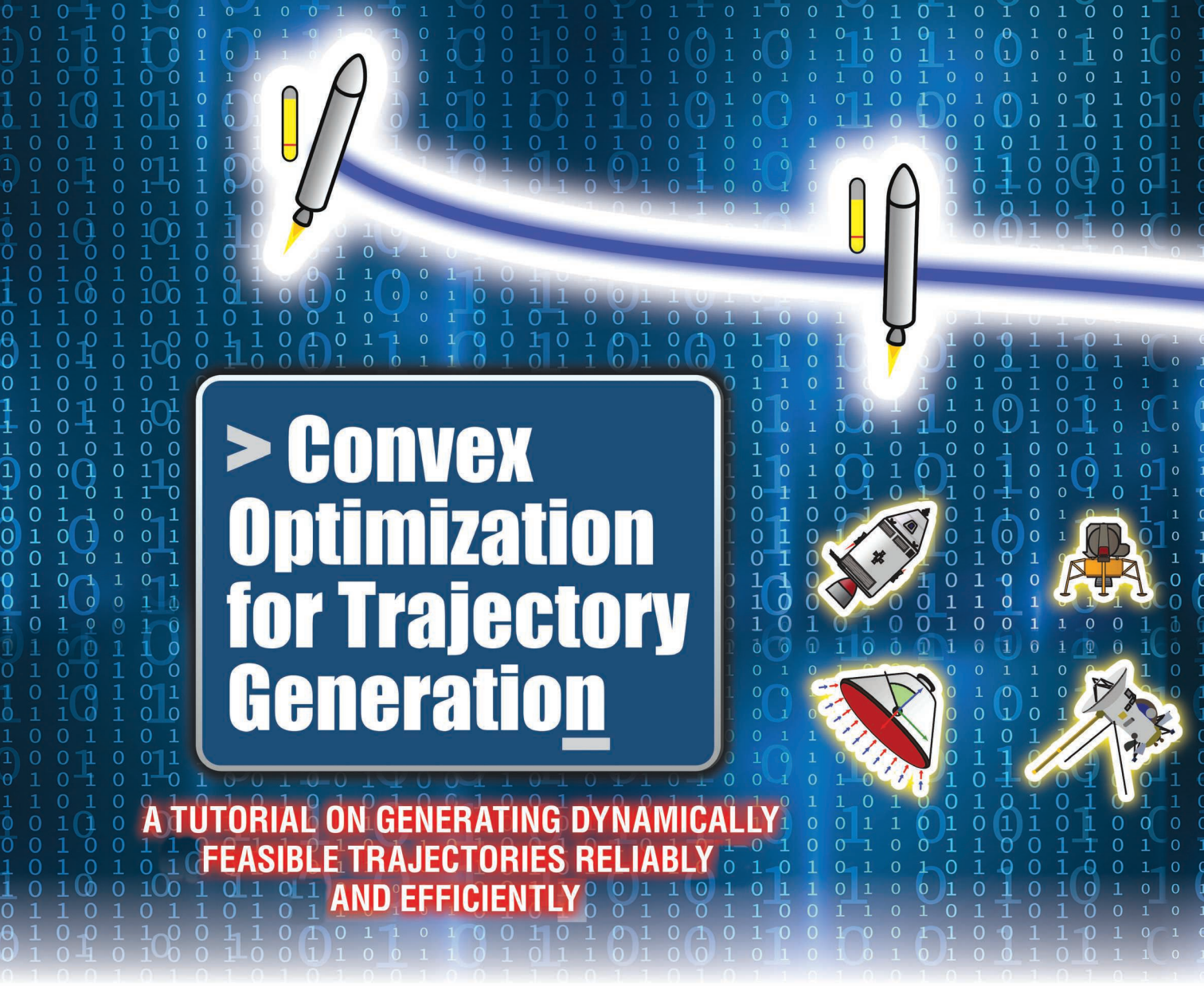


> Convex Optimization for Trajectory Generation

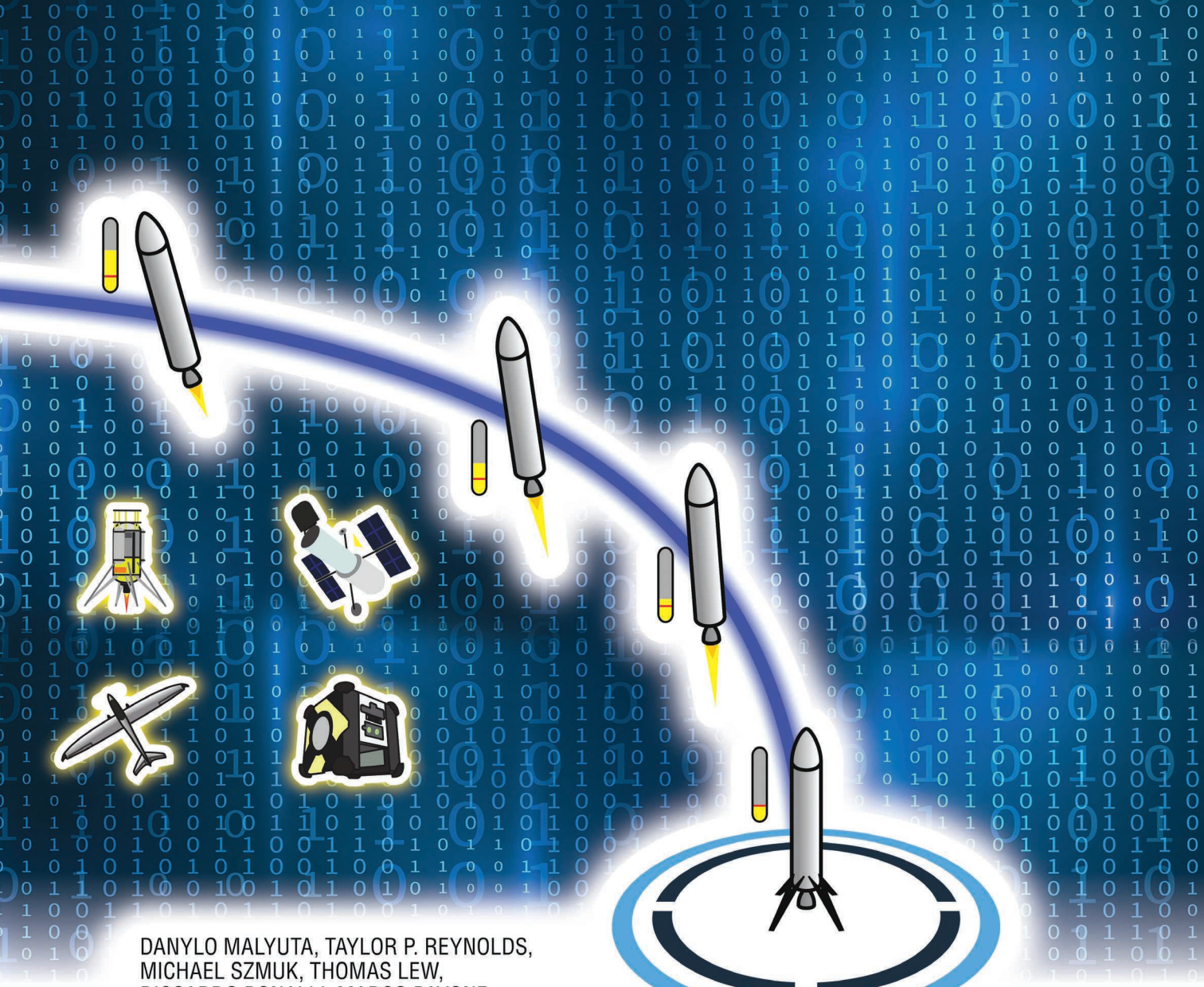
A TUTORIAL ON GENERATING DYNAMICALLY FEASIBLE TRAJECTORIES RELIABLY AND EFFICIENTLY

Autonomous vehicles and robots promise many exciting new applications that will transform society. For example, autonomous aerial vehicles (AAVs) operating in urban environments could deliver commercial goods and emergency medical supplies, monitor traffic, and provide threat alerts for national security [1]. At the same time, these applications present significant engineering challenges related to performance, trustworthiness, and safety. For instance, AAVs can be a catastrophic safety hazard should they lose control or situational awareness over a populated area. Space missions, self-driving cars, and applications of autonomous underwater vehicles share similar concerns.

Digital Object Identifier 10.1109/MCS.2022.3187542
Date of current version: 15 September 2022

Generating a trajectory autonomously onboard the vehicle is not only desirable for many of these applications but also a necessity when considering the deployment of autonomous systems (either in remote areas with little to no communication or at scale in a dynamic and uncertain world). For example, it is not possible to remotely control a spacecraft during a Mars landing scenario [2]–[4], nor is it practical to coordinate the motion of tens of thousands of delivery drones from a centralized location [1]. In these and many other scenarios, individual vehicles must be endowed with their own high-quality decision-making capability. Failure to generate a safe trajectory can result in losing the vehicle, payload, and even human life. Reliable methods for trajectory generation are a fundamental need for maintaining public trust and the high safety standard that is expected from autonomous and automatic systems [5].

1066-033X/22©2022IEEE



DANYLO MALYUTA, TAYLOR P. REYNOLDS,
MICHAEL SZMUK, THOMAS LEW,
RICCARDO BONALLI, MARCO PAVONE,
and BEHÇET AÇIKMEŞE

DANYLO MALYUTA

Computational resource requirements are the second major consideration for onboard trajectory generation. Historically, this has been the driving factor for much of the practical algorithm development [6], [7]. Although the modern consumer desktop is an incredibly powerful machine, industrial CPUs can still be relatively modest [8]. This is especially true for spaceflight, where the harsh radiation environment of outer space prevents the rapid adoption of new computing technologies. For example, NASA's flagship Mars rover *Perseverance* landed in 2021 while using a BAE RAD750 PowerPC flight computer, which is a 20-year-old technology [9], [10]. Factoring in the energy requirements of powerful CPUs and the fact that trajectory generation is a small part of all the tasks that an autonomous vehicle must perform, it becomes clear that modern trajectory generation is still confined to a small computational footprint. Consequently, real-time, onboard trajectory generation algorithms must be computationally efficient.

Trajectory generation is the computation of a multidimensional temporal state and control signal that satisfies a set of specifications while optimizing key mission objectives. This article is concerned exclusively with dynamically feasible trajectories (see "Summary"), which are those that respect the equations of motion of the vehicle under consideration. Although it is commonplace to track dynamically *in*feasible trajectories by using feedback control, a system can evolve only along dynamically feasible paths (whether those are computed up front during trajectory generation or are the result of feedback tracking). Performing dynamically feasible trajectory generation carries two important advantages. First, it provides a

method to systematically satisfy constraints (that is, specifications) that are hard, if not impossible, to meet through feedback control. This includes, for example, translation–attitude coupled sensor pointing constraints. Second, dynamically feasible trajectories leave much less tracking error for feedback controllers to "clean up," which usually means that tracking performance can be vastly improved. These two advantages will become more apparent throughout the article.

Numerical optimization provides a systematic mathematical framework to specify mission objectives as *costs or rewards* to be optimized and enforces state and control specifications as well as the equations of motion via *constraints*. As a result, trajectory generation problems can be expressed as optimal control problems, which are infinite–dimensional optimization problems over function spaces [11], [12]. Since the early 1960s, optimal control theory has proved to be extremely powerful [6], [13]. Early developments were driven by aerospace applications, where every gram of mass matters, and trajectories were typically sought to minimize fuel or some other mass-reducing metric (such as the aerodynamic load and thereby the structural mass). This led to work on trajectory algorithms for climbing aircraft, ascending and landing rockets, and spacecraft orbit transfer, to name a few [14]–[19]. Following these early applications, trajectory optimization problems have now been formulated in many practical areas, including aerial, underwater, and space vehicles, as well as for chemical processes [20]–[22], building climate control [23]–[25], robotics, and medicine [13].

At its core, optimization-based trajectory generation requires solving an optimal control problem of the following form (at this point, the problem is kept very general):

$$\min_{u,p,t_f} \; J(x, u, p, t_f), \tag{1a}$$

$$\text{s.t. } \dot{x}(t) = f(x, u, p, t), \tag{1b}$$

$$(x(t), u(t), p, t_f) \in \mathcal{C}(t), \quad \forall t \in [0, t_f], \tag{1c}$$

$$(x(0), p) \in \mathcal{X}_0, \quad (x(t_f), p) \in \mathcal{X}_f. \tag{1d}$$

The cost (1a) encodes the mission goal, the system dynamics are modeled by the differential equation constraint (1b), the state and control specifications are enforced through (1c), and the boundary conditions are fixed by (1d). Note that (1) makes a distinction between a control vector $u(t)$ (which is a temporal signal) and a so-called parameter vector p (which is a static variable that encodes other decision variables, such as temporal scaling).

In most cases, the solution of such an infinite–dimensional optimization problem is neither available in closed form nor computationally tractable to numerically compute (especially in real time). Instead, different solution methods have been proposed that typically share the following three main components:

- *Formulation*: specification of how the functions J and f and the sets $\mathcal{C}$, $\mathcal{X}_0$, and $\mathcal{X}_f$ are expressed mathematically
- *Discretization*: approximation of the infinite–dimensional state and control signal by a finite–dimensional set of basis functions
- *Numerical optimization*: iterative computation of an optimal solution of the discretized problem.

Choosing the most suitable combination of these three components is truly a mathematical art form that is highly problem dependent and not an established plug-and-play process like least-squares regression [26]. No single recipe or method is the best, and methods that work well for some

Summary

Reliable and efficient trajectory generation methods are a fundamental need for autonomous dynamical systems. The goal of this article is to provide a comprehensive tutorial of three major convex optimization-based trajectory generation methods: lossless convexification (LCvx) and two sequential convex programming algorithms, successive convexification (SCvx) and guaranteed sequential trajectory optimization (GuSTO). Trajectory generation is defined as the computation of a dynamically feasible state and control signal that satisfies a set of constraints while optimizing key mission objectives. The trajectory generation problem is almost always nonconvex, which typically means that it is difficult to solve efficiently and reliably onboard an autonomous vehicle. The three algorithms that we discuss use problem reformulation and a systematic algorithmic strategy to nonetheless solve nonconvex trajectory generation tasks using a convex optimizer. The theoretical guarantees and computational speed offered by convex optimization have made the algorithms popular in both research and industry circles. The growing list of applications includes rocket landing, spacecraft hypersonic reentry, spacecraft rendezvous and docking, aerial motion planning for fixed-wing and quadrotor vehicles, robot motion planning, and more. Among these applications are high-profile rocket flights conducted by organizations such as NASA, Masten Space Systems, SpaceX, and Blue Origin. This article equips the reader with the tools and understanding necessary to work with each algorithm and know their advantages and limitations. An open source tool called the SCP Toolbox accompanies the article and provides a practical implementation of every numerical example. By the end of the article, the reader will not only be ready to use the lossless convexification and sequential convex programming algorithms, but also to extend them and to contribute to their many exciting modern applications.

problems can fare much worse for others. Trajectory algorithm design is replete with problem-dependent tradeoffs in performance, optimality, and robustness, among others. Still, the optimization literature attempts to provide some formal guidance and intuition about the process. The rest of this article considers methods that solve the *exact* form of (1), subject only to the initial approximation made by discretizing the continuous-time dynamics.

Many excellent references discuss the discretization component [13], [27], [28]. Once the problem is discretized, numerical optimization methods can be used to obtain a solution. This is where the real technical challenges for reliable trajectory generation arise. Depending on the problem formulation and discretization method, the finite–dimensional optimization problem that must be solved can end up being a nonlinear (that is, *nonconvex*) optimization problem (NLP). However, NLP optimization has high computational complexity, and there are no general guarantees of either obtaining a solution or even certifying that a solution does not exist [26], [29], [30]. Hence, general NLP methods may not be appropriate for control applications since deterministic guarantees are needed for reliable trajectory generation in autonomous control.

In contrast, if the discretized problem is convex, then it can be solved reliably and with an efficiency that exceeds all other areas of numerical optimization except linear programming and least squares [26], [29], [31]–[34]. This is the key motivation behind this article's focus on a convex optimization-based problem formulations for trajectory generation. Building methods on top of convex optimization leverages fast iterative algorithms with polynomial time complexity [35, Th. 4.7]. That is, given any desired solution accuracy, a convex optimization problem can be solved to within this accuracy in a predetermined number of arithmetic operations that is a polynomial function of the problem size. Hence, there is a deterministic bound on how much computation is needed to solve a given convex optimization problem, and the number of iterations cannot grow indefinitely as it can for NLP. These special properties, together with a mature theoretical understanding and an ever-growing list of successful real-world use cases, leave little doubt that convex optimization-based solution methods are uniquely well suited to be used in the solution of optimal control problems.

Among the successful real-world use cases of convex, optimization-based trajectory generation are several inspiring examples from the aerospace domain. These include autonomous drones, spacecraft rendezvous and docking, and, most notably, planetary landing. The latter came into high profile as an enabling technology for the reusability of the SpaceX Falcon 9 and Falcon Heavy rockets [36]. Even earlier, the NASA Jet Propulsion Laboratory (JPL) demonstrated the use of a similar method for Mars pinpoint landing aboard the Masten *Xombie* sounding rocket [37]–[39]. Today, these methods are being studied and adopted for several Mars, Moon, and Earth landing applications [8], [36]. Although each application has its own set of unique challenges, they all share the need to use the full spacecraft motion envelope with limited sensing, actuation, and fuel/power [8]. These considerations are not unique to space applications and can be found in almost all autonomous vehicles, such as cars [40], [41], walking robots [42], [43], and quadrotors [44].

Having motivated the use of convex optimization, we note that many trajectory generation problems have common sources of nonconvexity, among which are nonconvex control constraints, nonconvex and coupled state–control constraints, and nonlinear dynamics. The goal of a convex, optimization-based trajectory generation algorithm is to provide a systematic way of handling these nonconvexities and generate a trajectory using a convex solver at its core.

Two methods stand out to achieve this goal. In special cases, it is possible to reformulate the problem into a convex one through a variable substitution and "lifting" (that is, augmentation) of the control input into a higher-dimensional space. In this case, Pontryagin's maximum principle [11] can be used to show that solving the new problem recovers a globally optimal solution of the original problem. This gives the approach the name *lossless convexification* (*LCvx*), and the resulting problem can often be solved with a single call to a convex solver. As one can imagine, however, LCvx tends to apply only to very specific problems. Fortunately, these include some important and practically useful forms of rocket landing and other trajectory generation problems for spacecraft and AAVs [6], [45], [46]. When LCvx cannot be used, convex optimization can be applied via *sequential convex programming* (*SCP*). This natural extension linearizes all nonconvex elements of (1) and solves the convex problem in a local neighborhood where the linearization is accurate. Roughly speaking, the problem is then relinearized at the new solution, and the whole process is repeated until a stopping criterion is met. In the overall classification of optimization algorithms, SCP is a trust region method [29], [47], [48]. While SCP is a whole class of algorithms, the primary focus is on two particular and closely related methods called *successive convexification (SCvx)* and *guaranteed sequential trajectory optimization (GuSTO)* [49], [50].

Let us go through the numerous applications where the LCvx, SCvx, and GuSTO methods have been used. The LCvx method was originally developed for rocket landing [45], [51]. This was the method at the center of the aforementioned JPL multiyear flight test campaign for Mars pinpoint landing [37]. The method also appears in applications for fixed-wing and quadrotor AAV trajectory generation [46], [52], spacecraft hypersonic reentry [53]–[55], and spacecraft rendezvous and docking [56], [57]. The SCvx method, which applies to far more general problems (albeit with fewer runtime guarantees), has been used extensively for the general rocket landing problem [58]–[63], quadrotor trajectory generation [52], [64], [65], spacecraft rendezvous and docking [66], and CubeSat attitude control [67]. Recently (as

part of the NASA Safe and Precise Landing—Integrated Capabilities Evolution project to develop a next-generation planetary landing computer [8]), the SCvx algorithm has been tested as an experimental payload aboard the Blue Origin New Shepard rocket [68]. The GuSTO method has been applied to free-flyer robots, such as those used aboard the International Space Station (ISS) [50], [69]–[71] and for car motion planning [72], [73], aircraft motion planning [50], and robot manipulator arms [69]. It is likely that LCvx and SCP concepts are also closely related to the SpaceX Falcon 9 terminal landing algorithm, which is known to use convex optimization [36].

This article provides a first-ever comprehensive tutorial of the LCvx, SCvx, and GuSTO algorithms. Placing these related methods under the umbrella of a single article provides a unified description that highlights common ideas and helps the practitioner know how, where, and when to deploy each approach. Previous tutorials on LCvx and SCvx provide a complementary technical discussion [74], [75]. There are two reasons for focusing on LCvx, SCvx, and GuSTO specifically. First, the authors are the developers of the three algorithms. Hence, we feel best positioned to provide a thorough description of these particular methods, given our experience with their implementation. Second, these three methods have a significant history of real-world applications. This should provide confidence that the methods have withstood the test of time and proved themselves to be useful when the stakes are were high. By the end of the article, the hope is to have provided the understanding and tools necessary to adapt each method to the reader's particular engineering application. Although the discussion for SCP is restricted to SCvx and GuSTO, both techniques are closely related to other SCP algorithms. After reading this tutorial, the practitioner will be well positioned to understand most if not all other SCP methods for trajectory generation. Applications of these SCP alternatives are discussed in the recent survey article [6].

Finally, note that this article is focused on solving a trajectory optimization problem, such as (1), *once* in real time. As illustrated in Figure 1, this results in a single optimal trajectory that can be robustly tracked by a downstream control system. The ability to solve for the trajectory in real time, however, can allow for updating the trajectory as the mission evolves and more information is revealed to the autonomous vehicle. Repetitive trajectory generation provides a feedback action that can itself be used for control purposes. This approach is the driving force behind model predictive control (MPC), which has been applied to many application domains over the past three decades [76]–[78]. Except for a brief discussion at the end of the section on SCP, this article does not cover MPC, and the interested reader should consult existing literature [78], [79]. Specifically, the application of LCvx and SCP algorithms in an MPC-like fashion is discussed in [6].

The rest of this article is organized as follows. The discussion begins with a short section on convex optimization, where the primary objective is to highlight why it is so useful for trajectory generation. The article is then split into three main sections. The "Lossless Convexification" section surveys the major results of LCvx to solve nonconvex trajectory problems in one shot. The "Sequential Convex Programming" section discusses SCP, which can handle very general and highly nonconvex trajectory generation tasks by iteratively solving a number of convex optimization problems. In particular, this part provides a detailed tutorial on two modern SCP methods: SCvx and GuSTO [49], [50]. Finally, the "Application Examples" section applies LCvx, SCvx, and GuSTO to three complex trajectory generation problems: a rocket-powered planetary lander, a quadrotor, and a microgravity free-flying robotic assistant.

Some important naming conventions and notations used throughout the article are defined in "Abbreviations" and "Notation." The terms "optimization" and "programming" are used interchangeably, courtesy of linear optimization historically being used for planning military operations [80]. Similarly, the term "nonlinear programming" means, more precisely, "nonconvex programming." Convexity is now known to be the true separator of efficient

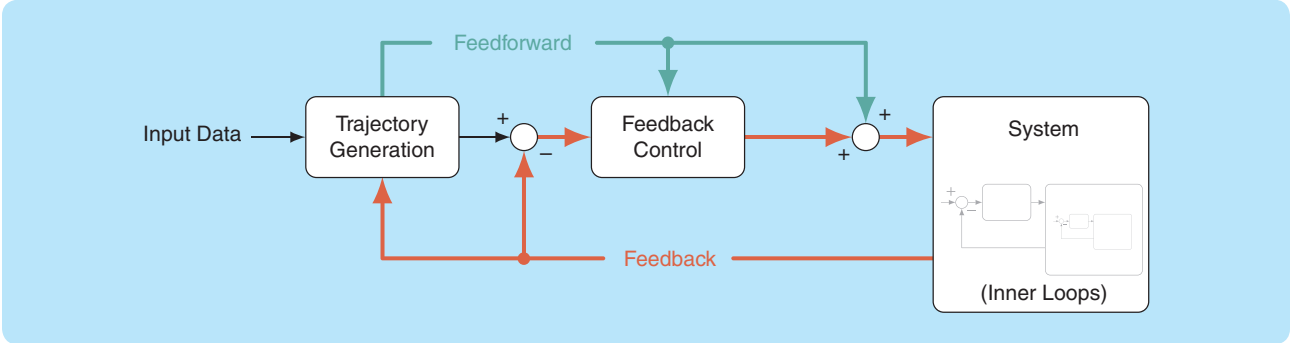


FIGURE 1 A typical control architecture consists of trajectory generation and feedback control elements. This article discusses algorithms for trajectory generation, which traditionally provides reference and feedforward control signals. By repeatedly generating new trajectories, a feedback action is created that can itself be used for control. Repeated trajectory generation for feedback control underlies the theory of model predictive control.

algorithms. However, this discovery came only after linear programming had already established itself as the dominant class of problems that can be efficiently solved via the Simplex method [30].

To complement the tutorial nature of this article, the numerical instances in the "Application Examples" section are accompanied by open source code linked in Figure 2. This software, called the *SCP Toolbox*, provides a parser interface to SCvx and GuSTO that is similar to other popular optimization tools such as CVX [248], [249]. The toolbox allows the reader to quickly get started on solving his or her own problems by using the methods developed in this article. The code is written in Julia because the programming language is simple to read, like Python, and can be as fast as C/C++ [81]. By downloading and running the code, the reader can recreate the exact plots seen in this article.

CONVEX OPTIMIZATION BACKGROUND

Convex optimization seeks to minimize a convex objective function while satisfying a set of convex constraints. The technique is expressive enough to capture many trajectory generation and control applications, and it is appealing due to the availability of solution algorithms with the following properties [26], [29]:

- A globally optimal solution is found if a feasible solution exists.
- A certificate of infeasibility is provided when a feasible solution does not exist.
- The runtime complexity is polynomial in the problem size.
- The algorithms can self-initialize, eliminating the need for an expert initial guess.

These properties are inherited by trajectory generation and control algorithms that use convex optimization. This makes convex programming safer and faster than other optimization methods for autonomous applications.

To appreciate what makes an optimization problem convex, let us introduce some basic definitions here and refer to [26] and [82] for further details. Two fundamental objects must be considered: a convex function and convex set. For reference, Figure 3 illustrates a notional convex set and function. By definition, $\mathcal{C} \subseteq \mathbb{R}^n$ is a convex set if and

Abbreviations

LCvx:	lossless convexification
LTV:	linear time varying
SCP:	sequential convex programming
SQP:	sequential quadratic programming
QP:	quadratic program
SOCP:	second-order cone program
SDP:	semidefinite program
NLP:	nonlinear (nonconvex) program
IPM:	interior point method
MPC:	model predictive control
ISS:	International Space Station
KKT:	Karush–Kuhn–Tucker
ODE:	ordinary differential equation
ZOH:	zeroth-order hold
FOH:	first-order hold
DoF:	degree of freedom
Programming:	Optimization

Notation

M	Matrices are capitalized letters.
z	This represents a scalar or vector variable.
$\mathcal{S}$	Sets are calligraphic capitalized letters.
(a, b, c)	This represents a concatenation of column or row vectors.
e_i	This is the ith standard basis vector.
$I_n \in \mathbb{R}^{n\times n}$	This represents the identity matrix.
$v^\times$	This is the skew-symmetric matrix representation of a cross product.
$\text{diag}(a, B, \ldots)$	This represents a (block) diagonal matrix from scalars and/or matrices.
$\text{dom}\, f$	This indicates the domain of a function.
$\nabla_x f$	This is the gradient vector or Jacobian matrix of f with respect to x.
$f[t]$	This is shorthand for a function evaluated at a particular time [for example, $f(t, x(t), u(t))$].
$\mathbb{R}_+, \mathbb{R}_{++}$	These indicate nonnegative and positive real numbers, respectively.
$\|x\|_{2+}$	This is the two-norm squared; it is the same as $x^\top x$.

github.com/UW-ACL/SCPToolbox.jl/tree/csm

FIGURE 2 The complete implementation source code for the numerical examples at the end of this article can be found in the GitHub repository of our open source sequential convex programming (SCP) trajectory generation tool, called the *SCP Toolbox*. The master branch provides even more algorithms and examples that are not covered here.

only if it contains the line segment connecting any two of its points:

$$x, y \in C \Rightarrow [x, y]_\theta \in C \tag{2}$$

for all $\theta \in [0, 1]$, where $[x, y]_\theta \triangleq \theta x + (1-\theta)y$. An important property is that convexity is preserved under set intersection. This allows us to build complicated convex sets by intersecting simpler sets. By replacing the word "sets" with "constraints," it can readily be appreciated how this property plays into modeling trajectory generation problems using convex optimization.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if dom f is a convex set and f lies below the line segment connecting any two of its points:

$$x, y \in \text{dom } f \Rightarrow f([x, y]_\theta) \leq [f(x), f(y)]_\theta \tag{3}$$

for all $\theta \in [0, 1]$. A convex optimization problem is simply the minimization of a convex function subject to a number of convex constraints that act to restrict the search space:

$$\max_{x \in \mathbb{R}^n} f_0(x), \tag{4a}$$

$$\text{s.t. } f_i(x) \leq 0, \quad i = 1, \ldots, n_{\text{ineq}}, \tag{4b}$$

$$g_i(x) = 0, \quad i = 1, \ldots, n_{\text{eq}}, \tag{4c}$$

where $f_0 : \mathbb{R}^n \to \mathbb{R}$ is a convex *cost* function, $f_i : \mathbb{R}^n \to \mathbb{R}$ are convex inequality constraints, and $g_i : \mathbb{R}^n \to \mathbb{R}$ are affine equality constraints. The problem contains n_{ineq} inequality and n_{eq} equality constraints. Note that the equality constraints must be affine, which means that each function g_i is a linear expression in x plus a constant offset. The equations of motion are equality constraints. Therefore, basic convex optimization restricts the dynamics to be affine [that is, linear time varying (LTV), at most]. Handling nonlinear dynamics will be a major topic of discussion throughout this article.

Each constraint defines a convex set so that, together, (4b) and (4c) form a convex, feasible set of values that the decision variable x may take. To explicitly connect this discussion back to the generic convex set introduced in (2), the feasible set can be written as

$$C = \{x \in \mathbb{R}^n : f_i(x) \leq 0, i = 1, \ldots, n_{\text{ineq}}, \; g_i(x) = 0, i = 1, \ldots, n_{\text{eq}}\}. \tag{5}$$

A fundamental consequence of convexity is that any local minimum of a convex function is a global minimum [26, Sec. 4.2.2]. More generally, convex functions come with a plethora of properties that allow algorithm designers to obtain global information about function behavior from local measurements. For example, a differentiable convex function is globally lower bounded by its local first-order approximation [26]. Thus, convexity may be viewed as a highly beneficial assumption about function behavior that enables efficient algorithm design. Indeed, a landmark discovery of the 20th century was that it is convexity, not linearity, that separates "hard" and "easy" problems [30].

For practitioners, the utility of convex optimization stems not so much from the ability to find the global minimum but rather the ability to find it (or, indeed, any other feasible solution) *quickly*. The field of numerical convex optimization was invigorated by the interior point method (IPM) family of optimization algorithms introduced in 1984 by Karmarkar [83]. Today, convex optimization problems can be solved by primal–dual IPMs in a few tens of iterations [26], [31]. Roughly speaking, IPMs can solve most convex trajectory generation problems in less than 1 s [9], [37], [63]. In technical parlance, IPMs have a favorable polynomial problem complexity: the number of iterations required to solve the problem to a given tolerance grows polynomially in the number of constraints $n_{\text{ineq}} + n_{\text{eq}}$. With some further assumptions, it is even possible to provide an upper bound on the number of iterations [35], [84]. Further details about convex optimization algorithms can be found in [29, Chs. 14 and 19]. Throughout this article, the goal will be to leverage existing convex problem solvers to create higher-level frameworks for the solution of trajectory generation problems.

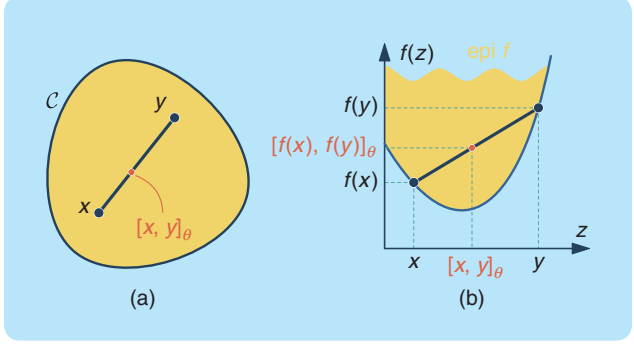


FIGURE 3 A notional convex set and convex function. In both cases, the variable $\theta \in [0, 1]$ generates a line segment between two points. The epigraph epi $f \subseteq \mathbb{R}^n \times \mathbb{R}$ is the set of points that lie above the function and itself defines a convex set. (a) A convex set contains all line segments connecting its points. (b) A convex function lies below all line segments connecting its points.

LOSSLESS CONVEXIFICATION

LCvx is a modeling technique that solves nonconvex optimal control problems through a convex relaxation. In this method, Pontryagin's maximum principle [11] is used to show that a convex relaxation of a nonconvex problem finds the globally optimal solution to the original problem (hence the method's name). To date, the method has been extended as far as relaxing certain classes of nonconvex control constraints, such as an input norm lower bound (see "Convex Relaxation of an Input Lower Bound") and a nonconvex pointing constraint (see "Convex Relaxation of an Input Pointing Constraint").

The LCvx method has been shown to work for a large class of state-constrained optimal control problems; however, a working assumption is that state constraints are

Convex Relaxation of an Input Lower Bound

A typical rocket engine's thrust is upper bounded by the engine's performance and lower bounded by combustion instability issues for small thrusts. Thus, (6c) can be written as

$$\rho_{\min} \le \|u\|_2 \le \rho_{\max}, \tag{S1}$$

so that $g_0(u) = g_1(u) \triangleq \|u\|_2$. The relaxation (7c) and (7d) then becomes

$$\rho_{\min} \le \sigma, \ \|u\|_2 \le \min(\sigma, \rho_{\max}). \tag{S2}$$

Figure S1 shows how for $u \in \mathbb{R}^2$, going from (S1) to (S2) lifts a nonconvex annulus to a convex volume $\mathcal{V}$:

$$\mathcal{V} \triangleq \{(u,\sigma) \in \mathbb{R}^3 : \rho_{\min} \le \sigma, \|u\|_2 \le \min(\sigma, \rho_{\max})\}. \tag{S3}$$

The drawback is that inputs that were not feasible for (S1) become feasible for (S2), in particular, any u for which $\|u\|_2 < \rho_{\min}$ is now feasible. However, for the special case of $(u,\sigma) \in \partial_{\text{LCvx}}\mathcal{V}$, where

$$\partial_{\text{LCvx}}\mathcal{V} \triangleq \{(u,\sigma) \in \mathbb{R}^3 : \rho_{\min} \le \sigma, \|u\|_2 = \min(\sigma, \rho_{\max})\}, \tag{S4}$$

the input u is feasible. The goal of lossless convexification is to show that this occurs at the global optimum of (7); in other words, $(u^*, \sigma^*) \in \partial_{\text{LCvx}}\mathcal{V}$. Theorem 1 guarantees this.

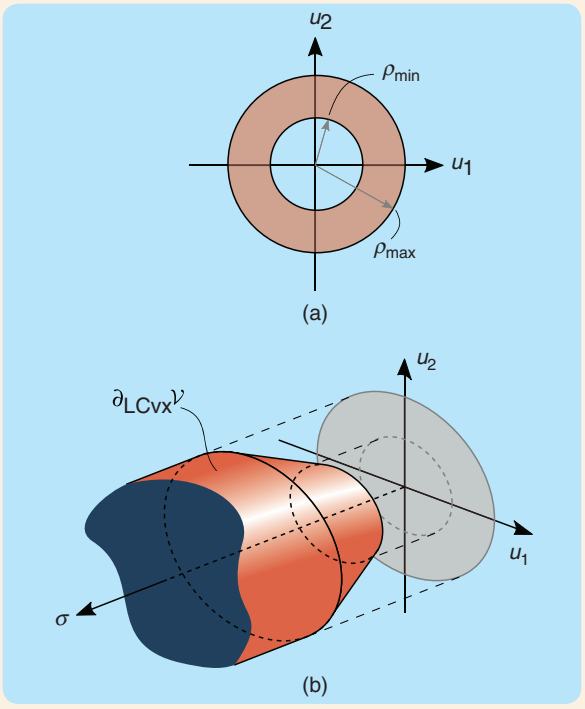


FIGURE S1 The relaxation of the nonconvex input constraint set (S1) in (a) to the convex set (S2) in (b) by using a slack input σ. If the optimal input $(u^*, \sigma^*) \in \partial_{\text{LCvx}}\mathcal{V}$ (S4), then u^* is feasible for the nonconvex constraint (S1).

Convex Relaxation of an Input Pointing Constraint

Practical mechanical systems, such as Segways and rockets, may have a constraint on their tilt angle away from an upright orientation. To model such cases, the inequality (9d) can be specialized to an input pointing constraint. By choosing $\hat{n}_g = \cos(\theta_{\max})$ and $g(u) = \|u\|_2$, the constraint becomes

$$\hat{n}_u^\top u \ge \|u\|_2 \cos(\theta_{\max}), \tag{S5}$$

which constrains u to be no more than $\theta_{\max}$ radians from a nominal pointing direction $\hat{n}_u$. The relaxed constraint (10e) becomes

$$\hat{n}_u^\top u \ge \sigma \cos(\theta_{\max}), \tag{S6}$$

which is a half-space constraint and thus convex, even for $\theta_{\max} > \pi/2$, when (S5) becomes nonconvex.

Figure S2 shows how for $u \in \mathbb{R}^2$, relaxing (S5) to (S6) lifts an obtuse "pie slice" to a convex half-space $\mathcal{V}$:

$$\mathcal{V} \triangleq \{(u,\sigma) \in \mathbb{R}^3 : \hat{n}_u^\top u \ge \sigma \cos(\theta_{\max}), \ \|u\|_2 \le \sigma\}. \tag{S7}$$

The drawback is that inputs that were not feasible for (S5) become feasible for (S6), in particular, low-magnitude inputs for which $\hat{n}_u^\top u < \|u\|_2 \cos(\theta_{\max})$ become feasible. However, if $(u,\sigma) \in \partial_{\text{LCvx}}\mathcal{V}$,

$$\partial_{\text{LCvx}}\mathcal{V} \triangleq \{(u,\sigma) \in \mathbb{R}^3 : \hat{n}_u^\top u \ge \sigma \cos(\theta_{\max}), \ \|u\|_2 = \sigma\}, \tag{S8}$$

then the input u satisfies the original constraint (S5). The goal of lossless convexification is to show that this occurs at the global optimum; in other words, $(u^*, \sigma^*) \in \partial_{\text{LCvx}}\mathcal{V}$. Theorem 2 guarantees this.

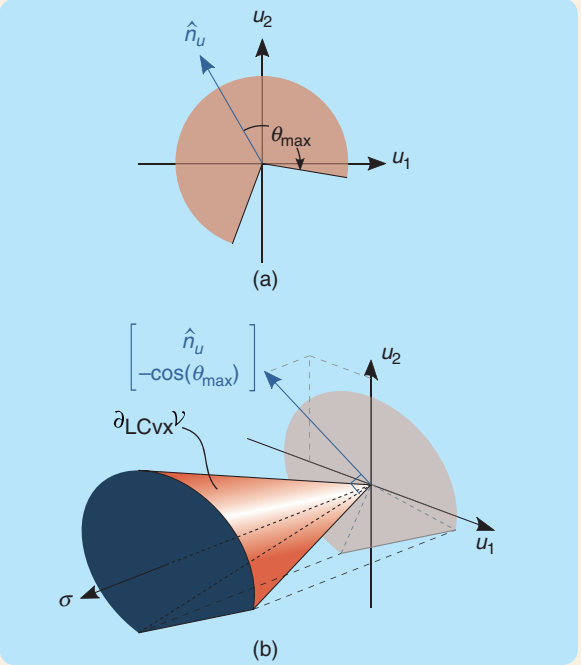


FIGURE S2 The relaxation of (a) the nonconvex input set (S5) to a convex set (S6) via an intersection with (b) a half-space in the lifted (u,σ) space. If the optimal input $(u^*, \sigma^*) \in \partial_{\text{LCvx}}\mathcal{V}$, then u^* is feasible for the nonconvex constraint (S5).

convex. The lossless relaxation of nonconvex state constraints remains under active research, and some related results are available [46] (which will be covered in this section). As the reader goes through this part of the article, note that the main concerns of LCvx are as follows:

- to find a convex lifting of the feasible input set
- to show that the optimal input of the lifted problem projects back to a feasible (and, in fact, optimal) input of the original nonlifted problem.

Figure 4 chronicles the development history of LCvx. The aim of this part of the article is to provide a tutorial overview of the key results, so theoretical proofs are omitted in favor of a more practical and action-oriented description. Ultimately, the aim is for the reader to come away with a clear understanding of how LCvx can be applied to his or her own problems. Table 1 summarizes and illustrates the LCvx results presented in this part of the article. Looking at the table from left to right provides a road map for this section, which presents the LCvx results in the same order. Our discussion begins by introducing LCvx for the input norm lower bound and pointing nonconvexities using a baseline problem with no state constraints. Then, the method is extended to handle affine and quadratic state constraints, followed by general convex state constraints. It is then described how the LCvx method can also handle a class of dynamical systems with nonlinear dynamics. For more general applications, embedding LCvx into nonlinear optimization algorithms is also discussed. At the very end of this part of the article, some of the newest LCvx results from the past year are covered, and a toy example is provided to demonstrate how LCvx can be used in practice.

