaternions in mechanics is their association with three-dimensional rotations of a rigid body. It was shown in §2.3.1 that this association is a result of the fact that the multiplication of two specially defined quaternions provides the same result as successive rotations using other parameterizations of rotation. The core piece is the multiplication rules that quaternions obey, and we can use Clifford algebras to derive them. The choice of vector space becomes $\mathcal{V} = \mathbb{R}^3$ and the usual Euclidean scalar product in $\mathbb{R}^3$ is used, denoted by $\cdot : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$. The Clifford algebra is $C(\mathbb{R}^3)$ and has dimension $2^3 = 8$. The basis

elements are

$$\{1,\, \boldsymbol{e}_1,\, \boldsymbol{e}_2,\, \boldsymbol{e}_3,\, \boldsymbol{e}_1 \otimes \boldsymbol{e}_2,\, \boldsymbol{e}_2 \otimes \boldsymbol{e}_3,\, \boldsymbol{e}_3 \otimes \boldsymbol{e}_1,\, \boldsymbol{e}_1 \otimes \boldsymbol{e}_2 \otimes \boldsymbol{e}_3\}.$$

The basis elements with even rank are $\{1,\, \boldsymbol{e}_1 \otimes \boldsymbol{e}_2,\, \boldsymbol{e}_2 \otimes \boldsymbol{e}_3,\, \boldsymbol{e}_3 \otimes \boldsymbol{e}_1\}$. The even Clifford subalgebra, $C^+(\mathbb{R}^3)$ is therefore spanned by these *four* vectors and has dimension $2^2 = 4$. Each of the pairwise products of basis elements of $C^+(\mathbb{R}^3)$ satisfies (A.6), and hence we make the associations

$$i \equiv \boldsymbol{e}_2 \otimes \boldsymbol{e}_3, \quad j \equiv \boldsymbol{e}_3 \otimes \boldsymbol{e}_1, \quad k \equiv \boldsymbol{e}_1 \otimes \boldsymbol{e}_2 \tag{A.8}$$

from which it can be verified that each relationship in (A.3) holds. A general element $\boldsymbol{x} \in C^+(\mathbb{R}^3)$ can therefore be written as

$$\boldsymbol{x} = x_4 + ix_1 + jx_2 + kx_3 \tag{A.9}$$

where $x_1, x_2, x_3, x_4 \in \mathbb{R}$. This as a quaternion as described by Hamilton [30]. Hence the even Clifford subalgebra for the vector space $\mathbb{R}^3$ with the usual Euclidean inner product is isomorphic to the set of quaternions, and we write $C^+(\mathbb{R}^3) \cong \mathbb{Q}$.

Consider the quaternion $\boldsymbol{q} \in \mathbb{Q}$, where using our notation we have $\boldsymbol{q} = q_4 + iq_1 + jq_2 + kq_3$. The *scalar part* is q_4 and the *vector part* is $\boldsymbol{q}_v = iq_1 + jq_2 + kq_3$. Given a second quaternion $\boldsymbol{p} \in \mathbb{Q}$, the product $\boldsymbol{q} \otimes \boldsymbol{p}$ is expressed using the rules (A.3) as

$$\begin{aligned} \boldsymbol{q} \otimes \boldsymbol{p} = {} & q_4p_4 - q_1p_1 - q_2p_2 - q_3p_4 \\ & + (q_4p_1 + q_1p_4 + q_2p_3 - q_3p_2)\, i \\ & + (q_4p_2 + q_2p_4 + q_3p_1 - q_1p_3)\, j \\ & + (q_4p_3 + q_3p_4 + q_1p_2 - q_2p_1)\, k. \end{aligned} \tag{A.10}$$

This equation is the basis for (2.21) in §2.3.1.

The conjugation operation that is used often in complex analysis generalizes to quater-

nions in the expected way according to

$$\boldsymbol{q}^* = q_4 - iq_1 - jq_2 - kq_3 \tag{A.11}$$

The scalar product in $\mathbb{Q}$ is $\odot : \mathbb{Q} \times \mathbb{Q} \to \mathbb{R}$, defined by

$$\boldsymbol{q} \odot \boldsymbol{p} = \boldsymbol{q} \cdot \boldsymbol{p}^* = q_4 p_4 + q_1 p_1 + q_2 p_2 + q_3 p_3. \tag{A.12}$$

The conjugation and scalar product may be used to define a *quaternion division* by using the so-called multiplicative inverse property that $\boldsymbol{q}_{\text{id}} = \boldsymbol{q} \otimes \boldsymbol{q}^{-1} = 1 + 0i + 0j + 0k$, where the quaternion inverse is defined to be $\boldsymbol{q}^{-1} = \boldsymbol{q}^* / \big(\boldsymbol{q} \odot \boldsymbol{q}\big)$. Evidently the inverse and conjugate coincide for quaternions that satisfy $\boldsymbol{q} \odot \boldsymbol{q} = 1$, which are exactly the *unit quaternions* that were shown to represent rotations in three dimensions in §2.3.1. This division is the basis for the (multiplicative) quaternion error. Because successive rotations are represented by quaternion multiplication, we can think of "adding" rotations as being equivalent to multiplying quaternions, whereas "subtracting" rotations to define an error is equivalent to dividing quaternions.

A.3 Dual Quaternions

The more general dual quaternions extend the utility of quaternions to include both three-dimensional rotation and three-dimensional translation of a rigid body. Using a similar but subtly different construction, they too can be derived using Clifford algebras. The vector space is $\mathcal{V} = \mathbb{R}^4$, but we define the scalar product $\cdot : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}$ such that for all $\boldsymbol{x}$, $\boldsymbol{y} \in \mathbb{R}^4$

$$\boldsymbol{x} \cdot \boldsymbol{y} = x_1 y_1 + x_2 y_2 + x_3 y_3. \tag{A.13}$$

This scalar product omits the fourth coordinate. It can be thought of as the usual Euclidean inner product restricted to the hyperplane defined by $\{\boldsymbol{x} \in \mathbb{R}^4 \mid x_4 = 1\}$. According to (A.4d), this implies that $\boldsymbol{e}_4 \otimes \boldsymbol{e}_4 = 0$.