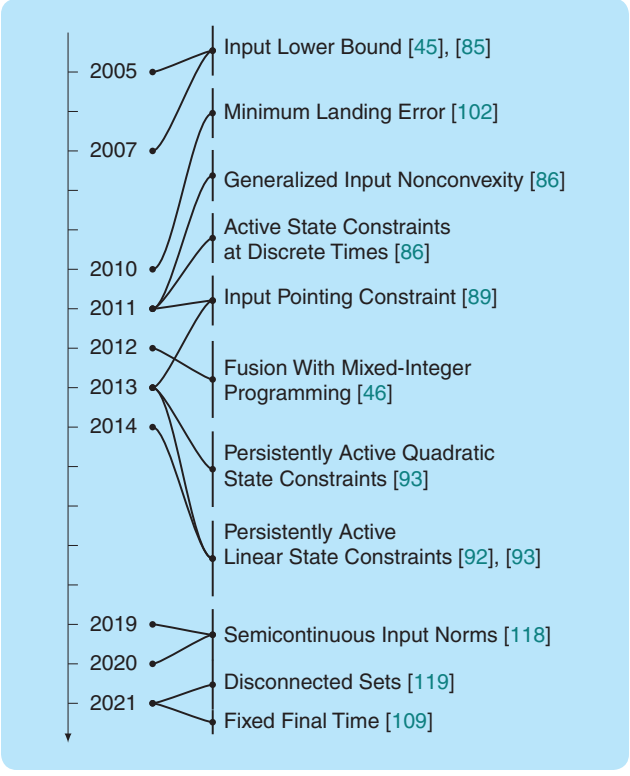


FIGURE 4 The chronology of lossless convexification theory development. Note the progression from state-unconstrained problems to those with progressively more general state constraints and (finally) to problems that contain integer variables.

No State Constraints

It is natural to begin by stating perhaps the simplest optimal control problem for which an LCvx result is available. Its salient features are a distinct absence of state constraints (except for the boundary conditions), and its only source of nonconvexity is a lower bound on the input given by (6c). A detailed description of LCvx for this problem may be found in [45], [85], and [86]:

$$\min_{u,t_f} \; m(t_f, x(t_f)) + \zeta \int_0^{t_f} \ell(g_1(u(t)))\,\mathrm{d}t, \tag{6a}$$

$$\text{s.t. } \dot{x}(t) = A(t)x(t) + B(t)u(t) + E(t)w(t), \tag{6b}$$

$$\rho_{\min} \le g_1(u(t)), \quad g_0(u(t)) \le \rho_{\max}, \tag{6c}$$

$$x(0) = x_0, \quad b(t_f, x(t_f)) = 0. \tag{6d}$$

In (6), $t_f > 0$ is the terminal time; $x(\cdot) \in \mathbb{R}^n$ is the state trajectory; $u(\cdot) \in \mathbb{R}^m$ is the input trajectory; $w(\cdot) \in \mathbb{R}^p$ is an exogenous additive disturbance; $m : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ is a convex terminal cost; $\ell : \mathbb{R} \to \mathbb{R}$ is a convex and nondecreasing running cost modifier; $\zeta \in \{0, 1\}$ is a fixed, user-chosen parameter to toggle the running cost; g_0, $g_1 : \mathbb{R}^m \to \mathbb{R}_+$ are convex functions; $\rho_{\min} > 0$ and $\rho_{\max} > \rho_{\min}$ are user-chosen bounds; and $b : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^{n_b}$ is an affine terminal constraint function. Note that the dynamics in (6) define an LTV system. For the LCvx discussion, the following two assumptions are made about the problem data.

Assumption 1

If any part of the terminal state is constrained, then the Jacobian $\nabla_x b[t_f] \in \mathbb{R}^{n_b \times n}$ is full row rank; that is, rank $\nabla_x b[t_f] = n_b$. This implies that the terminal state is not overconstrained.

Assumption 2

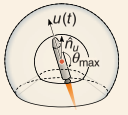
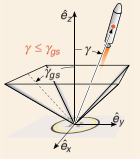
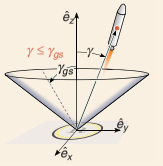
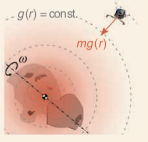
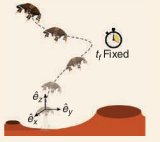
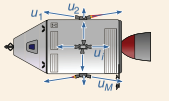
The running cost is positive definite; in other words, $\ell(z) > 0$ for all $z \neq 0$.

When faced with a nonconvex problem such as (6), an engineer has two choices. Either devise a nonlinear optimization algorithm or solve a simpler problem that is convex. The mantra of LCvx is to take the latter approach by "relaxing" the problem until it is convex. In the case of (6), the following relaxation is proposed, which introduces a new variable $\sigma(\cdot) \in \mathbb{R}$, called the *slack input*:

$$\min_{\sigma,u,t_f} \; m(t_f, x(t_f)) + \zeta \int_0^{t_f} \ell(\sigma(t))\,\mathrm{d}t, \tag{7a}$$

$$s.t. \; \dot{x}(t) = A(t)x(t) + B(t)u(t) + E(t)w(t), \tag{7b}$$

TABLE 1 A summary of state-of-the-art lossless convexification (LCvx) results. Each column displays elements of the original nonconvex optimal control problem that can be converted and solved as a convex problem using LCvx. The notation for each element is explained in the corresponding problem's section of the article.

	No State Constraints	Input Pointing Constraint	Affine State Constraints	Quadratic State Constraints	General Convex State Constraints	Nonlinear Dynamics	Fixed Final Time Problems	Hybrid System Problems
Terminal cost	$m(t_f, x(t_f))$	$m(t_f, x(t_f))$	$m(x(t_f))$	$m(x(t_f))$	$m(x(t_f))$	$m(t_f, x(t_f))$		
Running cost	$\zeta \ell(g_1(u(t)))$	$\ell(g(u(t)))$	$\zeta \ell(g_1(u(t)))$	$\zeta \ell(g_1(u(t)))$	$\zeta \ell(g_1(u(t)))$	$\zeta \ell(g_1(u(t)))$	$\ell(g(u(t)))$	$\sum_{i=1}^{M} \| u_i(t) \|_2$
Dynamics $\dot{x}(t) = \cdots$	$A(t)x(t) + B(t)u(t) + E(t)w(t)$	$A(t)x(t) + B(t)u(t) + E(t)w(t)$	$Ax(t) + Bu(t) + Ew$	$\dot{x}_1(t) = Ax_2(t)$ $\dot{x}_2(t) = Bu(t) + w$	$Ax(t) + Bu(t) + Ew$	$f(t, x(t), u(t), g(u(t)))$	$Ax(t) + Bu(t)$	$Ax(t) + B\Sigma_{i=1}^{M} u_i(t)$
Input norm bounds	$g_1(u(t)) \geq \rho_{\min}$ $g_0(u(t)) \leq \rho_{\max}$	$g(u(t)) \geq \rho_{\min}$ $g(u(t)) \leq \rho_{\max}$	$g_1(u(t)) \geq \rho_{\min}$ $g_0(u(t)) \leq \rho_{\max}$	$g_1(u(t)) \geq \rho_{\min}$ $g_0(u(t)) \leq \rho_{\max}$	$g_1(u(t)) \geq \rho_{\min}$ $g_0(u(t)) \leq \rho_{\max}$	$g(u(t)) \geq \rho_{\min}$ $g(u(t)) \leq \rho_{\max}$	$g(u(t)) \geq \rho_{\min}$ $g(u(t)) \leq \rho_{\max}$	$\| u_i(t) \|_2 \geq \nu_i(t)\rho_{\min,i}$ $\| u_i(t) \|_2 \leq \nu_i(t)\rho_{\max,i}$ $\gamma_i(t) \in \{0, 1\}$
Other input constraints		$\hat{n}_u^T u(t) \geq \hat{n}_g g(u(t))$	$Cu(t) \leq c$					$C_i u_i(t) \leq 0$
Other state constraints			$Hx(t) \leq h$	$x_2(t)^\top H x_2(t) \leq 1$	$x(t) \in \mathcal{X}$			
Boundary conditions	$x(0) = x_0$ $b(t_f, x(t_f)) = 0$	$x(0) = x_0$ $b(t_f, x(t_f)) = 0$	$x(0) = x_0$ $b(x(t_f)) = 0$	$x_1(0) = x_{1,0}$ $x_2(0) = x_{2,0}$ $b(x(t_f)) = 0$	$x(0) = x_0$ $b(x(t_f)) = 0$	$x(0) = x_0$ $b(t_f x(t_f)) = 0$	$x(0) = x_0$ $x(t_f) = x_f$	$x(0) = x_0$ $b(x(t_f)) = x_f$
Illustration*								

*The illustrations show a unique feature of the problem in the corresponding column. They do not show all the constraints.

$$\rho_{\min} \le \sigma(t), \quad g_0(u(t)) \le \rho_{\max}, \tag{7c}$$
$$g_1(u(t)) \le \sigma(t), \tag{7d}$$
$$x(0) = x_0, \quad b(t_f, x(t_f)) = 0. \tag{7e}$$

The relaxation of the nonconvex input constraint (6c) to the convex constraints (7c) and (7d) is illustrated in "Convex Relaxation of an Input Lower Bound" for the case of a throttleable rocket engine. Note that if (7d) is replaced with equality, then (7) is equivalent to (6). Indeed, the entire goal of LCvx is to *prove* that (7d) holds with equality at the globally optimal solution of (7). Because of the clear importance of constraint (7d) to the LCvx method, it is called the *LCvx equality constraint*. This special constraint will be highlighted in red in all subsequent LCvx optimization problems. Now, consider the following set of conditions, which arise naturally when using the maximum principle to prove LCvx. The theoretical details are provided in [86, Th. 2].

Condition 1

The pair $\{A(\cdot), B(\cdot)\}$ must be totally controllable. This means that any initial state can be transferred to any final state by a bounded control trajectory in any finite time interval $[0, t_f]$ [46], [87]. If the system is time invariant, this is equivalent to $\{A, B\}$ being controllable, and it can be verified by checking that the controllability matrix is full rank or, more robustly, via the Popov–Belevitch–Hautus (PBH) test [88].

Condition 2

Define the quantities

$$m_{\text{LCvx}} = \begin{bmatrix} \nabla_x m[t_f] \\ \nabla_t m[t_f] + \zeta \ell(\sigma(t_f)) \end{bmatrix} \in \mathbb{R}^{n+1}, \tag{8a}$$
$$B_{\text{LCvx}} = \begin{bmatrix} \nabla_x b[t_f]^\top \\ \nabla_t b[t_f]^\top \end{bmatrix} \in \mathbb{R}^{(n+1)\times n_b}. \tag{8b}$$

The vector m_{LCvx} and columns of B_{LCvx} must be linearly independent.

Theorem 1 can now be stated, which is the main LCvx result for (6). The practical implication of Theorem 1 is that the solution of (6) can be found in polynomial time by solving (7) instead.

Theorem 1

The solution of (7) is globally optimal for (6) if Condition 1 and Condition 2 hold.

Input Pointing Constraint

There is a partial generalization of Theorem 1. On the one hand, restrict (6) to the choices $\zeta = 1$ and $g_0 = g_1 \triangleq g$. On the other hand, introduce a new pointing-like input constraint (9d). The quantities $\hat{n}_u \in \mathbb{R}^m$ and $\hat{n}_g \in \mathbb{R}$ are user-chosen parameters. The new problem takes the following form:

$$\min_{u, t_f} \; m(t_f, x(t_f)) + \int_0^{t_f} \ell(g(u(t)))\,\mathrm{d}t, \tag{9a}$$
$$\text{s.t. } \dot{x}(t) = A(t)x(t) + B(t)u(t) + E(t)w(t), \tag{9b}$$
$$\rho_{\min} \le g(u(t)) \le \rho_{\max}, \tag{9c}$$
$$\hat{n}_u^\top u(t) \ge \hat{n}_g g(u(t)), \tag{9d}$$
$$x(0) = x_0, \quad b(t_f, x(t_f)) = 0. \tag{9e}$$

This problem is a partial generalization of (6) in the sense that the input constraints are more general due to (9d); however, there must be a running cost, and the form of nonconvexity (9c) is more restricted.

For example, an airborne vehicle's tilt angle can be constrained using (9d). This constraint, however, is nonconvex for $\hat{n}_g < 0$. This nonconvexity, along with the typical nonconvexity of the lower bound in (9c), is addressed by solving the following relaxation of the original problem:

$$\min_{\sigma, u, t_f} \; m(t_f, x(t_f)) + \int_0^{t_f} \ell(\sigma(t))\,\mathrm{d}t, \tag{10a}$$
$$\text{s.t. } \dot{x}(t) = A(t)x(t) + B(t)u(t) + E(t)w(t), \tag{10b}$$
$$\rho_{\min} \le \sigma(t) \le \rho_{\max}, \tag{10c}$$
$$g(u(t)) \le \sigma(t), \tag{10d}$$
$$\hat{n}_u^\top u(t) \ge \sigma(t)\hat{n}_g, \tag{10e}$$
$$x(0) = x_0, \quad b(t_f, x(t_f)) = 0. \tag{10f}$$

As in (7), a slack input $\sigma(\cdot) \in \mathbb{R}$ is introduced to strategically remove nonconvexity. Note again the appearance of the LCvx equality constraint (10d). Meanwhile, the relaxation of (9d) to (10e) corresponds to a half-space input constraint in the $(u, \sigma) \in \mathbb{R}^{m+1}$ space. A geometric intuition about the relaxation is illustrated in "Convex Relaxation of an Input Pointing Constraint" for a typical vehicle tilt constraint. LCvx can again be shown under an extra Condition 3, yielding Theorem 2. Theoretical details are provided in [89] and [90].

Condition 3

Let $N \in \mathbb{R}^{m\times(m-1)}$ be a matrix whose columns span the null space of $\hat{n}_u$ in (9d). The pair $\{A(\cdot), B(\cdot)N\}$ must be totally controllable.

Theorem 2

The solution of (10) is globally optimal for (9) if Conditions 1, 2, and 3 hold.

Affine State Constraints

The logical next step after Theorem 1 and Theorem 2 is to ask whether (6) can incorporate state constraints. It was already shown in the context of the input pointing constraint (9d) that the addition of a convex input constraint can change the LCvx guarantee. In particular, Theorem 2 is subject to more conditions than Theorem 1. The situation is even more intricate in the case of state constraints since Pontryagin's maximum principle statement is more

complicated for problems with state constraints [91]. Nevertheless, an LCvx guarantee can be made in the presence of state constraints by introducing a fairly mild set of extra conditions.

Affine inequality constraints are the simplest class of state constraints that can be handled in LCvx. The results presented in this section originate from [92]–[94]. The nonconvex statement of the original problem is a close relative of (6):

$$\min_{u,t_f} \; m(x(t_f)) + \zeta \int_0^{t_f} \ell(g_1(u(t)))\,\mathrm{d}t, \tag{11a}$$

$$\text{s.t. } \dot{x}(t) = Ax(t) + Bu(t) + Ew, \tag{11b}$$

$$\rho_{\min} \le g_1(u(t)), \quad g_0(u(t)) \le \rho_{\max}, \tag{11c}$$

$$Cu(t) \le c, \tag{11d}$$

$$Hx(t) \le h, \tag{11e}$$

$$x(0) = x_0, \quad b(x(t_f)) = 0. \tag{11f}$$

First, note that (11) is *autonomous*; that is, the terminal cost in (11a), the dynamics (11b), and the boundary constraint (11f) are all independent of time. A limited form of time variance can still be included by introducing an additional time integrator state whose dynamics are $\dot{z}(t) = 1$. The limitation here is that time variance must not introduce nonconvexity in the cost, dynamics, and terminal constraint (11f). The matrix of facet normals $H \in \mathbb{R}^{n_h \times n}$ and the vector of facet offsets $h \in \mathbb{R}^{n_h}$ define a new polytopic (affine) state constraint set. A practical use case for this constraint is described in "Landing Glideslope as an Affine State Constraint." Similarly, $C \in \mathbb{R}^{n_c \times m}$ and $c \in \mathbb{R}^{n_c}$ define a new polytopic subset of the input constraint set, as illustrated in "Using Half Spaces to Further Constrain the Input Set."

Note the following convex relaxation of (11), which takes the familiar form of (7):

$$\min_{\sigma,u,t_f} \; m(x(t_f)) + \zeta \int_0^{t_f} \ell(\sigma(t))\,\mathrm{d}t, \tag{12a}$$

$$\text{s.t. } \dot{x}(t) = Ax(t) + Bu(t) + Ew, \tag{12b}$$

$$\rho_{\min} \le \sigma(t), \quad g_0(u(t)) \le \rho_{\max}, \tag{12c}$$

$$g_1(u(t)) \le \sigma(t), \tag{12d}$$

$$Cu(t) \le c, \tag{12e}$$

$$Hx(t) \le h, \tag{12f}$$

$$x(0) = x_0, \quad b(x(t_f)) = 0. \tag{12g}$$

To guarantee LCvx for this convex relaxation, Condition 1 can be modified to handle the new state and input constraints (11d) and (11e). To this end, the following notion of cyclic coordinates from mechanics is used [95].

Definition 1

For a dynamical system $\dot{x} = f(x)$ with state $x \in \mathbb{R}^n$, any components of x that do not appear explicitly in $f(\cdot)$ are said to be *cyclic coordinates*. Without a loss of generality, the state can be decomposed as

$$x = \begin{bmatrix} x_c \\ x_{nc} \end{bmatrix}, \tag{13}$$

Landing Glideslope as an Affine State Constraint

A typical planetary landing problem may include a glideslope constraint to ensure sufficient elevation during approach that prevents the spacecraft from colliding with nearby terrain [45]. Letting $\hat{e}_3 \in \mathbb{R}^3$ represent the local vertical unit vector at the landing site, the glideslope requirement can be expressed as a convex, second-order cone constraint:

$$\hat{e}_3^{\mathsf{T}} x \ge \|x\|_2 \cos(\gamma_{\text{gs}}), \tag{S9}$$

where $\gamma_{\text{gs}} \in [0, \pi/2]$ is the glideslope angle, that is, the maximum angle that the spacecraft position vector is allowed to make with the local vertical. As illustrated in Figure S3, (S9) can be approximated as an intersection of four half spaces with the following outward normal vectors:

$$\hat{n}_1 \triangleq \begin{bmatrix} \cos(\gamma_{\text{gs}}) \\ 0 \\ -\sin(\gamma_{\text{gs}}) \end{bmatrix}, \quad \hat{n}_2 \triangleq \begin{bmatrix} 0 \\ \cos(\gamma_{\text{gs}}) \\ -\sin(\gamma_{\text{gs}}) \end{bmatrix}, \tag{S10a}$$

$$\hat{n}_3 \triangleq \begin{bmatrix} -\cos(\gamma_{\text{gs}}) \\ 0 \\ -\sin(\gamma_{\text{gs}}) \end{bmatrix}, \quad \hat{n}_4 \triangleq \begin{bmatrix} 0 \\ -\cos(\gamma_{\text{gs}}) \\ -\sin(\gamma_{\text{gs}}) \end{bmatrix}. \tag{S10b}$$

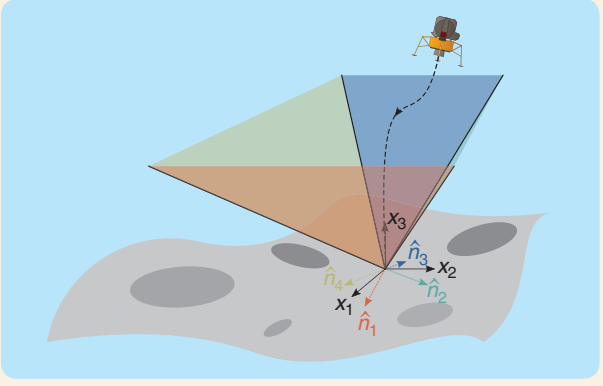


FIGURE S3 A spacecraft planetary landing glideslope cone can be approximated as an affine state constraint (11e). By adding more facets, a cone with circular cross sections can be approximated to arbitrary precision.

Thus, (S9) can be written in the form (11e) by setting

$$H = \begin{bmatrix} \hat{n}_1^{\mathsf{T}} \\ \hat{n}_2^{\mathsf{T}} \\ \hat{n}_3^{\mathsf{T}} \\ \hat{n}_4^{\mathsf{T}} \end{bmatrix}, \; h = 0 \in \mathbb{R}^4. \tag{S11}$$

Using Half Spaces to Further Constrain the Input Set

The input constraint set defined by (6c) generally lacks the ability to describe constraints on individual input components and combinations thereof. To enhance the descriptiveness of the input constraint set, half-space constraints in (11d) may be used. This is illustrated in Figure S4, where $g_0(\cdot) = \|\cdot\|_1$, $g_1(\cdot) = \|\cdot\|_2$, and

$$C = \begin{bmatrix} \hat{n}_1^\top \\ \hat{n}_2^\top \end{bmatrix}, \; c = \begin{bmatrix} c_1 \\ 0 \end{bmatrix}. \tag{S12}$$

For example, u_1 and u_2 may describe the concentrations of two chemical reactants—hydrochloric acid and ammonia—to produce ammonium chloride. Then, the first affine constraint may describe the maximum concentration of ammonia, while the second affine constraint may describe an interdependency in the concentrations of both reactants. Meanwhile, the now familiar constraint (11c) describes the joint reactant concentration upper and lower bounds.

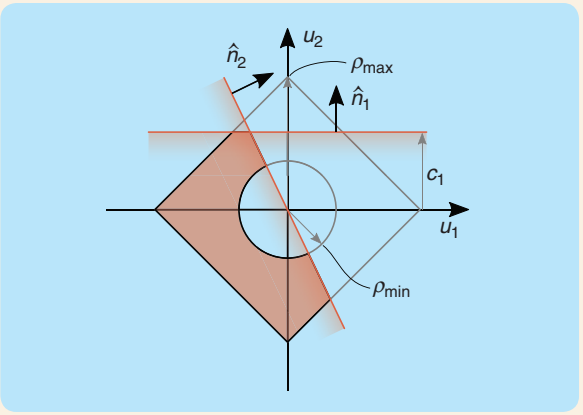


FIGURE S4 The half-space constraints from (11d) can be used to select only portions of the nonconvex input set defined by (11c). The remaining feasible input set is filled in red.

where $x_c \in \mathbb{R}^{n_c}$ are the cyclic and $x_{nc} \in \mathbb{R}^{n_{nc}}$ are the noncyclic coordinates such that $n_c + n_{nc} = n$. One can then write $\dot{x} = f(x_{nc})$.

Many mechanical systems have cyclic coordinates. For example, quadrotor drone and fixed-wing aircraft dynamics do not depend on the position [96]. Satellite dynamics in a circular low-Earth orbit, frequently approximated with the Clohessy–Wiltshire–Hill equations [97], do not depend on the true anomaly angle that locates the spacecraft along the orbit.

Using Definition 1, a *cyclic transformation* $\Psi_c : \mathbb{R}^n \to \mathbb{R}^n$ is any mapping from the state space to itself that translates the state vector along the cyclic coordinates. In other words, without loss of generality, assuming that the state is given by (13),

$$\Psi_c(x) = \begin{bmatrix} x_c + \Delta x_c \\ x_{nc} \end{bmatrix} \tag{14}$$

for some translation $\Delta x_c \in \mathbb{R}^{n_c}$. The following condition must then hold for the polytopic state constraint (12f).

Condition 4

Let $H_i^\top$ be the ith row of H in (12f). The set $\mathcal{H}_i = \{x \in \mathbb{R}^n : H_i^\top x = h_i\}$ thus corresponds to the ith facet of the constraint polytope. For each facet where $h_i \neq 0$, there must exist a cyclic transformation Ψ_c such that

$$H_i^\top \Psi_c(x) = 0. \tag{15}$$

To visualize the implication of Condition 4, consider the case of the landing glideslope constraint from "Landing Glideslope as an Affine State Constraint." Because the position of a spacecraft in a constant gravity field is a cyclic coordinate, Condition 4 confirms the intuitive understanding that landing can be imposed at a coordinate other than the origin without compromising LCvx. Figure 5 provides an example.

According to linear systems theory [92], [98], [99], $x(t) \in \mathcal{H}_i$ can hold for a nonzero time interval (that is, the state "sticks" to a facet) if and only if there exists a triplet of matrices $\{F_i \in \mathbb{R}^{m \times n}, G_i \in \mathbb{R}^{m \times n_v}, \mathbb{R}_i \in \mathbb{R}^{m \times p}\}$ such that

$$u(t) = F_i x(t) + G_i v(t) + R_i w. \tag{16}$$

The "new" control input $v(t) \in \mathbb{R}^{n_v}$ effectively gets filtered through (16) to produce a control signal that maintains $x(\cdot)$ on the hyperplane $\mathcal{H}_i$. The situation is depicted in Figure 6 in a familiar block diagram form. While the matrix triplet is not unique, a valid triplet can be computed using

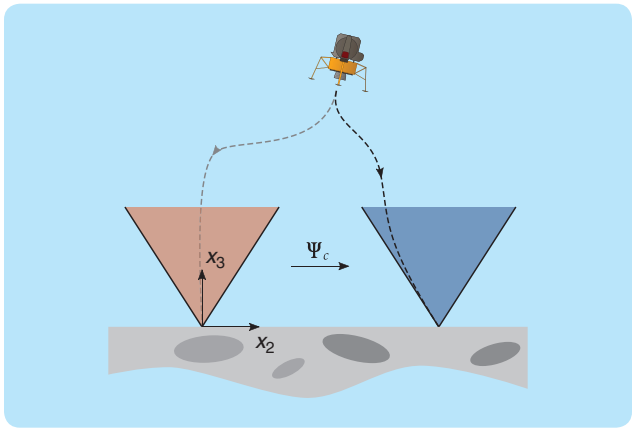


FIGURE 5 A landing glideslope constraint (shown in red, it is a side view of Figure S3) that undergoes a cyclic shift in position along the positive x_2-axis to arrive at a new landing location (in blue). Thanks to Condition 4, the lossless convexification guarantee continues to hold for the new glideslope constraint even though the new constraint facets are not subspaces.

standard linear algebra operations [92], [98]. The proof of LCvx for (11) was originally developed in [92] and [93]. The theory behind (16) is among the most abstract in all of LCvx, and a proof is not attempted here. The ultimate outcome of the proof is that the following condition must hold.

Condition 5

For each facet $\mathcal{H}_i \subseteq \mathbb{R}^n$ of the polytopic state constraint (12f), the following "dual" linear system has no transmission zeros:

$$\dot{\lambda}(t) = -(A + BF_i)^\top \lambda(t) - (CF_i)^\top \mu(t),$$
$$y(t) = (BG_i)^\top \lambda(t) + (CG_i)^\top \mu(t).$$

Transmission zeros are defined in [100, Sec. 4.5.1]. Roughly speaking, if there are no transmission zeros, then there cannot exist an initial condition $\lambda(0) \in \mathbb{R}^n$ and input trajectory $\mu(\cdot) \in \mathbb{R}^{n_c}$ such that $y(t) = 0$ for a nonzero time interval.

The conditions for when LCvx holds for (12) can now be stated. Note that the statement is very similar to Theorem 1. Indeed, the primary contribution of [92] and [93] was to introduce Condition 5 and show that LCvx holds by using a version of the maximum principle that includes state constraints [91], [101]. Meanwhile, the actual LCvx procedure does not change.

Theorem 3

The solution of (12) is globally optimal for (11) if Condition 2 and Condition 5 hold.

Quadratic State Constraints

The preceding section showed an LCvx result for state constraints that can be represented by the affine description (11e). The natural next question is whether LCvx extends to more complicated state constraints. A generalization of LCvx exists for quadratic state constraints if one can accept a slight restriction to the system dynamics. This result was originally presented in [94] and [99]. The nonconvex problem statement is:

$$\min_{u, t_f} \; m(x(t_f)) + \zeta \int_0^{t_f} \ell(g_1(u(t)))\,\mathrm{d}t, \tag{17a}$$
$$\text{s.t. } \dot{x}_1(t) = Ax_2(t), \tag{17b}$$
$$\dot{x}_2(t) = Bu(t) + w, \tag{17c}$$
$$\rho_{\min} \le g_1(u(t)), \quad g_0(u(t)) \le \rho_{\max}, \tag{17d}$$
$$x_2(t)^\top H x_2(t) \le 1, \tag{17e}$$
$$x_1(0) = x_{1,0}, \quad x_2(0) = x_{2,0}, \quad b(x(t_f)) = 0, \tag{17f}$$

where $(x_1, x_2) \in \mathbb{R}^n \times \mathbb{R}^n$ is the state that has been partitioned into two distinct parts, $u \in \mathbb{R}^n$ is an input of the same dimension, and the exogenous disturbance $w \in \mathbb{R}^n$ is some fixed constant. The matrix $H \in \mathbb{R}^{n \times n}$ is symmetric positive definite such that (17e) maintains the state in an ellipsoid. An example of such a constraint is illustrated in "Maximum Velocity as a Quadratic State Constraint."

Although the dynamics (17b) and (17c) are less general than (11b), they can still accommodate problems related to vehicle trajectory generation. In such problems, the vehicle is usually closely related to a double integrator system for which $A = I_n$ and $B = I_n$ such that x_1 is the position and x_2 is the velocity of the vehicle. The control u in this case is the acceleration. The following assumption further restricts the problem setup and is a consequence of the LCvx proof [93].

Assumption 3

The matrices A, B, and H in (11) are invertible. The functions $g_0(\cdot)$ and $g_1(\cdot)$ satisfy $\rho_{\min} \le g_1(-B^{-1}w)$ and $g_0(-B^{-1}w) \le \rho_{\max}$.

Assumption 3 has the direct interpretation of requiring that the disturbance w can be counteracted by an input that

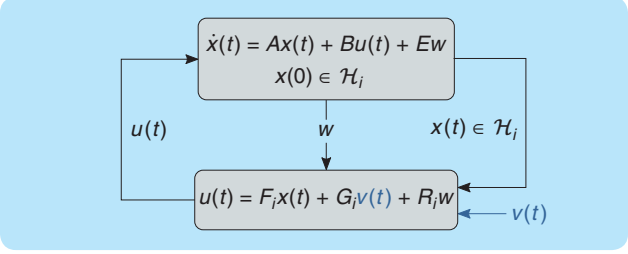


FIGURE 6 Given $x(0) \in \mathcal{H}_i$, the dynamical system (11b) evolves on $\mathcal{H}_i$ if and only if $u(t)$ is of the form (16).

Maximum Velocity as a Quadratic State Constraint

By setting $A = I_n$ and $B = I_n$ in (17), the state x_2 can be interpreted as a vehicle's velocity. A maximum velocity constraint $\|x_2\|_2 \le v_{\max}$ can then be equivalently written in the form of (17e) as

$$x_2^\top (v_{\max}^{-2} I_n) x_2 \le 1. \tag{S13}$$

Figure S5 illustrates the maximum velocity constraint.

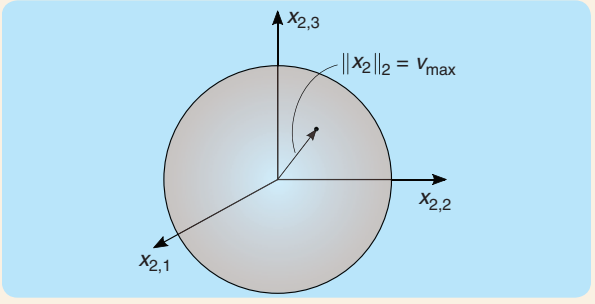


FIGURE S5 The boundary of the maximum velocity constraint, which can be expressed as (17e).

is feasible with respect to (17d). The relaxed problem once again simply convexifies the nonconvex input lower bound by introducing a slack input:

$$\min_{\sigma,u,t_f} \; m(x(t_f)) + \zeta \int_0^{t_f} \ell(\sigma(t))\,\mathrm{d}t, \tag{18a}$$

$$\text{s.t. } \dot{x}_1(t) = Ax_2(t), \tag{18b}$$

$$\dot{x}_2(t) = Bu(t) + w, \tag{18c}$$

$$\rho_{\min} \le \sigma(t), \quad g_0(u(t)) \le \rho_{\max}, \tag{18d}$$

$$g_1(u(t)) \le \sigma(t), \tag{18e}$$

$$x_2(t)^\top H x_2(t) \le 1, \tag{18f}$$

$$x_1(0) = x_{1,0}, \quad x_2(0) = x_{2,0}, \quad b(x(t_f)) = 0. \tag{18g}$$

Thanks to the structure of the dynamics (18b) and (18c), it can be shown that Condition 1 is automatically satisfied. On the other hand, Condition 2 must be modified to account for the quadratic state constraint.

Condition 6

If $\zeta = 0$, the vector $\nabla_x m[t_f] \in \mathbb{R}^{2n}$ and columns of the following matrix must be linearly independent:

$$\tilde{B}_{\text{LCvx}} = \begin{bmatrix} \nabla_{x_1} b[t_f]^\top & 0 \\ \nabla_{x_2} b[t_f]^\top & 2Hx_2(t_f) \end{bmatrix} \in \mathbb{R}^{2n \times (n_b+1)}. \tag{19}$$

Note that Condition 6 carries a subtle but important implication. Recall that due to Assumption 1, $\nabla_x b[t_f]^\top$ must be full column rank. Hence, if $\zeta = 0$, then we must have $n_b < 2n$, and if $n_b = 2n-1$, the vector $(0, 2Hx_2(t_f)) \in \mathbb{R}^{2n}$ and columns of $\nabla_x b[t_f]^\top$ must be linearly dependent. Otherwise, $\tilde{B}_{\text{LCvx}}$ is full column rank, and Condition 6 cannot be satisfied. With this in mind, the following LCvx result was proved in [99, Th. 2].

Theorem 4

The solution of (18) is globally optimal for (17) if Condition 6 holds.

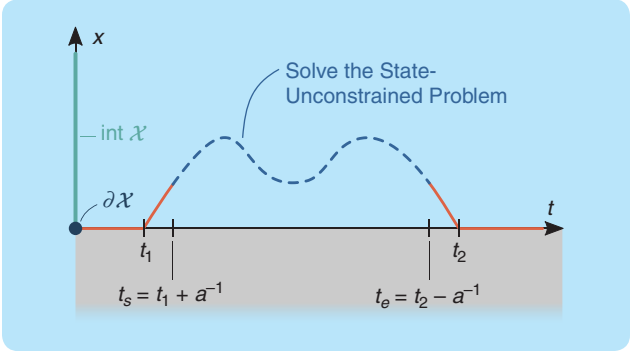


FIGURE 7 The dashed curve represents any segment of the optimal state trajectory for (20) that evolves in the interior of the state constraint set (20d). Because the optimal control problem is autonomous, any such segment is the solution to the state-unconstrained problem (6). When $\zeta = 1$ and taking the limit as $a \to \infty$, lossless convexification applies to the entire (open) segment inside int $\mathcal{X}$ [86].

General Convex State Constraints

The preceding two sections discussed problem classes where an LCvx guarantee is available even in the presence of affine and quadratic state constraints. For obvious reasons, an engineer may want to impose more exotic constraints than afforded by (11) and (17). Luckily, an LCvx guarantee is available for general convex state constraints.

As may be expected, generality comes at the price of a somewhat weaker result. In the preceding sections, the LCvx guarantee was independent from the way in which the affine and quadratic state constraints get activated: instantaneously, for periods of time, or for the entire optimal trajectory duration. In contrast, for the case of general convex state constraints, an LCvx guarantee will hold only as long as the state constraints are active *pointwise* in time. In other words, they get activated at isolated time instances and never persistently over a time interval. This result was originally provided in [86].

The nonconvex problem statement is

$$\min_{u,t_f} \; m(x(t_f)) + \zeta \int_0^{t_f} \ell(g_1(u(t)))\,\mathrm{d}t, \tag{20a}$$

$$\text{s.t. } \dot{x}(t) = Ax(t) + Bu(t) + Ew, \tag{20b}$$

$$\rho_{\min} \le g_1(u(t)), \quad g_0(u(t)) \le \rho_{\max}, \tag{20c}$$

$$x(t) \in \mathcal{X}, \tag{20d}$$

$$x(0) = x_0, \quad b(x(t_f)) = 0, \tag{20e}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ is a convex set that defines the state constraints. Without the state constraint, (20) is nothing but the autonomous version of (6). As for (11), time variance can be introduced in a limited way by using a time integrator state (as long as this does not introduce nonconvexity).

The relaxed problem uses the now familiar slack variable relaxation technique for (20c):

$$\min_{\sigma,u,t_f} \; m(x(t_f)) + \zeta \int_0^{t_f} \ell(\sigma(t))\,\mathrm{d}t, \tag{21a}$$

$$\text{s.t. } \dot{x}(t) = Ax(t) + Bu(t) + Ew, \tag{21b}$$

$$\rho_{\min} \le \sigma(t), \quad g_0(u(t)) \le \rho_{\max}, \tag{21c}$$

$$g_1(u(t)) \le \sigma(t), \tag{21d}$$

$$x(t) \in \mathcal{X}, \tag{21e}$$

$$x(0) = x_0, \quad b(x(t_f)) = 0. \tag{21f}$$

The LCvx proof is provided in [86, Corollary 3] and relies on recognizing the following two key facts:

1) When $x(t) \in \text{int}\mathcal{X}$ for any time interval $t \in [t_1, t_2]$, the state of the optimal control problem is unconstrained along that time interval.
2) For autonomous problems [recall the description after (11)], every segment of the trajectory is itself optimal [99].

As a result, whenever $x(t) \in \text{int}\,\mathcal{X}$, the solution of (21) is equivalent to the solution of (7). Consider an *interior* trajectory segment, as illustrated in Figure 7. The optimal trajectory for the dashed portion in Figure 7 is the solution of the following fixed final state, free final time problem:

$$\min_{u,t_f} \; m(z(t_e)) + \zeta \int_{t_s}^{t_e} \ell(g_1(u(t)))\,\mathrm{d}t, \tag{22a}$$

$$\text{s.t. } \dot{z}(t) = Az(t) + Bu(t) + Ew, \tag{22b}$$

$$\rho_{\min} \le g_1(u(t)), \quad g_0(u(t)) \le \rho_{\max}, \tag{22c}$$

$$z(t_s) = x(t_s), \quad z(t_e) = x(t_e). \tag{22d}$$

Note that (22) is an instance of (6), and, as long as $\zeta = 1$ (for Condition 2 to hold), Theorem 1 applies. Because $a > 0$ can be arbitrarily large in Figure 7, LCvx applies over the open time interval (t_1, t_2). Thus, the solution segments of the relaxed problem that lie in the interior of the state constraint set are feasible and globally optimal for the original (20). The same cannot be said when $x(t) \in \partial\mathcal{X}$. During these segments, the solution can become infeasible for (20). However, as long as $x(t) \in \partial\mathcal{X}$ at isolated time instances, LCvx can be guaranteed to hold. This idea is further illustrated in "Permissible State Constraint Activation for General Convex State Constraints."

When $\zeta = 0$, the situation becomes more complicated because Condition 2 does not hold for (22). This is clear from the fact that the terms defined in Condition 2 become

$$m_{\mathrm{LCvx}} = \begin{bmatrix} \nabla_z m[t_e] \\ 0 \end{bmatrix}, \quad B_{\mathrm{LCvx}} = \begin{bmatrix} I_n \\ 0 \end{bmatrix},$$

which are clearly not linearly independent since B_{LCvx} is full column rank. Thus, even for interior segments, the solution may be infeasible for (20). To remedy this, [86, Corollary 4] suggests Algorithm 1. At its core, the algorithm

Permissible State Constraint Activation for General Convex State Constraints

Return to the landing glideslope constraint for a spacecraft, which was motivated in "Landing Glideslope as an Affine State Constraint." Because (20) allows for using any convex state constraint, the second-order cone constraint (S9) can be used directly. As illustrated in Figure S6, lossless convexification (LCvx) in this case will hold only if the spacecraft touches the "landing cone" a finite number of times.

In Figure S6(a), the glideslope constraint is activated once at time t_1 prior to landing. Hence, $\mathcal{T} = \{t \in [0, t_f] : x(t) \in \partial\mathcal{X}\} = \{t_1, t_f\}$ is a discrete set, and Theorem 5 holds. Note that the constraint may be activated at other times (for example, $\mathcal{T} = \{t_1, t_2, t_f\}$) as long as these times are all isolated points. In Figure S6(b) there is an interval of time for which the glideslope constraint is active. This results in a nondiscrete set of state constraint activation times $\mathcal{T} = [t_1, t_2] \cup \{t_f\}$. For $t \in [t_1, t_2]$, there is no LCvx guarantee, and the solution to (21) may be infeasible for (20) over that interval.

For historical context, LCvx theory was initially developed specifically for planetary rocket landing. For this application, glideslope constraint activation behaves like Figure S6(a), so LCvx holds for that application. Indeed, the constraint (S9) was part of the NASA Jet Propulsion Laboratory's optimization-based rocket landing algorithm flight tests [S1], [S2], [37]–[39].

REFERENCES

[S1] B. Açikmeşe *et al.*, "Flight testing of trajectories computed by G-FOLD: Fuel optimal large divert guidance algorithm for planetary landing," in *Proc. 23rd Amer. Astronaut. Soc. Amer. Inst. Aeronaut. Astronaut. Space Flight Mech. Meeting*, Kauai, HI, USA, Feb. 2013, pp. 1–14.

[S2] D. P. Scharf *et al.*, "ADAPT demonstrations of onboard large-divert guidance with a VTVL rocket," in *Proc. IEEE Aerosp. Conf.*, Mar. 2014, pp. 1–18, doi: 10.1109/aero.2014.6836462.