The Clifford algebra is $C(\mathbb{R}^4)$ and has dimension $2^4 = 16$. The even Clifford sub-algebra is $C^+(\mathbb{R}^4)$, has dimension $2^3 = 8$, and is represented by the span of the basis elements of even rank:

$$\{1,\, \boldsymbol{e}_1 \otimes \boldsymbol{e}_2,\, \boldsymbol{e}_2 \otimes \boldsymbol{e}_3,\, \boldsymbol{e}_3 \otimes \boldsymbol{e}_1,\, \boldsymbol{e}_4 \otimes \boldsymbol{e}_1,\, \boldsymbol{e}_4 \otimes \boldsymbol{e}_2,\, \boldsymbol{e}_4 \otimes \boldsymbol{e}_3,\, \boldsymbol{e}_1 \otimes \boldsymbol{e}_2 \otimes \boldsymbol{e}_3 \otimes \boldsymbol{e}_4\}\,. \tag{A.14}$$

The first four basis elements are exactly those that appeared for quaternions. Moreover, the scalar product (A.13) produces identical results to that which was used to derive quaternions for any vector in the span of these first four basis elements. Hence the same associations can be made as in (A.8) for the subalgebra $C^+(\mathbb{R}^4)$. Furthermore, the eighth basis vector satisfies

$$(\boldsymbol{e}_1 \otimes \boldsymbol{e}_2 \otimes \boldsymbol{e}_3 \otimes \boldsymbol{e}_4)^2 = (\boldsymbol{e}_1 \otimes \boldsymbol{e}_2 \otimes \boldsymbol{e}_3 \otimes \boldsymbol{e}_4) \otimes (\boldsymbol{e}_1 \otimes \boldsymbol{e}_2 \otimes \boldsymbol{e}_3 \otimes \boldsymbol{e}_4) = 0,$$

because $\boldsymbol{e}_4 \otimes \boldsymbol{e}_4 = 0$, and this leads to the association of the *dual unit*

$$\epsilon \equiv \boldsymbol{e}_1 \otimes \boldsymbol{e}_2 \otimes \boldsymbol{e}_3 \otimes \boldsymbol{e}_4, \tag{A.15}$$

where $\epsilon \neq 0$ but $\epsilon^2 = 0$. This (seemingly bizarre) definition should be no less acceptable to the reader than that of the complex unit $i^2 = -1$. The product of the eighth basis vector and each of those given by (A.8) can then be computed, for example, to be

$$\underbrace{(\boldsymbol{e}_1 \otimes \boldsymbol{e}_2)}_{\equiv k} \otimes \underbrace{(\boldsymbol{e}_1 \otimes \boldsymbol{e}_2 \otimes \boldsymbol{e}_3 \otimes \boldsymbol{e}_4)}_{\equiv \epsilon} = -\boldsymbol{e}_3 \otimes \boldsymbol{e}_4 = \boldsymbol{e}_4 \otimes \boldsymbol{e}_3 \equiv k\epsilon.$$

Similarly, it can be shown that the remaining two basis elements in (A.14) are $i\epsilon$ and $j\epsilon$.

A general element $\boldsymbol{x} \in C^+(\mathbb{R}^4)$ can therefore be written as

$$\boldsymbol{x} = x_4 + ix_1 + jx_2 + kx_3 + \epsilon x_4' + i\epsilon x_1' + j\epsilon x_2' + k\epsilon x_3' \tag{A.16}$$

$$= x_4 + ix_1 + jx_2 + kx_3 + \epsilon\,(x_4' + ix_1' + jx_2' + kx_3') \tag{A.17}$$

where $x_1, x_2, x_3, x_4, x'_1, x'_2, x'_3, x'_4 \in \mathbb{R}$. Hence the even Clifford subalgebra for the vector space $\mathbb{R}^4$ with the inner product defined in (A.13) is isomorphic to the set of dual quaternions, and we have $C^+(\mathbb{R}^4) \cong \widetilde{\mathbb{Q}}$.

Consider the dual quaternion $\tilde{\boldsymbol{q}} \in \widetilde{\mathbb{Q}}$, which in our notation is written

$$\tilde{\boldsymbol{q}} = \boldsymbol{q}_1 + \epsilon \boldsymbol{q}_2 = q_{1,4} + iq_{1,1} + jq_{1,2} + kq_{1,3} + \epsilon \left(q_{2,4} + iq_{2,1} + jq_{2,2} + kq_{2,3} \right).$$

We label the *real part* as the quaternion $\boldsymbol{q}_1 \in \mathbb{Q}$ and the *dual part* as the quaternion $\boldsymbol{q}_2 \in \mathbb{Q}$. Given a second dual quaternion $\tilde{\boldsymbol{p}} \in \widetilde{\mathbb{Q}}$, the product $\tilde{\boldsymbol{q}} \otimes \tilde{\boldsymbol{p}}$ is expressed using the rules (A.3), in addition to $\epsilon^2 = 0$, as

$$\begin{aligned} \tilde{\boldsymbol{q}} \otimes \tilde{\boldsymbol{p}} &= (\boldsymbol{q}_1 + \epsilon \boldsymbol{q}_2) \otimes (\boldsymbol{p}_1 + \epsilon \boldsymbol{p}_2) \\ &= \boldsymbol{q}_1 \otimes \boldsymbol{p}_1 + \epsilon \left(\boldsymbol{q}_1 \otimes \boldsymbol{p}_2 + \boldsymbol{q}_2 \otimes \boldsymbol{p}_1 \right) \end{aligned} \tag{A.18}$$

where the quaternion products in (A.18) also follow the rules defined in (A.10). This equation is the basis for (2.34) in §2.3.2.