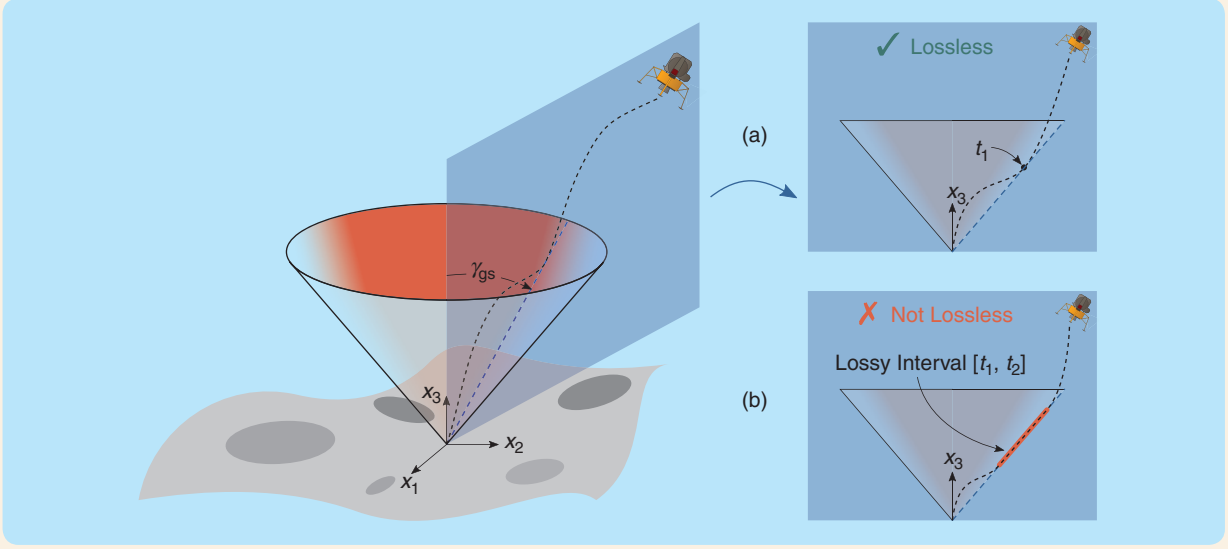


FIGURE S6 An illustration of when lossless convexification with general convex state constraints may fail. In (a), the glideslope constraint is activated once prior to landing. Hence, the solution is lossless. In (b), the constraint is activated for a nontrivial duration, and the solution may be infeasible over that interval. Note that the two figures (a) and (b) are in-plane projections of the 3D figure on the left.

relies on the following simple idea. By solving (21) with the suggested modifications on line 3 of Algorithm 1, every interior segment once again becomes an instance of (6), for which Theorem 1 holds. Furthermore, due to the constraint $x(t_f) = x^*(t_f^*)$, any solution to the modified problem will be optimal for the original formulation, where $\zeta = 0$ [since $m(x(t_f)) = m(x^*(t_f^*))$].

This modification can be viewed as a search for an equivalent solution for which LCvx holds. As a concrete example, (21) may be searching for a minimum miss distance solution for a planetary rocket landing trajectory [102]. The ancillary problem in Algorithm 1 can search for a minimum-fuel solution that achieves the same miss distance. Clearly, other running cost choices are possible. Thus, the ancillary problem's running cost becomes an extra tuning parameter.

It is now possible to summarize the LCvx result for problems with general convex state constraints.

Theorem 5

Algorithm 1 returns the globally optimal solution of (20) if the state constraint (20d) is activated at isolated time instances and Condition 1 and Condition 2 hold.

Nonlinear Dynamics

A commonality among the previous sections is the assumption that the system dynamics are linear. Across all LCvx results that were mentioned so far, the dynamics did not vary much from the first formulation in (6b). Many engineering applications, however, involve nonnegligible nonlinearities. A natural question is then whether the theory of LCvx can be extended to systems with general nonlinear dynamics.

An LCvx result is available for a class of nonlinear dynamical systems. The groundwork for this extension was presented in [46]. The goal here is to show that the standard input set relaxation based on the LCvx equality constraint is also lossless when the dynamics are nonlinear. Importantly, note that the dynamics themselves are not convexified, so the relaxed optimization problem is still nonconvex. Reliably solving for the global optimum is possible in special cases, for example, if the nonlinearities are approximated by piecewise affine functions. This yields a mixed-integer convex problem whose globally optimal solution can be found via mixed-integer programming [103].

ALGORITHM 1 The solution algorithm for (20).

When $\zeta = 0$, a two-step procedure is used, where an auxiliary problem with $\zeta = 1$ searches over the optimal solutions to the original problem.

1: Solve (21) to obtain $x^*(t_f^*)$
2: **if** $\zeta = 0$ **then**
3: Solve (21) again, with the modifications
- Use the cost $\int_0^{t_f} \ell(\sigma(t))\,dt$
- Set $b(x(t_f)) = x(t_f) - x^*(t_f^*)$

Since the relaxed problem is still nonconvex, it is instructive to dwell on the distinction between locally and globally optimal solutions. So far, only globally optimal solutions have been used since every locally optimal solution is globally optimal for a convex problem [26]. However, LCvx proofs rely on Pontryagin's maximum principle, which provides necessary conditions of optimality [104]. As a result, LCvx holds even for *locally* optimal solutions in the sense that such solutions are also locally optimal for the original problem and vice versa. Similarly, a globally optimal solution of the relaxed problem is also globally optimal for the original problem.

With this nuance in mind, the generalization of (6) is introduced that will be solved using LCvx:

$$\min_{u,t_f} \; m(t_f, x(t_f)) + \zeta \int_0^{t_f} \ell(g(u(t)))\,dt, \tag{23a}$$

$$\text{s.t. } \dot{x}(t) = f(t, x(t), u(t), g(u(t))), \tag{23b}$$

$$\rho_{\min} \le g(u(t)) \le \rho_{\max}, \tag{23c}$$

$$x(0) = x_0, \quad b(t_f, x(t_f)) = 0, \tag{23d}$$

where $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^n$ defines the nonlinear dynamics. As for (9), it is required that $g_0 = g_1 \triangleq g$.

Consider the following convex relaxation of the input constraint by using a slack input:

$$\min_{\sigma,u,t_f} \; m(t_f, x(t_f)) + \zeta \int_0^{t_f} \ell(\sigma(t))\,dt, \tag{24a}$$

$$\text{s.t. } \dot{x}(t) = f(t, x(t), u(t), \sigma(t)), \tag{24b}$$

$$\rho_{\min} \le \sigma(t) \le \rho_{\max}, \tag{24c}$$

$$g(u(t)) \le \sigma(t), \tag{24d}$$

$$x(0) = x_0, \quad b(t_f, x(t_f)) = 0. \tag{24e}$$

Note that the slack input σ makes a new appearance in the dynamics (24b). The more complicated dynamics require an updated version of Condition 1 to guarantee that LCvx holds.

Condition 7

The pair $\{\nabla_x f[t], \nabla_u f[t]\}$ must be totally controllable on $[0, t_f]$ for all feasible sequences of $x(\cdot)$ and $u(\cdot)$ for (24) [46], [87].

Using this condition, the following quite general LCvx guarantee is stated for problems that fit the template of problem (23).

Theorem 6

A locally (globally) optimal solution of (24) is locally (globally) optimal for (23) if Condition 2 and Condition 7 hold.

Condition 7 is generally quite difficult to check. Nevertheless, two general classes of systems have been shown to automatically satisfy this condition, thanks to the structure of their dynamics [46]. These classes accommodate vehicle trajectory generation problems with double integrator

dynamics and nonlinearities, such as mass depletion, aerodynamic drag, and nonlinear gravity. The following discussion of these system classes can appear hard to parse at first sight. For this reason, two practical examples of systems that belong to each class are provided in "Examples of Losslessly Convexifiable Nonlinear Systems."

The first corollary of Theorem 6 introduces the first class of systems. A key insight is that the null space conditions of the corollary require that $2m \geq n$; that is, there are at least half as many control variables as there are state variables. This is satisfied by some vehicle trajectory generation problems in which $2m = n$ (for example, when the state consists of the position and velocity, while the control is an acceleration that acts on all velocity states). This is a common approximation for flying vehicles. An example for rocket landing is shown in the third part of the article.

Corollary 1

Suppose that the dynamics (23b) are of the form

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad f(t, x, u, g(u)) = \begin{bmatrix} f_1(t, x) \\ f_2(t, x, u) \end{bmatrix}, \tag{25}$$

where $\operatorname{null}(\nabla_u f_2) = \{0\}$ and $\operatorname{null}(\nabla_{x_2} f_1) = \{0\}$. Then, Theorem 6 applies if Condition 2 holds.

The next corollary to Theorem 6 introduces the second class of systems, for which $2m < n$ is allowed. This class is once again useful for vehicle trajectory generation problems where the dynamics are given by (26) and $g(u)$ is a function that measures control effort. A practical example is when the state x_2 is mass, which is depleted as a function of the control effort (such as thrust for a rocket).

Corollary 2

Suppose that the dynamics (23b) are of the form

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad f(t, x, u, g(u)) = \begin{bmatrix} f_1(t, x, u) \\ f_2(t, g(u)) \end{bmatrix}. \tag{26}$$

Define the matrix

$$M \triangleq \begin{bmatrix} (\nabla_u f_1)^\top \\ \dfrac{\mathrm{d}(\nabla_u f_1)^\top}{\mathrm{d}t} - (\nabla_u f)^\top (\nabla_x f_1)^\top \end{bmatrix}. \tag{27}$$

Furthermore, suppose that the terminal constraint function b is affine and that $x_2(t_f)$ is unconstrained such that $\nabla_{x_2} b = 0$. Then Theorem 6 applies if $\operatorname{null}(M) = \{0\}$, and Condition 2 holds.

It must be emphasized that, due to the nonlinear dynamics, (24) is still a nonconvex program that can be hard to solve. For Theorem 6 to hold, at least a locally optimal solution must be found. In the special case when the dynamics f are piecewise affine, the problem can be solved to global optimality via mixed-integer programming [103], [105], [106]. In this case, the convexification of the nonconvex input lower bound reduces the number of disjunctions in the branch-and-bound tree and hence lowers the problem complexity [46]. Several examples of nonlinear systems that can be modeled in this way and comply with Corollary 1 and Corollary 2 are provided in "Approximating Nonlinear Systems With Piecewise Affine Functions."

Embedded Lossless Convexification

Note that the LCvx theory of the previous sections addresses special cases of problems whose nonconvexity is "just right" for an LCvx guarantee to be provable using the maximum principle. Although such problems have found practical use in problems such as spaceflight [37] and quadrotor path planning [52], this restriction leaves out many trajectory generation applications that do not fit the tight mold of original problems and conditions of the previous sections.

Despite this apparent limitation, LCvx is still highly relevant for problems that simply do not conform to one of the forms given in the previous sections. For such problems, suppose that the reader is facing the challenge of solving a nonconvex optimal control problem that fits the mold of (38) (the subject of the "Sequential Convex Programming" section) and considering whether LCvx can help. There is evidence that the answer is affirmative if one uses LCvx theory only on the constraints that are losslessly convexifiable. This is called *embedded LCvx* because it is used to convexify only part of the problem, while the rest is handled by another nonconvex optimization method, such as presented in the second part of this article. Because LCvx reduces the amount of nonconvexity present in the problem, it can significantly improve the convergence properties and reduce the computational cost to solve the resulting problem. An example of this approach for quadrotor trajectory generation is demonstrated in the "Application Examples" section at the end of this article.

The basic procedure for applying embedded LCvx is detailed in Figure 8. As shown in Figure 8(b), LCvx is not a computation scheme but a convex relaxation with an accompanying proof of equivalence to the original problem. Thus, it happens prior to the solution and simply changes the problem description seen by the subsequent numerical optimization algorithm. There are a number of examples of embedded LCvx worth mentioning. First, the previous section on nonlinear dynamics can be interpreted as embedded LCvx. For example, [46] solves a rocket landing problem in which only the nonconvex input constraint (23c) is convexified. This leaves behind a nonconvex problem due to nonlinear dynamics, and mixed-integer programming is used to solve it.

Another example is provided in [107], where LCvx is embedded in a mixed-integer AAV trajectory generation problem to convexify a stall speed constraint of the form

$$0 < v_{\min} \le \| v_{\text{cmd}}(t) \|_2 \le v_{\max}, \tag{28}$$

where the input $v_{\text{cmd}}(\cdot) \in \mathbb{R}^3$ is the commanded velocity while $v_{\min}$ and $v_{\max}$ are lower and upper bounds that guarantee a stable flight envelope. The same constraint is also considered in [46]. In [53], the authors develop a highly nonlinear planetary entry trajectory optimization problem, where the control input is the bank angle $\beta \in \mathbb{R}$ parameterized via two inputs $u_1 \triangleq \cos(\beta)$ and $u_2 \triangleq \sin(\beta)$. The associated constraint $\| u \|_2^2 = 1$ is convexified to $\| u \|_2^2 \le 1$, and equality at the optimal solution is shown in an LCvx-like fashion (the authors call it "assurance of active control constraint"). Similar methods are used in [55] in the context of rocket landing with aerodynamic controls.

A survey of related methods is available in [108]. Finally, a noteworthy approach is taken in [52] and [58], where embedded LCvx is used to convexify an input lower bound and an attitude-pointing constraint for rocket landing and agile quadrotor flight. SCP from the "Sequential Convex Programming" section is then used to solve the remaining nonlinear optimal control problems. The quadrotor application, in particular, is demonstrated as a numerical example in the "Application Examples" section at the end of this article. As a result of the success of these applications, there are likely to be further unexplored opportunities to use LCvx as a strategy to derive simpler problem formulations. The results would speed up computation for optimization-based trajectory generation.

The Future of Lossless Convexification

LCvx is a method that solves nonconvex trajectory generation problems with one or a small number of calls to a convex solver. This places it among the most reliable and robust methods for nonconvex trajectory generation. The future of LCvx therefore has an obvious motivation: to expand the class of problems that can be losslessly convexified. The most recent result discussed in the previous sections is for problems with affine state constraints [92], which dates back to 2014. In the last several years, LCvx research has been rejuvenated by several fundamental discoveries and practical methods that expand the approach to new and interesting problem types. This section briefly surveys these new results.

Fixed Final Time Problems

The first new LCvx result applies to a fixed final time and fixed final state version of (6) with no state constraints. To begin, recognize that the classical LCvx result from Theorem 1 does not apply when both t_f and $x(t_f)$ are fixed. In

Examples of Losslessly Convexifiable Nonlinear Systems

At first sight, the discussion around Corollary 1 and Corollary 2 may be hard to parse into something useful. On the contrary, let us describe two concrete and very practical examples of dynamical systems that satisfy these corollaries. Both examples originate from [46].

NONLINEAR ROCKET LANDING

First, lossless convexification (LCvx) can be used for rocket landing with nonlinear gravity and aerodynamic drag. Both effects are important for landing either on small celestial bodies with a weak gravitational pull or planets with a thick atmosphere (such as Earth). In this case, the lander dynamics can be written as

$$\ddot{r}(t) = g(r(t)) - \frac{c}{m(t)} \| \dot{r}(t) \|_2 \dot{r}(t) + \frac{T(t)}{m(t)}, \tag{S14a}$$

$$\dot{m}(t) = -\alpha \| T(t) \|_2, \tag{S14b}$$

where r denotes the position, m is the mass, T is the thrust, c is the drag coefficient, α is inversely proportional to the rocket engine's specific impulse, and $g : \mathbb{R}^3 \to \mathbb{R}^3$ is the nonlinear gravity model. An illustration is given in Figure S7(a).

The preceding dynamics can be rewritten using the template of (23b) by defining the state $x \triangleq (r, \dot{r}, m) \in \mathbb{R}^7$, input $u \triangleq T \in \mathbb{R}^3$, and input penalty function $g(u) \triangleq \| u \|_2$. The equations of motion can then be written as (omitting the time argument for concision)

$$f(t, x, u, g(u)) = \begin{bmatrix} f_r(x, u) \\ f_{\dot{r}}(x) \\ f_m(g(u)) \end{bmatrix} = \begin{bmatrix} \dot{r} \\ g(r) - cm^{-1} \| \dot{r} \|_2 \dot{r} + Tm^{-1} \\ -\alpha \| T \|_2 \end{bmatrix}.$$

This system belongs to the class in Corollary 2. In particular, let $f_1 = (f_r, f_{\dot{r}})$ and $f_2 = f_m$. Applying the algebra in (27),

$$M = \begin{bmatrix} 0 & m^{-1} I \\ m^{-1} I & M_{22} \end{bmatrix}, \tag{S15}$$

where $M_{22} = -(c/m^2)(\| \dot{r} \|_2 I + \dot{r}\dot{r}^\top / \| \dot{r} \|_2) + \alpha T T^\top / (m^2 \| T \|_2)$. Thanks to the off-diagonal terms, $\text{null}(M) = \{0\}$ unconditionally, so the rocket lander dynamics satisfy the requirements of Corollary 2.

AUTONOMOUS AERIAL VEHICLE TRAJECTORY GENERATION WITHOUT STALLING

An autonomous aerial vehicle can lose control and fall out of the sky if its airspeed drops below a certain value. This occurs because the wings fail to generate enough lift, resulting in an aerodynamic *stall*. It was shown in [46] that a stall speed constraint of the form $v_{\min} \le \| v \|_2$ can be handled by LCvx via the following dynamics:

this case, $B_{\text{LCvx}} = I_{n+1}$ in (8b), and therefore its columns (which span all of $\mathbb{R}^{n+1}$) cannot be linearly independent from m_{LCvx}. Thus, one traditionally could not fix the final time and the final state simultaneously. Recently, Kunhippurayil et al. [109] showed that Condition 2, in fact, is not necessary for the following version of (6):

$$\min_{u} \int_0^{t_f} \ell(g(u(t)))\,\mathrm{d}t, \tag{29a}$$

$$\text{s.t. } \dot{x}(t) = Ax(t) + Bu(t), \tag{29b}$$

$$\rho_{\min} \le g(u(t)) \le \rho_{\max}, \tag{29c}$$

$$x(0) = x_0, \quad x(t_f) = x_f, \tag{29d}$$

where t_f is fixed and $x_f \in \mathbb{R}^n$ specifies the final state.

The lossless relaxation is the usual one, which is a specialization of (7) for (29):

$$\min_{\sigma,u} \int_0^{t_f} \ell(\sigma(t))\,\mathrm{d}t, \tag{30a}$$

$$\text{s.t. } \dot{x}(t) = Ax(t) + Bu(t)w(t), \tag{30b}$$

$$\rho_{\min} \le \sigma(t) \le \rho_{\max}, \tag{30c}$$

$$g(u(t)) \le \sigma(t), \tag{30d}$$

$$x(0) = x_0, \quad x(t_f) = x_f. \tag{30e}$$

The following result is then proved in [109]. By dropping Condition 2, the result generalizes Theorem 1 and significantly expands the reach of LCvx to problems without state constraints.

Theorem 7

The solution of (30) is globally optimal for (29) if Condition 1 holds and t_f is between the minimum feasible time and the time that minimizes (29a). For longer trajectory durations, there exists a feasible solution of (30) that is globally optimal for (29).

Perhaps the most important part of Theorem 7, and a significant future direction for LCvx, is in its final sentence. Although a lossless solution "exists," how does one find it? An *algorithm* is provided in [109] to find the lossless solution, that is, one solution among many others that may not be lossless. This is similar to Theorem 5 and Algorithm 1: it is known that slackness in (30d) may occur, so an algorithm is devised that works around the issue and recovers an input for which (30d) holds with equality. Most traditional LCvx results place further restrictions on the original problem to "avoid" slackness. However, this limits the applicability of LCvx. By instead providing algorithms that recover lossless inputs from problems that do not admit LCvx

$$\dot{r}(t) = v(t) + v_\infty(t), \tag{S16a}$$

$$\dot{v}(t) = \kappa(v_d(t) - v(t)) - c\|v(t)\|_2 v(t), \tag{S16b}$$

where v is the airspeed, $\dot{r}$ is the ground velocity, and v_∞ is the velocity of the air mass relative to the ground (also known as the *freestream velocity*). An illustration is provided in Figure S7(b).

The key transformation in (S16) is to make the desired airspeed v_d the control variable and include a velocity control inner loop via a proportional controller with gain κ. Because the velocity dynamics are first order, v converges to v_d along a straight path in the velocity phase plane. Hence, the stall speed constraint is closely approximated for small control errors $\|v_d - v\|_2$. The dynamics can then be rewritten using the template of (23b) by defining the state $x \triangleq (r, v) \in \mathbb{R}^6$ and input $u \triangleq v_d \in \mathbb{R}^3$. The equations of motions can then be written as

$$f(t, x, u, g(u)) = \begin{bmatrix} f_r(t, x) \\ f_v(x, u) \end{bmatrix}, = \begin{bmatrix} v + v_\infty \\ -\kappa v - c\|v\|_2 v + \kappa v_d \end{bmatrix}.$$

This system belongs to the class in Corollary 1. In particular, let $f_1 = f_r$ and $f_2 = f_v$. Then, $\nabla_u f_2 = \kappa I$ and $\nabla_{x_2} f_1 = I$. Therefore, $\text{null}(\nabla_u f_2) = \{0\}$ and $\text{null}(\nabla_{x_2} f_1) = \{0\}$, so the aircraft dynamics satisfy the requirements of Corollary 1.

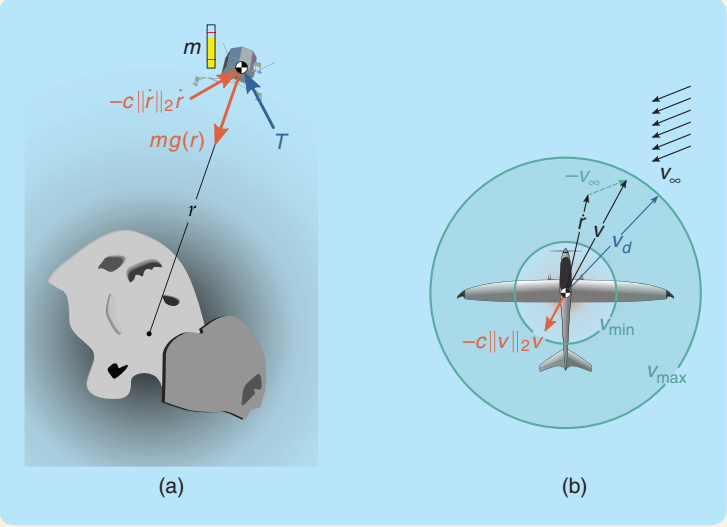


FIGURE S7 Examples of two nonlinear systems for which the lossless convexification result of Theorem 6 can be applied. (a) Landing a rocket in the presence of nonlinear gravity and atmospheric drag. (b) Autonomous aircraft trajectory generation with a stall constraint and aerodynamic drag.

naturally, LCvx can be addressed "head on," and the class of losslessly convexifiable problems can be expanded. A similar approach is used for spacecraft rendezvous in [56], where an iterative algorithm modifies the dynamics to extract bang–bang controls from a solution that exhibits slackness.

Hybrid System Problems

Many physical systems contain discrete on–off elements, such as valves, relays, and switches [46], [106], [110]–[112]. Discrete behavior can also appear through interactions between the autonomous agent and its environment, such as through foot contact for walking robots [113]–[115]. Modeling discrete behavior is the province of hybrid system theory, and the resulting trajectory problems typically combine continuous variables and discrete logic elements (that is, "and" and "or" gates) [110], [112], [116]. Because there is no concept of an infinitesimally small local perturbation for values that, for example, can be equal only to zero or one, problems with discrete logic are fundamentally more difficult to solve. Traditional solution methods use mixed-integer programming [103]. The underlying branch-and-bound method, however, has poor (combinatorial) worst-case complexity. Historically, this made it very difficult to put optimization with discrete logic onboard computationally constrained and safety-critical systems throughout aerospace, automotive, and even state-of-the-art robotics sectors [117].

Two recent results showed that LCvx can be applied to certain classes of hybrid optimal control problems that are

Approximating Nonlinear Systems With Piecewise Affine Functions

Piecewise affine functions can be used for arbitrarily accurate approximation of any nonlinear function. This is the technique used by [46] to write the nonlinear dynamics (24b) in piecewise affine form. Doing so enables solving (24) to global optimality via mixed-integer programming, which is one possible way to meet the lossless convexification requirements of Theorem 6. It is now shown how a piecewise affine approximation can be obtained in the general case and concretely in the case of the dynamics from "Examples of Losslessly Convexifiable Nonlinear Systems."

GENERAL CASE

Begin by finding a first-order approximation of the nonlinear equation $f:\mathbb{R}\times\mathbb{R}^n\times\mathbb{R}^m\times\mathbb{R}\to\mathbb{R}^n$ from (24b) about an operating point $(t,\bar{x}^i,\bar{u}^i)\in\mathbb{R}\times\mathbb{R}^n\times\mathbb{R}^m$, where i is an integer index that will become clear in a moment. Without loss of generality, assume that f is decomposable into an affine and a nonaffine part:

$$f = f_a + f_{\text{na}}. \tag{S17}$$

The first-order Taylor expansion of f is given by

$$\begin{aligned} f &\approx f_a + f^i_{\text{na}},\\ f^i_{\text{na}} &\triangleq \bar{f}_{\text{na}} + (\nabla_x \bar{f}_{\text{na}})(x-\bar{x}) + (\tilde{\nabla}_u \bar{f}_{\text{na}})(u-\bar{u}), \end{aligned} \tag{S18}$$

where the following shorthand is used:

$$f^i_{\text{na}} = f^i_{\text{na}}(t,x,u,g(u)), \tag{S19a}$$

$$\bar{f}_{\text{na}} = f_{\text{na}}(t,\bar{x}^i,\bar{u}^i,g(\bar{u}^i)), \tag{S19b}$$

$$\tilde{\nabla}_u f_{\text{na}} = \nabla_u f_{\text{na}} + (\nabla_g f_{\text{na}})(\nabla g)^\top. \tag{S19c}$$

In (S19c), it is understood that $\nabla_g f_{\text{na}}$ is the Jacobian of f_{na} with respect to its fourth argument. Suppose that the linearization in (S18) is sufficiently accurate over only the hyperrectangular region $\mathcal{R}^i\subset\mathbb{R}^n\times\mathbb{R}^m$ defined by the following inequalities:

$$\bar{x}^i + b^i_{L,x} \le x \le \bar{x}^i + b^i_{U,x}, \tag{S20a}$$

$$\bar{u}^i + b^i_{L,u} \le u \le \bar{u}^i + b^i_{U,u}, \tag{S20b}$$

where $b^i_{L,x}$ and $b^i_{U,x}$ represent the upper and lower bounds on the state, while $b^i_{L,u}$ and $b^i_{U,u}$ relate to the input. The index i now takes on a clear meaning: it represents the ith "validity" region. Without loss of generality, assume $\mathcal{R}^i\cap\mathcal{R}^j=\emptyset$ if $i\neq j$. In general, $i=1,\ldots,N$, which means that f is approximated by affine functions over N regions. The piecewise affine approximation of f can then be written as

$$f_{\text{pwa}} \triangleq f_a + f^i_{\text{na}}, \text{ for } i \text{ such that } (x,u)\in\mathcal{R}^i. \tag{S21}$$

The big-M formulation can be used to write (S21) in a form that is readily employed in a mixed-integer program [106]. To this end, let $M>0$ be a sufficiently large fixed scalar parameter, and let $z^i\in\{0,1\}$ be a binary variable indicating that $(x,u)\in\mathcal{R}^i$ if and only if $z^i=1$. Then, the piecewise affine dynamics in mixed-integer programming form are encoded by the following set of constraints:

$$\dot{x} = f_a(t,x,u,g(u)) + \sum_{i=1}^N z^i f^i_{\text{na}}(t,x,u,g(u)), \tag{S22a}$$

$$x \ge \bar{x}^i + b^i_{L,x} - M(1-z^i), \tag{S22b}$$

$$x \le \bar{x}^i + b^i_{U,x} + M(1-z^i), \tag{S22c}$$

$$u \ge \bar{u}^i + b^i_{L,u} - M(1-z^i), \tag{S22d}$$

$$u \le \bar{u}^i + b^i_{U,u} + M(1-z^i), \tag{S22e}$$

$$1 = \sum_{i=1}^N z^i. \tag{S22f}$$

NONLINEAR ROCKET LANDING

Consider the rocket dynamics in (S14a), and, for simplicity, suppose that the mass is constant. Define the state $x\triangleq(r,\dot{r})\in\mathbb{R}^6$ and input $u\triangleq T\in\mathbb{R}^3$. The nonaffine part of the dynamics (S17) then takes the particular form

$$f_{\text{na}} = \begin{bmatrix} 0 \\ g(r) - cm^{-1}\|\dot{r}\|_2\dot{r} \end{bmatrix}. \tag{S23}$$

useful for trajectory generation [118], [119]. While the results are more general, the following basic problem will help ground the discussion:

$$\min_{u,\gamma,t_f} \int_0^{t_f} \sum_{i=1}^{M} \| u_i(t) \|_2 \mathrm{d}t, \tag{31a}$$

$$\text{s.t. } \dot{x}(t) = Ax(t) + B\sum_{i=1}^{M} u_i(t), \tag{31b}$$

$$\gamma_i(t)\rho_{\min,i} \le \| u_i(t) \|_2 \le \gamma_i(t)\rho_{\max,i}, \tag{31c}$$

$$\gamma_i(t) \in \{0, 1\}, \tag{31d}$$

$$C_i u_i(t) \le 0, \tag{31e}$$

$$x(0) = x_0, \quad b(x(t_f)) = x_f, \tag{31f}$$

where M is the number of individual input vectors and the binary variables γ_i are used to model the on–off nature of each input. Compared to the traditional (6), this new problem can be seen as a system controlled by M actuators that can be either "off" or "on" and norm bounded in the $[\rho_{\min,i}, \rho_{\max,i}]$ interval. The affine input constraint (31e) represents an affine cone and is a specialized version of the earlier constraint (11d). Figure 9 illustrates the kind of input set that can be modeled.

Imitating the previous results, the convex relaxation uses a slack input for each control vector:

$$\min_{\sigma,u,\gamma,t_f} \int_0^{t_f} \sum_{i=1}^{M} \sigma_i(t) \mathrm{d}t, \tag{32a}$$

$$\text{s.t. } \dot{x}(t) = Ax(t) + B\sum_{i=1}^{M} u_i(t), \tag{32b}$$

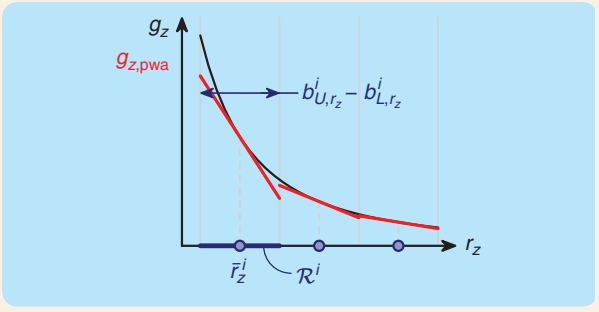


FIGURE S8 A piecewise affine approximation of g_z, the z-component of the nonlinear gravity force, using (S24).

For ease, consider only the nonlinear gravity term, $g(r)$. Atmospheric drag is addressed below for autonomous aerial vehicle (AAV) trajectory generation. Suppose that $g(r) = (0, 0, g_z(r_z))$, which means that only the vertical component $g_z : \mathbb{R} \to \mathbb{R}$ must be considered. A typical profile is $g_z(r_z) = -\mu/r_z^2$, where μ is the gravitational parameter. Given a reference point $\bar{r}^i_z$,

$$g^i_z = g_z(\bar{r}^i_z) + g'_z(\bar{r}^i_z)(r_z - \bar{r}_z). \tag{S24}$$

Using (S21) and (S24), Figure S8 draws $g_{z,\mathrm{pwa}}$.

AUTONOMOUS AERIAL VEHICLE TRAJECTORY GENERATION WITHOUT STALLING

Consider the AAV dynamics in (S16) and, for simplicity, assume constant-altitude flight. Define the state $x \triangleq (r, v) \in \mathbb{R}^4$ and input $u = v_d \in \mathbb{R}^2$. The nonaffine part of the dynamics (S17) then takes the particular form

$$f_{\mathrm{na}} = \begin{bmatrix} 0 \\ -c\| v \|_2 v \end{bmatrix}. \tag{S25}$$

Examine the aerodynamic drag term, $f_d = -c\| v \|_2 v$. The first-order Taylor expansion about a reference airspeed $\bar{v}^i$ is

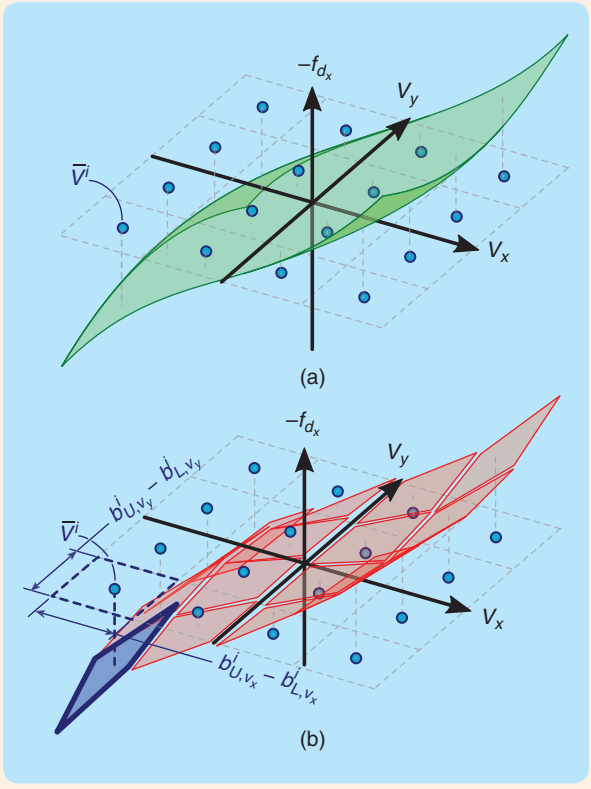


FIGURE S9 A piecewise affine approximation of f_{d_x}, the x-component of the aerodynamic drag force, using (S26). (a) The surface of the continuous function $-f_{d,x}$. The dots display operating points at which the gradient is evaluated. (b) The surface of the discontinuous piecewise affine approximation $-f_{d_x,\mathrm{pwa}}$. The dashed rectangle in the airspeed space shows the boundary of the approximation validity region $\mathcal{R}^i$.

$$f^i_d = -c\| \bar{v}^i \|_2 [I + \| \bar{v}^i \|_2^{-2} \bar{v}^i \bar{v}^{i\top}](v - \bar{v}^i). \tag{S26}$$

Using (S21) and (S26), Figure S9 draws $f_{d,\mathrm{pwa}}$.

$$\gamma_i(t)\rho_{\min,i} \le \sigma_i(t) \le \gamma_i(t)\rho_{\max,i}, \tag{32c}$$

$$\| u_i(t) \|_2 \le \sigma_i(t), \tag{32d}$$

$$0 \le \gamma_i(t) \le 1, \tag{32e}$$

$$C_i u_i(t) \le 0, \tag{32f}$$

$$x(0) = x_0, \quad b(x(t_f)) = x_f, \tag{32g}$$

where the novelty is in that the γ_i variables have also been relaxed to the continuous $[0, 1]$ interval.

Taking (32) as an example, [118] and [119] prove LCvx from slightly different angles. In [119], it is recognized that (32) will losslessly convexify (31) if the dynamical system is "normal" due to the so-called bang–bang principle [12], [120]. Normality is related to, but much stronger than, the notion of controllability from Condition 1. Nevertheless, it is shown that the dynamical system can be perturbed by an arbitrarily small amount to induce normality. This phenomenon was previously observed in a practical context for rocket landing LCvx with a pointing constraint, which was discussed for (9) [89]. Practical examples are given for spacecraft orbit reshaping, minimum energy transfer, and CubeSat differential drag and thrust maneuvering. Note that while mixed-integer programming fails to solve the latter problem, the convex relaxation is solved in less than 100 ms.

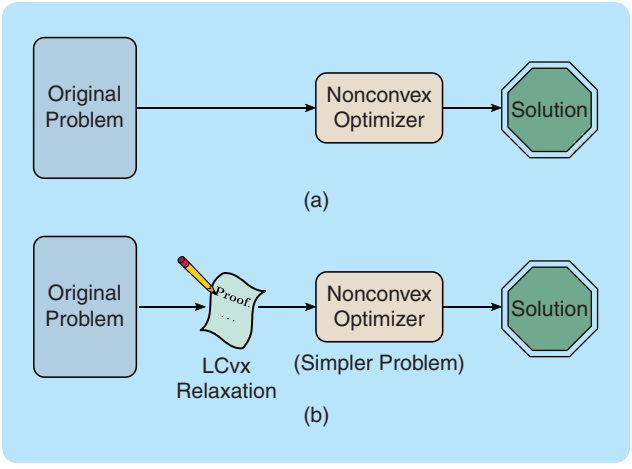


FIGURE 8 An illustration of how embedded lossless convexification (LCvx) can be used to solve an optimal control problem that does not fit any of the templates presented in the "Lossless Convexification" section. (a) The solution process for a nonconvex optimal control problem, without using LCvx. (b) The solution process for a nonconvex optimal control problem, where LCvx is embedded to convexify part of the original problem.

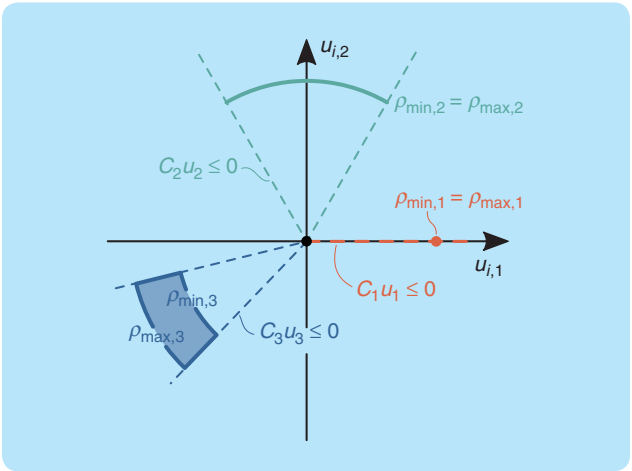


FIGURE 9 A feasible input set that can be modeled in (31). It is a nonconvex, disconnected set composed of the origin, a point, an arc, and a nonconvex set with an interior. For example, this setup can represent a satellite equipped with thrusters and drag plates or a rocket with a thrust–gimbal coupled engine [118], [119].

The results in [57] and [118] also prove LCvx for (32). However, instead of leveraging normality and perturbing the dynamics, the nonsmooth maximum principle [91], [121], [122] is used directly to develop a set of conditions for which LCvx holds. These conditions are an interesting mix of problem geometry [that is, the shapes and orientations of the constraint cones (32f)] and Condition 1 and Condition 2. Notably, they are more general than normality, so they can be satisfied by systems that are not normal. Practical examples are given for spacecraft rendezvous and rocket landing with a coupled thrust–gimbal constraint. The solution is observed to take on the order of a few seconds and be more than 100 times faster than mixed-integer programming.

The developments in [118] and [119] are viewed as complementary: the work of [118] shows that for some systems, the perturbation proposed by [119] is not necessary. On the other hand, the authors of [119] provide a method to recover LCvx when the conditions of [118] fail. Altogether, the fact that an arbitrarily small perturbation of the dynamics can recover LCvx suggests a deeper underlying theory for how and why problems can be losslessly convexified. The search for this theory will be a running theme of future LCvx research, and its eventual discovery will lead to more general LCvx algorithms.

Developing New Lossless Convexifications

The previous sections covered a variety of LCvx guarantees. With some luck, one of these results might fit the reader's particular problem. The corresponding relaxation can then be applied to solve the nonconvex trajectory generation problem through convex optimization. However, it is quite likely that the reader's problem will not fit the rigid mold of "original problems" from the previous sections. In this case, the more important lesson of this article is how to think about developing a new or adapted LCvx guarantee.

Suppose that one is faced with a problem that does not fit the aforementioned LCvx results. In this case, a viable first approach is to consider embedded LCvx from the previous section. The combination of an LCvx relaxation and an SCP method from the "Sequential Convex Programming" section can result in a real-time algorithm that is

applicable to problem types with a more general set of nonconvexities [6], [108].

When developing a new convex relaxation, the suggestion is to adopt a "simulate first, prove later" approach. Begin by proposing a relaxation, then implement the problem and verify that the solution is "empirically lossless." In other words, the method seems to work but without a proof for why or when it does. Next, use the maximum principle to develop a rigorous set of conditions under which LCvx holds. This will most likely be an iterative process in which the insights from the proof affect the numerical implementation and vice versa. In fact, this is exactly the process by which the lossless convexification in (32) was obtained. If the reader is able to complete the proof, he or she can expect to be rewarded with a much faster and more robust solution method as well as a newfound deep insight into the original problem.

The most difficult case arises when the problem's nonconvexities are all different from those discussed in the preceding sections and it is unclear which constraint to relax. For example, there may be no constraint such as (6c) that can be relaxed to create the LCvx equality constraint (7d). In this case, the reader must propose a different convex relaxation. Fortunately, developing an LCvx guarantee remains fundamentally the same. Once the relaxed convex problem is written, Pontryagin's maximum principle is used to show that its optimal solution achieves the global optimum of the original problem. Some researchers have used this approach extensively, and inspiration can be gained from their published methods [53], [55], [108], [123], [124].