The scalar product in $\widetilde{\mathbb{Q}}$ is defined as the pairwise quaternion scalar product between the real and dual parts of two dual quaternions [31]. Denoting this product by $\odot : \widetilde{\mathbb{Q}} \times \widetilde{\mathbb{Q}} \to \mathbb{D}$, where $\mathbb{D} \coloneqq \{x + \epsilon y \mid x, y \in \mathbb{R}\}$ is the set of *dual numbers*, permits the expression of the dual quaternion dot product as

$$\tilde{\boldsymbol{q}} \odot \tilde{\boldsymbol{p}} = \boldsymbol{q}_1 \odot \boldsymbol{p}_1 + \epsilon \left(\boldsymbol{q}_1 \odot \boldsymbol{p}_2 + \boldsymbol{q}_2 \odot \boldsymbol{p}_1 \right). \tag{A.19}$$

A unit dual quaternion is one for which the dual quaternion scalar product (A.19) evaluates to the identity element of $\mathbb{D}$, which is $1 + \epsilon 0$. Hence, a unit dual quaternion satisfies

$$\tilde{\boldsymbol{q}} \odot \tilde{\boldsymbol{q}} = \boldsymbol{q}_1 \odot \boldsymbol{q}_1 + \epsilon \left(\boldsymbol{q}_1 \odot \boldsymbol{q}_2 + \boldsymbol{q}_2 \odot \boldsymbol{q}_1 \right) = 1 + \epsilon 0, \tag{A.20}$$

from which it can be seen that $\boldsymbol{q}_1 \odot \boldsymbol{q}_1 = 1$ and $\boldsymbol{q}_1 \odot \boldsymbol{q}_2 = 0$. This results in the definition of

$\mathbb{R}^8_u$ in (2.33).

A.3.1 Connection to Matrix-Vector Algebra

We now take a brief aside to explore the connection between the product $\otimes$ and regular matrix-vector algebra that was used in the main text. Consider the general Clifford subalgebra $C^+(\mathcal{V})$ with dimension 2^{n-1}. Assume that the basis elements have been given some ordering – the specific ordering does not matter, only that one has been assumed and that it is kept consistent. Let $\boldsymbol{x}$, $\boldsymbol{y} \in C^+(\mathcal{V})$ and consider the product $\boldsymbol{x} \otimes \boldsymbol{y}$. The product can be expressed by the matrix equations

$$\boldsymbol{x} \otimes \boldsymbol{y} = [\boldsymbol{x}]_{\otimes}\boldsymbol{y} = [\boldsymbol{y}]^*_{\otimes}\boldsymbol{x} \tag{A.21}$$

where $[\boldsymbol{x}]_{\otimes}, [\boldsymbol{y}]^*_{\otimes}$ are each $2^{n-1} \times 2^{n-1}$ matrices that are defined as follows. The columns of $[\boldsymbol{x}]_{\otimes}$ are the product of $\boldsymbol{x}$ with each of the basis elements of $C^+(\mathcal{V})$ *from the right*. That is,

$$[\boldsymbol{x}]_{\otimes} = \begin{bmatrix} \boldsymbol{x} & \boldsymbol{x} \otimes \boldsymbol{e}_1 & \cdots & \boldsymbol{x} \otimes \boldsymbol{e}_n & \boldsymbol{x} \otimes (\boldsymbol{e}_1 \otimes \boldsymbol{e}_2) & \cdots & \boldsymbol{x} \otimes (\boldsymbol{e}_1 \otimes \cdots \otimes \boldsymbol{e}_n) \end{bmatrix},$$

where the ordering of the columns is consistent with the assumed ordering of the basis elements. Similarly, the matrix $[\boldsymbol{y}]^*_{\otimes}$ is obtained by the same procedure, though the basis elements are now multiplied *from the left*. Thus,

$$[\boldsymbol{y}]^*_{\otimes} = \begin{bmatrix} \boldsymbol{y} & \boldsymbol{e}_1 \otimes \boldsymbol{y} & \cdots & \boldsymbol{e}_n \otimes \boldsymbol{y} & (\boldsymbol{e}_1 \otimes \boldsymbol{e}_2) \otimes \boldsymbol{y} & \cdots & (\boldsymbol{e}_1 \otimes \cdots \otimes \boldsymbol{e}_n) \otimes \boldsymbol{y} \end{bmatrix}$$

where again the ordering of the columns is consistent with the assumed ordering of the basis elements. These representations of the vector product $\boldsymbol{x} \otimes \boldsymbol{y}$ are what give rise to the matrix-vector definitions of the quaternion product (2.21) and dual quaternion product (2.34).

A.4 Beyond Quaternions and Dual Quaternions

Interestingly, if we keep $\mathcal{V} = \mathbb{R}^3$ but change the inner product to a similar version of the that was used to "derive" dual quaternions, we can obtain the *planar quaternions* [31]. These four-dimensional parameters can then be used to model two planar position coordinates and one angular coordinate [222] – just like were used to study the planar landing problems.

If we change the underlying vector space, $\mathcal{V}$, or the scalar product on this vector space, then the Clifford algebra will change. It is not clear that we have exhausted all useful avenues in this regard when it comes to obtaining parameter sets for trajectory generation. For example, perhaps there is another inner product that one might use with $\mathcal{V} = \mathbb{R}^4$ and derive a different parameterization that is able to represent 6-DOF motion with properties that are different than those of dual quaternions. Perhaps by using higher dimensional vector spaces but the usual Euclidean inner product, we can embed 6-DOF motion in a higher-dimensional setting where the nonlinear equations of motion and/or constraint functions have properties (e.g., convexity) more amenable to optimization using the techniques presented in this work. This is not as crazy as it sounds; rotational kinematics represented using non-quaternion parameter sets (e.g., Euler angles) do not come close to the relatively benign bilinear nature of the quaternion kinematics. Moreover, unit quaternions have two other properties not shared by all attitude parameterization based on Euclidean geometry; namely that they are naturally well-scaled (all entries have magnitude at most one), and they require fewer floating point operations to perform basic operations such as successive rotations and evaluating the kinematics. These kinds of properties are important to develop well-conditioned and computationally efficient algorithms, and it is not unreasonable to suspect that yet undiscovered parameter sets that represent more general motion could be derived with similar properties.

Appendix B
SELECT ANALYTICAL JACOBIANS

This appendix provides expressions for the Jacobians of various functions used throughout the main text. The Jacobians are necessary to recreate the case studies that were presented for each algorithm. First, some basic relations for quaternions are provided in §B.1. Next, all partial derivatives required for the convexification step used by the 6-DOF powered descent case studies presented in §4.3 are provided. These use the dual quaternion formulation. Lastly, the Jacobians of auxiliary computations used for funnel synthesis are provided, namely those that pertain to the solution of Problem (6.82).

B.1 Quaternions

Recall first from (2.29) that for a general vector $\boldsymbol{z} \in \mathbb{R}^3$ the following relations hold

$$\boldsymbol{z}_{\mathcal{B}} = C_{\mathcal{BL}}\boldsymbol{z}_{\mathcal{L}} \iff \boldsymbol{z}_{\mathcal{B}} = \boldsymbol{q}^* \otimes \boldsymbol{z}_{\mathcal{L}} \otimes \boldsymbol{q}, \tag{B.1a}$$

$$\boldsymbol{z}_{\mathcal{L}} = C_{\mathcal{LB}}\boldsymbol{z}_{\mathcal{B}} \iff \boldsymbol{z}_{\mathcal{L}} = \boldsymbol{q} \otimes \boldsymbol{z}_{\mathcal{B}} \otimes \boldsymbol{q}^*. \tag{B.1b}$$

Consider the derivative of either $\boldsymbol{z}_{\mathcal{B}}$ or $\boldsymbol{z}_{\mathcal{I}}$ with respect to the attitude quaternion (the matrix $C_{\mathcal{BL}}$ is a function of this quaternion).

Consider first (B.1a), where $C_{\mathcal{BL}}$ is given by

$$C_{\mathcal{BL}} = \left(q_4^2 - \boldsymbol{q}_v^\top \boldsymbol{q}_v\right) - 2q_4\boldsymbol{q}_v^\times + 2\boldsymbol{q}_v\boldsymbol{q}_v^\top \tag{B.2}$$

The i^{th} column of $C_{\mathcal{BL}}$ will multiply the i^{th} entry of $\boldsymbol{z}_{\mathcal{L}}$, $z_{\mathcal{L},i}$, for each $i = 1, 2, 3$. Since the complete derivative of $\boldsymbol{z}_{\mathcal{B}}$ with respect to $\boldsymbol{q}$ should be a 3×4 matrix, *differentiating the columns of the matrix $C_{\mathcal{IB}}$ with respect to the quaternion* will provide three 3×4 matrices

that are then multiplied by the appropriate scalar entry in the vector $\boldsymbol{z}_{\mathcal{L}}$ to compute the resulting Jacobian. Formally, these are

$$\frac{\partial {C_{\mathcal{BL}}}^{(1)}}{\partial \boldsymbol{q}} = 2\begin{bmatrix} q_1 & -q_2 & -q_3 & q_0 \\ q_2 & q_1 & q_0 & q_3 \\ q_3 & -q_0 & q_1 & -q_2 \end{bmatrix}, \quad \frac{\partial {C_{\mathcal{BL}}}^{(2)}}{\partial \boldsymbol{q}} = 2\begin{bmatrix} q_2 & q_1 & -q_0 & -q_3 \\ -q_1 & q_2 & -q_3 & q_0 \\ q_0 & q_3 & q_2 & q_1 \end{bmatrix},$$

$$\frac{\partial C_{\mathcal{BL}}^{(3)}}{\partial \boldsymbol{q}} = 2\begin{bmatrix} q_3 & q_0 & q_1 & q_2 \\ -q_0 & q_3 & q_2 & -q_1 \\ -q_1 & -q_2 & q_3 & q_0 \end{bmatrix},$$

where $C_{\mathcal{BL}}^{(i)}$ denotes the i^{th} column of $C_{\mathcal{BL}}$. The Jacobian is then evaluated as

$$\frac{\partial \boldsymbol{z}_{\mathcal{B}}}{\partial \boldsymbol{q}} = \frac{\partial {C_{\mathcal{LB}}}^{1}}{\partial \boldsymbol{q}} z_{\mathcal{L},1} + \frac{\partial {C_{\mathcal{LB}}}^{2}}{\partial \boldsymbol{q}} z_{\mathcal{L},2} + \frac{\partial {C_{\mathcal{LB}}}^{3}}{\partial \boldsymbol{q}} z_{\mathcal{L},3}. \tag{B.3}$$

Similarly, the linearization of (B.1b) is achieved by observing that the entries in the i^{th} column of $C_{\mathcal{LB}}$ multiply the scalar terms $z_{\mathcal{B},i}$ for each $i = 1, 2, 3$. The i^{th} column of $C_{\mathcal{LB}}$ is the same as the i^{th} row of $C_{\mathcal{BL}}$. The same process carried out to differentiate the columns of $C_{\mathcal{BL}} = C_{\mathcal{LB}}^{\top}$ with respect to the quaternion provides the three necessary 3×4 matrices that are then used in the analogous way to (B.3).

The quaternion case is more easily computed by using the matrix expression given in (2.22). In this case,

$$\frac{\partial \boldsymbol{z}_{\mathcal{B}}}{\partial \boldsymbol{q}} = [\boldsymbol{q}^* \otimes \boldsymbol{z}_{\mathcal{L}}]_{\otimes} + [\boldsymbol{z}_{\mathcal{L}} \otimes \boldsymbol{q}]_{\otimes}^{*} \hat{I}, \quad \hat{I} = \mathbf{diag}\left\{-I_3, 1\right\}, \tag{B.4a}$$

$$\frac{\partial \boldsymbol{z}_{\mathcal{L}}}{\partial \boldsymbol{q}} = [\boldsymbol{q} \otimes \boldsymbol{z}_{\mathcal{B}}]_{\otimes} \hat{I} + [\boldsymbol{z}_{\mathcal{B}} \otimes \boldsymbol{q}^*]_{\otimes}^{*}. \tag{B.4b}$$

where the fourth row is removed to obtain the desired 3×4 matrix (we've implicitly assumed that $\boldsymbol{z}_{\mathcal{B}}$ or $\boldsymbol{z}_{\mathcal{L}}$ is a pure quaternion in (B.1), the removal of this row reconciles this assumption with the fact that $\boldsymbol{z}_{\mathcal{B}}$ and $\boldsymbol{z}_{\mathcal{L}}$ are in fact three-dimensional vectors).