Toy Example

The following example provides a simple illustration of how LCvx can be used to solve a nonconvex problem. This example is meant to be a "preview" of the practical application of LCvx. More challenging and realistic cases are given in the "Application Examples" section at the end of this article. The problem to be solved concerns the minimum effort control of a double integrator system (such as a car) with a constant "friction" term, g. This can be written as a linear time-invariant instance of (6):

$$\min_{u} \int_0^{t_f} u(t)^2 \mathrm{d}t, \tag{33a}$$

$$\text{s.t. } \dot{x}_1(t) = x_2(t), \tag{33b}$$

$$\dot{x}_2(t) = u(t) - g, \tag{33c}$$

$$1 \le |u(t)| \le 2, \tag{33d}$$

$$x_1(0) = x_2(0) = 0, \tag{33e}$$

$$x_1(t_f) = s, \quad x_2(t_f) = 0, \quad t_f = 10. \tag{33f}$$

The input $u(\cdot) \in \mathbb{R}$ is the acceleration of the car. The constraint (33d) is a nonconvex, 1D version of the constraint (S1). Assuming that the car has unit mass, the integrand in (33a) has units of watts. The objective of (33) is therefore to move a car by a distance s in $t_f = 10$ s while minimizing the average power. Following the relaxation template provided by (7), the following convex relaxation of the problem is proposed:

$$\min_{\sigma,u} \int_0^{t_f} \sigma(t)^2 \mathrm{d}t, \tag{34a}$$

$$\text{s.t. } \dot{x}_1(t) = x_2(t), \tag{34b}$$

$$\dot{x}_2(t) = u(t) - g, \tag{34c}$$

$$1 \le \sigma(t) \le 2, \tag{34d}$$

$$|u(t)| \le \sigma(t), \tag{34e}$$

$$x_1(0) = x_2(0) = 0, \tag{34f}$$

$$x_1(t_f) = s, \quad x_2(t_f) = 0, \quad t_f = 10. \tag{34g}$$

To guarantee that LCvx holds [in other words, (34) finds the globally optimal solution of (33)], the first attempt is to verify the conditions of Theorem 1. In particular, it must be shown that Condition 1 and Condition 2 hold. First, from (33b) and (33c), the following state-space matrices can be extracted:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{35}$$

It can be verified that Condition 1 holds by either showing that the controllability matrix is full rank or by using the PBH test [88]. Next, from (34a) and (34g), extract the following terminal cost and terminal constraint functions:

$$m(t_f, x(t_f)) = 0, \quad b(t_f, x(t_f)) = \begin{bmatrix} t_f - 10 \\ x_1(t_f) - s \\ x_2(t_f) \end{bmatrix}. \tag{36}$$

Now substitute (36) into (8) to obtain

$$m_{\mathrm{LCvx}} = \begin{bmatrix} 0 \\ 0 \\ \sigma(t_f)^2 \end{bmatrix}, \quad B_{\mathrm{LCvx}} = I_3. \tag{37}$$

Thus, B_{LCvx} is full column rank, and its columns cannot be linearly independent from m_{LCvx}. It is concluded that Condition 2 does not hold, so Theorem 1 cannot be applied. In fact, (33) has both a fixed final time and a fixed final state. This is exactly the edge case for which traditional LCvx does not apply, as mentioned in the previous section on the future of LCvx. Instead, refer to Theorem 7, which says that Condition 2 is not needed as long as t_f is between the minimum and optimal times for (33). This holds for the problem parameters used in Figure 10. The minimum time is just slightly shorter than 10 s, and the optimal time is ≈ 13.8 s for Figure 10(a) and ≈ 13.3 s for Figure 10(b). Most interestingly, LCvx fails [that is, (34e) does not hold with equality] for t_f values almost exactly past the optimal time for Figure 10(a) and just slightly past it for Figure 10(b).

Although (34) is convex, it has an infinite number of solution variables because time is continuous. To find an approximation of the optimal solution by using a numerical convex optimization algorithm, the problem must be

temporally discretized. To this end, apply a first-order hold (FOH) discretization with $N = 50$ temporal nodes, as explained in "Discretizing Continuous-Time Optimal Control Problems."

Reviewing the solutions in Figure 10 for two values of the friction parameter g, the nonconvex constraint (33d) holds in both cases. Note that this is despite the fact that trajectories with $|u(t)| < 1$ constitute feasible solutions of (34). The fact that this does not occur is the salient feature of LCvx theory, and for this problem this outcome is guaranteed by Theorem 7. Finally, note that Figure 10 also plots the analytical globally optimal solution obtained via the maximum principle, where no relaxation or discretization is made. The close match between this solution and the numerical LCvx solution further confirms the theory as well as the accuracy of the FOH discretization method. Note that the mismatch at $t = 0$ in the acceleration plot in Figure 10(b) is a benign, single-time-step discretization artifact that is commonly observed in LCvx numerical solutions.

SEQUENTIAL CONVEX PROGRAMMING

Now consider a different kind of convex optimization-based trajectory generation algorithm: sequential convex programming (SCP). This opens a world of possibilities beyond the restricted capabilities of LCvx. One could say that if LCvx is a surgical knife to remove acute nonconvexity, then SCP is a catch-all sledgehammer for nonconvex trajectory design [6]. A wealth of industrial and research applications, including high-profile experiments, support this statement. Examples can be found in many engineering domains, ranging from aerospace [59], [124]–[126] and mechanical design [127], [128] to power grid technology [129], chemical processes [130], and computer vision [131], [132]. Just last year, the Tipping Point Partnership between NASA and Blue Origin started testing an SCP algorithm aboard the New Shepard rocket [8], [68]. Another application of SCP methods is for the SpaceX Starship landing flip maneuver [133]. Although SpaceX's methods are undisclosed, convex optimization is used by the Falcon 9 rocket, and SCP algorithms are highly capable of solving such challenging trajectories [36], [134].

Further afield, examples of SCP can be found in medicine [135], [136], economics [137], [138], biology [139], [140], and fisheries [141]. Of course, in any of these applications, SCP is not the only methodology that can be used to obtain good solutions. Others might include IPMs [13], [142], dynamic programming [143], augmented Lagrangian techniques [144], genetic and evolutionary algorithms [145], and machine learning and neural networks [146], to name a few. However, it is commonly accepted that SCP methods are

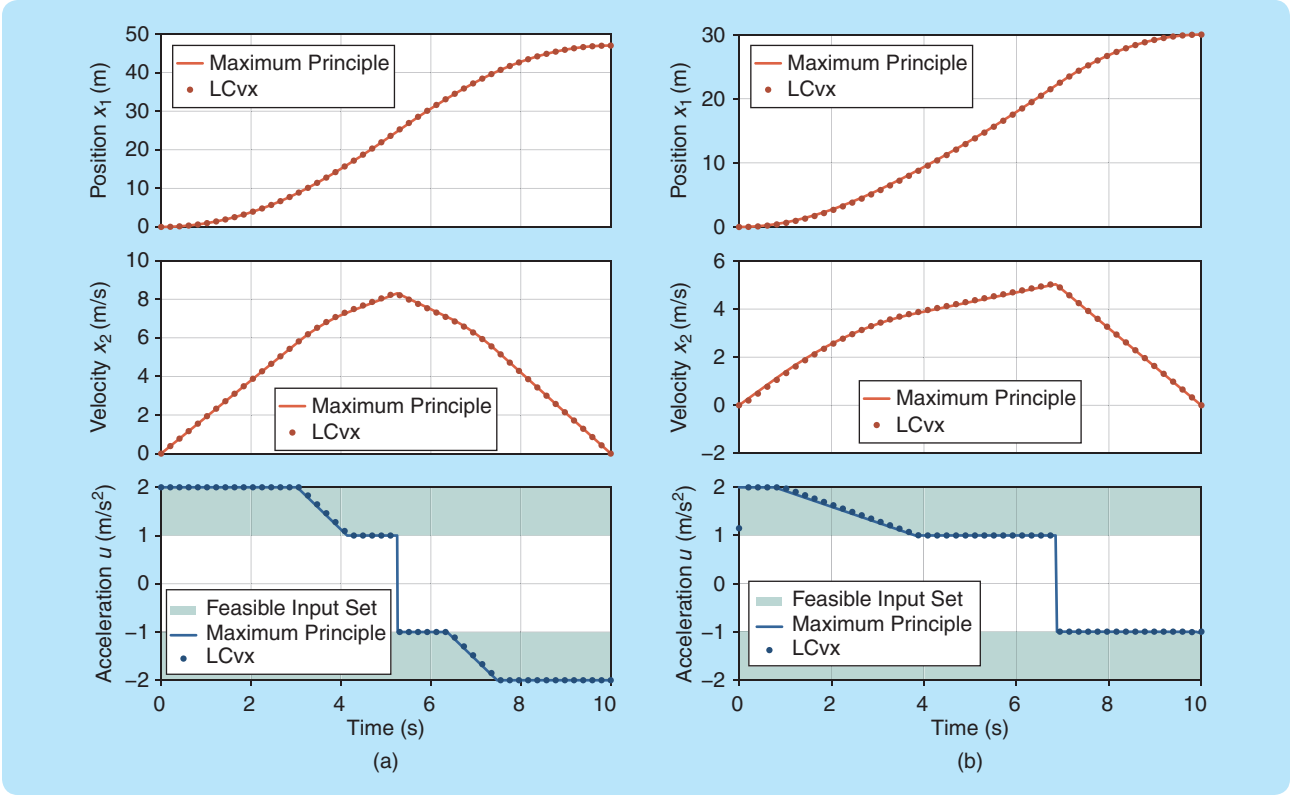


FIGURE 10 The lossless convexification (LCvx) solutions of (34) for two scenarios. The close match of the analytic solution using the maximum principle (drawn as a continuous line) and the discretized solution using LCvx (drawn as discrete dots) confirms that LCvx finds the globally optimal solution of the problem. (a) The solution of (34) for g = 0.1 m/s^2 and s = 47 m. (b) The solution of (34) for g = 0.6 m/s^2 and s = 30 m.

fast, flexible, and efficient local optimization algorithms for trajectory generation. They are a powerful tool to have in a trajectory engineer's toolbox, and they will be the focus of this part of the article.

As the name suggests, at the core of SCP is the idea of iterative convex approximation. Most, if not all, SCP algorithms for trajectory generation can be cast in the form demonstrated by Figure 11. Strictly speaking, SCP methods are nonlinear local optimization algorithms. In particular, the reader will recognize that SCP algorithms are specialized trust region methods for continuous-time optimal control problems [29], [47], [48].

All SCP methods solve a sequence of convex approximations, called *subproblems*, to the original nonconvex problem and update the approximation as new solutions are obtained. Going around the loop of Figure 11, all algorithms start with a user-supplied initial guess, which can be very coarse (more on this later). At ①, the SCP algorithm has available a so-called reference trajectory, which may be infeasible with respect to the problem dynamics and constraints. The nonconvexities of the problem are removed by a local linearization around the reference trajectory, while convex elements are kept unchanged. Well-designed SCP algorithms add extra features to the problem to maintain subproblem feasibility after linearization. The resulting convex continuous-time subproblem is then temporally discretized to yield a finite–dimensional convex optimization problem. The optimal solution to the discretized subproblem is computed at ②, where the SCP algorithm makes a call to any appropriate convex optimization solver. The solution is tested at ③ against stopping criteria. If the test passes, the algorithm has *converged*, and the most recent solution from ② is returned. Otherwise, the solution is used to update the trust region (and possibly other parameters) that are internal to the SCP algorithm. The solution then becomes the new reference trajectory for the next iteration of the algorithm.

The SCP approach offers two main advantages. First, a wide range of algorithms exists to reliably solve each convex subproblem at ②. Because SCP is agnostic to the particular choice of subproblem optimizer, well-tested algorithms can be used. This makes SCP very attractive for safety-critical applications, which are ubiquitous throughout disciplines such as aerospace and automotive engineering. Second, one can derive meaningful theoretical guarantees for algorithm performance and computational complexity, as opposed to general NLP optimization, where the convergence guarantees are much weaker. Taken together, these advantages have led to the development of very efficient SCP algorithms with runtimes short enough to enable real-time deployment for some applications [63].

A fundamental dilemma of NLP optimization is that one can either compute locally optimal solutions quickly or globally optimal solutions slowly. SCP techniques are not immune to this tradeoff despite the fact that certain subclasses of convex optimization can be viewed as "easy" from a computational perspective, due to the availability of IPMs. Some of the aforementioned applications may favor the ability to compute solutions quickly (that is, in near real time), such as aerospace and power grid technologies. Others, such as economics and structural truss design, may favor global optimality and put less emphasis on solution time (although early trade studies may still benefit from a fast optimization method). Given the motivation from the beginning of this article, the focus is on the former class of algorithms that provide locally optimal solutions in near real time.

This part of the article provides an overview of the algorithmic design choices and assumptions that lead to effective SCP implementations. The core tradeoffs include how the convex approximations are formulated, what structure is devised for updating the solutions, how progress toward a solution is measured, and how all of these enable theoretical convergence and performance guarantees. The goal is for the reader to develop the following view of SCP: it is an effective and flexible way to do trajectory optimization that inherits some but not all of the theoretical properties of convex optimization. SCP works well for complex problems; however, it is definitely not a panacea for all of nonconvex optimization. SCP can fail to find a solution, but a slight change to the algorithm parameters usually recovers feasibility and local optimality. This part of the article provides the reader with all the necessary insights to get started with SCP. The numerical examples in the "Application Examples" section provide a practical and open source implementation of the algorithms herein.

Historical Development of Sequential Convex Programming

Tracing the origins of SCP is not a simple task. Since the field of nonlinear programming gained traction as a popular discipline in the 1960s and 1970s, many researchers have explored the solution of nonconvex optimization problems via convex approximations. This section attempts to catalog some of the key developments, with a focus on providing insight into how the field moved toward the present version of SCP for trajectory generation.

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 [147]–[150]. Early results were of mainly academic interest, and computationally tractable methods remained elusive. One of the most important ideas that emerged from these early investigations appears to be that of McCormick relaxations [151]. These are a set of atomic rules for constructing convex/concave relaxations of a specific class of functions that everywhere under/overestimate the original functions. These rules result in a class of SCP methods, and algorithms based on McCormick relaxations

continue to be developed with increasing computational capabilities [152]–[155].

Difference-of-convex programming is a related class of SCP methods [156], [157]. These types of algorithms rely on the formulation of nonconvex constraints as the difference between two convex functions, say, $f = f_1 - f_2$, where both f_1 and f_2 are convex functions. The advantage of this decomposition is that only the function f_2 must be linearized to approximate the nonconvex function f. The convex–concave procedure presented in [158] is one example of a successful implementation of this idea, and it has been applied (among other places) in machine learning to support vector machines and principal component analysis [159].

Perhaps the earliest and simplest class of SCP methods whose structure resembles the one in Figure 11 is, unsurprisingly, sequential linear programming (SLP). These algorithms linearize all nonlinear functions about a current reference solution so that each subproblem is a linear program. These linear programs are then 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-scale problems [160], [161]. 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 computational resources at ② in Figure 11 has become unnecessary except, perhaps, for very large-scale problems.

Another important class of SCP methods is that of sequential quadratic programming (SQP). The works of Han [162], [163], Powell [164]–[166], Boggs and Tolle [167]–[169], and Fukushima [170] 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 [171], and an exhaustive monograph is available by Conn et al. [47]. SQP methods approximate a nonconvex program with a quadratic program using some reference solution, then use the solution to this quadratic program to update the approximation. Byrd et al. provide a general theory for inexact SQP methods [172]. The proliferation of SQP-type algorithms

Discretizing Continuous-Time Optimal Control Problems

Assume a continuous-time linear time-varying (LTV) dynamical system governed by the ordinary differential equation (ODE) (44b). To temporally discretize this system, choose a set of N temporal nodes

$$0 = t_1 < t_2 < \cdots < t_N = 1 \tag{S27}$$

that do not need to be evenly spaced. The objective of any discretization method is to represent the state trajectory $x(\cdot) : [0,1] \to \mathbb{R}^n$ at each of these temporal nodes as a function of the state, control, and parameters at the temporal nodes. Mathematically, this transforms the ODE (44b) into an LTV difference equation.

There are countless ways to achieve this objective, and [191] provides a comparative study of several such methods for motion planning problems. When choosing a method, it is important to remember that not all discretizations perform equally well. Coarse discretization techniques can render a convex subproblem infeasible even if the original optimal control problem is feasible. Moreover, different discretization methods will change the sparsity properties of the resulting convex subproblem, impacting the computational requirements for solving each subproblem at stage ② in Figure 11.

In practice, two methods appear to be the most appropriate for sequential convex programming-based trajectory generation: pseudospectral and interpolating polynomial methods. Pseudospectral discretization has gained popularity after its original introduction to the optimal control community, by Vlassenbroeck [S3], [S4]. This method approximates both the state and control trajectories using a basis of so-called Lagrange interpolating polynomials. Time is discretized into a nonuniform temporal grid defined by the roots of a polynomial that is a member of a family of orthogonal polynomials [S5]–[S7], [28]. In contrast, interpolating polynomial methods approximate only the control signal. This enables the exact solution of the LTV system (44b) over each time interval, yielding an "exact" discrete-time representation [59]. Here, *exact* means that the continuous- and discrete-time states match at the temporal nodes. A key attribute of interpolating polynomial discretization is that upon convergence, a sequential convex programming solution satisfying the discretized dynamics (S32a) will exactly satisfy the original nonlinear differential equation (38b). On the other hand, pseudospectral methods have the advantage of being able to discretize a nonlinear system "directly," without first going through a linearization process to obtain the LTV system (44b). Pseudospectral and interpolating polynomial methods for trajectory generation are described in detail in [6] and [191].

As an example, consider a discretization approach from the class of interpolating polynomial methods. This method, called first-order hold (FOH) interpolation, provides a suitable balance between optimality, feasibility, and computation time. It was found to work well for many problems [37], [59], [191]. In FOH, the control signal $u(\cdot) : [0,1] \to \mathbb{R}^m$ is constrained to be a continuous and piecewise affine function of time. By defining the inputs at the temporal nodes, $\{u_k\}_{k=1}^N$, the continuous-time signal is obtained by linear interpolation inside each time interval $[t_k, t_{k+1}]$:

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 existence of algorithms to quickly and reliably solve quadratic programs. The iterates obtained by SQP algorithms can be interpreted either as solutions to quadratic programs or the application of Newton's method to the optimality conditions of the original problem [29]. SQP algorithms are arguably the most mature class of SCP methods [173], and modern developments continue to address both theoretical and applied aspects [174], [175].

Their long history of successful deployment in NLP solvers notwithstanding [173], SQP methods do come with several drawbacks. Gill and Wong nicely summarize the difficulties that can arise when using SQP methods [176], and only the basic ideas are outlined here. Most importantly (and the same applies for any "second-order" method), it is difficult to accurately and reliably estimate the Hessian of the nonconvex program's Lagrangian. Even if this is done (for example, analytically), there is no guarantee that it will be positive semidefinite, and an indefinite Hessian results 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 Broyden–Fletcher–Goldfarb–Shanno update [177]. 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 curtail the real-time applicability of an SQP-type algorithm. Fortunately, a great deal of effort has gone into making SQP algorithms highly practical, resulting in mature algorithm packages, such as the Sparse Nonlinear OPTimizer (SNOPT) [173].

One truly insurmountable drawback of SQP methods for trajectory generation in particular is that quadratic programs require all constraints to be affine in the solution variables. Alas, many motion planning problems are naturally subject to nonaffine convex constraints. An example of a second-order cone constraint was already presented that arises from a spacecraft glideslope requirement in "Landing Glideslope as an Affine State Constraint," as illustrated in

$$u(t) = \frac{t_{k+1}-t}{t_{k+1}-t_k}u_k + \frac{t-t_k}{t_{k+1}-t_k}u_{k+1}, \triangleq \sigma_-(t)u_k + \sigma_+(t)u_{k+1}. \tag{S28}$$

The dynamics (44b) can then be expressed for $t \in [t_k, t_{k+1}]$ as

$$\dot{x}(t) = A(t)x(t) + B(t)\sigma_-(t)u_k + B(t)\sigma_+(t)u_{k+1} + F(t)p + r(t), \tag{S29}$$

which is again an LTV system except for the fact that the control is no longer a continuous-time function $u(\cdot)$. Instead, it is fully determined by the two vectors $u_k, u_{k+1} \in \mathbb{R}^m$. A standard result from linear systems theory is that the unique solution of (S29) is given by [88]:

$$x(t) = \Phi(t,t_k)x(t_k) + \int_{t_k}^{t} \Phi(t,\tau)[B(\tau)\sigma_-(\tau)u_k + B(\tau)\sigma_+(\tau)u_{k+1} + F(\tau)p + r(\tau)]d\tau, \tag{S30}$$

where $\Phi(t,t_k)$ is the state transition matrix that satisfies the following matrix differential equation:

$$\dot{\Phi}(t,t_k) = A(t)\Phi(t,t_k), \;\; \Phi(t_k,t_k) = I_n. \tag{S31}$$

By setting $t = t_{k+1}$, (S30) creates an LTV difference equation that updates the discrete-time state $x_k = x(t_k)$ to the next discrete-time state $x_{k+1} = x(t_{k+1})$:

$$x_{k+1} = A_k x_k + B_k^- u_k + B_k^+ u_{k+1} + F_k p + r_k, \tag{S32a}$$

$$A_k = \Phi(t_{k+1}, t_k), \tag{S32b}$$

$$B_k^- = A_k \int_{t_k}^{t_{k+1}} \Phi(\tau,t_k)^{-1} B(\tau)\sigma_-(\tau)\,d\tau, \tag{S32c}$$

$$B_k^+ = A_k \int_{t_k}^{t_{k+1}} \Phi(\tau,t_k)^{-1} B(\tau)\sigma_+(\tau)\,d\tau, \tag{S32d}$$

$$F_k = A_k \int_{t_k}^{t_{k+1}} \Phi(\tau,t_k)^{-1} F(\tau)\,d\tau, \tag{S32e}$$

$$r_k = A_k \int_{t_k}^{t_{k+1}} \Phi(\tau,t_k)^{-1} r(\tau)\,d\tau. \tag{S32f}$$

In a practical implementation, the state transition matrix, the integrals (S32c)–(S32f), and the reference trajectory's state $\bar{x}(\cdot)$ are computed simultaneously over each interval $[t_k, t_{k+1}]$. This procedure is explained in more detail in [191], and pseudocode can be found in [63]. Ultimately, the update equation (S32a) is used to write $N-1$ affine equality constraints for $k = 1, \ldots, N-1$, that represent the discretized dynamic feasibility constraint on the trajectory. Equation (54b) provides an example of embedding such constraints into an optimization problem.

REFERENCES

[S3] J. Vlassenbroeck, "A Chebyshev polynomial method for optimal control with state constraints," *Automatica*, vol. 24, no. 4, pp. 499–506, 1988, doi: 10.1016/0005-1098(88)90094-5.

[S4] J. Vlassenbroeck and R. Van Dooren, "A Chebyshev technique for solving nonlinear optimal control problems," *IEEE Trans. Autom. Control*, vol. 33, no. 4, pp. 333–340, 1988, doi: 10.1109/9.192187.

[S5] C. de Boor and B. Swartz, "Collocation at Gaussian points," *SIAM J. Numer. Anal.*, vol. 10, no. 4, pp. 582–606, Sep. 1973, doi: 10.1137/0710052.

[S6] V. Rao, "A survey of numerical methods for optimal control," *Adv. Astronaut. Sci.*, vol. 135, no. 1, pp. 497–528, 2010.

[S7] M. Ross and M. Karpenko, "A review of pseudospectral optimal control: From theory to flight," *Annu. Rev. Contr.*, vol. 36, no. 2, pp. 182–197, Dec. 2012, doi: 10.1016/j.arcontrol.2012.09.002.

Figure S6. For problems with nonaffine convex constraints, the use of an SQP algorithm may require more iterations to converge compared to a more general SCP algorithm, leading to a reduction in computational efficiency. Moreover, since nonaffine convex constraints are linearized, each SQP iterate is not guaranteed to be feasible with respect to the original convex constraints, whereas the SCP iterates will be.

There are several classes of SCP algorithms that generalize the idea of SQP to address this limitation. Semidefinite programs (SDPs) are the most general class of convex optimization problems for which efficient off-the-shelf solvers are available. Fares et al. introduced sequential semidefinite programming [178], which uses matrix variables that are subject to definiteness constraints. Such algorithms find application most commonly in robust control, where problems are formulated as (nonconvex) SDPs with linear and bilinear matrix inequalities [179]. Recent examples have appeared for robust planetary rocket landing [180], [181]. Sequential semidefinite programming can be viewed as the furthest possible generalization of SLP to the idea of exploiting existing convexity in the subproblems.

This article focuses 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 [for example, LPs, QPs, second-order cone programs (SOCPs), and SDPs]. This class of SCP methods has been developed largely over the past decade and represents the most active area of current development, with successful applications in robot and spacecraft trajectory optimization [6], [53], [59], [62], [63], [123]–[125], [182]–[184].

Two specific algorithms are discussed that belong to this class of SCP methods: SCvx and GuSTO. These two algorithms are complementary in a number of ways and enjoy favorable theoretical guarantees. The theoretical analysis of SCvx works with the temporally discretized problem and provides guarantees in terms of the Karush–Kuhn–Tucker (KKT) optimality conditions [49], [64], [185]. On the other hand, GuSTO is analyzed for the continuous-time problem and provides theoretical guarantees in terms of the Pontryagin maximum principle [11], [12], [50], [69]. The trajectory problems in the "Application Examples" section at the end of this article are solved using both SCvx and GuSTO exactly as they are presented here. These examples illustrate that the methods are, to some degree, interchangeable.

Typically, although not necessarily, the convex solver used at ② in Figure 11 is based on an IPM [31], [142]. This leads to a nice interpretation of SCP as the "next layer up" in a hierarchy of optimization algorithms described in [26, Ch. 11] and illustrated in Figure 12. The bottommost layer contains the unconstrained Newton's method, which solves a sequence of unconstrained QPs. The next layer solves linear equality constrained convex problems. This again uses Newton's method but with a more complicated step computation. The third layer is the IPM family of methods that solve a convex problem with linear equality and convex inequality constraints as a sequence of linear equality constrained problems. Thus, IPMs iteratively call the algorithm in layer ② of Figure 12. Analogously, SCP solves a nonconvex problem as a sequence of convex problems with linear equality and convex inequality constraints. Thus, SCP iteratively calls an IPM algorithm from layer ③. Numerical experience has shown that for most problems, IPMs require on the order of tens of iterations (that is, calls to layer ②) to converge [26]. Similarly,

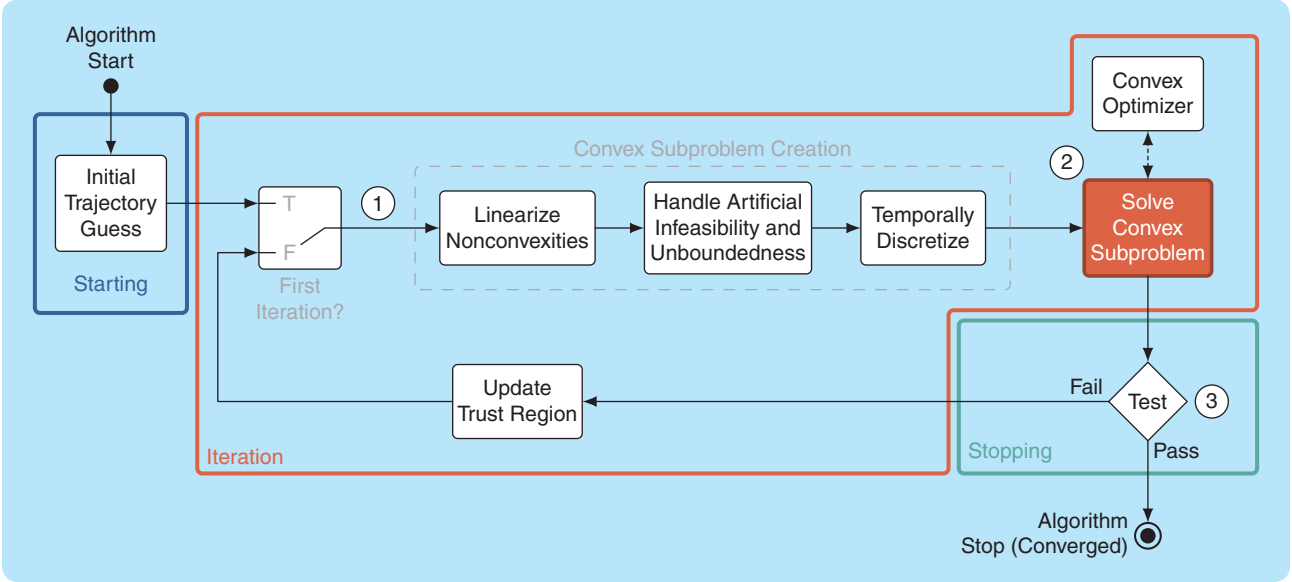


FIGURE 11 A typical sequential convex programming (SCP) algorithm. Every SCP-based trajectory generation method is composed of three major components: a way to guess the initial trajectory ("Starting"), an iteration scheme that refines the trajectory until it is feasible and locally optimal ("Iteration"), and an exit criterion to stop once a sufficiently accurate trajectory has been computed ("Stopping"). In a well-designed SCP scheme, the test (convergence) criterion is guaranteed to trigger, but the solution may be infeasible for the original problem.

our experience has been that SCP requires on the order of tens of iterations (that is, calls to layer ③) to converge.

The rest of this part of the article is organized as follows. First, the general continuous-time optimal control problem is described, and the common algorithmic underpinnings of SCP are discussed. The SCvx and GuSTO algorithms are then described in full detail. The end of this part compares SCvx and GuSTO and gives some advice on using SCP in the real world. The "Application Examples" section that follows presents two numerical experiments that provide practical insight and highlight the capabilities of each algorithm.

Problem Formulation

The goal of SCP methods is to solve continuous-time optimal control problems of the following form:

$$\min_{u,p} \; J(x,u,p), \tag{38a}$$

$$\text{s.t. } \dot{x}(t) = f(t,x(t),u(t),p), \tag{38b}$$

$$(x(t),p) \in \mathcal{X}(t), \tag{38c}$$

$$(u(t),p) \in \mathcal{U}(t), \tag{38d}$$

$$s(t,x(t),u(t),p) \le 0, \tag{38e}$$

$$g_{\text{ic}}(x(0),p) = 0, \tag{38f}$$

$$g_{\text{tc}}(x(1),p) = 0, \tag{38g}$$

where $x(\cdot) \in \mathbb{R}^n$ is the state trajectory, $u(\cdot) \in \mathbb{R}^n$ is the control trajectory, and $p \in \mathbb{R}^d$ is a vector of parameters. The function $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^n$ represents the (nonlinear) dynamics, which are assumed to be at least once continuously differentiable. Initial and terminal boundary conditions are enforced by using the continuously differentiable functions $g_{\text{ic}} : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^{n_{\text{ic}}}$ and $g_{\text{tc}} : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^{n_{\text{tc}}}$. Convex and nonconvex path (that is, state and control) constraints are separated by using the convex sets $\mathcal{X}(t)$ and $\mathcal{U}(t)$ to represent convex path constraints, and the continuously differentiable function $s : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^{n_s}$ represents nonconvex path constraints. It is assumed that the sets $\mathcal{X}(t)$ and $\mathcal{U}(t)$ are compact (that is, closed and bounded). This means the vehicle cannot escape to infinity or apply infinite control action, which is obviously reasonable for all practical applications. Finally, (38) is defined on the $[0,1]$ time interval, and the constraints (38b)–(38e) must hold at each time instant.

Note that the parameter vector p can be used, among other things, to capture free initial and/or free final time problems by making t_0 and t_f elements of p. In particular, an appropriate scaling of time can transform the $[0,1]$ time interval in (38) into a $[t_0, t_f]$ interval. Therefore the problem statement's restriction to the $[0,1]$ time interval is without loss of generality [27]. This transformation is used in the numerical examples at the end of the article.

Hybrid systems, such as bouncing balls, colliding objects, and bipedal robots, require integer variables in their optimization models. The integer variable type, however, is missing from (38). Nevertheless, methods exist to embed integer variables into the continuous-variable formulation. Among these methods are state-triggered constraints [59], [61], [62], [65], [66], [111], [186] and homotopy techniques, such as the relaxed autonomously switched hybrid system and composite smooth control [6], [187], [188]. This work therefore moves forward using (38) "without a loss of generality," noting that there are methods to embed integer solution variables exactly or as an arbitrarily accurate approximation [6].

Note that (38) is not the most general optimal control problem that SCP methods can solve. However, it is general enough for the introductory purpose of this article and can already cover the vast majority of trajectory optimization problems [6]. The numerical implementation code associated with this article (see Figure 2) was applied to solve problems ranging from quadrotor trajectory generation to spacecraft rendezvous and docking [66], [111].

The cost function in (38a) is assumed to be of the Bolza form [104]:

$$J(x,u,p) = \phi(x(1),p) + \int_0^1 \Gamma(x(t),u(t),p)\,\mathrm{d}t, \tag{39}$$

where the terminal cost $\phi : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}$ is assumed to be a convex function and the running cost $\Gamma : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}$ can be, in general, a nonconvex function. Note that convexity assumptions on ϕ are without a loss of generality. For example, a nonconvex ϕ can be replaced by a linear terminal cost τ_{f} (where τ_{f} becomes an element of p), and a nonconvex terminal boundary condition is added to the definition of g_{tc} in (38g):

$$\phi(x(1),p) = \tau_{\text{f}}. \tag{40}$$

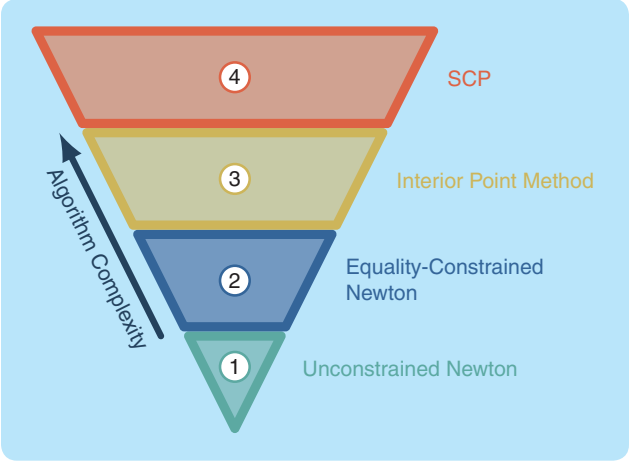


FIGURE 12 Sequential convex programming (SCP) can be placed atop of a hierarchy of classical optimization algorithms. Here, the "width" of each layer is representative of the corresponding algorithm's implementation and runtime complexity (to be used only as an intuitive guide). Each layer embeds within itself the algorithms from the layers below it.

Sequential Convex Programming Algorithm Foundations

All SCP methods work by solving a sequence of local convex approximations to (38), which are called *subproblems*. As shown in Figure 11, this requires access to an existing reference trajectory at location ①. This is called a *reference solution*, with the understanding that this trajectory need not be a feasible solution to the problem (neither for the dynamics nor for the constraints). SCP methods update this reference solution after each passage around the loop of Figure 11, with the solution obtained at ② becoming the reference for the next iteration. This begs the question: Where does the reference solution for the first iteration come from?

Initial Trajectory Guess

A user-supplied initial trajectory guess is responsible for providing the first SCP iteration with a reference solution. Henceforth, the notation $\{\bar{x}(t), \bar{u}(t), \bar{p}\}_0^1$ denotes a reference trajectory on the time interval $[0, 1]$. It is shown in the following sections that the SCP algorithms SCvx and GuSTO are guaranteed to converge almost regardless of the initial trajectory guess. In particular, this guess can be grossly infeasible with respect to both the dynamics (38b) and the nonconvex constraints (38e)–(38g). However, the algorithms do require the guess to be feasible with respect to the convex path constraints (38c) and (38d). Assuring this is almost always an easy task, either by manually constructing a simplistic solution that respects the convex constraints or by projecting an infeasible guess onto the $\mathcal{X}(t)$ and $\mathcal{U}(t)$ sets. For reference, both strategies are implemented in the open source code linked in Figure 2.

Numerical experience has shown that both SCvx and GuSTO are extremely adept at morphing coarse initial guesses into feasible and locally optimal trajectories. This represents a significant algorithmic benefit since most traditional methods, including SQP and NLP, require good (or even feasible) initial guesses, which can be very hard to obtain [27], [28]. To give an idea of what kind of initial guess can be provided, an initialization method called *straight-line interpolation* is presented. It is observed that this technique works well for a wide variety of problems, and it is used in the numerical examples at the end of this article. However, this is merely a rule of thumb and not a rigorously derived technique.

Begin by fixing the initial and final states x_{ic} and x_{tc} that represent either single-point boundary conditions or points in a desired initial and terminal set defined by (38f) and (38g). The state trajectory is then defined as a linear interpolation between the two endpoints:

$$\bar{x}(t) = (1-t)\,x_{\text{ic}} + t\,x_{\text{tc}}, \quad \text{for } t \in [0, 1]. \tag{41}$$

If a component of the state is a nonadditive quantity, such as a unit quaternion, then linear interpolation is not the most astute choice. In such cases, the simplest alternative to linear interpolation is chosen. For unit quaternions, this would be spherical linear interpolation (SLERP) [189].

Whenever possible, select the initial input trajectory based on insight from the physics of the problem. For example, for an aerial vehicle, choose an input that opposes the pull of gravity. In the case of a rocket, the choice can be $u_{\text{ic}} = -m_{\text{wet}} g_{\mathcal{I}}$ and $u_{\text{tc}} = -m_{\text{dry}} g_{\mathcal{I}}$, where m_{wet} and m_{dry} are the initial and estimated final masses of the vehicle and $g_{\mathcal{I}}$ is the inertial gravity vector. If the problem structure does not readily admit a physics-based choice of control input, the go-to approach is to set the input to the smallest feasible value that is compliant with (38d). The intuition is that small inputs are often associated with a small cost (38a). The initial control solution is interpolated using an expression similar to (41):

$$\bar{u}(t) = (1-t)\,u_{\text{ic}} + t\,u_{\text{tc}}, \quad \text{for } t \in [0, 1]. \tag{42}$$

The initial guess for $\bar{p}$ can have a significant impact on the number of SCP iterations required to obtain a solution. For example, if $\bar{p}$ represents the final time of a free final time problem that evolves on the $[0, t_f]$ interval, then it is best to guess a time dilation value that is reasonable for the expected trajectory. Since parameters are inherently problem specific, however, it is unlikely that any generic rule of thumb akin to (41) and (42) will prove reliable. Fortunately, since SCP runtime is usually on the order of a few seconds or shorter, the user can experiment with different initial guesses for $\bar{p}$ and identify a good initialization strategy relatively quickly.

For all but the simplest problems, the initial guess $\{\bar{x}(t), \bar{u}(t), \bar{p}\}_0^1$ constructed in the preceding is going to be (highly) infeasible with respect to the dynamics and constraints of (38). Nevertheless, SCP methods, such as SCvx and GuSTO, can, and often do, converge to usable trajectories through such a coarse initial guess. However, this does not relieve the user entirely from choosing an initial guess that exploits the salient features of the particular problem. A well-chosen initial guess will (likely) have the following three benefits for the solution process:

1) It will reduce the number of iterations and the time required to converge. This is almost always a driving objective in the design of a trajectory optimization algorithm since fast convergence is not only a welcome feature but also a hard requirement for onboard implementation in an autonomous system.
2) It will encourage the converged solution to be feasible for (38). As mentioned, SCP methods, including SCvx and GuSTO, will always converge to a trajectory but without a guarantee that the trajectory will be feasible for the original problem. The fact that the solution often *is* feasible with respect to (38) is a remarkable "observation" that researchers and engineers have made, and it is a driving reason for the modern interest in SCP methods. Nevertheless, an observation is not a proof, and there are limits to how bad an initial guess can be. The only rule of thumb that is always

valid is that one should embed as much problem knowledge as possible in the initial guess.

3) A better initial guess may also improve the converged trajectory's optimality. However, the level of optimality is usually difficult to measure because a globally optimal solution is rarely available for the kinds of difficult trajectory problems that are typically solved using SCP. Nevertheless, some attempts to characterize the optimality level have been made in recent years [63], [190].

Linearization

Now consider location ① in Figure 11, and imagine that the algorithm is at some iteration during the SCP solution process. The first task for constructing a convex subproblem is to remove the nonconvexities of (38). For this purpose, recall that the algorithm has access to the reference trajectory $\{\bar{x}(t), \bar{u}(t), \bar{p}\}_0^1$. If every nonconvexity is replaced by its first-order approximation around the reference trajectory, then the resulting subproblem is guaranteed to be convex. Furthermore, first-order approximations are computationally inexpensive to compute relative to second-order approximations (that is, those involving Hessian matrices). As noted in the previous section on SCP history, the linearization of all nonconvex elements is not the only choice for SCP; it is simply a very common one, and it is used for this article.

To formulate the linearized nonconvex terms, the following Jacobians must be computed (the time argument is omitted where necessary to keep the notation short):

$$A(t) \triangleq \nabla_x f(t, \bar{x}(t), \bar{u}(t), \bar{p}), \tag{43a}$$

$$B(t) \triangleq \nabla_u f(t, \bar{x}(t), \bar{u}(t), \bar{p}), \tag{43b}$$

$$F(t) \triangleq \nabla_p f(t, \bar{x}(t), \bar{u}(t), \bar{p}), \tag{43c}$$

$$r(t) \triangleq f(t, \bar{x}(t), \bar{u}(t), \bar{p}) - A\bar{x}(t) - B\bar{u}(t) - F\bar{p}, \tag{43d}$$

$$C(t) \triangleq \nabla_x s(t, \bar{x}(t), \bar{u}(t), \bar{p}), \tag{43e}$$

$$D(t) \triangleq \nabla_u s(t, \bar{x}(t), \bar{u}(t), \bar{p}), \tag{43f}$$

$$G(t) \triangleq \nabla_p s(t, \bar{x}(t), \bar{u}(t), \bar{p}), \tag{43g}$$

$$r'(t) \triangleq s(t, \bar{x}(t), \bar{u}(t), \bar{p}) - C\bar{x}(t) - D\bar{u}(t) - G\bar{p}, \tag{43h}$$

$$H_0 \triangleq \nabla_x g_{\mathrm{ic}}(\bar{x}(0), \bar{p}), \tag{43i}$$

$$K_0 \triangleq \nabla_p g_{\mathrm{ic}}(\bar{x}(0), \bar{p}), \tag{43j}$$

$$\ell_0 \triangleq g_{\mathrm{ic}}(\bar{x}(0), \bar{p}) - H_0\bar{x}(0) - K_0\bar{p}, \tag{43k}$$

$$H_{\mathrm{f}} \triangleq \nabla_x g_{\mathrm{tc}}(\bar{x}(1), \bar{p}), \tag{43l}$$

$$K_{\mathrm{f}} \triangleq \nabla_p g_{\mathrm{tc}}(\bar{x}(1), \bar{p}), \tag{43m}$$

$$\ell_{\mathrm{f}} \triangleq g_{\mathrm{tc}}(\bar{x}(1), \bar{p}) - H_{\mathrm{f}}\bar{x}(1) - K_{\mathrm{f}}\bar{p}. \tag{43n}$$

These matrices can be used to write the first-order Taylor series approximations for f, s, g_{ic}, and g_{tc}. Note that the cost function (39) is not linearized at this point since SCvx and GuSTO make different assumptions about its particular form. The convexification of the cost function will be addressed separately in later sections on SCvx and GuSTO.

Using the terms in (43) yields the following approximation of (38) around the reference trajectory:

$$\min_{u,p} \; J(x, u, p), \tag{44a}$$

$$\text{s.t. } \dot{x}(t) = A(t)x(t) + B(t)u(t) + F(t)p + r(t), \tag{44b}$$

$$(x(t), p) \in \mathcal{X}(t), \tag{44c}$$

$$(u(t), p) \in \mathcal{U}(t), \tag{44d}$$

$$C(t)x(t) + D(t)u(t) + G(t)p + r'(t) \leq 0, \tag{44e}$$

$$H_0x(0) + K_0p + \ell_0 = 0, \tag{44f}$$

$$H_{\mathrm{f}}x(1) + K_{\mathrm{f}}p + \ell_{\mathrm{f}} = 0. \tag{44g}$$

Equation (44) is convex in the constraints and potentially nonconvex in the cost. Note that the convex path constraints in (38c) and (38d) do not require an approximation. This is a key advantage of SCP over methods such as SLP and SQP, as discussed in the previous section on SCP history.

Because the control trajectory $u(\cdot)$ belongs to an infinite–dimensional vector space of continuous-time functions, (44) cannot be implemented and solved numerically on a digital computer. To do so, consider a finite–dimensional representation of the control function $u(t)$, which can be obtained via temporal discretization and direct collocation [27], [191]. These representations turn the original infinite–dimensional optimal control problem into a finite–dimensional parameter optimization problem that can be solved on a digital computer.

In general, and rather unsurprisingly, solutions to discretized problems are only approximately optimal and feasible with respect to the original problem. In particular, a discrete-time control signal has fewer degrees of freedom (DoF) than its continuous-time counterpart. Therefore, it may lack the flexibility required to exactly match the true continuous-time optimal control signal. By adding more temporal nodes, the approximation can become arbitrarily accurate, albeit at the expense of problem size and computation time. Another problem with discretization is that the path constraints are usually enforced only at the discrete temporal nodes and not over the entire time horizon. This can lead to (typically mild) constraint violation between the discrete-time nodes, although some techniques exist to remedy this artifact [192], [193].

The bad news notwithstanding, there are well-established discretization methods that ensure the exact satisfaction of the original continuous-time nonlinear dynamics (38b). Thus, the discretized solution can still produce strictly dynamically feasible continuous-time trajectories. Refer to [6], [59], [62], and [191] for detailed explanations of discretization methods that ensure the exact satisfaction of the continuous-time nonlinear dynamics. An introduction to the technique that is used for the numerical examples at the end of this article is given in "Discretizing Continuous-Time Optimal Control Problems."

The systematic linearization of all nonconvex elements has ensured that (44) is convex in the constraints, which is good

news. However, linearization unsurprisingly has a price. It has inadvertently introduced two artifacts that must be addressed: artificial unboundedness and artificial infeasibility.

Artificial Unboundedness

Linear approximations are accurate only in a neighborhood around the reference solution $\{\bar{x}(t), \bar{u}(t), \bar{p}\}_0^1$. Thus, for each $t \in [0, 1]$, the subproblem solution must be kept "sufficiently close" to the linearization point defined by the reference solution. Another reason to not deviate too far from the reference is that, in certain malicious cases, linearization can render the solution unbounded below [that is, the convex cost (44a) can be driven to negative infinity]. This phenomenon is referred to as *artificial unboundedness*. To mitigate this problem and quantify the meaning of "sufficiently close," add the following trust region constraint:

$$\begin{aligned} &\delta x(t) = x(t) - \bar{x}(t), \\ &\delta u(t) = u(t) - \bar{u}(t), \\ &\delta p = p - \bar{p}, \\ &\alpha_x \| \delta x(t) \|_q + \alpha_u \| \delta u(t) \|_q + \alpha_p \| \delta p \|_q \leq \eta, \text{ for } t \in [0, 1], \end{aligned} \quad (45)$$

for some choice of $q \in \{1, 2, 2^+, \infty\}$ and constants $\alpha_x, \alpha_u, \alpha_p \in \{0, 1\}$. Use $q = 2^+$ to denote the ability to impose the trust region as the quadratic two-norm squared. The trust region radius η is a fixed scalar that is updated between SCP iterations (that is, passages around the loop in Figure 11). The update rule associated with the trust region measures how well the linearization approximates the original nonconvex elements at each iterate. This informs the algorithm whether to shrink, grow, or maintain the trust region radius. SCP methods can be differentiated by how they update the trust region, and this is discussed separately for SCvx and GuSTO in the upcoming sections.

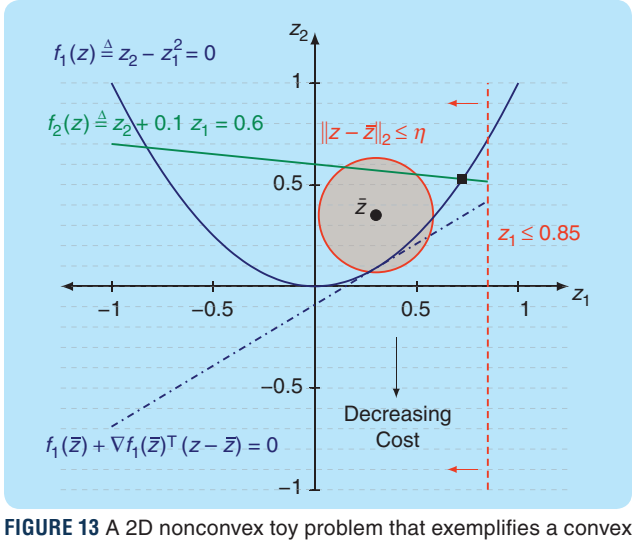


FIGURE 13 A 2D nonconvex toy problem that exemplifies a convex subproblem obtained during a sequential convex programming iteration. In this case, the cost function $J(z) = z_2$ and level curves of the cost are shown as gray dashed lines. The blue curve represents a nonconvex equality constraint, and its linearization is shown as the blue dash–dot line. Another convex equality constraint is shown in green, and a convex inequality constraint is shown as the vertical, red dashed line. The trust region is the red circle centered at the linearization point $\bar{z}$, and has radius η. The optimal solution of the original (nonconvex) problem is shown as the black square. The convex subproblem is artificially infeasible. Without the trust region and green constraint, it would also be artificially unbounded.

Figure 13 shows a 2D toy problem that exemplifies a single iteration of an SCP convergence process. In this example, the "original problem" consists of one parabolic (nonconvex) equality constraint (blue), a convex equality constraint (green), and a convex half-space inequality constraint (feasible to the left of the vertical red, dashed line). The original problem is approximated around the reference solution $\bar{z}$, resulting in the blue dash–dot equality constraint and the same convex equality and inequality constraints. The trust region is shown as the red circle and represents the region in which the SCP algorithm has deemed the convex approximation valid. If the new solution z deviates too much from $\bar{z}$, the linear approximation of the parabola becomes poor. Moreover, had the green equality constraint been removed, eliminating the trust region would lead to artificial unboundedness, as the cost could be decreased indefinitely.

Clearly, there is another problem with the linearization in Figure 13: the resulting subproblem is infeasible. This is because the green and blue dash–dot equality constraints do not intersect inside the set defined by the trust region and the convex inequality constraint half space. This issue is known as *artificial infeasibility*.

Artificial Infeasibility

Linearization can make the resulting subproblem infeasible. Two independent cases can arise wherein the constraints imposed in (44) are inconsistent (that is, no feasible solution exists), even though the original constraints admit a nonempty feasible set. The cases are as follows:

- In the first case, the intersection of the convexified path constraints may be empty. This occurs in the example of Figure 13, where no feasible solution exists because the linearized constraints (green and blue dash–dot lines) do not intersect to the left of the red inequality constraint.
- In the second case, the trust region may be so small that it restricts the solution variables to a part of the solution space that is outside of the feasible set. In other words, the intersection of the trust region with the (nonempty) feasible region of the convexified constraints may itself be empty. This would have been the case in Figure 13 if the green and blue dash–dot lines were to intersect outside of the red trust region circle but to the left of the half-space inequality (for example, if $\bar{z}$ were slightly farther to the right).

The occurrence of either case would prevent SCP from finding a new reference solution at ② in Figure 11. Thus, the solution loop would not be able to continue, and the algorithm would fail. As a result, even if the original problem admits a feasible solution (shown as the black square in Figure 13), either of the two aforementioned scenarios would prevent SCP from finding it. This phenomenon is referred to as *artificial infeasibility*.

This edge case was recognized early in the development of SCP algorithms [161], [164]. Two equivalent remedies exist. One approach adds an unconstrained, but penalized, slack variable to each linearized constraint. This variable is sometimes called a *virtual control* when applied to the dynamics constraint (38b) and a *virtual buffer* when applied to other constraints [49]. To keep the language succinct, both applications will be referred to as *virtual control*. The second approach penalizes constraint violations by augmenting the original cost function (44a) with *soft penalty* terms. When the same functions and weights are used to penalize the virtual control terms, this strategy results in the same optimality conditions. The ultimate result of both strategies is that the subproblem is guaranteed to be feasible.

Because the virtual control is a new term that is added to the problem, it follows that the converged solution must have a zero virtual control in order to be feasible and physically meaningful with respect to the original problem. If, instead, the second strategy is used and constraint violations are penalized in the cost, the converged solution must not violate the constraints (that is, the penalty terms should be zero). Intuitively, if the converged solution uses a nonzero virtual control or has a nonzero constraint violation penalty, then it is not a solution of the original optimal control problem.

The fact that trajectories can converge to an infeasible solution is one of the salient limitations of SCP. However, it is not unlike the drawback of any other NLP optimization method, which may fail to find a solution entirely, even if one exists. When SCP converges to an infeasible solution, it is called a "soft" failure since usually only a few virtual control terms are nonzero. A soft failure carries important information because the temporal nodes and constraints with nonzero virtual control hint at how and where the solution is infeasible. Usually, relatively mild tuning of the algorithm parameters or problem definition will recover convergence to a feasible solution. In relation to the optimization literature at large, the soft failure exhibited by SCP is related to one-norm regularization, lasso regression, and basis pursuit, all of which are used to find sparse approximate solutions [26, Sec. 11.4.1].

The specific algorithmic choices made to address artificial unboundedness (for example, selection of the trust region radius update rule) and artificial infeasibility (for example, virtual control versus constraint penalization) lead to SCP algorithms with different characteristics. The next two sections review two such algorithms, SCvx and GuSTO, and highlight their design choices and algorithmic properties. To facilitate a concise presentation, the time argument t is suppressed whenever possible.

The SCvx Algorithm

In light of the preceding discussion, the SCvx algorithm makes the following algorithmic choices:

- The terminal and running costs in (39) are both assumed to be convex functions. It was previously mentioned that this is without loss of generality for the terminal cost, and the same reasoning applies for the running cost. Any nonconvex term in the cost can be offloaded into the constraints, and an example was given in (40).
- To handle artificial unboundedness, SCvx enforces (45) as a hard constraint. While several choices are possible [49], [185], this article uses $\alpha_x = \alpha_u = \alpha_p = 1$. The trust region radius η is adjusted at each iteration by an update rule, which is discussed in this section.
- To handle artificial infeasibility, SCvx uses virtual control terms.

Begin by describing how SCvx uses virtual control to handle artificial infeasibility. This is done by augmenting the linear approximations (44b) and (44e)–(44g) as follows:

$$\dot{x} = Ax + Bu + Fp + r + E\nu, \tag{46a}$$

$$\nu_s \geq Cx + Du + Gp + r', \tag{46b}$$

$$0 = H_0 x(0) + K_0 p + \ell_0 + \nu_{\text{ic}}, \tag{46c}$$

$$0 = H_{\text{f}} x(1) + K_{\text{f}} p + \ell_{\text{f}} + \nu_{\text{tc}}, \tag{46d}$$

where $\nu(\cdot) \in \mathbb{R}^{n_\nu}$, $\nu_s(\cdot) \in \mathbb{R}^{n_s}$, $\nu_{\text{ic}} \in \mathbb{R}^{n_{\text{ic}}}$, and $\nu_{\text{tc}} \in \mathbb{R}^{n_{\text{tc}}}$ are the virtual control terms. To keep the notation manageable, use the symbol $\hat{\nu}$ as a shorthand for the argument list $(\nu, \nu_s, \nu_{\text{ic}}, \nu_{\text{tc}})$.

The virtual control ν in (46a) can be viewed simply as another control variable that can be used to influence the state trajectory. Like the other virtual control terms, the goal is for ν to be zero for any converged trajectory because it is a synthetic input that cannot be used in reality. Note that it is required that the pair (A, E) in (46a) is controllable, which is easy to verify using any of several available controllability tests [88]. A common choice is $E = I_n$, in which case (A, E) is unconditionally controllable.

The use of nonzero virtual control in the subproblem solution is discouraged by augmenting the cost function with a virtual control penalty term. Intuitively, this means that virtual control is used only when it is necessary to avoid subproblem infeasibility. To this end, define a positive definite penalty function $P : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}_+$, where p is any appropriate integer. The following choice is typical in practice:

$$P(y, z) = \|y\|_1 + \|z\|_1, \tag{47}$$

where y and z are placeholder arguments. The cost function (44a) is augmented using the penalty function, as follows:

$$J_\lambda(x, u, p, \hat{\nu}) \triangleq \phi_\lambda(x(1), p, \nu_{\text{ic}}, \nu_{\text{tc}}) + \int_0^1 \Gamma_\lambda(x, u, p, E\nu, \nu_s)\, \mathrm{d}t, \tag{48}$$

$$\phi_\lambda(x(1), p, \nu_{\text{ic}}, \nu_{\text{tc}}) = \phi(x(1), p) + \lambda P(0, \nu_{\text{ic}}) + \lambda P(0, \nu_{\text{tc}}),$$
$$\Gamma_\lambda(x, u, p, E\nu, \nu_s) = \Gamma(x, u, p) + \lambda P(E\nu, \nu_s). \tag{49}$$

The positive weight $\lambda \in \mathbb{R}_{++}$ is selected by the user and must be sufficiently large. This statement is made more precise later in the section on SCvx convergence guarantees. For now, note that in practice, it is quite easy to find an appropriately large λ value by selecting a power of 10. Although λ can generally be a function of time, this rarely appears in practice.

Observe an important notational feature of (49), where $E\nu$ is used in place of ν for the argument list of Γ_λ. This will help to highlight that the continuous-time matrix E is substituted with its discrete-time version after temporal discretization [see (52)]. The continuous-time convex subproblem that is solved at each iteration of the SCvx algorithm can then be stated formally as

$$\min_{u,p,\hat{\nu}} J_\lambda(x, u, p, \hat{\nu}), \tag{50a}$$
$$\text{s.t. } \dot{x} = Ax + Bu + Fp + r + E\nu, \tag{50b}$$
$$(x, p) \in \mathcal{X}, \quad (u, p) \in \mathcal{U}, \tag{50c}$$
$$Cx + Du + Gp + r' \le \nu_s, \tag{50d}$$
$$H_0 x(0) + K_0 p + \ell_0 + \nu_{\text{ic}} = 0, \tag{50e}$$
$$H_{\text{f}} x(1) + K_{\text{f}} p + \ell_{\text{f}} + \nu_{\text{tc}} = 0, \tag{50f}$$
$$\| \delta x \|_q + \| \delta u \|_q + \| \delta p \|_q \le \eta. \tag{50g}$$

It was mentioned in the previous section that (50) is not readily implementable on a computer because it is a continuous-time, hence, infinite–dimensional, optimization problem. To solve the problem numerically, a temporal discretization is applied, such as the one discussed in "Discretizing Continuous-Time Optimal Control Problems." In particular, select a set of temporal nodes $t_k \in [0, 1]$ for $k = 1, \ldots, N$, and recast the subproblem as a parameter optimization problem in the (overloaded) variables $x = \{x_k\}_{k=1}^N$, $u = \{u_k\}_{k=1}^N$, p, $\nu = \{\nu_k\}_{k=1}^{N-1}$, $\nu_s = \{\nu_{s,k}\}_{k=1}^N$, ν_{ic}, and ν_{tc}.

Depending on the details of the discretization scheme, there may be fewer decision variables than there are temporal nodes. For simplicity, a zeroth-order hold (ZOH) assumption (that is, a piecewise constant function) is used to discretize the dynamics virtual control $\nu(\cdot)$. This means that the virtual control takes the value $\nu(t) = \nu_k$ inside each time interval $[t_k, t_{k+1})$, and the discretization process works like any other interpolating polynomial method from "Discretizing Continuous-Time Optimal Control Problems." Because this leaves ν_N undefined, $\nu_N = 0$ is used for notational convenience whenever it appears in future equations.

The continuous-time cost function (48) can be discretized using any appropriate method. Pseudospectral methods, for example, specify a numerical quadrature that must be used to discretize the integral [28]. For simplicity, assume that the time grid is uniform (that is, $t_{k+1} - t_k = \Delta t$ for all $k = 1, \ldots, N-1$) and that trapezoidal numerical integration is used. This allows for writing the discrete-time version of (48) as

$$\mathcal{L}_\lambda(x, u, p, \hat{\nu}) = \phi_\lambda(x(1), p, \nu_{\text{ic}}, \nu_{\text{tc}}) + \texttt{trapz}(\Gamma_\lambda^N), \tag{51}$$
$$\Gamma_{\lambda,k}^N = \Gamma_\lambda(x_k, u_k, p, E_k\nu_k, \nu_{s,k}), \tag{52}$$

where trapezoidal integration is implemented by the function $\texttt{trapz}: \mathbb{R}^N \to \mathbb{R}$, defined as

$$\texttt{trapz}(z) \triangleq \frac{\Delta t}{2} \sum_{k=1}^{N-1} z_k + z_{k+1}. \tag{53}$$

Equation (51) is called the *linear augmented cost function*. This name is a slight misnomer because (51) is, in fact, a generally nonlinear convex function. However, the "linear" qualifier emphasizes that the cost relates to the convex subproblem for which all nonconvexities have been linearized. In particular, the virtual control terms can be viewed as linear measurements of dynamic and nonconvex path and boundary constraint infeasibility.

Finally, the constraints (50c), (50d), and (50g) are enforced at the discrete temporal nodes t_k for each $k = 1, \ldots, N$. In summary, the following discrete-time convex subproblem is solved at each SCvx iteration (that is, location ② in Figure 11):

$$\min_{x,u,p,\hat{\nu}} \mathcal{L}_\lambda(x, u, p, \hat{\nu}), \tag{54a}$$
$$\text{s.t. } x_{k+1} = A_k x_k + B_k u_k + F_k p + r_k + E_k \nu_k, \tag{54b}$$
$$(x_k, p) \in \mathcal{X}_k, \quad (u_k, p) \in \mathcal{U}_k, \tag{54c}$$
$$C_k x_k + D_k u_k + G_k p + r_k' \le \nu_{s,k}, \tag{54d}$$
$$H_0 x_1 + K_0 p + \ell_0 + \nu_{\text{ic}} = 0, \tag{54e}$$
$$H_{\text{f}} x_N + K_{\text{f}} p + \ell_{\text{f}} + \nu_{\text{tc}} = 0, \tag{54f}$$
$$\| \delta x_k \|_q + \| \delta u_k \|_q + \| \delta p \|_q \le \eta. \tag{54g}$$

Note that (54b) is written as a shorthand convenience for discretized dynamics and it is not representative of every possible discretization choice. For example, (54b) is correct if ZOH discretization is used. However, as specified in (S32a), FOH discretization would lead to the following constraint that replaces (54b):

$$x_{k+1} = A_k x_k + B_k^- u_k + B_k^+ u_{k+1} + F_k p + r_k + E_k \nu_k.$$

The most general interpolating polynomial discretization fits into the following discrete-time dynamics constraint:

$$x_{k+1} = A_k x_k + \sum_{j=1}^{N} B_k^j u_j + F_k p + r_k + E_k \nu_k,$$

where the j superscript indexes different input coefficient matrices. Other discretization methods lead to yet other affine equality constraints, all of which simply replace (54b) [191]. With this in mind, (54b) is written for simplicity. Furthermore, it is implicitly understood that the constraints (54b)–(54d) and (54g) hold at each temporal node. Because (54) is a finite–dimensional convex optimization problem, it can be solved to global optimality using an off-the-shelf convex optimization solver [26]. Denote the optimal solution by $x^* = \{x_k^*\}_{k=1}^N$, $u^* = \{u_k^*\}_{k=1}^N$, p^*, $\nu^* = \{\nu_k^*\}_{k=1}^{N-1}$, $\nu_s^* = \{\nu_{s,k}^*\}_{k=1}^N$, ν_{ic}^*, and ν_{tc}^*.

SCvx Update Rule

The preceding discussion enables taking a nonconvex optimal control problem such as (38) and 1) convexify it to (44), 2) add a trust region (45) to avoid artificial unboundedness, 3) add virtual control terms (46) to avoid artificial infeasibility, 4) penalize virtual control usage in the cost (48), and 5) apply discretization to obtain a finite–dimensional convex problem (54). This gives all the necessary ingredients to go around the loop in Figure 11 except for one thing: how to update the trust region radius η in (45). In general, the trust region changes after each pass around the loop. This section discusses this missing ingredient.

First, define a linearization accuracy metric, called the *defect*,

$$\delta_k \triangleq x_{k+1} - \psi(t_k, t_{k+1}, x_k, u, p) \tag{55}$$

for $k = 1, \ldots, N-1$. The function ψ is called the *flow map*, and its role is to "propagate" the control input u through the continuous-time nonlinear dynamics (38b), starting at state x_k at time t_k and evolving until the next temporal grid node t_{k+1} [194]. It is important that the flow map be implemented in a way that is consistent with the chosen discretization scheme, as defined below.

Definition 2

The flow map ψ in (55) is *consistent* with the discretization used for (54) if the following equation holds for all $k = 1, \ldots, N-1$:

$$\psi(t_k, t_{k+1}, \bar{x}_k, \bar{u}, \bar{p}) = A_k \bar{x}_k + B_k \bar{u}_k + F_k \bar{p} + r_k. \tag{56}$$

There is an intuitive way to think about the consistency property of ψ. The reader may follow along using the illustration in Figure 14. On the one hand, $\psi(t_k, t_{k+1}, \bar{x}_k, \bar{u}, \bar{p})$ maps an initial state $\bar{x}_k$ through the continuous-time nonlinear dynamics (38b) to a new state $\tilde{x}_{k+1}$. On the other hand, the right-hand side of (56) does the same except that it uses the linearized and discretized dynamics and outputs a new state $\hat{x}_{k+1}$. Because the linearization is being evaluated at the reference trajectory (that is, at the linearization point), the linearized continuous-time dynamics will yield the exact same trajectory. Thus, the only difference between the left- and right-hand sides of (56) is that the right-hand side works in discrete time. Consistency, then, simply means that propagating the continuous-time dynamics yields the same point as performing the discrete-time update (that is, $\tilde{x}_{k+1} = \hat{x}_{k+1}$). For every discretization method that is used to construct (54), there exists a consistent flow map.

The reader is likely already familiar with several flow maps, even if the term sounds new. Consider the following concrete examples. When using forward Euler discretization, the corresponding consistent flow map is simply

$$\psi(t_k, t_{k+1}, x_k, u, p) = x_k + \Delta t \cdot f(t_k, x_k, u_k, p). \tag{57}$$

When using an interpolating polynomial discretization scheme, such as the one described in "Discretizing Continuous-Time Optimal Control Problems," the corresponding consistent flow map is the solution to the dynamics (38b) obtained through numerical integration. In other words, the flow map satisfies the following conditions at each time instant $t \in [t_k, t_{k+1}]$:

$$\psi(t_k, t_k, x_k, u, p) = x_k, \tag{58a}$$

$$\dot{\psi}(t_k, t, x_k, u, p) = f(t, \psi(t_k, t, x_k, u, p), u(t), p). \tag{58b}$$

As demonstrated in Figure 15, the defect (55) captures the discrepancy between the next discrete-time state x_{k+1} and the state obtained by using the flow map starting at

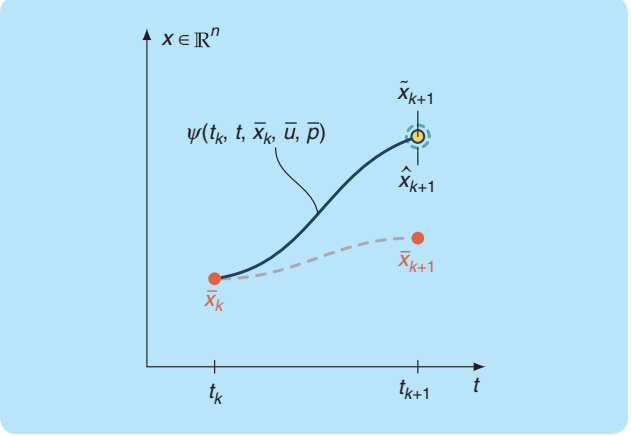


FIGURE 14 The flow map consistency property in Definition 2. When the flow map is consistent with the discretization scheme, the state $\tilde{x}_{k+1}$ propagated through the flow map (the yellow circle) and the state $\hat{x}_{k+1}$ propagated through the discrete-time linearized update equation (the dashed green circle) match. When the reference trajectory is not dynamically feasible, it generally deviates from the flow map trajectory; hence, $\bar{x}_{k+1} \neq \tilde{x}_{k+1}$.

time t_k. The defect has the following interpretation: a nonzero defect indicates that the solution to the subproblem is dynamically infeasible with respect to the original nonconvex dynamics. For a dynamically feasible subproblem solution, the flow map trajectories in Figure 15 coincide with the discrete-time states at the discrete-time nodes. This is a direct consequence of the consistency property from Definition 2. It will be shown that this happens for the converged solutions of the numerical examples presented in the "Application Examples" section.

The defects obtained from (55) can now be leveraged to update the trust region radius in SCvx. First, define a nonlinear version of (51) as follows:

$$\mathcal{J}_\lambda(x, u, p) = \phi_\lambda(x(1), p, g_{\text{ic}}(x(0), p), g_{\text{tc}}(x(1), p)) + \texttt{trapz}\,(\Gamma^N_\lambda), \tag{59}$$

$$\Gamma^N_{\lambda,k} = \Gamma_\lambda(x_k, u_k, p, \delta_k, [s(t_k, x_k, u_k, p)]^+), \tag{60}$$

where the positive part function $[\,\cdot\,]^+$ returns zero when its argument is negative and otherwise just returns the argument. A salient feature of (59) is that it evaluates the penalty function (47) based on how well the actual nonconvex constraints are satisfied. To do so, when compared to (51), $E_k\nu_k$ is replaced with the defect δ_k measuring dynamic infeasibility, and $\nu_{s,k}$ is replaced with the actual nonconvex path constraints (38e), while ν_{ic} and ν_{tc} are replaced by the actual boundary conditions (38f) and (38g). Because evaluation of the defect and nonconvex path and boundary constraints is a nonlinear operation, (59) is called the *nonlinear augmented cost function*.

The SCvx algorithm uses the linear augmented cost (51) and nonlinear augmented cost (59) to, roughly speaking, measure the accuracy of the convex approximation of (38) by (44). Using the reference solution and the optimal solution to (54), SCvx defines the following scalar metric to measure convexification accuracy:

$$\rho \triangleq \frac{\mathcal{J}_\lambda(\bar{x}, \bar{u}, \bar{p}) - \mathcal{J}_\lambda(x^*, u^*, p^*)}{\mathcal{J}_\lambda(\bar{x}, \bar{u}, \bar{p}) - \mathcal{L}_\lambda(x^*, u^*, p^*, \hat{\nu}^*)}. \tag{61}$$

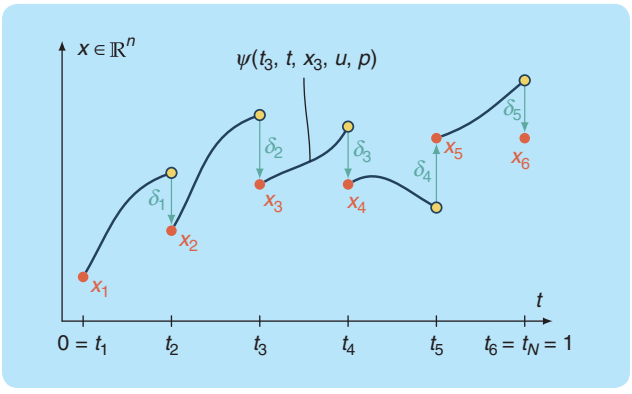


FIGURE 15 The defect calculation according to (55). The flow map computation restarts at each discrete-time node. At each temporal node, the defect is computed as the difference between the discrete solution output by (54) and the corresponding flow map value.

Let us carefully analyze the elements of (61). First, the denominator is always nonnegative because the following relation holds [49, Th. 3.10]:

$$\mathcal{J}_\lambda(\bar{x}, \bar{u}, \bar{p}) \geq \mathcal{L}_\lambda(x^*, u^*, p^*, \hat{\nu}^*). \tag{62}$$

The proof of (62) works by constructing a feasible subproblem solution that matches the cost $\mathcal{J}_\lambda(\bar{x}, \bar{u}, \bar{p})$. Begin by setting the state, input, and parameter values to the reference solution $\{\bar{x}(t), \bar{u}(t), \bar{p}\}_0^1$. The virtual controls must be chosen to make this solution feasible for (54) and yield a matching cost (54a). The consistency property from Definition 2 ensures that this is always possible to do. In particular, choose the dynamics virtual control to match the defects and the other virtual controls to match the constraint values at the reference solution. This represents a feasible solution of (54) whose cost equals $\mathcal{J}_\lambda(\bar{x}, \bar{u}, \bar{p})$. The optimal cost for the subproblem cannot be worse, so the inequality (62) follows. If the denominator of (61) is zero, it follows from the preceding discussion that the reference trajectory is an optimal solution of the convex subproblem. This signals that it is appropriate to terminate SCvx and exit the loop in Figure 11. Hence, the denominator of (61) can be used as part of a stopping criterion that avoids a division by zero. The stopping criterion is discussed further in the next section.

Examining (61) holistically, it is essentially a ratio between the actual cost improvement (the numerator) and the predicted cost improvement (the denominator), achieved during one SCvx iteration [48]:

$$\rho = \frac{\text{actual improvement}}{\text{predicted improvement}}. \tag{63}$$

In fact, the denominator can be loosely interpreted as a lower-bound prediction: the algorithm "thinks" that the actual cost improves by at least that much. The following intuition can thus be adopted based on seeing (54) as a local model of (38). A small ρ value indicates that the model is inaccurate because the actual cost decrease is much smaller than predicted. If ρ is close to unity, the model is accurate because the actual cost decrease is similar to the prediction. If the ρ value is greater than unity, the model is "conservative" because it underestimates the cost reduction. As a result, large ρ values incentivize growing the trust region because the model is trustworthy and the algorithm can make faster progress by utilizing more of it. On the other hand, small ρ values incentivize shrinking the trust region to not "overstep" an inaccurate model [49].

The SCvx update rule for the trust region radius η formalizes the preceding intuition. Employing three user-defined scalars $\rho_0, \rho_1, \rho_2 \in (0, 1)$ that split the real number line into four parts, the trust region radius and reference trajectory are updated at the end of each SCvx iteration according to Figure 16. The user-defined constants $\beta_{\text{sh}}, \beta_{\text{gr}} > 1$

are the trust region shrink and growth rates, respectively. Practical implementations of SCvx also let the user define minimum and maximum trust region radii by using $\eta_0, \eta_1 > 0$. The choice of the user-defined scalars ρ_0, ρ_1, and ρ_2 greatly influences the algorithm runtime. As shown in Figure 16, when $\rho < \rho_0$, the algorithm will actually outright reject the subproblem solution and solve the subproblem again with a smaller trust region. This begs the question: Can rejection go on indefinitely? If this occurs, the algorithm will be stuck in an infinite loop. The answer is no; the metric (61) must eventually rise above ρ_0 [49, Lemma 3.11].

SCvx Stopping Criterion

The previous sections provide a complete description of the "Starting" and "Iteration" regions in Figure 11. A crucial remaining element is how and when to exit the "SCP loop" of the "Iteration" region. This is defined by a stopping criterion (also called an *exit criterion*), which is implemented at location ③ in Figure 11. When the stopping criterion triggers, the algorithm has converged, and the final iteration's trajectory $\{x^*(t), u^*(t), p^*\}_0^1$ is output.

The basic idea of the stopping criterion is to measure how different the new trajectory $\{x^*(t), u^*(t), p^*\}_0^1$ is from the reference trajectory $\{\bar{x}(t), \bar{u}(t), \bar{p}\}_0^1$. Intuitively, if the difference is small, then the algorithm considers the trajectory unworthy of further improvement, and it is appropriate to exit the SCP loop. The formal SCvx exit criterion uses the denominator of (61) (that is, the predicted cost improvement) as the stopping criterion [49], [64], [185], [193]:

$$\mathcal{J}_\lambda(\bar{x}, \bar{u}, \bar{p}) - \mathcal{L}_\lambda(x^*, u^*, p^*, \hat{\nu}^*) \leq \varepsilon, \tag{64}$$

where $\varepsilon \in \mathbb{R}_{++}$ is a user-defined (small) threshold. For notational simplicity, write (64) as

$$\bar{\mathcal{J}}_\lambda - \mathcal{L}_\lambda^* \leq \varepsilon.$$

Numerical experience has shown a control-dependent stopping criterion to sometimes lead to an unnecessarily conservative definition of convergence [62]. For example, in an application such as rocket landing, common vehicle characteristics (for example, inertia and engine-specific impulse) imply that relatively large changes in control can have little effect on the state trajectory and optimality. When the cost is control dependent (which it often is), (64) may be a conservative choice that will result in more iterations without providing a more optimal solution. In such cases, a simpler and less conservative stopping criterion may be used:

$$\| p^* - \bar{p} \|_{\hat{q}} + \max_{k \in \{1,\ldots,N\}} \| x_k^* - \bar{x}_k \|_{\hat{q}} \leq \varepsilon, \tag{65}$$

where $\hat{q} \in \{1, 2, 2^+, \infty\}$ defines a norm similarly to (45). Importantly, the following correspondence holds between (64) and (65). For any ε choice in (65), there is a (generally different) ε choice in (64) such that if (64) holds, then (65) holds. This is called an "implication correspondence" between stopping criteria.

SCvx guarantees that there will be an iteration for which (64) holds. By implication correspondence, this guarantee extends to (65). In general, the user can define yet another stopping criterion that is tailored to the specific trajectory problem, as long as implication correspondence holds. This is done for the numerical examples at the end of the article, where the following stopping criterion that combines (64) and (65) is used:

$$\text{(65) holds, or } \bar{\mathcal{J}}_\lambda - \mathcal{L}_\lambda^* \leq \varepsilon_r |\bar{\mathcal{J}}_\lambda|, \tag{66}$$

where $\varepsilon_r \in \mathbb{R}_{++}$ is a user-defined (small) threshold on the relative cost improvement. The second term in (66) allows the algorithm to terminate when relatively little progress is being made in decreasing the cost, which signals that the algorithm has achieved local optimality.

SCvx Convergence Guarantee

The SCvx trust region update rule in Figure 16 is designed to permit a rigorous convergence analysis of the algorithm. A detailed discussion is given in [49]. To arrive at the convergence result, the proof requires the following (mild) technical condition that is common to most, if not all, optimization

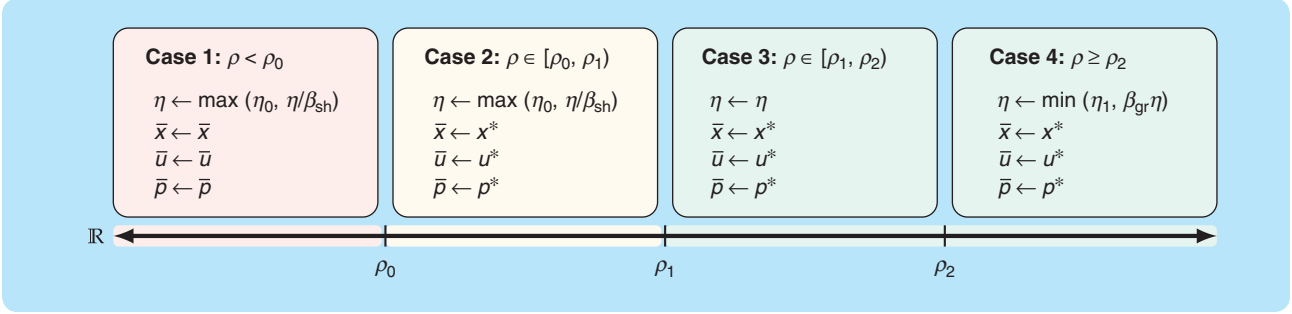


FIGURE 16 The sequential convex programming trust region update rule. The accuracy metric ρ is defined in (61) and provides a measure of how accurately the convex subproblem given by (54) describes the original problem (38). Note that case 1 actually rejects the solution to (54) and shrinks the trust region before proceeding to the next iteration. In this case, the convex approximation is deemed so poor that it is unusable.

algorithms. This condition is truly "mild" because it is almost never checked in practice, and SCvx just works.

Condition 8

The gradients of the equality and active inequality constraints of (38) must be linearly independent for the final converged trajectory output by SCvx. This is known as a *linear independence constraint qualification* [29, Ch. 12].

It is also required that the weight λ in (48) be sufficiently large. This ensures that the integral penalty term in (48) is a so-called exact penalty function. A precise condition for "large enough" is provided in [49, Th. 3.9], and a possible strategy is outlined following the theorem in that article. However, in practice, simply iterate over a few powers of 10 until SCvx converges with zero virtual control. An approximate range that works for most problems is 10^2 to 10^4. Once a magnitude is identified, it usually works well across a wide range of parameter variations for the problem.

The SCvx convergence guarantee is stated in the following theorem that is proved in [49, Th. 3.21]. Besides Condition 8, the theorem requires a few more mild assumptions on the inequality constraints in (38) and the penalized cost (51). They are not stated here due to their technical nature. It is enough to say that, like Condition 8, these assumptions are mild enough that they are rarely checked in practice.

Theorem 8

Suppose that Condition 8 holds and that the weight λ in (48) is large enough. Regardless of the initial reference trajectory provided in Figure 11, the SCvx algorithm will always converge at a superlinear rate by iteratively solving (54). Furthermore, if the virtual controls are zero, then the converged trajectory is a stationary point of (38) in the sense of having satisfied the KKT conditions.

Theorem 8 confirms that SCvx solves the KKT conditions of the original (38). Because these are first-order, necessary conditions of optimality, they are also satisfied by local maxima and saddle points. Nevertheless, because each convex subproblem is minimized by SCvx, the likelihood of converging to a stationary point that is not a local minimum is very small.

Theorem 8 also states that the convergence rate is superlinear [29], which is to say that the distance from the converged solution decreases superlinearly [49, Th. 4.7]. This is better than general NLP methods and on par with SQP methods that usually also attain, at most, superlinear convergence [171].

In conclusion, note that Theorem 8 is quite intuitive and confirms the basic intuition. If one is "lucky" to obtain a solution with zero virtual control, then it is a local minimum of the original nonconvex problem (38). The reader will be surprised, however, at just how easy it is to be "lucky." In most cases, it is a simple matter of ensuring that the penalty weight λ in (48) is large enough. If the problem is feasible but SCvx is not converging or is converging with nonzero virtual control, the first thing to try is to increase λ. In more difficult cases, some nonconvex constraints may need to be reformulated as equivalent versions that play nicer with the linearization process (the free-flyer example in the "Application Examples" section discusses this in detail). However, there is no a priori hard guarantee that the converged solution will satisfy $\hat{\nu} \equiv 0$. In fact, since SCvx always converges, even an infeasible optimal control problem can be solved, and it will return a solution for which $\hat{\nu}$ is nonzero.