B.2 6-DOF Powered Descent

There are numerous partial derivatives needed to solve any nonconvex optimal control problem using the PTR algorithm. Recall that the (nonlinear) equations of motion are linearized as part of the transcription process, and any nonconvex algebraic constraints must be approximated somehow by using convex functions of the solution variables. Regardless of the method used for the latter, the Jacobian of the constraint with respect to the solution variables that it depends on is useful. First-order approximations are relatively cheap to compute, are guaranteed to be convex, and are likely the most widely applicable strategy without further assumptions on the functional form of the nonconvex constraint.

For the 6-DOF powered descent guidance problem, the problem formulation was described in §3.4. The presentation of the Jacobians is split into three parts: the equations of motion, the boundary conditions, and the algebraic constraints (including the state-triggered constraints). We include the aerodynamic effects and coast time, t_c, in the presentation of the Jacobians, even though neither of these were used in the case studies presented (problems solved using each of them can be found in [78]). The parameter vector $\boldsymbol{p}$ is therefore composed of t_f and t_c in that order.

Equations of Motion

Recall that the state vector is composed of the mass, position and velocity in the surface-fixed landing frame, and the attitude quaternion and angular velocity in the body frame. The position vector and attitude quaternion were combined into the unit dual quaternion, $\tilde{\boldsymbol{q}}$, whereas the velocity and angular velocity were combined into the dual quaternion, $\tilde{\boldsymbol{\omega}}$. The equations of motion were then derived to be

$$\dot{\boldsymbol{x}} = f\big(\boldsymbol{x}, \boldsymbol{u}\big) = \begin{bmatrix} -\alpha\|\boldsymbol{u}\|_2 - \beta \\ \frac{1}{2}\tilde{\boldsymbol{q}} \otimes \tilde{\boldsymbol{\omega}} \\ \boldsymbol{J}^{-1}\left[\Phi\boldsymbol{u}_{\mathcal{B}} + m\tilde{\boldsymbol{g}}_{\mathcal{B}} + \Phi_a\boldsymbol{a}_{\mathcal{B}} - \tilde{\boldsymbol{\omega}} \oslash \boldsymbol{J}\tilde{\boldsymbol{\omega}}\right] \end{bmatrix}. \tag{B.5}$$

Recall that after temporally normalizing (see (4.7)), these dynamics are written in terms of the normalized time variable τ as

$$\boldsymbol{x}' = F\big(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}\big) = t_f f\big(\boldsymbol{x}, \boldsymbol{u}\big). \tag{B.6}$$

The partial derivatives of F with respect to each of its inputs are:

$$A = \nabla_x F\big(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{p}}\big) = \begin{bmatrix} 0_{1\times 1} & 0_{1\times 8} & 0_{1\times 8} \\ 0_{8\times 1} & A_{qq} & A_{qw} \\ A_{\omega m} & A_{\omega q} & A_{\omega\omega} \end{bmatrix} \tag{B.7a}$$

$$B = \nabla_u F\big(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{p}}\big) = \begin{bmatrix} B_m \\ 0_{8\times 3} \\ B_\omega \end{bmatrix} \tag{B.7b}$$

$$E = \nabla_p F\big(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{p}}\big) = \begin{bmatrix} f\big(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}\big) & 0_{n_x\times 1} \end{bmatrix} \tag{B.7c}$$

where,

$$A_{qq} = \frac{1}{2}[\tilde{\boldsymbol{\omega}}]^*_\otimes, \quad A_{q\omega} = \frac{1}{2}[\tilde{\boldsymbol{q}}]_\otimes, \quad A_{\omega m} = \begin{bmatrix} 0_{4\times 1} \\ -\frac{1}{m^2}\boldsymbol{u} \end{bmatrix} \tag{B.8a}$$

$$A_{\omega q} = \boldsymbol{J}^{-1}\left[m\tilde{I}_8\left([\tilde{\boldsymbol{q}}^* \otimes \tilde{\boldsymbol{g}}_{\mathcal{L}}]_\otimes + [\tilde{\boldsymbol{g}}_{\mathcal{L}} \otimes \tilde{\boldsymbol{q}}]^*_\otimes \hat{I}_8\right)\right], \quad \tilde{\boldsymbol{g}}_{\mathcal{L}} = \begin{bmatrix} 0_{4\times 1} \\ \boldsymbol{g}_{\mathcal{L}} \end{bmatrix} \tag{B.8b}$$

$$A_{\omega\omega} = -\boldsymbol{J}^{-1}\left([\tilde{\boldsymbol{\omega}}]_\oslash \boldsymbol{J} + [\boldsymbol{J}\tilde{\boldsymbol{\omega}}]^*_\oslash + \Phi_a \nabla_{\tilde{\boldsymbol{\omega}}} \boldsymbol{a}_{\mathcal{B}}\right) \tag{B.8c}$$

$$B_m = -\alpha \frac{\boldsymbol{u}^\top}{\|\boldsymbol{u}\|_2}, \quad B_\omega = \boldsymbol{J}^{-1}\Phi \tag{B.8d}$$

$$\nabla_{\tilde{\boldsymbol{\omega}}} \boldsymbol{a}_{\mathcal{B}} = -\frac{1}{2}\rho A C_a \left(E_v \|E_v \tilde{\boldsymbol{\omega}}\|_2 + \frac{E_v \tilde{\boldsymbol{\omega}} \tilde{\boldsymbol{\omega}}^\top E_v^\top E_v}{\|E_v \tilde{\boldsymbol{\omega}}\|_2}\right), \tag{B.8e}$$

$$\tilde{I}_8 = \begin{bmatrix} 0_{4\times 4} & I_4 \\ I_4 & 0_{4\times 4} \end{bmatrix}, \quad \hat{I}_8 = I_2 \otimes_K \hat{I}. \tag{B.8f}$$

Note that the rows and columns that correspond to the (dual) scalar part of the dual quaternion $\tilde{\boldsymbol{\omega}}$ will always be zero in each of A, B and E. As mentioned in the text below (3.54), we typically treat $\tilde{\boldsymbol{\omega}}$ as a six-dimensional vector in implementation in order to reduce the state dimension from $n_x = 17$ to $n_x = 15$.