The GuSTO Algorithm

The GuSTO algorithm is another SCP method that can be used for trajectory generation. The reader will find that GuSTO has a familiar feel to that of SCvx. Nevertheless, the algorithm is subtly different from both computational and theoretical standpoints. For example, while SCvx works directly with the temporally discretized (54) and derives its convergence guarantees from the KKT conditions, GuSTO performs all of its analysis in continuous time by using Pontryagin's maximum principle [11], [12], [49], [50]. Temporal discretization is introduced only at the end to enable numerical solutions of the problem. GuSTO applies to versions of (38) where the running cost in (39) is quadratic in the control variable:

$$\Gamma(x, u, p) = u^\top S(p)\, u + u^\top \ell(x, p) + g(x, p), \tag{67}$$

where the parameter p, as before, can be used to capture free final time problems. The functions S, ℓ, and g must all be continuously differentiable. Furthermore, the function S must be positive semidefinite. The dynamics must be affine in the control variable:

$$f(t, x, u, p) = f_0(t, x, p) + \sum_{i=1}^{m} u_i f_i(t, x, p), \tag{68}$$

where $f_i : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^n$ are nonlinear functions representing a decomposition of the dynamics into terms that are control independent and terms that linearly depend on each of the control variables u_i. Note that any Lagrangian mechanical system can be expressed in the control affine form, so (68) is applicable to the vast majority of real-world vehicle trajectory generation applications [195]. Finally, the nonconvex path constraints (38e) are independent of the control terms; that is, $s(t, x, u, p) = s(t, x, p)$. Altogether, these assumptions specialize (38) to problems that are convex in the control variable.

At its core, GuSTO is an SCP trust region method just like SCvx. Thus, it has to address the same issues of artificial infeasibility and artificial unboundedness. To this end, the GuSTO algorithm makes the following algorithmic choices:

- To handle artificial unboundedness, GuSTO augments the cost function with a soft penalty on the violation of (45). Because the original problem (38) is convex in the control variables, there is no need for a

trust region with respect to the control, and $\alpha_u = 0$ is used. In its standard form, GuSTO works with default values $\alpha_x = \alpha_p = 1$. However, one can choose different values $\alpha_x, \alpha_p > 0$ without affecting the convergence guarantees.

» To handle artificial infeasibility, GuSTO augments the cost function with a soft penalty on nonconvex path constraint violation. As the algorithm progresses, the penalty weight increases.

From these choices, it can already be deduced that a salient feature of the GuSTO algorithm is its use of soft penalties to enforce nonconvex constraints. Recall that in SCvx, the trust region (45) is imposed exactly, and the linearized constraints are (equivalently) relaxed by using virtual control (46). By penalizing constraints, GuSTO can be analyzed in continuous time via the classical Pontryagin maximum principle [11], [12], [50], [69]. An additional benefit of having moved the state constraints into the cost is that the virtual control term employed in the dynamics (46a) can be safely removed since linearized dynamics are almost always controllable [72].

Begin by formulating the augmented cost function used by the GuSTO convex subproblem at ② in Figure 11. This involves three elements: the original cost (39), soft penalties on path constraints that involve the state, and a soft penalty on trust region violation. The latter two elements are now discussed. To formulate the soft penalties, consider a continuously differentiable, convex, and nondecreasing penalty function $h_\lambda : \mathbb{R} \to \mathbb{R}_+$ that depends on a scalar weight $\lambda \geq 1$ [50], [72]. The goal of h_λ is to penalize any positive value and be agnostic to nonpositive values. Thus, a simple example is a quadratic rectifier,

$$h_\lambda(z) = \lambda([z]^+)^2, \tag{69}$$

where higher λ values indicate higher penalization. Another example is the SoftPlus function [196]:

$$h_\lambda(z) = \lambda\sigma^{-1}\log(1 + e^{\sigma z}), \tag{70}$$

where $\sigma \in \mathbb{R}_{++}$ is a sharpness parameter. As σ grows, the SoftPlus function becomes an increasingly accurate approximation of $\lambda\max\{0, z\}$.

Note that h_λ is used to enforce soft penalties for violating the constraints (38c) and (38e) that involve the state and parameter. To this end, let $w : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^{n_w}$ be a convex, continuously differentiable indicator function that is nonpositive if and only if the convex state constraints in (38c) are satisfied:

$$w(x, p) \leq 0 \;\Leftrightarrow\; (x, p) \in \mathcal{X}.$$

Observe that because $\mathcal{X}$ is a convex set, such a function always exists. Using w, s from (38e), and h_λ, all the state constraints can be grouped into a single soft penalty function:

$$g_{\text{sp}}(t, x, p) \triangleq \sum_{i=1}^{n_w} h_\lambda(w_i(t, x)) + \sum_{i=1}^{n_s} h_\lambda(s_i(t, x, p)). \tag{71}$$

Another term is added to the augmented cost function to handle artificial unboundedness. This term penalizes violations of the trust region constraint and is defined in a similar fashion to (71):

$$g_{\text{tr}}(x, p) \triangleq h_\lambda\left(\|\delta x\|_q + \|\delta p\|_q - \eta\right), \tag{72}$$

although hard-enforced versions of the trust region constraint can also be used without changing the algorithm's properties [72].

The overall augmented cost function is obtained by combining the original cost (38a) with (71) and (72). The resulting function is generally nonconvex, and it can be decomposed into its convex and nonconvex parts:

$$J_\lambda(x, u, p) \triangleq \breve{J}_\lambda(x, p) + \tilde{J}_\lambda(x, u, p), \tag{73a}$$

$$\breve{J}_\lambda(x, p) = \phi(x(1), p) + \int_0^1 g_{\text{tr}}(x, p) + \sum_{i=1}^{n_w} h_\lambda(w_i(t, x))\,\mathrm{d}t, \tag{73b}$$

$$\tilde{J}_\lambda(x, u, p) = \int_0^1 \Gamma(x, u, p) + \sum_{i=1}^{n_s} h_\lambda(s_i(t, x, p))\,\mathrm{d}t. \tag{73c}$$

The terms $g_{\text{tr}}(x, p)$ and $h_\lambda(w_i(x))$ in (73b) are convex functions since they are formed by composing a convex nondecreasing function h_λ with a convex function [26]. Thus, (73b) is the convex part of the cost, while (73c) is the nonconvex part.

The ultimate goal is to construct a convex subproblem that can be solved at location ② in Figure 11. Thus, the nonconvex part of the cost (73c) must be convexified around the reference trajectory $\{\bar{x}(t), \bar{u}(t), \bar{p}\}_0^1$. This requires the following Jacobians in addition to the initial list (43):

$$A^\Gamma \triangleq \nabla_x \Gamma(\bar{x}, \bar{u}, \bar{p}), \tag{74a}$$

$$B^\Gamma \triangleq \nabla_u \Gamma(\bar{x}, \bar{u}, \bar{p}), \tag{74b}$$

$$F^\Gamma \triangleq \nabla_p \Gamma(\bar{x}, \bar{u}, \bar{p}). \tag{74c}$$

Using (43e), (43 g), and (74), the convex approximation of (73c) can be written as

$$\tilde{L}_\lambda(x, u, p) = \int_0^1 \Gamma(\bar{x}, \bar{u}, \bar{p}) + A^\Gamma \delta x + B^\Gamma \delta u + F^\Gamma \delta p + \sum_{i=1}^{n_s} h_\lambda(s_i(t, \bar{x}, \bar{p}) + C_i \delta x + G_i \delta p)\,\mathrm{d}t, \tag{75}$$

where C_i and G_i are the ith rows of the Jacobians (43e) and (43g), respectively. Note that, strictly speaking, $\tilde{L}_\lambda$ is not a linearized version of $\tilde{J}_\lambda$ because the second term in (73c) is linearized only inside the $h_\lambda(\cdot)$ function. Replacing $\tilde{J}_\lambda$ in (73a) with $\tilde{L}_\lambda$ yields a convexified augmented cost function:

$$L_\lambda(x, u, p) = \breve{J}_\lambda(x, p) + \tilde{L}_\lambda(x, u, p). \tag{76}$$

To summarize the preceding discussion, the continuous-time convex subproblem that is solved at each iteration of the GuSTO algorithm can be stated formally as:

$$\min_{u,p} \; L_\lambda(x, u, p), \tag{77a}$$

$$\text{s.t. } \dot{x} = Ax + Bu + Fp + r, \tag{77b}$$

$$(u, p) \in \mathcal{U}, \tag{77c}$$

$$H_0 x(0) + K_0 p + \ell_0 = 0, \tag{77d}$$

$$H_f x(1) + K_f p + \ell_f = 0. \tag{77e}$$

Equation 77 can be compared to the SCvx continuous-time convex subproblem given by (50). Broadly speaking, the problems are quite similar. Their main differences stem from how the SCvx and GuSTO algorithms handle artificial infeasibility and artificial unboundedness. In the case of SCvx, virtual control terms are introduced, and a hard trust region (50g) is imposed. In the case of GuSTO, everything is handled via soft penalties in the augmented cost. The result is that penalty terms in the cost (77a) replace the constraints (50c), (50d), and (50g).

There is another subtle difference between (77) and (50), which is that GuSTO does not use virtual control terms for the linearized boundary conditions (77d) and (77e). In SCvx, these virtual control terms maintain subproblem feasibility when the linearized boundary conditions define hyperplanes that do not intersect with the hard-enforced convex state path constraints in (50c). This is not a problem in GuSTO since the convex state path constraints are penalized only in the cost (71), and violating them for the sake of retaining feasibility is allowed. Furthermore, the linearized dynamics (77b) are theoretically guaranteed to be almost always controllable [72]. This implies that the dynamics (77b) can always be steered between the linearized boundary conditions (77d) and (77e) [88].

Similar to the treatment of (50), a temporal discretization scheme must be applied to numerically solve the subproblem. Discretization proceeds in the same way as for SCvx: select a set of temporal points $t_k \in [0, 1]$ for $k = 1, \ldots, N$, and recast (77) as a parameter optimization problem in the (overloaded) variables $x = \{x_k\}_{k=1}^N$, $u = \{u_k\}_{k=1}^N$, and p. The same discretization choices are available as for SCvx, as described in "Discretizing Continuous-Time Optimal Control Problems."

The integrals of the continuous-time cost function (77a) must also be discretized. This is done in a similar fashion to the way (51) was obtained from (48) for SCvx. For simplicity, again assume that the time grid is uniform, with step size Δt, and that trapezoidal numerical integration (53) is used. For notational convenience, by combining the integral terms in (73b) and (75), the convexified augmented cost function (76) can be written compactly as

$$L_\lambda(x, u, p) = \phi(x(1), p) + \int_0^1 L_\lambda^\Gamma(t, x, u, p)\,\mathrm{d}t, \tag{78}$$

where the convex function L_λ^Γ is formed by summing the integrands of (73b) and (75). Then, compute the discrete-time version of (78), which is the GuSTO equivalent of its SCvx counterpart (51):

$$\mathcal{L}_\lambda(x, u, p) = \phi(x(1), p) + \texttt{trapz}(L_\lambda^{\Gamma,N}), \tag{79}$$

$$L_{\lambda,k}^{\Gamma,N} = L_\lambda^\Gamma(k, x_k, u_k, p). \tag{80}$$

Finally (as in SCvx), the constraint (77c) is enforced only at the discrete temporal nodes. In summary, the following discrete-time convex subproblem is solved at each GuSTO iteration (that is, location ② in Figure 11):

$$\min_{x,u,p} \; \mathcal{L}_\lambda(x, u, p), \tag{81a}$$

$$\text{s.t. } x_{k+1} = A_k x_k + B_k u_k + F_k p + r_k, \tag{81b}$$

$$(u_k, p) \in \mathcal{U}_k, \tag{81c}$$

$$H_0 x_1 + K_0 p + \ell_0 = 0, \tag{81d}$$

$$H_f x_N + K_f p + \ell_f = 0, \tag{81e}$$

where it is implicitly understood that the constraints (81b) and (81c) hold at each temporal node.

GuSTO Update Rule

The development has now reached a similar point to SCvx from the previous section: by using (81) as the discrete-time convex subproblem, all the necessary elements are available to go around the loop in Figure 11, except for how to update the trust region radius η and penalty weight λ. Both values are generally different at each GuSTO iteration, and the method by which they are updated is now described. The reader will find the concept to be similar to SCvx.

First, recall that GuSTO imposes the trust region (45) as a soft constraint via the penalty function (72). This means that the trust region constraint can possibly be violated. If the solution to (81) violates (45), GuSTO rejects the solution and increases the penalty weight λ by a user-defined factor $\gamma_{\text{fail}} > 1$. Otherwise, if (45) holds, the algorithm proceeds by computing the following convexification accuracy metric that is analogous to (61):

$$\rho \triangleq \frac{|\, J_\lambda(x^*, u^*, p^*) - L_\lambda(x^*, u^*, p^*)\,| + \Theta^*}{|\, L_\lambda(x^*, u^*, p^*)\,| + \int_0^1 \|\, \dot{x}^*\,\|_2 \,\mathrm{d}t}, \tag{82}$$

where, based on (38b) and (77b), the following quantities are defined:

$$\dot{x}^* \triangleq Ax^* + Bu^* + Fp^* + r, \tag{83a}$$

$$\Theta^* \triangleq \int_0^1 \|\, f(t, x^*, u^*, p^*) - \dot{x}^*\,\|_2 \,\mathrm{d}t. \tag{83b}$$

Equation (83a) integrates to yield the continuous-time state solution trajectory. This is done in accordance with the temporal discretization scheme, such as the one described in "Discretizing Continuous-Time Optimal Control Problems." As a result, the value of Θ^* is nothing but a measure of the

total accumulated error that results from linearizing the dynamics along the subproblem's optimal solution. In a way, Θ^* is the counterpart of SCvx defects defined in (55), with the subtle difference that while defects measure the discrepancy between the linearized and nonlinear state trajectories, Θ^* measures the discrepancy between the linearized and nonlinear state dynamics.

The continuous-time integrals in (82), in principle, can be evaluated exactly (that is, to within numerical precision) based on the discretization scheme used. In practice, they can be approximated by directly numerically integrating the solution of (81) by using (for example) trapezoidal integration (53). This means the following approximation of (82) is evaluated:

$$\begin{aligned}\rho &\approx \frac{|\mathcal{J}_\lambda(x^*, u^*, p^*) - \mathcal{L}_\lambda(x^*, u^*, p^*)| + \Theta^*}{|\mathcal{L}_\lambda(x^*, u^*, p^*)| + \texttt{trapz}\,(\mathbf{x}^*)},\\ \Theta^* &\approx \texttt{trapz}\,(\Delta f^*),\\ \Delta f_k^* &= \|f(t_k, x_k^*, u_k^*, p^*) - x_{k+1}^*\|_2, \quad \mathbf{x}_k^* = \|x_{k+1}^*\|_2. \end{aligned} \tag{84}$$

The nonlinear augmented cost $\mathcal{J}_\lambda$ in the numerator of (84) is a temporally discretized version of (73a). In particular, combine the integrals of (73b) and (73c) to express (73a) as

$$J_\lambda(x, u, p) = \phi(x(1), p) + \int_0^1 J_\lambda^\Gamma(t, x, u, p)\,\mathrm{d}t, \tag{85}$$

which enables computing $\mathcal{J}_\lambda$ as follows:

$$\mathcal{J}_\lambda(x, u, p) = \phi(x(1), p) + \texttt{trapz}\,(J_\lambda^{\Gamma,N}), \tag{86}$$

$$J_{\lambda,k}^{\Gamma,N} = J_\lambda^\Gamma(k, x_k, u_k, p). \tag{87}$$

Examining (82) holistically, it can be interpreted as a normalized error that results from linearizing the cost and dynamics:

$$\rho = \frac{\text{cost error} + \text{dynamics error}}{\text{normalization term}}. \tag{88}$$

Note that as long as the solution of (81) is nontrivial (that is, $x^*(t) = 0$ for all $t \in [0, 1]$ does not hold), the normalization term is guaranteed to be strictly positive. Thus, there is no danger of dividing by zero.

For comparison, SCvx evaluates convexification accuracy through (63), which measures accuracy as a relative error in the cost improvement prediction. This prediction is a "higher-order" effect: linearization error indirectly influences cost prediction error through the optimization of (54). GuSTO takes a more direct route with (88) and measures convexification accuracy directly as a normalized error that results from linearizing both the cost and dynamics.

Examining (88), the following intuition about the size of ρ can be adopted. Similarly to SCvx, (81) can be viewed as a local model of (38). A large ρ value indicates an inaccurate model since the linearization error is large. A small ρ value indicates an accurate model since the linearization error is relatively small compared to the normalization term. Hence, large ρ values incentivize shrinking the trust region to not overstep the model, while small ρ values incentivize growing the trust region to exploit a larger area of an accurate model. Note that the opposite intuition holds for SCvx, where small ρ values are associated with shrinking the trust region.

The GuSTO update rule formalizes the preceding intuition. Employing two user-defined constants $\rho_0,\ \rho_1 \in (0, 1)$ that split the real number line into three parts, the trust region radius η and penalty weight λ are updated at the end of each GuSTO iteration according to Figure 17. Just like in SCvx, the user-defined constants $\beta_{\text{sh}},\ \beta_{\text{gr}} > 1$ are the trust region shrink and growth rates, respectively. The sizes of the trust region and penalty weight are restricted by user-defined constants: η_0 is the minimum trust region radius, η_1 is the maximum trust region radius, and $\lambda_0 \geq 1$ is the minimum penalty weight. Importantly, for cases 1 and 2 in Figure 17, whenever the solution of (81) is accepted,

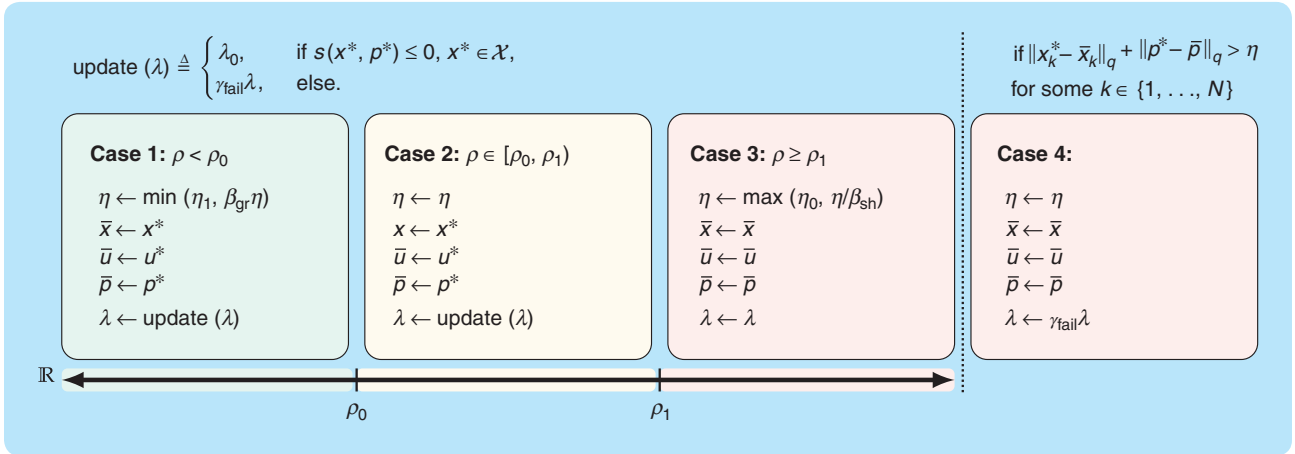


FIGURE 17 The GuSTO trust region update rule. The accuracy metric ρ is defined in (82) and provides a measure of how accurately the convex subproblem given by (81) describes the original (38). Note that cases 3 and 4 reject the solution to (81). In case 3, this is due to the convex approximation being deemed so inaccurate that it is unusable, and the trust region is shrunk accordingly. In case 4, this is due to the trust region constraint (45) being violated.

the penalty weight λ is increased by a factor γ_{fail} if any of the nonconvex state constraints in (38e) are violated. This incentivizes subsequent iterates to become feasible with respect to (38e).

In addition, GuSTO requires that η eventually shrinks to zero as the SCP iterations progress (this is addressed in more detail below in the section on GuSTO convergence) [72]. However, if the trajectory is converging, then δx, δu, $\delta p \to 0$ and the linearizations in (77) are bound to become more accurate. Hence, near-zero ρ values are expected when the trajectory is converging, which means that Figure 17 will grow the trust region instead of shrinking it. A simple remedy is to apply the following exponential shrink factor following the update in Figure 17:

$$\eta \leftarrow \mu^{[1+k-k_*]^+}\eta, \tag{89}$$

where $\mu \in (0, 1)$ is an exponential shrink rate and $k_* \geq 1$ is the first SCP iteration where shrinking is applied. The user sets both μ and k_* and, in this manner, can regulate how fast the trust region shrinks to zero. Low μ values and high k_* values are viewed as setting the algorithm up for a longer "exploration" phase prior to tightening the trust region.

GuSTO Stopping Criterion

To complete the description of the GuSTO algorithm, it remains to define the stopping criterion used at location ③ of Figure 11. The formal GuSTO exit criterion directly checks if the control and parameters are almost unchanged [50], [71]:

$$\left(\|\bar{p} - p^*\|_{\hat{q}} \leq \varepsilon \text{ and } \int_0^1 \|u^*(t) - \bar{u}(t)\|_{\hat{q}} \leq \varepsilon\right) \text{or } \lambda > \lambda_{\max}, \tag{90}$$

where $\varepsilon \in \mathbb{R}_{++}$ and $\hat{q} \in \{1, 2, 2^+, \infty\}$ have the same meaning as before in (65) and the new parameter $\lambda_{\max} \in \mathbb{R}_{++}$ is a (large) maximum penalty weight.

When (90) triggers due to $\lambda_{\max}$, GuSTO exits with an unconverged trajectory that violates the state and/or trust region constraints. This is equivalent to SCvx exiting with nonzero virtual control, and it indicates that the algorithm has failed to solve the problem, due to inherent infeasibility or numerical issues. Computing (90) can be computationally expensive due to numerical integration of the control deviation. The calculation can be simplified by directly using the discrete-time solution. This leads to the following stopping criterion:

$$\begin{aligned} \|p^* - \bar{p}\|_{\hat{q}} + \texttt{trapz}\,(\Delta u^*) \leq \varepsilon \text{ or } \lambda > \lambda_{\max}, \\ \Delta u_k^* = \|u_k^* - \bar{u}_k\|_{\hat{q}}. \end{aligned} \tag{91}$$

As discussed for SCvx, an implication correspondence holds between the stopping criteria (90) and (91). In practice, following the example of (66), an option is typically added for exiting when relatively little progress is made in decreasing the cost, which can often signal local optimality sooner than (91) is satisfied:

$$\text{(91) holds, or } |\bar{\mathcal{J}}_\lambda - \mathcal{J}_\lambda^*| \leq \varepsilon_{\text{r}} |\bar{\mathcal{J}}_\lambda|, \tag{92}$$

where (as before) $\varepsilon_{\text{r}} \in \mathbb{R}_{++}$ is a user-defined (small) threshold on the relative cost improvement. The numerical examples at the end of the article implement (92).

GuSTO Convergence Guarantee

Each iteration of the GuSTO numerical implementation is composed of three stages. Reviewing Figure 11, the convex problem (81) is first constructed and solved with a convex optimizer at location ②. Using the solution, the stopping criterion (92) is checked at ③. If the test fails, the third and final stage updates the trust region radius and soft penalty weight according to Figure 17 and (89). In this context, a convergence guarantee ensures that the stopping criterion at ③ in Figure 11 eventually triggers. GuSTO is an SCP algorithm that was designed and analyzed in continuous time by using the Pontryagin maximum principle [50], [72]. Thus, the first convergence guarantee assumes that the GuSTO algorithm solves the continuous-time subproblem [that is, (77)] at each iteration [50, Corollary 3.1], [72, Th. 3.2].

Theorem 9

Regardless of the initial reference trajectory provided in Figure 11, the GuSTO algorithm will always converge by iteratively solving (50). Furthermore, if $\lambda \leq \lambda_{\max}$ (in particular, the state constraints are exactly satisfied), then the solution is a stationary point of (38) in the sense of having satisfied the necessary optimality conditions of the Pontryagin maximum principle.

Note once again the following duality between the GuSTO and SCvx convergence guarantees. Both algorithms can converge to infeasible points of the original problem (38). For SCvx, this corresponds to a nonzero virtual control (recall Theorem 8). For GuSTO, this corresponds to $\lambda > \lambda_{\max}$. The situations are completely equivalent and correspond simply to the different choices made by the algorithms to impose nonconvex constraints either with virtual control (as in SCvx) or as soft cost penalties (as in GuSTO).

Theorem 9 should be viewed as the GuSTO counterpart of Theorem 8 for SCvx. Despite the similarities between the two statements, there are three important nuances of Theorem 9 that are now discussed.

The first difference concerns how the GuSTO update rule in Figure 17 and (89) plays into the convergence proof. In SCvx, the update rule from Figure 16 plays a critical role in proving convergence. Thus, SCvx is an SCP algorithm that is intimately linked to its trust region update rule. This is not the case for GuSTO, whose convergence proof does not rely on the update rule definition at all. The only thing assumed by Theorem 9 is that the trust region radius η eventually shrinks to zero [72]. Ensuring this is the reason

for (89). Thus, GuSTO is an SCP algorithm that accepts any update rule that eventually shrinks to zero. Clearly, some update rules may work better than others, and the one presented in this article is simply a choice that is implemented in practice. Another simple update rule is given in [72]. Overall, the GuSTO update rule can be viewed as a mechanism by which to accept or reject solutions based on their convexification accuracy and not as a function designed to facilitate formal convergence proofs.

The reason why it is necessary to shrink the trust region to zero is the second nuanced detail of Theorem 9. In the nonlinear optimization literature, most claims of convergence fall into the so-called weak convergence category [29], [197]. This is a technical term, which means that a *subsequence* among the trajectory iterates $\{x^i(t), u^i(t), p^i\}_0^1$ converges. For example, the subsequence may be $i = 1, 3, 5, 7, \ldots$, while the iterates $i = 2, 4, 6, \ldots$ behave differently. As a baseline, both SCvx and GuSTO provide the weak convergence guarantee. The next step is to provide a strong convergence guarantee, which ensures that the full sequence of iterates converges. For SCvx, this is possible by leveraging properties of the update rule in Figure 16. As mentioned in the previous paragraph, GuSTO is more flexible in its choice of update rule. To get a converging iterate sequence, the additional element (89) is introduced to force all subsequences to converge to the same value. Because analysis of SCvx has shown the strong convergence guarantee to be intimately tied to the choice of update rule, GuSTO can likely achieve a similar guarantee with an appropriately designed update rule.

The third and final nuanced detail is that Theorem 9 assumes that a continuous-time subproblem [that is, (77)] is solved at each iteration. In reality, a discrete-time version (81) is implemented on the computer. This difference between proof and reality is similar to LCvx, where proofs are also given using the continuous-time Pontryagin's maximum principle even though the implementation is in discrete time. If the discretization error is small, the numerically implemented GuSTO algorithm will remain in the vicinity of a convergent sequence of continuous-time subproblem solutions. Thus, given an accurate temporal discretization, Theorem 9 can be reasonably assumed to apply for the numerical GuSTO algorithm [72].

The default choice in research has been to use an interpolating polynomial method, such as described in "Discretizing Continuous-Time Optimal Control Problems." This approach has three advantages [191]: 1) the continuous-time dynamics (77b) are satisfied exactly, 2) there is a cheap way to convert the discrete-time numerical solution into a continuous-time control signal, and 3) the discretized dynamics (81b) result in a sparser problem than alternative formulations (for example, pseudospectral methods), which benefits a real-time solution [191]. With this choice, discretization introduces only two artifacts: the control signal has fewer degrees of freedom (DoF), and the objective function (81a) is off from (77a) by a discretization error. Thus, using an interpolating polynomial discretization enables rigorously stating that (81) finds a "local" optimum for a problem that is "almost" (77). The term "local" is used because the control function has fewer degrees of freedom, and "almost" is employed because of discretization error in the objective function. At convergence, this means that the GuSTO solution satisfies the Pontryagin maximum principle for a slightly different problem than (38). In practice, this technical discrepancy makes little to no difference.

Implementation Details

A complete description of the SCvx and GuSTO algorithms has now been provided, in particular, all the elements that are needed to go around, and eventually exit, the SCP loop in Figure 11. This section discusses two implementation details that significantly improve the performance of both algorithms. The first detail concerns the temporal discretization procedure, and the second detail is about variable scaling.

Temporal Discretization

The core task of discretization is to convert a continuous-time LTV dynamics constraint into a discrete-time constraint. For SCvx, this means converting (50b) to (54b). For GuSTO, this means converting (77b) to (81b). In all cases, the approach is to find the equivalent discrete-time representation of the consistent flow map ψ from Definition 2. The details of this conversion depend entirely on the type of discretization. One example is given in detail for FOH, which is an interpolating polynomial method, in "Discretizing Continuous-Time Optimal Control Problems." This example encapsulates a core issue with many other discretization schemes, so it will be used to ground the discussion.

In FOH discretization, the integrals in (S32) require the continuous-time reference trajectory $\{\bar{x}(t), \bar{u}(t), \bar{p}\}_0^1$ to evaluate the corresponding Jacobians in (43). However, the convex subproblem solution yields only a discrete-time trajectory. To obtain the continuous-time reference, use the continuous-time input obtained directly from (S28) and integrate (58) in tandem with the integrals in (S32). This operation is implemented over a time interval $[t_k, t_{k+1}]$ as one big integration of a concatenated time derivative composed of (58) and all the integrands in (S32) [63]. This saves computational resources by not repeating multiple integrals and has the numerical advantage of letting an adaptive step integrator automatically regulate the temporal resolution of the continuous-time reference trajectory. Because the integration is reset at the end of each time interval, the continuous-time reference state trajectory is discontinuous, as illustrated in Figure 15. Further details are provided in [63] and [191] and the source code of the numerical examples for this article, which is linked in Figure 2.

In theory, discretization occurs in the forward path of the SCP loop, just before the subproblem solution, as shown

in Figure 11. However, by integrating (58) as described previously, it is actually possible obtain the SCvx defects that are needed at stage ③ in Figure 11. Integrating the flow map twice (once for discretization and another time for the defects) is clearly wasteful. Hence, in practice, discretization is implemented at stage ③ in Figure 11, and the result is stored in memory to be used at the next iteration. Note that there is no computational benefit when the flow map does not need to be integrated, such as in (57). This is the case for many discretization schemes, including forward Euler, Runge–Kutta, and pseudospectral methods. The unifying theme of these methods is that the next discrete-time state is obtained algebraically as a function of (a subset of) the other discrete-time states rather than through a numerical integration process.

Variable Scaling

Convex SCP subproblems consist of the following optimization variables: the states, inputs, parameter vector, and possibly a number of virtual controls. The *variable scaling* operation uses an invertible function to transform the optimization variables into a set of new "scaled" variables. The resulting subproblems are completely equivalent, but the magnitudes of the optimization variables are different [26], [29]. SCvx and GuSTO are not scale-invariant algorithms. This means that good variable scaling can significantly impact not only how quickly a locally optimal solution is found but also the solution quality (that is, the level of optimality it achieves). Hence, it is imperative that the reader applies good variable scaling when using SCP to solve trajectory problems. A standard variable scaling technique is now presented that is routinely used successfully in practice.

To motivate the scaling approach, let us review the two major effects of variable magnitude on SCP algorithms. The first effect is common throughout scientific computing and arises from the finite precision arithmetic used by modern computers [13], [29], [32], [198], [199]. When variables have very different magnitudes, a lot of numerical error can accumulate over the iterations of a numerical optimization algorithm [29]. Managing variables of very different magnitudes is a common requirement for numerical optimal control, where optimization problems describe physical processes. For example, the state may include the energy (measured in Joules) and angle (measured in radians). The former might be on the order of 10^6, while the latter is on the order of 10^0 [200]. Most algorithms will struggle to navigate the resulting decision space, which is extremely elongated along the energy axis.

The second effect of different variable magnitudes occurs in the formulation of the trust region constraint (45). This constraint mixes all the states, inputs, and parameters into a single sum on its left-hand side. We have long been taught not to compare apples and oranges, and yet (without scaling), this is exactly what is done in (45). Reusing the previous example, a trust region radius $\eta = 1$ means different things for an angle state than it does for an energy state. It would effectively bias progress to the angle state while allowing almost no progress in the energy state. However, if the variables are scaled to nondimensionalize their values, then the components in the left-hand side of (45) become comparable, and the sum is valid. Thus, variable scaling plays an important role in ensuring that the trust region radius η is "meaningful" across states, inputs, and parameters. Without variable scaling, SCP algorithms usually have a very hard time converging to feasible solutions.

There is now an understanding that variable scaling should seek to nondimensionalize the state, input, and parameter vector to make them comparable. Furthermore, it should bound the values to a region where finite precision arithmetic is accurate. To this end, the following affine transformation has been successfully deployed across a broad spectrum of trajectory optimization research:

$$x = S_x \hat{x} + c_x, \tag{93a}$$

$$u = S_u \hat{u} + c_u, \tag{93b}$$

$$p = S_p \hat{p} + c_p, \tag{93c}$$

where $\hat{x} \in \mathbb{R}^n$, $\hat{u} \in \mathbb{R}^m$, and $\hat{p} \in \mathbb{R}^d$ are the new scaled variables. The user-defined matrices and offset vectors in (93) are chosen so that the scaled state, input, and parameter vectors are roughly bounded by a unit hypercube: $\hat{x} \in [0, 1]^n$, $\hat{u} \in [0, 1]^m$, and $\hat{p} \in [0, 1]^d$. Another advantage of using a $[0, 1]$ interval is that box constraint lower bounds on the variables can be enforced "for free" if the low-level convex solver operates on nonnegative variables [63].

To give a concrete example of (93), suppose that the state is composed of two quantities: a position that takes values in $[100, 1000]$ m and a velocity that takes values in $[-10, 10]$ ms^{-1}. Then, choose $S_x = \text{diag}(900, 20) \in \mathbb{R}^{2\times 2}$ and $c_x = (100, -10) \in \mathbb{R}^2$. Most trajectory problems have enough problem-specific information to find an appropriate scaling. When exact bounds on possible variable values are unknown, an engineering approximation is sufficient.

Discussion

The previous two sections make it clear that SCvx and GuSTO are two instances of SCP algorithms that share many practical and theoretical properties. The algorithms even share similar parameters sets, which are listed in Table 2. From a practical point of view, SCvx can be shown to have superlinear convergence rates under some mild additional assumptions. On the other hand, the theory of GuSTO allows one to leverage additional information from dual variables to accelerate the algorithm, providing quadratic convergence rates. Finally, the theoretical analysis of both methods can be extended to account for dynamics manifold-type constraints without modifying the algorithms [69], [72]. For example, the orientation of a quadrotor can be parameterized with a unit

quaternion by representing it as a 4D vector that is constrained to lie on the surface of a sphere [201]. The GuSTO algorithm has also been extended to handle stochastic problem settings [71], [73].

Making Algorithm Modifications

The end of the "Lossless Convexification" section gave advice for the development of new LCvx methods. For LCvx, new developments are primarily motivated by the fact that the existing set of LCvx results is restricted to a rather specific set of trajectory generation problems. Although SCP methods can solve a much more general set of optimal control tasks, it is still true that particular problem instances might encourage the reader to introduce new elements into the solution algorithm to improve performance and handle new features. In this case, the reader will wonder whether the properties of SCvx and GuSTO will be inherited by the new algorithm.

Although theoretical convergence proofs of SCvx and GuSTO do rely on a set of assumptions, these assumptions are not strictly necessary for the algorithms to work well in practice. Numerical experience suggests that SCP methods can be deployed to solve diverse and challenging problem formulations that do not necessarily satisfy the theory of the previous two sections. It is often the case that what works best in practice cannot (yet) be proved rigorously in theory [186]. As Renegar writes regarding convex optimization [202, p. 51], "It is one of the ironies of the IPM literature that algorithms which are more efficient in practice often have somewhat worse complexity bounds." The same applies for SCP methods, where algorithms that work better in practice may admit weaker theoretical guarantees in the general case [203].

The reader should thus feel free to modify and adapt the methods based on the requirements of their particular problem. In general, a "modify first, prove later" approach to algorithm development is suggested. For example, when using SCvx, it is possible to significantly speed up convergence by entirely replacing the trust region update step with a soft trust region penalty in the cost function. This results in a so-called penalized trust region method that has been used successfully to solve a variety of rocket landing and quadrotor trajectory generation tasks [6], [52], [59], [62], [63], [65].

Trajectory Replanning

This article focuses on the problem of generating complete trajectories between certain boundary conditions. Once generated, the trajectory remains fixed, and a separate feedback control law is used to make the vehicle follow the trajectory robustly and reliably [37], [204], [205]. Figure 1 shows a block

TABLE 2 **A summary of the user-selected parameters for the SCvx and GuSTO algorithms.**

Parameter Dependence

SCvx	N	q	$\hat{q}$	P		λ			η	η_0	η_1	ρ_0	ρ_1	ρ_2	β_{sh}	β_{gr}				ε	ε_r
GuSTO	N	q	$\hat{q}$	P	h_λ		λ_0	λ_{max}	η	η_0	η_1	ρ_0	ρ_1		β_{sh}	β_{gr}	γ_{fail}	μ	k_*	ε	ε_r

Parameter Key

N	$\mathbb{N}$	Density of temporal discretization
q	$\{1,2,2^+,\infty\}$	Trust region norm
$\hat{q}$	$\{1,2,2^+,\infty\}$	Stopping criterion norm
P	$\mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}_+$	Positive definite penalty function
h_λ	$\mathbb{R} \to \mathbb{R}_+$	Convex nondecreasing soft penalty function
λ	$\mathbb{R}_{++}$	Penalty weight
λ_0, λ_{max}	≥ 1	Minimum (initial) and maximum penalty weight
η	$\mathbb{R}_{++}$	Initial trust region radius
η_0, η_1	$\mathbb{R}_{++}$	Minimum and maximum trust region radii
ρ_0, ρ_1, ρ_2	(0,1)	Trust region update parameters
β_{sh}, β_{gr}	>1	Trust region shrinking and growth factors
γ_{fail}	$\mathbb{R}_{++}$	Penalty weight growth factor
μ	(0, 1)	Trust region exponential shrink rate
k_*	$\mathbb{N}$	First iteration when shrinking is applied
ε	$\mathbb{R}_{++}$	Absolute trajectory change tolerance
ε_r	$\mathbb{R}_{++}$	Relative cost improvement tolerance

diagram of this setup. However, one may naturally be curious to apply the same SCP (or LCvx) methods to *replan* trajectories at some frequency. This can allow the trajectory generation algorithm to account for newly arriving information about the vehicle's state and the environment.

Several variations of the replanning idea have appeared in the literature. One standard approach is to simply solve the trajectory problem again, using the current trajectory as the initial guess. As illustrated in Figure 18, the final time is reduced such that only the remaining portion of the trajectory is computed. This method was demonstrated for real-time mobile obstacle avoidance with a quadrotor [206]. However, there is no guarantee, at present, of recursive feasibility, meaning that a replanning step might fail to converge despite a sequence of prior successful solutions. Recursive feasibility and the associated task of characterizing the "domain of attraction" for SCP convergence are open research questions.

Funnel libraries offer an alternative methodology for trajectory replanning that guarantees recursive feasibility. The core idea is to synthesize robust controllers around a batch of SCP trajectories. Each controller induces a funnel-like region of space, where it locally asymptotically stabilizes the vehicle around the associated trajectory. By composing these controllers and trajectories, a so-called funnel library is created that can be used as a lookup table for robust and optimization-free onboard trajectory generation. This is a relatively new approach, and examples of its usage have appeared for vehicles such as quadrotors, fixed-wing airplanes, and rocket-powered planetary landers [180], [181], [207]–[209]. The primary challenge of this family of algorithms is to efficiently fill the required portion of the state space with funnels, which becomes an increasingly challenging proposition for high-dimensional spaces. A secondary challenge is numerical sensitivity since robust controller synthesis relies on semidefinite programming, which is the most algorithmically challenging form of convex optimization.

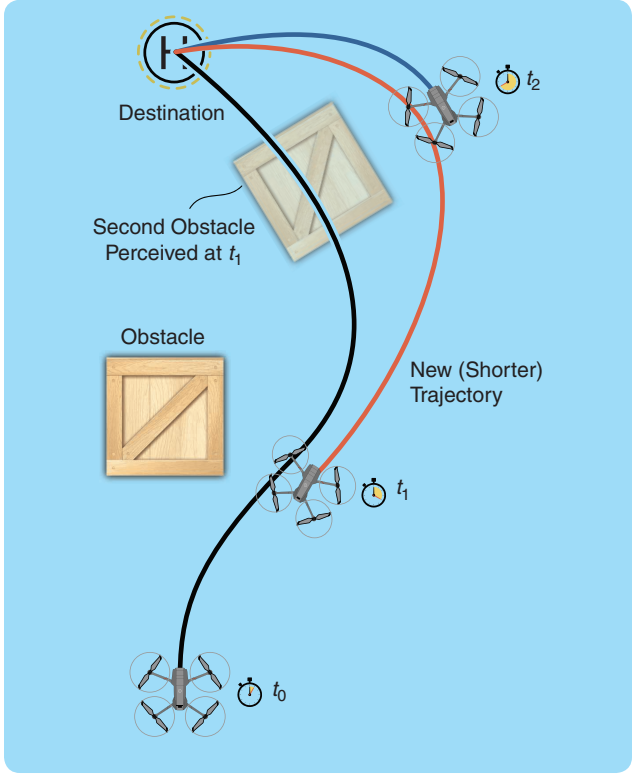


FIGURE 18 Illustration of trajectory replanning with sequential convex programming. At time t_0, the quadrotor plans a trajectory to the destination, with one obstacle in the environment. At time t_1, there is a tracking error in addition to a second obstacle that comes into the quadrotor camera's field of view. A new trajectory is planned for the remaining segment of the flight, which avoids the new obstacle. At time t_2, a third trajectory is generated that accounts for tracking error, this time with no new obstacles. At each solution instance, the previous trajectory serves as the initial guess. The trajectories are tracked using the quadrotor's own feedback controllers.