Boundary Conditions

The boundary conditions, including the coast time, are given by (3.65a) and (3.67a). These boundary conditions do not depend on the control vector, and the terminal velocity constraints (3.67a) are all fixed values; hence we do not need to compute any partial derivatives of them. Only the partial derivatives of (3.65a) with respect to the initial state vector and the parameter vector are required.

For the mass, we have, using Leibniz's rule:

$$\nabla_{t_c} m(t_0) = -\alpha \Gamma_{\mathrm{c}}(t_c) - \beta. \tag{B.9}$$

The initial mass does not depend on the state vector in any way as we have formulated it.

For the initial dual quaternion, we have

$$\nabla_{\boldsymbol{q}(t_0)} \boldsymbol{b}_{\tilde{\boldsymbol{q}}}(\boldsymbol{q}(t_0), t_c) = \begin{bmatrix} I_4 \\ \frac{1}{2}[\boldsymbol{b}_{\boldsymbol{r},0}(t_c)]_{\otimes} \end{bmatrix}, \tag{B.10a}$$

$$\nabla_{t_c} \boldsymbol{b}_{\tilde{\boldsymbol{q}}}(\boldsymbol{q}(t_0), t_c) = \frac{1}{2}[\boldsymbol{q}(t_0)]^*_{\otimes} \nabla_{t_c} \boldsymbol{b}_{\boldsymbol{r},0}(t_c), \tag{B.10b}$$

$$\nabla_{\boldsymbol{q}(t_0)} \boldsymbol{b}_{\tilde{\boldsymbol{\omega}}}(\boldsymbol{q}(t_0), t_c) = \begin{bmatrix} 0 \\ [\boldsymbol{q}(t_0)^* \otimes \boldsymbol{b}_{\boldsymbol{v},0}(t_c)]_{\otimes} + [\boldsymbol{b}_{\boldsymbol{v},0}(t_c) \otimes \boldsymbol{q}(t_0)]^*_{\otimes} \hat{I} \end{bmatrix}, \tag{B.10c}$$

$$\nabla_{t_c} \boldsymbol{b}_{\tilde{\boldsymbol{\omega}}}(\boldsymbol{q}(t_0), t_c) = [\boldsymbol{q}^*(t_0)]_{\otimes} [\boldsymbol{q}(t_0)]^*_{\otimes} \nabla_{t_c} \boldsymbol{b}_{\boldsymbol{v},0}(t_c), \tag{B.10d}$$

where the particular forms of $\nabla_{t_c} \boldsymbol{b}_{\boldsymbol{r},0}(t_c)$ and $\nabla_{t_c} \boldsymbol{b}_{\boldsymbol{v},0}(t_c)$ depend on the chosen coast trajec-

tory. For the example engine-off aerodynamics-free case given in (3.66), these are

$$\nabla_{t_c}\boldsymbol{b}_{\boldsymbol{r},0}(t_c) = \boldsymbol{v}_{\mathcal{L},ic} + \boldsymbol{g}_{\mathcal{L}}t_c, \quad \text{and} \quad \nabla_{t_c}\boldsymbol{b}_{\boldsymbol{v},0}(t_c) = \boldsymbol{g}_{\mathcal{L}}.$$

State Constraints

All of the control constraints for the 6-DOF powered descent guidance problem, as presented, are convex except for the thrust lower bound constraint. It is straightforward to derive the partial derivative of this constraint, and so we omit it here in favor of presenting the necessary Jacobians of the dual quaternion-based state constraints and the state-triggered constraints.

The approach cone constraint (3.57), while convex over the given domain, is not a second-order cone, and therefore must be linearized when using the real-time implementation methods of Chapter 5. Moreover, it does not satisfy the disciplined convex programming rule set required to prototype code using the popular CVX tool [19] – and therefore needs to be linearized in that setting as well. The required Jacobian of the constraint (3.57) is

$$\nabla_{\tilde{\boldsymbol{q}}}c_\gamma = -2\tilde{\boldsymbol{q}}^\top M_\gamma + \frac{2\tilde{\boldsymbol{q}}^\top E_d}{\|E_d\tilde{\boldsymbol{q}}\|_2}\cos\gamma_{\max} \tag{B.11}$$

For the state-triggered constraints, we provide expressions for the slant-range-triggered LOS constraint and the speed-triggered AOA constraint. The former is an example of a compound STC, and we begin by noting that all partial derivatives are given assuming that all trigger conditions are satisfied. In this case, we have from (3.70) that$\sigma^*(\cdot) = -g(\cdot)$.

The slant-range-triggered LOS constraint is given by (3.79). This constraint is a function of the unit dual quaternion $\tilde{\boldsymbol{q}}$ only, and the associated Jacobians are

$$\begin{aligned}\nabla_{\tilde{\boldsymbol{q}}}h_\xi\big(\tilde{\boldsymbol{q}}\big) = \nabla_{\tilde{\boldsymbol{q}}}g_{\xi,1}\big(\tilde{\boldsymbol{q}}\big)g_{\xi,2}(\tilde{\boldsymbol{q}})c_\xi(\tilde{\boldsymbol{q}}) + g_{\xi,1}(\tilde{\boldsymbol{q}})\nabla_{\tilde{\boldsymbol{q}}}g_{\xi,2}\big(\tilde{\boldsymbol{q}}\big)c_\xi(\tilde{\boldsymbol{q}}) \\ + g_{\xi,1}(\tilde{\boldsymbol{q}})g_{\xi,2}(\tilde{\boldsymbol{q}})\nabla_{\tilde{\boldsymbol{q}}}c_\xi\big(\tilde{\boldsymbol{q}}\big)\end{aligned} \tag{B.12a}$$

$$\nabla_{\tilde{\boldsymbol{q}}}g_{\xi,1}\big(\tilde{\boldsymbol{q}}\big) = -\frac{2\tilde{\boldsymbol{q}}^\top E_d}{\|E_d\tilde{\boldsymbol{q}}\|_2}, \quad \nabla_{\tilde{\boldsymbol{q}}}g_{\xi,2}\big(\tilde{\boldsymbol{q}}\big) = \frac{2\tilde{\boldsymbol{q}}^\top E_d}{\|E_d\tilde{\boldsymbol{q}}\|_2}, \tag{B.12b}$$

$$\nabla_{\tilde{\boldsymbol{q}}} c_\xi\big(\tilde{\boldsymbol{q}}\big) = 2\tilde{\boldsymbol{q}}^\top M_\xi + \frac{2\tilde{\boldsymbol{q}}^\top E_d}{\|E_d\tilde{\boldsymbol{q}}\|_2}\cos\xi_{\max}. \tag{B.12c}$$

The speed-triggered angle of attack constraint is given in (3.84). This constraint is a function of the dual velocity $\tilde{\boldsymbol{\omega}}$ only, and the associated Jacobians are

$$\nabla_{\tilde{\omega}} h_\alpha\big(\tilde{\boldsymbol{\omega}}\big) = -\nabla_{\tilde{\omega}} g_\alpha\big(\tilde{\boldsymbol{\omega}}\big) c_\alpha(\tilde{\boldsymbol{\omega}}) - g_\alpha(\tilde{\boldsymbol{\omega}})\nabla_{\tilde{\omega}} c_\alpha\big(\tilde{\boldsymbol{\omega}}\big) \tag{B.13a}$$

$$\nabla_{\tilde{\omega}} g_\alpha\big(\tilde{\boldsymbol{\omega}}\big) = \frac{\tilde{\boldsymbol{\omega}}^\top E_v^\top E_v}{\|E_v\tilde{\boldsymbol{\omega}}\|_2}, \quad \nabla_{\tilde{\omega}} c_\alpha\big(\tilde{\boldsymbol{\omega}}\big) = \boldsymbol{z}_{\mathcal{B}}^\top E_v + \frac{\tilde{\boldsymbol{\omega}}^\top E_v^\top E_v}{\|E_v\tilde{\boldsymbol{\omega}}\|_2}\cos\alpha_{\max} \tag{B.13b}$$

B.3 Funnel Synthesis

This section provides the Jacobians for the cost and constraint functions for the optimization problem (6.82). The cost function was constructed by creating the matrix H that satisfies the equation

$$E\Delta C\boldsymbol{\eta} = H(\boldsymbol{\eta})\boldsymbol{\delta},$$

where $\boldsymbol{\delta} = \operatorname{vec}\Delta$ for some matrices $E \in \mathbb{R}^{n_x \times n_p}$, $\Delta \in \mathbb{R}^{n_p \times n_q}$, $C \in \mathbb{R}^{n_q \times n_x}$ and $\boldsymbol{\eta} \in \mathbb{R}^{n_x}$. Note that C has been used in place of C_{cl} here to avoid excessive subscripts. The matrix H is defined in (6.79) to be

$$H(\boldsymbol{\eta}) = \begin{bmatrix} E\big(\boldsymbol{e}_1^\top C\boldsymbol{\eta}\big) & \cdots & E\big(\boldsymbol{e}_{n_q}^\top C\boldsymbol{\eta}\big) \end{bmatrix} \in \mathbb{R}^{n_x \times n_q n_p}.$$

The Jacobian of H with respect to $\boldsymbol{\eta}$ is an $n_x \times n_q n_p \times n_x$ tensor that can be computed via

$$\nabla_{\eta_k} H\big(\boldsymbol{\eta}\big) = \begin{bmatrix} EC_{1,k} & \cdots & EC_{n_q,k} \end{bmatrix} \in \mathbb{R}^{n_x \times n_q n_p} \tag{B.14}$$

for each $k = 1, \ldots, n_x$ along the third dimension. The cost function for the optimization problem (6.82) is

$$J = \frac{1}{2} y(\boldsymbol{\eta})^\top {H^\dagger}^\top(\boldsymbol{\eta}) H^\dagger(\boldsymbol{\eta}) y(\boldsymbol{\eta}).$$

where $y(\boldsymbol{\eta}) = \dot{\boldsymbol{\eta}} - A_{cl}\boldsymbol{\eta}$. The Jacobian of the cost with respect to $\boldsymbol{\eta}$ is then

$$\nabla_{\boldsymbol{\eta}} J = y(\boldsymbol{\eta})^\top {H^\dagger}^\top(\boldsymbol{\eta}) \left[\nabla_{\boldsymbol{\eta}} H^\dagger\big(\boldsymbol{\eta}\big) y(\boldsymbol{\eta}) + H^\dagger(\boldsymbol{\eta}) \nabla_{\boldsymbol{\eta}} y\big(\boldsymbol{\eta}\big)\right], \tag{B.15}$$

where, as shown in [223],

$$\begin{aligned}\nabla_{\boldsymbol{\eta}} H^\dagger\big(\boldsymbol{\eta}\big) = -H^\dagger \nabla_{\boldsymbol{\eta}} H H^\dagger + H^\dagger {H^\dagger}^\top \nabla_{\boldsymbol{\eta}} H^\top \left(I_{n_x} - H H^\dagger\right) \\ + \left(I_{n_p} - H^\dagger H\right) \nabla_{\boldsymbol{\eta}} H^\top {H^\dagger}^\top H^\dagger\end{aligned} \tag{B.16a}$$

is an $(n_q n_p \times n_x \times n_x)$ tensor, and,

$$\nabla_{\boldsymbol{\eta}} y\big(\boldsymbol{\eta}\big) = \nabla_{\boldsymbol{\eta}} \left(f\big(\bar{\boldsymbol{x}} + \boldsymbol{\eta}, \bar{\boldsymbol{u}} + K\boldsymbol{\eta}\big) - A_{cl}\boldsymbol{\eta}\right) = A + BK - A_{cl} \tag{B.16b}$$