Distinct from the approach of tracking trajectories with a separate feedback controller, the theory and practice of MPC have also been applied to simultaneous trajectory generation and control. For example, the methodology was used for planetary as well as asteroid rocket landing [125], [210], [211]. Notably, the temporal aspect of MPC can be leveraged to approximate a nonconvex trajectory generation problem as a convex quadratic optimization problem at each time step. Some research has also appeared on using SCP to solve nonlinear (nonconvex) MPC problems. Applications include the control of hydroelectric power plants [212], ground vehicles [213], swarm reconfiguration for orbiting spacecraft [214], [215], continuous stirred-tank chemical reactors, and electromagnetically actuated mass–spring–damper systems [216]. The interested reader can find more discussion of MPC for trajectory generation in the dedicated survey articles [6] and [78].

To conclude this section, note that SCvx, GuSTO, and related SCP methods have been used to solve a plethora of trajectory generation problems, including reusable launch vehicles [217]–[219], robotic manipulators [182], [220], [221], robot motion planning [207], [222], [223], and other examples mentioned in the introduction and at the beginning of this part of the article. All these SCP variants and applications should inspire the reader to develop his or her own SCP method that works best for a particular application.

APPLICATION EXAMPLES

The first and second parts of this article provided the theoretical background necessary to start solving nonconvex trajectory generation problems by using LCvx, SCvx, and GuSTO. This part is dedicated to doing just that: providing examples of how trajectory generation problems can be solved using the three algorithms. Rocket landing is covered first, followed by quadrotor flight with obstacle avoidance and, finally, 6-DoF flight of a robot inside a space station. The implementation code that produces the exact

plots in this part of the article is entirely available in the *SCP Toolbox* linked in Figure 2.

Lossless Convexification: Three-Degrees-of-Freedom, Fuel-Optimal Rocket Landing

Rocket-powered planetary landing guidance, also known as *powered descent guidance* (*PDG*), was the original motivation for the development of LCvx. It makes use of several LCvx results presented in the "Lossless Convexification" section, making it a fitting first example. Work on PDG began in the late 1960s [19] and has since been extensively studied [224]. The objective is to find a sequence of thrust commands that transfer the vehicle from a specified initial state to a desired landing site without consuming more propellant than what is available. Using optimization to compute this sequence can greatly increase the range of feasible landing sites and precision of the final landing location [225]–[227].

LCvx for PDG was first introduced in [45] and [85], where minimizing fuel usage was the objective. It was the first time that convex optimization was shown to be applicable to PDG. This discovery unlocked a polynomial time algorithm with guaranteed convergence properties for generating optimal landing trajectories. The work was later expanded to handle state constraints [86], [93], [99], minimize the landing site miss distance [102], include nonconvex pointing constraints [S1], [89], and handle nonlinear terms in the dynamics, such as aerodynamic drag and nonlinear gravity [46]. Today, it is known that SpaceX uses convex optimization for the Falcon 9 rocket landing algorithm [36]. Flight tests have also been performed using LCvx in a collaboration between NASA and Masten Space Systems [37], as shown in Figure 19.

FIGURE 19 The Masten *Xombie* rocket near the end of a 750-m divert maneuver. (Source: [37, Fig. 1]; used with permission.) (a) Close-up of the rocket executing an optimal divert trajectory. (b) Preflight checkout by the Masten Space Systems staff. (c) A zoomed-out view with the Mojave desert in the background.

An LCvx PDG example is now presented based on a mixture of original ideas from [45] and [89]. Note that LCvx considers the 3-DoF PDG problem, where the vehicle is modeled as a point mass. This model is a good approximation as long as the rotational and translational states are weakly coupled such that the attitude can be changed rapidly and without significantly perturbing the vehicle's position. This is a valid approximation for many vehicles, including quadrotors and rockets with high attitude control authority. For example, the Mars Science Laboratory uses differential throttling for attitude control and can be approximated in this way [225], [228]. In the 3-DoF model, the thrust vector is taken as the control input, where its direction serves as a proxy for the vehicle's attitude.

Let us begin by stating the raw minimum-fuel 3-DoF PDG problem:

$$\min_{T_c, t_f} \int_0^{t_f} \| T_c(t) \|_2 \, \mathrm{d}t, \tag{94a}$$

$$\text{s.t. } \dot{r}(t) = v(t), \tag{94b}$$

$$\dot{v}(t) = g + \frac{T_c(t)}{m(t)} - \omega^\times \omega^\times r(t) - 2\omega^\times v(t), \tag{94c}$$

$$\dot{m}(t) = -\alpha \| T_c(t) \|_2, \tag{94d}$$

$$\rho_{\min} \leq \| T_c(t) \|_2 \leq \rho_{\max}, \tag{94e}$$

$$T_c(t)^\top \hat{e}_z \geq \| T_c(t) \|_2 \cos(\gamma_p), \tag{94f}$$

$$H_{\mathrm{gs}} r(t) \leq h_{\mathrm{gs}}, \tag{94g}$$

$$\| v(t) \|_2 \leq v_{\max}, \tag{94h}$$

$$m_{\mathrm{dry}} \leq m(t_f), \tag{94i}$$

$$r(0) = r_0,\ v(0) = v_0,\ m(0) = m_{\mathrm{wet}}, \tag{94j}$$

$$r(t_f) = v(t_f) = 0. \tag{94k}$$

The vehicle translational dynamics correspond to a double integrator with variable mass, moving in a constant gravitational field and viewed in the planet's rotating frame. In particular, $r \in \mathbb{R}^3$ is the position, $v \in \mathbb{R}^3$ is the velocity, $g \in \mathbb{R}^3$ is the gravitational acceleration, and $\omega \in \mathbb{R}^3$ is the planet's constant angular velocity. The notation $\omega^\times$ denotes the skew-symmetric matrix representation of the cross product $\omega \times (\cdot)$. The mass $m \in \mathbb{R}$ is depleted by the rocket engine according to (94d), with the fuel consumption rate

$$\alpha \triangleq \frac{1}{I_{\mathrm{sp}} g_e}, \tag{95}$$

where $g_e \approx 9.807\ \mathrm{ms}^{-2}$ is Earth's standard gravitational acceleration and I_{sp} is the rocket engine's specific impulse.

As illustrated in Figure 20, $T_c \in \mathbb{R}^3$ is the rocket engine thrust vector, which is upper and lower bounded via (94e). The lower bound was motivated in "Convex Relaxation of an Input Lower Bound." The thrust vector, and therefore the vehicle attitude, also has a tilt angle constraint (94f) that prevents the vehicle from deviating by more than angle γ_p away from the vertical. Following the discussion in "Landing Glideslope as an Affine State Constraint," an affine glideslope constraint is also imposed via (94g) with a maximum glideslope angle γ_{gs}. The velocity is constrained to a maximum magnitude v_{max} by (94h). The final mass must be greater than m_{dry} (94i), which ensures that no more fuel is consumed than what is available. Constraints (94j) and (94k) impose fixed boundary conditions on the rocket's state. In particular, the starting mass is $m_{wet} > m_{dry}$.

To begin the LCvx process, the standard input slack variable $\sigma \in \mathbb{R}$ from "Convex Relaxation of an Input Lower Bound" is introduced to remove the nonconvex lower bound in (94e). As a consequence, the familiar LCvx equality constraint appears in (96f). Following "Convex Relaxation of an Input Pointing Constraint," this also replaces $\| T_c(t) \|_2$ in (94f) with $\sigma(t)$ in (96g), resulting in the following problem:

$$\min_{\sigma, T_c, t_f} \int_0^{t_f} \sigma(t)\,dt, \tag{96a}$$

$$\text{s.t. } \dot{r}(t) = v(t), \tag{96b}$$

$$\dot{v}(t) = g + \frac{T_c(t)}{m(t)} - \omega^\times \omega^\times r(t) - 2\omega^\times v(t), \tag{94c}$$

$$\dot{m}(t) = -\alpha\sigma(t), \tag{96d}$$

$$\rho_{min} \le \sigma(t) \le \rho_{max}, \tag{96e}$$

$$\| T_c(t) \|_2 \le \sigma(t), \tag{96f}$$

$$T_c(t)^\top \hat{e}_z \ge \sigma(t)\cos(\gamma_p), \tag{96g}$$

$$H_{gs} r(t) \le h_{gs}, \tag{96h}$$

$$\| v(t) \|_2 \le v_{max}, \tag{96i}$$

$$m_{dry} \le m(t_f), \tag{96j}$$

$$r(0) = r_0,\ v(0) = v_0,\ m(0) = m_{wet}, \tag{96k}$$

$$r(t_f) = v(t_f) = 0. \tag{96l}$$

Next, the nonlinearity T_c/m in (94c) must be approximated. To this end, [45] showed that the following change of variables can be made:

$$\xi \triangleq \frac{\sigma}{m},\ u \triangleq \frac{T_c}{m},\ z \triangleq \ln(m). \tag{97}$$

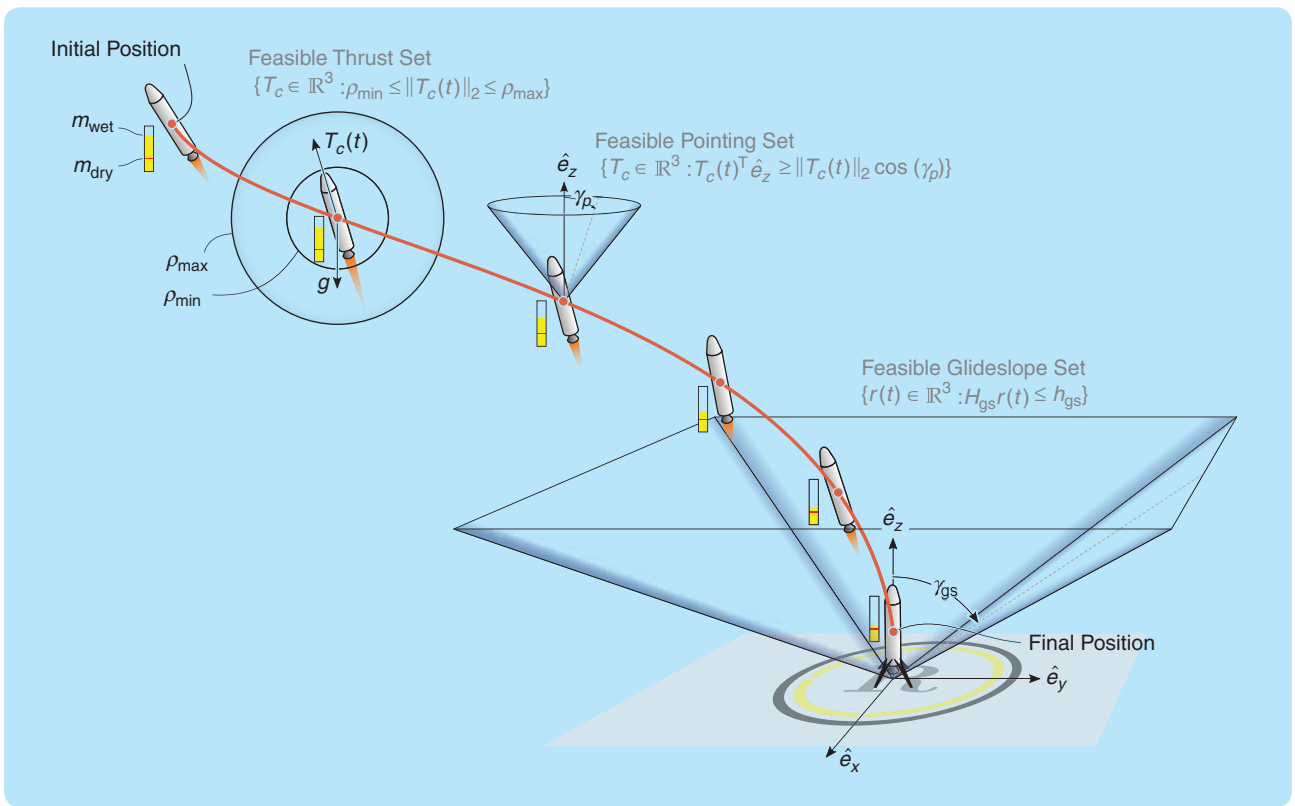


FIGURE 20 The three-degrees-of-freedom powered descent guidance problem, showing some of the relevant constraints on the rocket-powered lander's trajectory. The thrust direction $T_c(t)/\| T_c(t) \|_2$ servers as a proxy for the vehicle attitude.

Using the new variables, note that

$$\frac{\dot{m}(t)}{m(t)} = -\alpha\xi(t) \Rightarrow z(t) = z(0) - \alpha\int_0^t \xi(\tau)\,\mathrm{d}\tau. \tag{98}$$

Since the cost in (96a) maximizes $m(t_f)$ and because $\alpha > 0$, an equivalent cost is to minimize $\int_0^{t_f} \xi(t)\,\mathrm{d}t$. The new variables linearize all the constraints except for the upper bound part of (96e). In the new variables, (96e) becomes

$$\rho_{\min} e^{-z(t)} \le \xi(t) \le \rho_{\max} e^{-z(t)}. \tag{99}$$

To keep the optimization problem an SOCP, it is desirable to also do something about the lower bound in (99), which is a convex exponential cone. In [45], it was shown that the following Taylor series approximation is accurate to within a few percentage points and therefore acceptable:

$$\mu_{\min}(t)\left[1 - \delta z(t) + \frac{1}{2}\delta z(t)^2\right] \le \xi(t), \tag{100a}$$

$$\mu_{\max}(t)[1 - \delta z(t)] \ge \xi(t), \tag{100b}$$

where

$$\mu_{\min}(t) = \rho_{\min} e^{-z_0(t)}, \tag{101a}$$

$$\mu_{\max}(t) = \rho_{\max} e^{-z_0(t)}, \tag{101b}$$

$$\delta z(t) = z(t) - z_0(t), \tag{101c}$$

$$z_0(t) = \ln(m_{\text{wet}} - \alpha\rho_{\max} t). \tag{101d}$$

The reference profile $z_0(\cdot)$ corresponds to the maximum fuel rate; thus, $z_0(t)$ lower bounds $z(t)$. To ensure that physical bounds on $z(t)$ are not violated, the following extra constraint is imposed:

$$z_0(t) \le z(t) \le \ln(m_{\text{wet}} - \alpha\rho_{\min} t). \tag{102}$$

The constraints (100) and (102) together approximate (99) and have the important property of being conservative with respect to the original constraint [45, Lemma 2]. Hence, using this approximation *will not generate solutions that are infeasible for the original problem*. The finalized convex relaxation of (94) can now be written:

$$\min_{\xi,u,t_f} \int_0^{t_f} \xi(t)\,\mathrm{d}t, \tag{103a}$$

$$\text{s.t. } \dot{r}(t) = v(t), \tag{103b}$$

$$\dot{v}(t) = g + u(t) - \omega^\times\omega^\times r(t) - 2\omega^\times v(t), \tag{103c}$$

$$\dot{z}(t) = -\alpha\xi(t), \tag{103d}$$

$$\mu_{\min}(t)\left[1 - \delta z(t) + \frac{1}{2}\delta z(t)^2\right] \le \xi(t), \tag{103e}$$

$$\mu_{\max}(t)[1 - \delta z(t)] \ge \xi(t), \tag{103f}$$

$$\|u(t)\|_2 \le \xi(t), \tag{103g}$$

$$u(t)^\top \hat{e}_z \ge \xi(t)\cos(\gamma_p), \tag{103h}$$

$$H_{\text{gs}} r(t) \le h_{\text{gs}}, \tag{103i}$$

$$\|v(t)\|_2 \le v_{\max}, \tag{103j}$$

$$\ln(m_{\text{dry}}) \le z(t_f), \tag{103k}$$

$$z_0(t) \le z(t) \le \ln(m_{\text{wet}} - \alpha\rho_{\min} t), \tag{103l}$$

$$r(0) = r_0,\ v(0) = v_0,\ z(0) = \ln(m_{\text{wet}}), \tag{103m}$$

$$r(t_f) = v(t_f) = 0. \tag{103n}$$

Several conditions must now be checked to ensure LCvx, that is, the solution of (103) is globally optimal for (94). To begin, view z in (103d) as a "fictitious" state that is used for concise notation. In practice, it is explicitly known that $z(t) = \ln(m_{\text{wet}}) - \alpha\int_0^t \xi(t)\,\mathrm{d}t$, and thus, every instance of $z(t)$ in (103e)–(103l) can be replaced with this expression. Therefore, z is *not* considered part of the state. Furthermore, (103e), (103f), (103k), and (103l) are *input* and not state constraints. Defining the state as $x = (r, v) \in \mathbb{R}^6$, the state-space matrices are

$$A = \begin{bmatrix} 0 & I_3 \\ -\omega^\times\omega^\times & -2\omega^\times \end{bmatrix},\ B = \begin{bmatrix} 0 \\ I_3 \end{bmatrix},\ E = B. \tag{104}$$

The pair $\{A, B\}$ is unconditionally controllable, and hence, Condition 1 is satisfied. Condition 3 must also be verified due to the presence of the pointing constraint (103h). In this case $\hat{n}_u = \hat{e}_z$, and $N = [\hat{e}_x\ \ \hat{e}_y]$. Condition 3 holds as long as $\omega^\times\hat{e}_z \ne 0$, which means that the planet does not rotate about the local vertical of the landing frame.

The glideslope constraint (103i) can be treated either via Condition 5 or by checking that it can be only instantaneously active. In this case, the latter can be proved by considering a force balance in the glideslope plane's normal direction $\hat{n}_{gs}$, as illustrated in Figure 21. To aid the proof, recall that the optimal thrust profile for the rocket landing problem is a bang–bang one [45]. This allows for distilling the verification down to a conservative condition that the following inequalities hold for all $\theta \in [\pi/2 - \gamma_p - \gamma_{gs}, \pi/2 + \gamma_p - \gamma_{gs}]$:

$$\rho_{\min}\cos(\theta) < m_{\text{dry}}\|g\|_2\sin(\gamma_{\text{gs}}), \tag{105a}$$

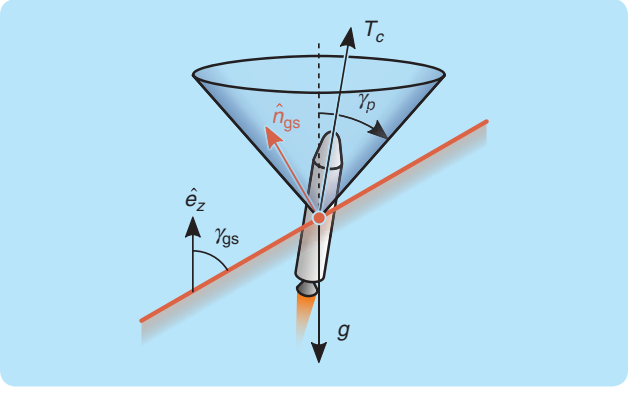


FIGURE 21 A force balance in the normal direction $\hat{n}_{\text{gs}}$ can be used to guarantee that the glideslope constraint can be activated only instantaneously.

$$\rho_{\max}\cos(\theta) > m_{\text{wet}} \| g \|_2 \sin(\gamma_{\text{gs}}). \tag{105b}$$

For the problem parameters (107) of this example, the preceding inequalities hold. This means that if the rocket is situated on the glideslope, either the glideslope constraint will be violated at the next time step when using minimum thrust, or a higher glideslope angle will be achieved when using maximum thrust. Therefore, (103i) can be only instantaneously active and does not pose a threat to LCvx.

Condition 2 must also be checked, which relates to the transversality condition of the maximum principle [11], [12], [104]. In the case of (103), the current problem has $m[t_f] = 0$ and $b[t_f] = x(t_f)$. Hence,

$$m_{\text{LCvx}} = \begin{bmatrix} 0 \in \mathbb{R}^6 \\ \xi(t_f) \end{bmatrix}, \; B_{\text{LCvx}} = \begin{bmatrix} I_6 \\ 0 \end{bmatrix}. \tag{106}$$

Thus, as long as $\xi(t_f) > 0$, Condition 2 holds. Since $\xi(t_f) > 0$ is guaranteed by the fact that the lower bound in (100) is greater than the exact lower bound $\rho_{\min} e^{-z(t_f)} > 0$ [45, Lemma 2], Condition 2 holds.

The remaining constraint to be checked is the maximum velocity bound (103j). Although it can be written as the quadratic constraint $v_{\max^{-2}} v(t)^\top v(t) \leq 1$, the dynamics (103b) and (103c) do not match the form (17b) and (17c) required by the LCvx result for quadratic state constraints. Thus, the more restricted statement for general state constraints in Theorem 5 must be used. According to Theorem 5, LCvx will hold as long as the maximum velocity bound (103j) is activated, at most, a discrete number of times. In summary, the solution of (103) is guaranteed to be globally optimal for (94) as long as (103j) is never persistently active.

A landing trajectory example is obtained by solving (103) using a ZOH discretized input signal with a $\Delta t = 1\ s$ time step (see "Discretizing Continuous-Time Optimal Control Problems"). Let us solve the problem with the following numerical parameters:

$$g = -3.71\hat{e}_z \text{ m s}^{-2}, \; m_{\text{dry}} = 1505 \text{ kg}, \tag{107a}$$

$$m_{\text{wet}} = 1905 \text{ kg}, \; I_{\text{sp}} = 225 \text{ s}, \tag{107b}$$

$$\omega = (3.5\hat{e}_x + 2\hat{e}_z) \cdot 10^{-3} \circ \text{s}^{-1}, \tag{107c}$$

$$\rho_{\min} = 4.971 \text{ kN}, \; \rho_{\max} = 13.258 \text{kN}, \tag{107d}$$

$$\gamma_{gs} = 86^\circ, \; \gamma_p = 40^\circ, \; v_{\max} = 500 \text{km/h}^{-1}, \tag{107e}$$

$$r_0 = 2\hat{e}_x + 1.5\hat{e}_z \text{ km}, \tag{107f}$$

$$v_0 = 288\hat{e}_x + 108\hat{e}_y - 270\hat{e}_z \text{ km/h}^{-1}. \tag{107g}$$

Golden search [48] is used to find the optimal time of flight t_f. This is a valid choice because the cost function is unimodal with respect to t_f [102]. For the problem parameters in (107), an optimal rocket landing trajectory is found with a minimum-fuel time of flight $t_f^* = 75$ s. Figure 22 visualizes the computed optimal landing trajectory. Figure 22(a) clearly shows that the glideslope constraint is not only satisfied, it is also activated only twice (once during midflight and another time at touchdown), as required by the LCvx guarantee. Figure 22(c) clearly shows that the velocity constraint is never activated. Hence, in this case, it does not pose a threat to LCvx.

Several other interesting views of the optimal trajectory are plotted in Figure 22(d)–(g). In all cases, the state and input constraints of (94) hold. The fact that (103) is an LCvx of (94) is most evident during the minimum-thrust segment from approximately 40 to 70 s in Figure 22(f). Here, it is important to realize that $\| T_c(t) \|_2 < \rho_{\min}$ is feasible for the relaxed problem. The fact that this never occurs is a consequence of the LCvx guarantee that (103g) holds with equality. In conclusion, it is worth emphasizing that the trajectory presented in Figure 22 is the *globally* optimal solution to this rocket landing problem. This means that one can do no better for the given problem parameters and description.

Sequential Convex Programming: Quadrotor Obstacle Avoidance

Now consider trajectory generation for problems that cannot be handled by LCvx. The objective of this first example is to compute a trajectory for a quadrotor that flies from one point to another through an obstacle-filled flight space. This example is sourced primarily from [52], and a practical demonstration is shown in Figure 23.

The quadrotor is modeled as a point mass, which is a reasonable approximation for small and agile quadrotors whose rotational states evolve over much shorter time scales than the translational states [45]. The equations of motion are expressed in an East–North–Up (ENU) inertial coordinate system, and the state vector is composed of the position and velocity. Using a simple double-integrator model, the continuous-time equations of motion are expressed in the ENU frame as

$$\ddot{r}(\mathsf{t}) = a(\mathsf{t}) - g\hat{n}, \tag{108}$$

where $r \in \mathbb{R}^3$ denotes the position, $g \in \mathbb{R}$ is the (constant) acceleration due to gravity, $\hat{n} = (0, 0, 1) \in \mathbb{R}^3$ is the "up" direction, and $a(\mathsf{t}) \in \mathbb{R}^3$ is the commanded acceleration, which is the control input.

The time t spans the interval $[0, t_f]$. Because the previous section on SCP used a normalized time $t \in [0, 1]$, t is called the *absolute time*, and a special font is used to denote it. The trajectory duration is optimized, and the following constraint is imposed to keep the final time bounded:

$$t_{f,\min} \leq t_f \leq t_{f,\max}, \tag{109}$$

where $t_{f,\min} \in \mathbb{R}$ and $t_{f,\max} \in \mathbb{R}$ are user-defined parameters. Boundary conditions on the position and velocity are imposed to ensure that the vehicle begins and ends at the desired states:

$$r(0) = r_0, \quad \dot{r}(0) = v_0, \tag{110a}$$

$$r(t_f) = r_f, \quad \dot{r}(t_f) = v_f. \tag{110b}$$

The magnitude of the commanded acceleration is limited from above and below by the electric motor and propeller

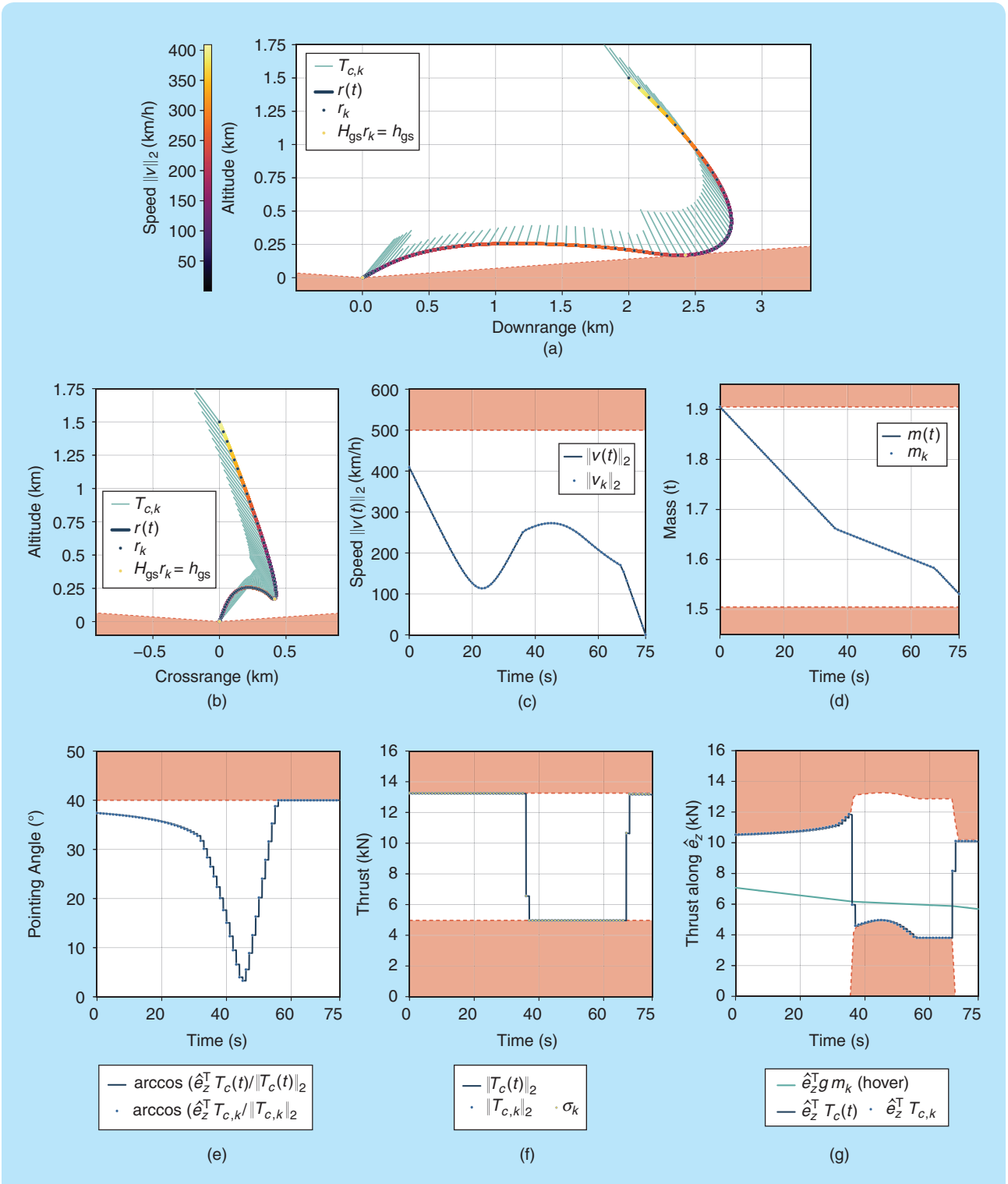


FIGURE 22 Various views of the globally optimal rocket landing trajectory obtained via lossless convexification (LCvx) for (94). Circular markers show the discrete-time trajectory directly returned from the convex solver. Lines show the continuous-time trajectory, which is obtained by numerically integrating the zeroth-order hold discretized control signal through the dynamics (94b)–(94d). The exact match at every discrete-time node demonstrates that the solution is dynamically feasible for the actual continuous-time vehicle. (a) The side view of the optimal landing trajectory. The glideslope constraint (104i) is activated at a single time instance in midflight. (b) The front-on view of the optimal landing trajectory. (c) The velocity history. Constraint (104j) is never activated. (d) The mass history. The vehicle uses a feasible amount of fuel. (e) The pointing angle history. Constraint (104h) gets activated. (f) The thrust magnitude history. The LCvx equality constraint (97f) is tight. (g) The vertical thrust projection. The rocket never hovers.

configuration, and its direction is constrained to model a tilt angle constraint on the quadrotor. In effect, the acceleration direction is used as a proxy for the vehicle attitude. This is an accurate approximation for a "flat" quadrotor configuration, where the propellers are not canted with respect to the plane of the quadrotor body. Specifically, the following control constraints are enforced:

$$a_{\min} \leq \| a(\mathrm{t}) \|_2 \leq a_{\max}, \tag{111a}$$

$$\| a(\mathrm{t}) \|_2 \cos\theta_{\max} \leq \hat{n}^\top a(\mathrm{t}), \tag{111b}$$

where $0 < a_{\min} < a_{\max}$ are the acceleration bounds and $\theta_{\max} \in (0°, 180°]$ is the maximum angle by which the acceleration vector is allowed to deviate from the "up" direction.

Finally, obstacles are modeled as 3D ellipsoidal keep-out zones described by the following nonconvex constraints:

$$\| H_j(r(\mathrm{t}) - c_j) \|_2 \geq 1, \quad j = 1, \ldots, n_{\text{obs}}, \tag{112}$$

where $c_j \in \mathbb{R}^3$ denotes the center and $H_j \in \mathbb{S}^3_{++}$ defines the size and shape of the jth obstacle. Note that the formulation allows the obstacles to intersect. Using the preceding equations, the following free final time optimal control problem is to be solved:

$$\min_{u, t_f} t_f^{-1} \int_0^{t_f} \| a(\mathrm{t}) \|_2^2 \mathrm{dt}, \tag{113a}$$

$$\text{s.t. (108)–(112).} \tag{113b}$$

The cost function (113a) seeks to minimize the average specific (that is, per-unit mass) control power, normalized by the flight time. The normalization yields a measure of the average control power sustained over a 1-s time interval. For short, refer to this as a "minimum control power" cost. In the literature, this cost is sometimes called "minimum control energy" despite the slight misnomer.

Due to the presence of nonconvex state constraints (112), only embedded LCvx applies to the preceding problem. In particular, LCvx can be used to handle the nonconvex input lower bound in (111a) along with the tilt constraint in (111b). This removes some of the nonconvexity. However, it is only a partial convexification, which still leaves behind a nonconvex optimal control problem. Thus, SCP techniques must be used for the solution.

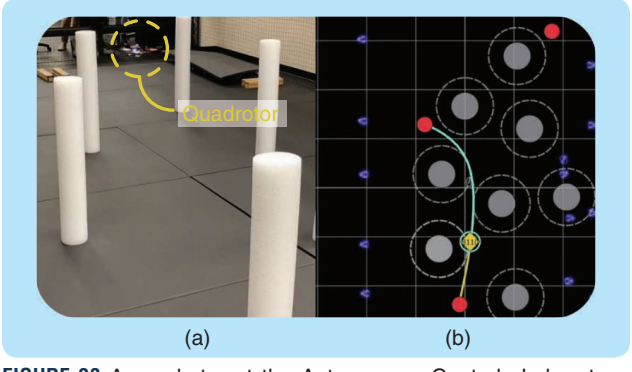


FIGURE 23 A quadrotor at the Autonomous Controls Laboratory, University of Washington, executing a collision-free trajectory computed by sequential convex programming [52], [245], [246]. (a) The quadrotor at the start of its trajectory in the indoor flight space. (b) A graphical user interface showing an obstacle-avoiding trajectory planned by SCP that is to be flown.

SCvx Formulation

Begin by demonstrating how SCvx can be used to solve (113). The first and main step is to cast the problem into the template of (38). Once this is done, the rest of the solution process is completely automated by the mechanics of the SCvx algorithm, as described in the previous section. To make the notation lighter, the argument of time is omitted whenever possible.

Start by defining the state and input vectors. For the input vector, note that the nonconvex input constraints in (111) can be convexified via embedded LCvx. In particular, the relaxation used for (10) can losslessly convexify both input constraints by introducing a slack input $\sigma \in \mathbb{R}$ and rewriting (111) as

$$a_{\min} \leq \sigma \leq a_{\max}, \tag{114a}$$

$$\| a \|_2 \leq \sigma, \tag{114b}$$

$$\sigma \cos\theta_{\max} \leq \hat{n}^\top a, \tag{114c}$$

where (114b) is the familiar LCvx equality constraint. Thus, the following state and "augmented" input vectors can be defined:

$$x = \begin{bmatrix} r \\ \dot{r} \end{bmatrix} \in \mathbb{R}^6, \quad u = \begin{bmatrix} a \\ \sigma \end{bmatrix} \in \mathbb{R}^4. \tag{115}$$

Next, one must deal with the fact that (38) uses normalized time $t \in [0,1]$, while (113) uses absolute time $\mathrm{t} \in [0, t_f]$. To reconcile the two quantities, a 1D parameter vector $p \in \mathbb{R}$ is used. The parameter defines a *time dilation* such that the following relation holds:

$$\mathrm{t} = pt, \tag{116}$$

from which it follows that $p \equiv t_f$. In absolute time, the dynamics are given directly by writing (108) in terms of (115), which gives a set of time-invariant, first-order, ordinary differential equations (ODEs):

$$f(x,u) = \begin{bmatrix} \dot{r} \\ a - g\hat{n} \end{bmatrix}. \tag{117}$$

For (38), convert the dynamics to normalized time by applying the chain rule:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{\mathrm{d}x}{\mathrm{dt}} \frac{\mathrm{dt}}{\mathrm{d}t} = pf(x,u), \tag{118}$$

which yields the dynamics (38b) in normalized time:

$$f(x,u,p) = pf(x,u). \tag{119}$$

The convex path constraints (38c) and (38d) are simple to write. Although there are no convex state constraints, there

are convex final time bounds (109). These can be included as a convex state path constraint (38c), which is mixed in the state and parameter. Using (116), define the convex state path constraints as

$$\mathcal{X} = \{(x,p) \in \mathbb{R}^6 \times \mathbb{R} : t_{f,\min} \le p \le t_{f,\max}\}. \tag{120}$$

On the other hand, the convex input constraint set $\mathcal{U}$ is given by all the input vectors that satisfy (114). The nonconvex path constraints (38e) are given by the vector function $s : \mathbb{R}^3 \to \mathbb{R}^{n_{\text{obs}}}$, whose elements encode the obstacle avoidance constraints:

$$s_j(r) = 1 - \| H_j(r - c_j) \|_2, \quad j = 1, \ldots, n_{\text{obs}}. \tag{121}$$

It is worthwhile to mention how to evaluate the Jacobian (43e) for (121). Suppose that a reference position trajectory $\{\bar{r}(t)\}_0^1$ is available, and consider the jth obstacle constraint in (121). The following gradient then allows for evaluating (43e):

$$\nabla s_j(r) = -\frac{H_j^\top H_j (r - c_j)}{\| H_j (r - c_j) \|_2}. \tag{122}$$

The boundary conditions (38f) and (38g) are obtained from (110):

$$g_{\text{ic}}(x(0), p) = \begin{bmatrix} r(0) - r_0 \\ \dot{r}(0) - v_0 \end{bmatrix}, \tag{123a}$$

$$g_{\text{tc}}(x(1), p) = \begin{bmatrix} r(1) - r_f \\ \dot{r}(1) - v_f \end{bmatrix}. \tag{123b}$$

Finally, the cost (113a) is converted into the Bolza form (39). There is no terminal cost; hence, $\phi \equiv 0$. On the other hand, the direct transcription of the running cost is $\Gamma(x,u,p) = \sigma^2$ since the time dilation p and inverse final time t_f^{-1} cancel out when the integral is expressed in normalized time. However, because SCvx augments the cost with penalty terms in (48), it is best to normalize the running cost by its nominal value to make it roughly equal to one for a "mild" trajectory. This greatly facilitates the selection of the penalty weight λ. By taking the nominal value of σ as the hover condition, the following running cost is defined:

$$\Gamma(x,u,p) = \left(\frac{\sigma}{g}\right)^2. \tag{124}$$

This completes the specialization of (38) for the quadrotor obstacle avoidance problem. The only remaining task is to choose the SCvx algorithm parameters listed in Table 2. The rest of the solution process is completely automated by the general SCvx algorithm description in the "Sequential Convex Programming" section.

GuSTO Formulation

The GuSTO algorithm can also be used to solve (113). The formulation is almost identical to SCvx, which underscores the fact that the two algorithms can be used interchangeably to solve many of the same problems. The quadratic running cost (67) encodes (124) as follows:

$$S(p) = \operatorname{diag}(0, 0, 0, g^{-2}), \tag{125a}$$

$$\ell(x,p) = 0, \tag{125b}$$

$$g(x,p) = 0. \tag{125c}$$

The dynamics (117) can also be cast into the control affine form (68):

$$f_0(x) = \begin{bmatrix} \dot{r} \\ -g\hat{n} \end{bmatrix}, \tag{126a}$$

$$f_i(x) = \begin{bmatrix} 0 \\ e_i \end{bmatrix}, \quad i = 1, 2, 3, \tag{126b}$$

$$f_4(x) = 0, \tag{126c}$$

where $e_i \in \mathbb{R}^3$ is the ith standard basis vector. The reader may be surprised that these are the only changes required to convert the SCvx formulation from the previous section into a form that can be ingested by GuSTO. The only remaining task is to choose the GuSTO algorithm parameters listed in Table 2. Just like for SCvx, the rest of the solution process is completely automated by the general GuSTO algorithm description in the "Sequential Convex Programming" section.

Initial Trajectory Guess

The state trajectory initial guess is obtained by a simple straight-line interpolation, as provided by (41):

$$x(t) = (1 - t) \begin{bmatrix} r_0 \\ v_0 \end{bmatrix} + t \begin{bmatrix} r_f \\ v_f \end{bmatrix}, \quad \text{for } t \in [0,1]. \tag{127}$$

The initial parameter vector, which is just the time dilation, is chosen to be in the middle of the allowed trajectory durations:

$$p = \frac{t_{f,\min} + t_{f,\max}}{2}. \tag{128}$$

The initial input trajectory is guessed based on physical insight about the quadrotor problem. Ignoring any particular trajectory task, it is known that the quadrotor must generally support its own weight under the influence of gravity. Thus, a constant initial input guess is chosen that would make a static quadrotor hover:

$$a(t) = g\hat{n}, \quad \sigma(t) = g, \quad \text{for } t \in [0,1]. \tag{129}$$

This initial guess is infeasible with respect to both the dynamics and the obstacle constraints, and it is extremely cheap to compute. The fact that it works well in practice highlights two facts: SCP methods are relatively easy to initialize, and they readily accept infeasible initial guesses. Note that in the particular case of this problem, an initial trajectory could also be computed using a convex version of

the problem obtained by removing the obstacle constraints (112) and applying the LCvx relaxation (114).

Numerical Results

The preceding discussion defines a specialized instance of (38) and an initialization strategy for the quadrotor obstacle avoidance problem. The solution is obtained via SCP according to the general algorithm descriptions for SCvx and GuSTO. For temporal discretization, use the FOH interpolating polynomial method from "Discretizing Continuous-Time Optimal Control Problems." The algorithm parameters are provided in Table 3, and the full implementation is available in the code repository linked in Figure 2. An embedded conic solver (ECOS) is used as the numerical convex optimizer at ② in Figure 11 [229]. The timing results are obtained on a Dell XPS 13 9260 laptop powered by an Intel Core i5-7200U CPU clocked at 2.5 GHz. The computer has 8 GiB of low-power double data rate third generation (LPDDR3) random-access memory and 128 KiB level 1, 512 KiB level 2, and 3 MiB level 3 caches.