where $A = \nabla_x f\big(\bar{\boldsymbol{x}} + \boldsymbol{\eta}, \bar{\boldsymbol{u}} + K\boldsymbol{\eta}\big)$ and $B = \nabla_u f\big(\bar{\boldsymbol{x}} + \boldsymbol{\eta}, \bar{\boldsymbol{u}} + K\boldsymbol{\eta}\big)$. The calculation of (B.15) then proceeds by multiplying each

$$\nabla_{\eta_k} H^\dagger\big(\boldsymbol{\eta}\big) y(\boldsymbol{\eta}), \quad k = 1, \ldots, n_x$$

to create an $(n_q n_p \times 1 \times n_x)$ tensor that can be reduced to an $n_q n_p \times n_x$ matrix by squeezing along the second dimension. The resulting matrix is then summed with the matrix $H^\dagger(\boldsymbol{\eta}) \nabla_{\boldsymbol{\eta}} y\big(\boldsymbol{\eta}\big) \in \mathbb{R}^{n_q n_p \times n_x}$ to produce the term in square brackets in (B.15).

Lastly, there is one constraint in the optimization problem (6.82) that ensures the solution resides in the quadratic funnel. The constraint is given by

$$c(\boldsymbol{\eta}) = \boldsymbol{\eta}^\top Q^{-1} \boldsymbol{\eta} - 1 \leq 0$$

and its Jacobian is

$$\nabla_{\boldsymbol{\eta}} c\big(\boldsymbol{\eta}\big) = 2\boldsymbol{\eta}^\top Q^{-1} \in \mathbb{R}^{1 \times n_x}. \tag{B.17}$$

Appendix C

TEMPORAL MATRIX DECOMPOSITIONS

Denote the standard n-dimensional simplex by

$$\Sigma^n = \left\{ \sigma \in \mathbb{R}^{n+1} \;\middle|\; \sum_{i=1}^{n+1} \sigma_i = 1,\ \sigma_i \geq 0,\ i = 1, \ldots, n \right\}. \tag{C.1}$$

Let $M(t) \in \mathbb{R}^{n \times m}$ and $N(t) \in \mathbb{R}^{m \times n}$ be matrix valued functions of time. Suppose that

$$M(t) = \sum_{i=1}^{n_M} \sigma_i(t) M_i \quad \text{and} \quad N(t) = \sum_{j=1}^{n_N} \varsigma_j(t) N_j, \tag{C.2}$$

where $\sigma(t) = \big(\sigma_1(t), \ldots, \sigma_{n_M}(t)\big) \in \Sigma^{n_M - 1}$ and $\varsigma(t) = \big(\varsigma_1(t), \ldots, \varsigma_{n_N}(t)\big) \in \Sigma^{n_N - 1}$.

We then have the following two results on the decomposition of the product $M(t)N(t)$.

Lemma C.1. *At any time $t \in \mathbb{R}_+$, the product $M(t)N(t) \in \mathbb{R}^{n \times n}$ can be written as a convex combination of the $n_M n_N$ matrices*

$$M_i N_j, \quad i = 1, \ldots, n_M,\ j = 1, \ldots, n_N. \tag{C.3}$$

Proof. Let $M(t)$ and $N(t)$ be as in (C.2). Then $M(t)N(t) = \sum_{i=1}^{n_M} \sum_{j=1}^{n_N} \sigma_i(t)\varsigma_j(t) M_i N_j$. It is straightforward to show that $\sum_{i=1}^{n_M} \sum_{j=1}^{n_N} \sigma_i(t)\varsigma_j(t) = 1$, and that $\sigma_i(t)\varsigma_j(t) \geq 0$ for any i, j. □

The second result is a special case of Lemma C.1 for which the dimensions of $M(t)$ and $N(t)$ are equal and the time-varying coefficients in the expansions (C.2) are the same.

Lemma C.2. *If $n_M = n_N$ and $\sigma(t) = \varsigma(t)$ for any $t \in \mathbb{R}_+$, then the product $M(t)N(t) \in \mathbb{R}^{n \times n}$*

can be written as a convex combination of the $\frac{1}{2}n_M(n_M+1)$ *matrices:*

$$M_iN_i, \qquad i=1,\ldots,n_M \tag{C.4a}$$

$$M_iN_j+M_jN_i, \quad i=1,\ldots,n_M-1,\ j=i+1,\ldots,n_M \tag{C.4b}$$

Proof. The proof is by construction. Let $M(t)$ and $N(t)$ be as in (C.2) with $n_M=n_N$. From the proof of Lemma C.1 we know that the product $M(t)N(t)$ can be written as the convex combination $M(t)N(t)=\sum_{i=1}^{n_M}\sum_{j=1}^{n_M}\sigma_i(t)\varsigma_j(t)M_iN_j$, where $\sum_{i=1}^{n_M}\sum_{j=1}^{n_M}\sigma_i(t)\varsigma_j(t)=1$. If $\sigma(t)=\varsigma(t)$ for any $t\in\mathbb{R}_+$ then we have $\sigma_i(t)\varsigma_j(t)=\sigma_j(t)\varsigma_i(t)$, and so:

$$\begin{aligned}
M(t)N(t) &= \sum_{i=1}^{n_M}\sum_{j=1}^{n_M}\sigma_i(t)\varsigma_j(t)M_iN_j \\
&= \sum_{i=1}^{n_M}\sigma_i(t)\varsigma_j(t)M_iN_i+\sum_{i=1}^{n_M}\sum_{j=1,j\neq i}^{n_M}\sigma_i(t)\varsigma_j(t)M_iN_j \\
&= \sum_{i=1}^{n_M}\sigma_i(t)\varsigma_j(t)M_iN_i+\sum_{i=1}^{n_M-1}\sum_{j=i+1}^{n_M}\sigma_i(t)\varsigma_j(t)\left(M_iN_j+M_jN_i\right)
\end{aligned}$$

The last expression establishes that the product $M(t)N(t)$ is a convex combination of the matrices in (C.4). □