TABLE 3 **The algorithm parameters for the quadrotor obstacle avoidance example.**

Problem Parameters

t_f,min	0	s	Minimum final time
t_f,max	2.5	s	Maximum final time
g	9.81	ms^{-2}	Gravity
a_{min}	0.6	ms^{-2}	Minimum acceleration
a_{max}	23.2	ms^{-2}	Maximum acceleration
θ_{max}	60	°	Maximum tilt angle
c_1	(1, 2, 0)	m	Center of obstacle 1
c_2	(2, 5, 0)	m	Center of obstacle 2
H_1	diag(2,2,0)	m^{-1}	Shape of obstacle 1
H_2	diag(1.5,1.5,0)	m^{-1}	Shape of obstacle 2
r_0	(0, 0, 0)	m	Initial position vector
v_0	(0, 0, 0)	ms^{-1}	Initial velocity vector
r_f	(2.5, 6, 0)	m	Final position vector
v_f	(0, 0, 0)	ms^{-1}	Final velocity vector

Algorithm Parameters

	SCvx	GuSTO
N	30	30
q	∞	∞
$\hat{q}$	∞	∞
P	(48)	(48)
h_λ		(71)
λ	30	
λ_0, λ_{max}		10^4,10^9
η	1	10
η_0, η_1	10^{-3},10	10^{-3},10
ρ_0, ρ_1, ρ_2	0, 0.1, 0.7	0.1, 0.9, –
β_{sh}, β_{gr}	2, 2	2, 2
γ_{fail}		5
μ		0.8
k_*		6
ε	0	0
ε_r	0	0

The convergence process for both algorithms is presented in Figure 24. The convergence tolerances $\varepsilon = \varepsilon_r = 0$ are used, and both algorithms are terminated after 15 iterations. At each iteration, the algorithms must solve subproblems of roughly equal sizes, as documented in Table 4. Differences in the sizes arise from the slightly different subproblem formulations of each algorithm. Among the primary contributors are how artificial infeasibility and unboundedness are treated as well as differences in the cost penalty terms. Notably, GuSTO has no one-norm cones because it does not have a dynamics virtual control term like SCvx [compare (54b) and (81b)].

Despite their differences, Figure 24 shows that GuSTO and SCvx are equally fast. Notably, temporal discretization takes much longer than actually solving the SCP subproblem. This is a simple consequence of the fact that discretization is done by unoptimized Julia code, whereas the solution is computed using the optimized C code of the ECOS solver. The timing results are thus not representative of the runtime distribution between the problem formulation and solution. Instead, the roughly equal per-iteration times show that the subproblem difficulty stays constant across iterations. Furthermore, this timing is a conservative upper bound for an optimized SCP implementation [63]. For example, SCP for similar quadrotor trajectory generation tasks has been demonstrated to execute in as few as 40 ms on embedded platforms [52].

The trajectory solutions for SCvx and GuSTO are given in Figures 25 and 26. The fact that these practically identical trajectories were produced by two different algorithms can be spotted only from the different convergence histories on the left side of Figure 25. These histories show how the SCP algorithms morph an infeasible initial guess into a feasible and locally optimal trajectory. Recall that this is a free final time problem, and both algorithms are able to increase the initial guess (128) until the maximum allowed flight time $t_{f,max}$. This is optimal since a quadrotor minimizing the average specific control power per unit time (113a) will opt for a slow trajectory with the lowest acceleration.

Finally, note that temporal discretization results in some clipping of the obstacle keep-out zones in Figure 25. This is the direct result of imposing constraints only at the discrete-time nodes. Various strategies exist to mitigate the

clipping effect, such as increasing the radius of the keep-out zones, increasing the number of discretization points, imposing a sufficiently low maximum velocity constraint, and numerically minimizing a function related to the state transition matrix [192], [193].

Sequential Convex Programming: Six-Degrees-of-Freedom Free Flyer

Having demonstrated the use of LCvx on a relatively simple quadrotor trajectory, a substantially more challenging example is now presented that involves nonlinear 6-DoF dynamics and a more complex set of obstacle avoidance constraints. The objective is to compute a trajectory for a 6-DoF free-flying robotic vehicle that must navigate through an environment akin to the International Space Station (ISS). Free flyers are robots that provide assistance to human operators in microgravity environments [230], [231]. As shown in Figure 27, the Astrobee, Japan Aerospace Exploration Agency Internal Ball Camera, and European Space Agency Crew Interactive Mobile Companion are

TABLE 4 **A breakdown of the subproblem size for the quadrotor obstacle avoidance example.**

	SCvx	GuSTO
Variables	640	542
Affine equalities	186	186
Affine inequalities	240	360
One-norm cones	29	0
Infinity-norm cones	61	31
Second-order cones	30	30

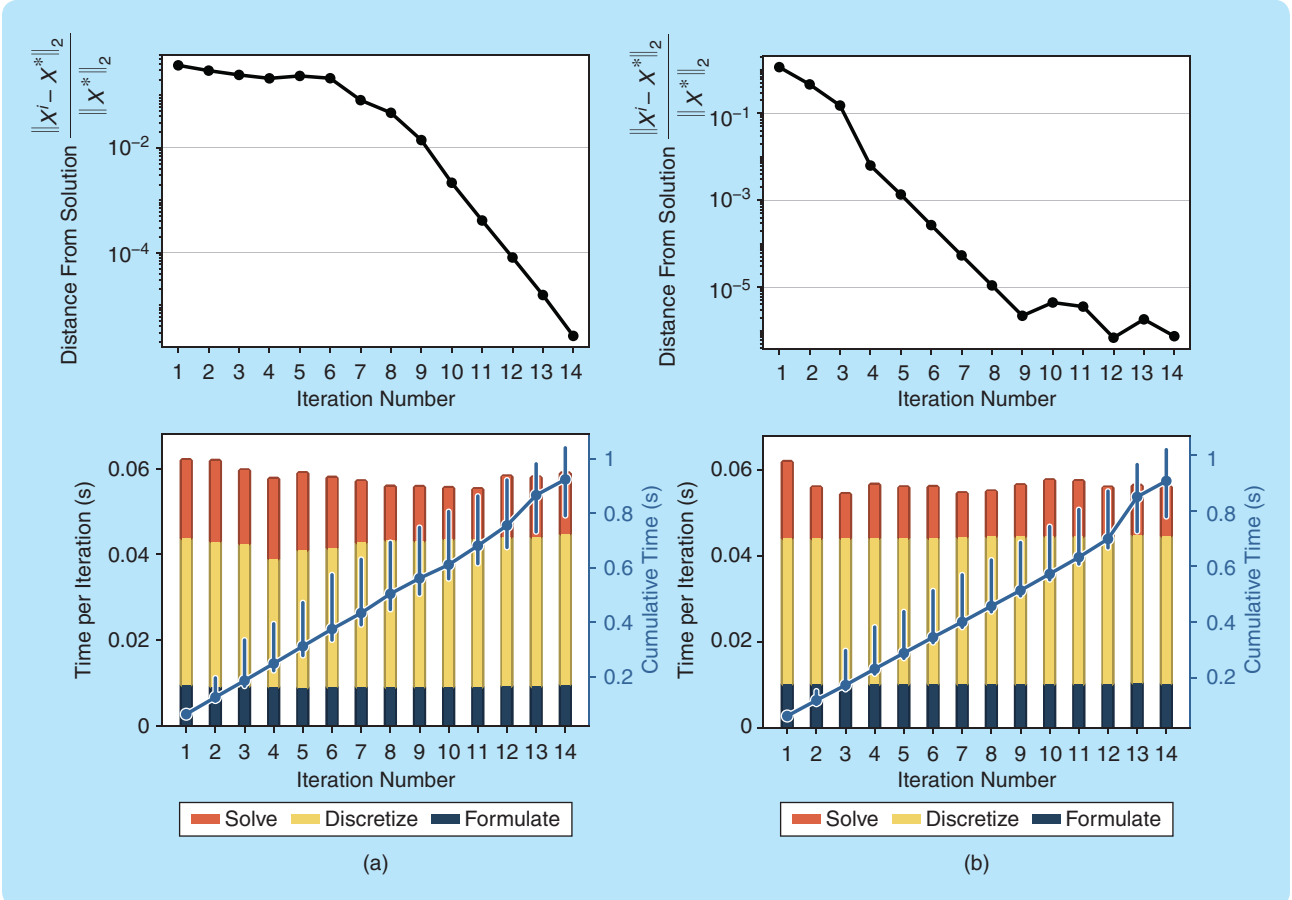


FIGURE 24 The convergence and runtime performance for the quadrotor obstacle avoidance problem. Both algorithms take a similar amount of time and number of iterations to converge to a given tolerance. The top row shows the distance of the concatenated solution vector at the ith iteration, X^i, from its final converged value, X^*. The runtime subplots in the bottom row show statistics on algorithm performance over 50 executions. The solution times per subproblem are roughly equal, which shows that the subproblem difficulty stays constant over the iterations. "Formulate" measures the time taken to parse the subproblem into the input format of the convex optimizer, "Discretize" measures the time taken to temporally discretize the linearized dynamics (44b), and "Solve" measures the time taken by the core convex numerical optimizer. Each bar shows the median time. The trace across the diagonal shows the cumulative time obtained by summing the runtimes of all preceding iterations. Its markers are placed at the median time, and the error bars show the 10% (bottom) and 90% (top) quantiles. Because small runtime differences accumulate over iterations, the error bars grow with the iteration count. (a) SCvx. (b) GuSTO.

three recent successful deployments of such robots. Their goals include filming the ISS, assisting with maintenance tasks, and looking after the astronauts' mental health [232]–[234]. The particulars of this SCP example are taken primarily from [50].

The quadrotor in the previous section was modeled as a point mass whose attitude was approximated by the direction of the acceleration vector. The free flyer, on the other hand, is a more complex vehicle that must generally perform coupled translational and rotational motion by using an assembly of multiple thrusters. Maneuvers may require pointing a camera at a fixed target and emulating nonholonomic behavior for the sake of predictability and operator comfort [235], [236]. This calls for whole-body motion planning, so the free flyer is modeled as a full 6-DoF rigid body with both translational and rotational dynamics.

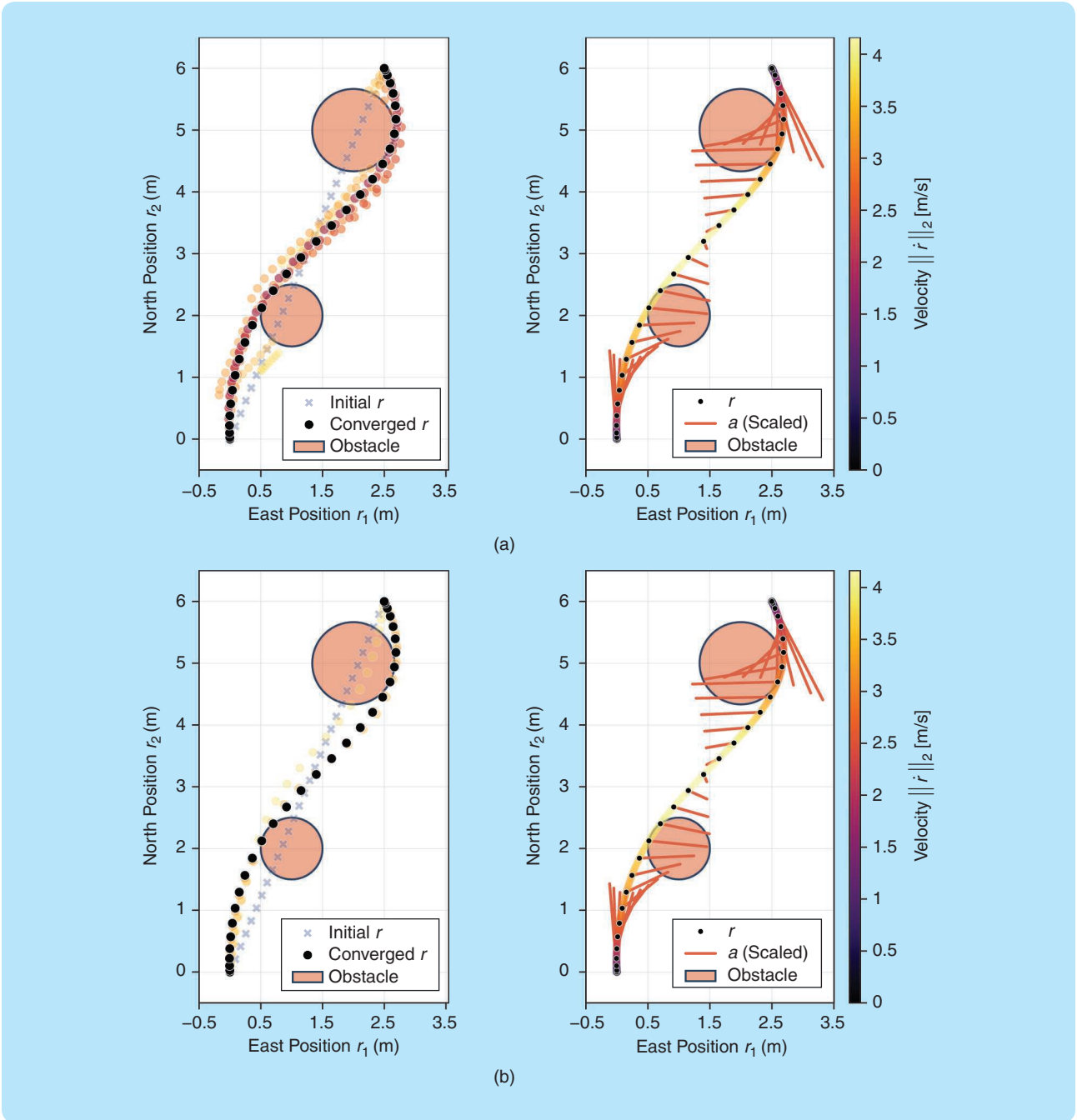


FIGURE 25 The position trajectory evolution (left) and final converged trajectory (right) for the quadrotor obstacle avoidance problem. The continuous-time trajectory in the right plots is obtained by numerically integrating the dynamics (130). The fact that this trajectory passes through the discrete-time subproblem solution confirms dynamic feasibility. The red lines in the right plots show the acceleration vector, as seen from above. (a) SCvx. (b) GuSTO.

To describe the equations of motion, introduce two reference frames. First, let $\mathcal{F}_\mathcal{I}$ be an inertial reference frame with a conveniently positioned, but otherwise arbitrary, origin. Second, let $\mathcal{F}_\mathcal{B}$ be a rotating reference frame affixed to the robot's center of mass, whose unit vectors are aligned with the robot's principal axes of inertia. Here, $\mathcal{F}_\mathcal{I}$ is the inertial frame, and $\mathcal{F}_\mathcal{B}$ is the body frame. Correspondingly, vectors expressed in $\mathcal{F}_\mathcal{I}$ are inertial vectors and carry an $\mathcal{I}$ subscript (for example, $x_\mathcal{I}$), while those expressed in $\mathcal{F}_\mathcal{B}$ are body vectors and carry a $\mathcal{B}$ subscript (for example, $x_\mathcal{B}$). For the purpose of trajectory generation, the orientation of $\mathcal{F}_\mathcal{B}$ with respect to $\mathcal{F}_\mathcal{I}$ is encoded using a vector representation of a unit quaternion, $q_{\mathcal{B}\leftarrow\mathcal{I}} \in \mathbb{R}^4$.

The convention is to represent the translational dynamics in $\mathcal{F}_\mathcal{I}$ and the attitude dynamics in $\mathcal{F}_\mathcal{B}$. This yields the following Newton–Euler equations that govern the free flyer's motion [59]:

$$\dot{r}_\mathcal{I}(t) = v_\mathcal{I}(t), \tag{130a}$$

$$\dot{v}_\mathcal{I}(t) = m^{-1} T_\mathcal{I}(t), \tag{130b}$$

$$\dot{q}_{\mathcal{B}\leftarrow\mathcal{I}}(t) = \frac{1}{2} q_{\mathcal{B}\leftarrow\mathcal{I}}(t) \otimes \omega_\mathcal{B}(t), \tag{130c}$$

$$\dot{\omega}_\mathcal{B}(t) = J^{-1}\left(M_\mathcal{B}(t) - \omega_\mathcal{B}(t)^\times J \omega_\mathcal{B}(t)\right). \tag{130d}$$

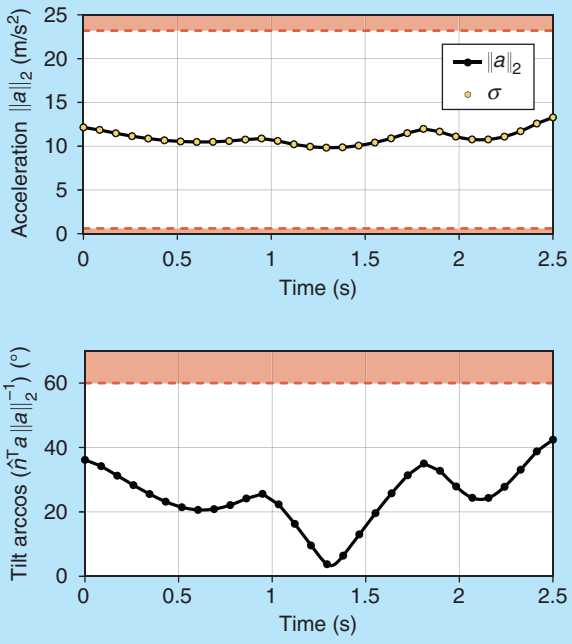


FIGURE 26 The acceleration norm and tilt angle time histories for the converged trajectory of the quadrotor obstacle avoidance problem. These are visually identical for SCvx and GuSTO, so a single plot is shown. The continuous-time acceleration norm is obtained from the first-order hold assumption (S28), while the continuous-time tilt angle is obtained by integrating the solution through the nonlinear dynamics. Similar to Figure 22, the acceleration time history plot confirms that lossless convexification holds [that is, constraint (114b) holds with equality].

The state variables in the preceding equations are the inertial position $r_\mathcal{I} \in \mathbb{R}^3$, inertial velocity $v_\mathcal{I} \in \mathbb{R}^3$, aforementioned unit quaternion attitude $q_{\mathcal{B}\leftarrow\mathcal{I}} \in \mathbb{R}^4$, and body angular velocity $\omega_\mathcal{B} \in \mathbb{R}^3$. The latter variable represents the rate at which $\mathcal{F}_\mathcal{B}$ rotates with respect to $\mathcal{F}_\mathcal{I}$. The free flyer's motion is controlled by an inertial thrust vector $T_\mathcal{I} \in \mathbb{R}^3$ and a body torque vector $M_\mathcal{B} \in \mathbb{R}^3$. The physical parameters of the free flyer (the mass $m > 0$ and principal moment of inertia matrix $J \in \mathbb{R}^{3\times3}$) are fixed. The dynamics are written in absolute time t that spans the interval $[0, t_f]$. The final time t_f is optimized and bounded using the previous constraint (109).

The initial and final conditions for each state are specified in this example to be fixed constants:

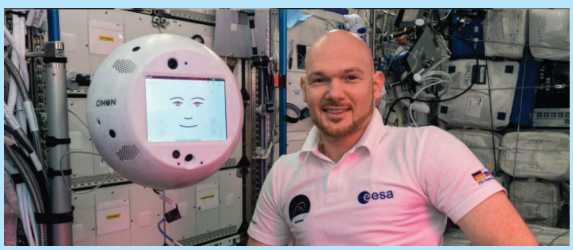
$$r_\mathcal{I}(0) = r_0, \quad r_\mathcal{I}(t_f) = r_f, \tag{131a}$$

$$v_\mathcal{I}(0) = v_0, \quad v_\mathcal{I}(t_f) = v_f, \tag{131b}$$

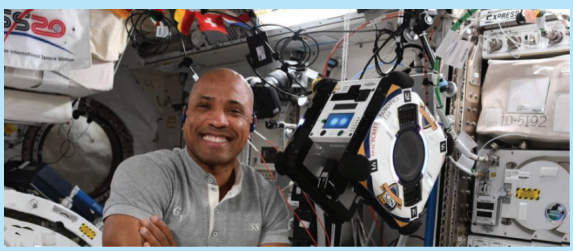
$$q_{\mathcal{B}\leftarrow\mathcal{I}}(0) = q_0, \quad q_{\mathcal{B}\leftarrow\mathcal{I}}(t_f) = q_f, \tag{131c}$$

$$\omega_\mathcal{B}(0) = 0, \quad \omega_\mathcal{B}(t_f) = 0. \tag{131d}$$

(a)



(b)



(c)

FIGURE 27 Three examples of free-flyer robots at the International Space Station. (a) The Japan Aerospace Exploration Agency Internal Ball Camera, with Peggy Whitson. (b) The European Space Agency/German Aerospace Center/Airbus/IBM Crew Interactive Mobile Companion, with Alexander Gerst. (c) The Naval Postgraduate School/NASA Astrobee, with Victor Glover [233], [234], [247]. These robots provide a helping hand aboard the space station. Photos used with permission.

The free-flyer robot implements a 6-DoF holonomic actuation system based on a centrifugal impeller that pressurizes air, which can then be vented from a set of nozzles distributed around the body [237], [238]. Holonomic actuation means that the thrust and torque vectors are independent of the vehicle's attitude [239]. The capability of this system can be modeled by the following control input constraints:

$$\| T_{\mathcal{I}}(t) \|_2 \leq T_{\max}, \quad \| M_{\mathcal{B}}(t) \|_2 \leq M_{\max}, \tag{132}$$

where $T_{\max} > 0$ and $M_{\max} > 0$ are user-defined constants representing the maximum thrust and torque.

This problem involves both convex and nonconvex state constraints. Convex constraints are used to bound the velocity and angular velocity magnitudes to user-defined constants:

$$\| v_{\mathcal{I}}(t) \|_2 \leq v_{\max}, \quad \| \omega_{\mathcal{B}}(t) \|_2 \leq \omega_{\max}. \tag{133}$$

Nonconvex state constraints are used to model the (fictional) ISS flight space and avoid floating obstacles. The latter are modeled exactly as in the quadrotor example, using the constraints in (112). The flight space, on the other hand, is represented by a union of rectangular rooms. This is a difficult nonconvex constraint, and its efficient modeling requires some explanation.

At the conceptual level, the space station flight space is represented by a function $d_{\text{ss}} : \mathbb{R}^3 \to \mathbb{R}$ that maps the inertial position to a scalar number. This is commonly referred to as a *signed distance function* (*SDF*) [240]. Let us denote the set of positions that are within the flight space by $\mathcal{O}_{ss} \subset \mathbb{R}^3$. A valid SDF is given by any function that satisfies the following property:

$$r_{\mathcal{I}} \in \mathcal{O}_{\text{ss}} \;\Leftrightarrow\; d_{\text{ss}}(r_{\mathcal{I}}) \geq 0. \tag{134}$$

If a continuously differentiable d_{ss} can be formulated, then the flight space can be modeled using (134) as the following nonconvex path constraint (38e):

$$d_{\text{ss}}(r_{\mathcal{I}}) \geq 0. \tag{135}$$

Open source libraries, such as Bullet [241], are available to compute the SDF for obstacles of arbitrary shape. This example uses a simpler custom implementation. To begin, the space station is modeled as an assembly of several rooms. This is expressed as a set union,

$$\mathcal{O}_{\text{ss}} \triangleq \bigcup_{i=1}^{n_{\text{ss}}} \mathcal{O}_i, \tag{136}$$

where each room $\mathcal{O}_i$ is taken to be a rectangular box:

$$\mathcal{O}_i \triangleq \{ r_{\mathcal{I}} \in \mathbb{R}^3 : l_i^{\text{ss}} \leq r_{\mathcal{I}} \leq u_i^{\text{ss}} \}. \tag{137}$$

The coordinates l_i^{ss} and u_i^{ss} represent the "rear bottom-left" and the "ahead top-right" corners of the ith room when looking along the positive axis directions. Equivalently, but more advantageously for implementing the SDF, the set (137) can be represented as

$$\mathcal{O}_i \triangleq \left\{ r_{\mathcal{I}} \in \mathbb{R}^3 : \left\| \frac{r_{\mathcal{I}} - c_i^{\text{ss}}}{s_i^{\text{ss}}} \right\|_\infty \leq 1 \right\}, \tag{138}$$

where vector division is entry-wise, and the new terms c_i^{ss} and s_i^{ss} are the room's centroid and diagonal:

$$c_i^{\text{ss}} = \frac{u_i^{\text{ss}} + l_i^{\text{ss}}}{2}, \quad s_i^{\text{ss}} = \frac{u_i^{\text{ss}} - l_i^{\text{ss}}}{2}. \tag{139}$$

To find the SDF for the overall flight space, begin with the simpler task of writing an SDF for a single room. This is straightforward using (138):

$$d_{\text{ss},i}(r_{\mathcal{I}}) \triangleq 1 - \left\| \frac{r_{\mathcal{I}} - c_i^{\text{ss}}}{s_i^{\text{ss}}} \right\|_\infty, \tag{140}$$

which is a concave function that satisfies a property similar to (134):

$$r_{\mathcal{I}} \in \mathcal{O}_i \;\Leftrightarrow\; d_{\text{ss},i}(r_{\mathcal{I}}) \geq 0. \tag{141}$$

Because $d_{\text{ss},i}$ is concave, the constraint on the right side of (141) is convex. This means that constraining the robot to be inside room $\mathcal{O}_i$ is a convex operation, which makes sense since $\mathcal{O}_i$ is a convex set.

As the free flyer traverses the flight space, one can imagine the individual room SDFs to evolve based on the robot's position. When the robot enters the ith room, $d_{\text{ss},i}$ becomes positive and grows to a maximum value of one as the robot approaches the room center. As the robot exits the room, $d_{\text{ss},i}$ becomes negative and decreases in value as the robot flies farther away. To keep the robot inside the space station flight space, the room SDFs must evolve such that there is always at least one nonnegative room SDF. This requirement precisely describes the overall SDF, which can be encoded mathematically as a maximization:

$$d_{\text{ss}}(r_{\mathcal{I}}) \triangleq \max_{i=1,\ldots,n_{\text{ss}}} d_{\text{ss},i}(r_{\mathcal{I}}). \tag{142}$$

This SDF definition satisfies the required property (134). Figure 28(a) provides an example scalar field generated by (142) for a typical space station layout. Visually, when restricted to a plane with a fixed altitude $r_{\mathcal{I},3}$, the individual room SDFs form four-sided "pyramids" above their corresponding room.

The room SDFs $d_{ss,i}$ are concave. However, the maximization in (142) generates a nonconvex function. Two possible strategies to encode (142) are to introduce integer variables or to work with a smooth approximation [46]. The former strategy generates a mixed-integer convex subproblem, which is possible to solve but does not fit the SCP algorithm mold of this article. As mentioned before for (38), the SCP subproblems do not involve integer variables. Thus, a smooth approximation strategy is pursued, which yields

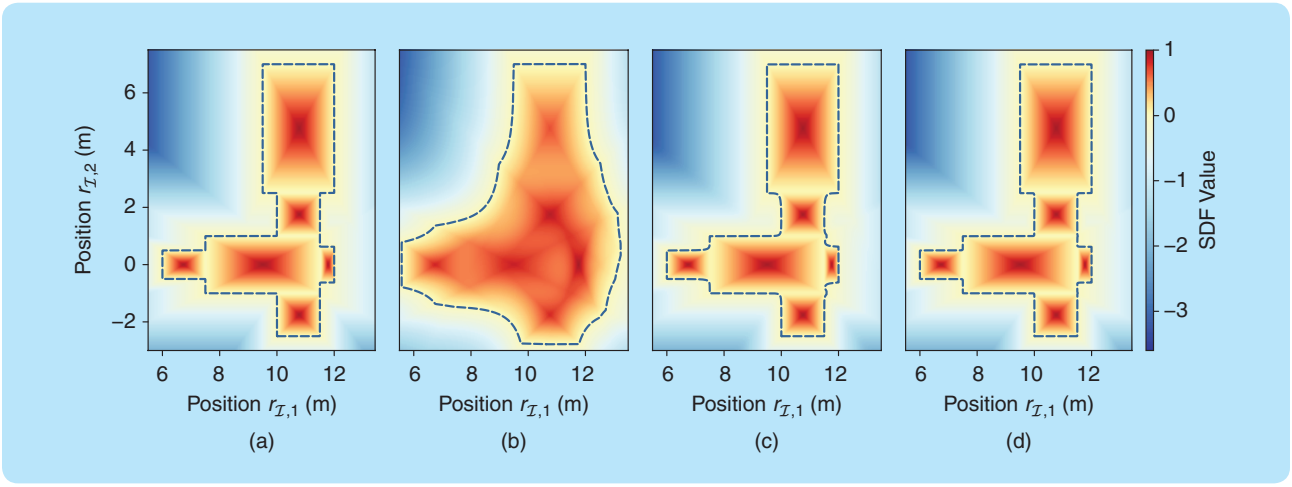


FIGURE 28 A heatmap visualization of the exact signed distance function (SDF) (142) and the approximate SDF (145) for several values of the sharpness parameter σ. Each plot also shows the SDF zero-level set boundary as a dashed line. This boundary encloses the feasible flight space, which corresponds to nonnegative values of the SDF. (a) d_{ss} (exact), (b) $\tilde{d}_{\text{ss}}$ for $\sigma = 1$, (c) $\tilde{d}_{\text{ss}}$ for $\sigma = 10$, and (d) $\tilde{d}_{\text{ss}}$ for $\sigma = 50$.

an arbitrarily accurate approximation of the feasible flight space $\mathcal{O}_{\text{ss}}$ and carries the significant computational benefit of avoiding mixed-integer programming.

The crux of this strategy is to replace the maximization in (142) with the softmax function. Given a general vector $v \in \mathbb{R}^n$, this function is defined by

$$\mathsf{L}_\sigma(v) = \sigma^{-1} \log \sum_{i=1}^{n} e^{\sigma v_i}, \tag{143}$$

where $\sigma > 0$ is a sharpness parameter such that L_σ upper bounds the exact value $\max_i v_i$ with an additive error of at most $\log(n)/\sigma$. To develop intuition, consider the SDF (142) for two adjacent rooms and restricted along the jth axis of the inertial frame. The SDF can then be written as

$$d_{\text{ss}}(r_{\mathcal{I},j}) = \max\left\{1 - \left|\frac{r_{\mathcal{I},j} - c_{1j}^{\text{ss}}}{s_{1j}^{\text{ss}}}\right|, 1 - \left|\frac{r_{\mathcal{I},j} - c_{2j}^{\text{ss}}}{s_{2j}^{\text{ss}}}\right|\right\}. \tag{144}$$

Figure 29 illustrates the relationship between the exact SDF (144) and its approximation, which is obtained by replacing the max operator with L_σ. Note that d_{ss} is indeed highly nonconvex, and the approximation quickly converges to the exact SDF as σ increases. This 1D example can now be generalized, and (142) is replaced with the following approximation:

$$\tilde{d}_{\text{ss}}(r_{\mathcal{I}}) \triangleq \mathsf{L}_\sigma(\delta_{\text{ss}}(r_{\mathcal{I}})), \tag{145a}$$

$$\delta_{\text{ss},i}(r_{\mathcal{I}}) \le d_{\text{ss},i}(r_{\mathcal{I}}), \quad i = 1, \ldots, n_{\text{ss}}, \tag{145b}$$

where the new functions $\delta_{\text{ss},i}$ are so-called slack room SDFs. The model (145) admits several favorable properties. First, (145a) is smooth in the new slack SDFs and can be included directly in (38e) as the following nonconvex path constraint:

$$\tilde{d}_{\text{ss}}(r_{\mathcal{I}}) \ge 0. \tag{146}$$

Second, the constraints in (145b) are convex and can be included directly in (38c). Overall, the approximate SDF (145a) satisfies the following property:

$$r_{\mathcal{I}} \in \tilde{\mathcal{O}}_{\text{ss}} \;\Leftrightarrow\; \exists \delta_{\text{ss}}(r_{\mathcal{I}}) \text{ such that (146) holds}, \tag{147}$$

where $\tilde{\mathcal{O}}_{\text{ss}} \subset \mathbb{R}^3$ is an approximation of $\mathcal{O}_{\text{ss}}$ that becomes arbitrarily more accurate as the sharpness parameter σ increases. The geometry of this convergence process is illustrated in Figure 28(b)–(d) for a typical space station layout. Crucially, the fact that L_σ is an upper bound of the maximum means that the approximate SDF $\tilde{d}_{\text{ss}}$ is nonnegative at the interfaces of adjacent rooms. In other words, the passage between adjacent rooms cannot be artificially blocked by the approximation.

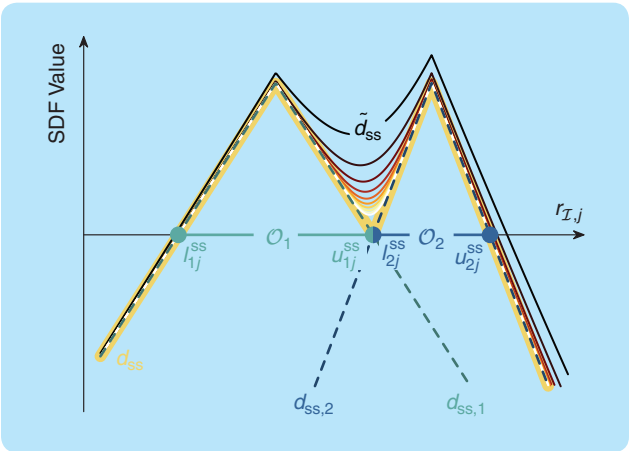


FIGURE 29 The correspondence between the exact signed distance function (SDF) (142) and its approximation $\tilde{d}_{\text{ss}}$ using the softmax function (143). As the sharpness parameter σ increases, the approximation quickly converges to the exact SDF. This figure illustrates a sweep for $\sigma \in [1, 5]$, where lower values are associated with darker colors.

Summarizing the preceding discussion, in a similar way to (113), the goal is to solve the following minimum control power, free final time optimal control problem:

$$\min_{T_{\mathcal{I}}, M_{\mathcal{B}}, t_f, \delta_{\text{ss}}} t_f^{-1} \int_0^{t_f} \| T_{\mathcal{I}}(\text{t}) \|_2^2 + \| M_{\mathcal{B}}(\text{t}) \|_2^2 \text{dt}, \tag{148a}$$

$$\text{s.t. (109), (112), (130)}-\text{(133), and (146)}. \tag{148b}$$

SCvx Formulation

Like for the quadrotor example, begin by demonstrating how to cast (148) into the standard template of (38). While this process is mostly similar to that of the quadrotor, the particularities of the flight space constraint (146) will reveal a salient feature of efficient modeling for SCP. Once the modeling step is done and an initial trajectory guess is defined, the solution process is completely automated by the general SCvx algorithm description in the "Sequential Convex Programming" section. To keep the notation light, the argument of time is omitted where possible.

Examining the dynamics (130), define the following state and control vectors:

$$x = (r_{\mathcal{I}}, v_{\mathcal{I}}, q_{\mathcal{B}\leftarrow\mathcal{I}}, \omega_{\mathcal{B}}) \in \mathbb{R}^{13}, \tag{149a}$$

$$u = (T_{\mathcal{I}}, M_{\mathcal{B}}) \in \mathbb{R}^{6}. \tag{149b}$$

Next, define the parameter vector to serve two purposes. First, as for the quadrotor, define a time dilation α_{t} such that (116) holds, yielding $\text{t} = \alpha_{\text{t}} t$. Second, take advantage of the fact that (38c) is mixed in the state and parameter in order to place the slack room SDFs in (145b) into the parameter vector. In particular, according to (54c), the constraint (145b) is imposed only at the discrete-time grid nodes. Thus, for a grid of N nodes, there are only Nn_{ss} instances of (145b) to be included. The following vector can therefore be defined:

$$\Delta_{\text{ss}} \triangleq (\delta_{\text{ss}}^1, \ldots, \delta_{\text{ss}}^N) \in \mathbb{R}^{Nn_{\text{ss}}}, \tag{150}$$

where $\delta_{\text{ss}}^k \equiv \delta_{\text{ss}}(r_{\mathcal{I}}(t_k))$ and $\Delta_{\text{ss},i+(k-1)n_{\text{ss}}}$ denotes the slack SDF value for the ith room at time t_k. To keep the notation concise, use the shorthand $\Delta_{\text{ss},ik} \equiv \Delta_{\text{ss},i+(k-1)n_{\text{ss}}}$. The overall parameter vector is then given by

$$p = \begin{bmatrix} \alpha_{\text{t}} \\ \Delta_{\text{ss}} \end{bmatrix} \in \mathbb{R}^{1+Nn_{\text{ss}}}. \tag{151}$$

In absolute time, the dynamics (38b) are given directly by (130). As for the quadrotor, this forms a set of time-invariant, first-order ODEs:

$$f(x,u) = \begin{bmatrix} v_{\mathcal{I}} \\ m^{-1} T_{\mathcal{I}} \\ \frac{1}{2} q_{\mathcal{B}\leftarrow\mathcal{I}} \otimes \omega_{\mathcal{B}} \\ J^{-1}(M_{\mathcal{B}} - \omega_{\mathcal{B}}^{\times} J \omega_{\mathcal{B}}) \end{bmatrix}. \tag{152}$$

The boundary conditions (38f) and (38g) are obtained from (131):

$$g_{\text{ic}}(x(0), p) = \begin{bmatrix} r_{\mathcal{I}}(0) - r_0 \\ v_{\mathcal{I}}(0) - v_0 \\ q_{\mathcal{B}\leftarrow\mathcal{I}}(0) - q_0 \\ \omega_{\mathcal{B}}(0) \end{bmatrix}, \tag{153a}$$

$$g_{\text{tc}}(x(1), p) = \begin{bmatrix} r_{\mathcal{I}}(1) - r_f \\ v_{\mathcal{I}}(1) - v_f \\ q_{\mathcal{B}\leftarrow\mathcal{I}}(1) - q_f \\ \omega_{\mathcal{B}}(1) \end{bmatrix}. \tag{153b}$$

The dynamics are converted to normalized time in the same way as (119):

$$f(x, u, p) = \alpha_{\text{t}} f(x, u). \tag{154}$$

The convex state and input path constraints (38c) and (38d) are straightforward. The quadrotor example leverages the mixed state–parameter nature of (38c) to include all the convex state and parameter constraints. In particular, these are (109), (133), and (145b). Using the definition of time dilation, (109) is translated into the constraint

$$t_{f,\min} \le \alpha_{\text{t}} \le t_{f,\max}. \tag{155}$$

Using the definition of the concatenated slack SDF vector (150), one can transform (145b) into the following constraints:

$$\Delta_{\text{ss},ik} \le d_{\text{ss},i}(r_{\mathcal{I}}(t_k)), \quad i = 1, \ldots, n_{\text{ss}}, \quad k = 1, \ldots, N. \tag{156}$$

Consequently, the convex path constraint set $\mathcal{X}$ in (38c) is given by

$$\mathcal{X} = \{(x, p) \in \mathbb{R}^{13} \times \mathbb{R}^{1+Nn_{\text{ss}}} : \text{(133), (155), and (156) hold}\}. \tag{157}$$

The convex input constraint set $\mathcal{U}$ in (38d) is given by all the input vectors that satisfy (132).

The nonconvex path constraints (38e) for the free-flyer problem involve the ellipsoidal floating obstacles (112) and approximate flight space constraint (146). The floating obstacle constraints are modeled exactly like for the quadrotor by using (121). The flight space constraint leverages the concatenated slack SDF vector (150) and eventual temporal discretization of the problem to impose (146) at each temporal grid node as

$$\mathsf{L}_{\sigma}(\delta_{\text{ss}}^k) \ge 0, \quad k = 1, \ldots, N. \tag{158}$$

Hence, the nonconvex path constraint function in (38e) can be written as $s : \mathbb{R}^3 \times \mathbb{R}^{Nn_{\text{ss}}} \to \mathbb{R}^{n_{\text{obs}}+1}$. The first n_{obs} components are given by (121), and the last component is given by the negative left-hand side of (158). It remains to define the running cost of the Bolza cost function (39). As for the quadrotor, the integrand in (148a) is scaled to be mindful of

The gradient acts like a torchlight that illuminates the local surroundings in a dark room and allows one to take a step closer to a light switch.

the penalty terms that will be added by the SCvx algorithm. This yields the following convex running cost definition:

$$\Gamma(x, u, p) = \left(\frac{\|T_{\mathcal{I}}\|_2}{T_{\max}}\right)^2 + \left(\frac{\|M_{\mathcal{B}}\|_2}{M_{\max}}\right)^2. \tag{159}$$

At this point, it may seem as though formulating (38) for SCvx is complete. However, the seemingly innocuous flight space constraint (158) actually hides an important difficulty that will now be addressed. The importance of the following discussion cannot be overstated, as it can mean the difference between successful trajectory generation and convergence to an infeasible trajectory (that is, one with nonzero virtual control). In the case of the 6-DoF free flyer, omission of the following discussion incurs a 40% optimality penalty for the value of (148a).

The investigation of (158) begins by writing down its Jacobians, which SCvx will use for linearizing the constraint. Note that (158) is a function of only $\delta_{\text{ss}}^k \in \mathbb{R}^{n_{\text{ss}}}$, which is part of the concatenated slack SDF (150) and thus resides in the parameter vector. Hence, only the Jacobian (43g) is nonzero. Using the general softmax definition (143), the ith element of $\nabla \mathsf{L}_\sigma(\delta_{\text{ss}}^k)$ is given by

$$\frac{\partial \mathsf{L}_\sigma(\delta_{\text{ss}}^k)}{\partial \delta_{\text{ss},i}^k} = \left(\sum_{j=1}^{n_{\text{ss}}} e^{\sigma \delta_{\text{ss},j}^k}\right)^{-1} e^{\sigma \delta_{\text{ss},i}^k}. \tag{160}$$

When the slack SDF satisfies the lower-bound (156) with equality, the Jacobian (160) is an accurate representation of how the overall SDF (145a) changes due to small perturbations in the robot's position. The problematic case occurs when this bound is loose. To illustrate the idea, suppose that the robot is located near the center of $\mathcal{O}_i$ such that $d_{\text{ss},i}(r_{\mathcal{I}}(t_k)) = 0.8$. Assume that the rooms do not overlap and that the slack SDF values of the other rooms satisfy $\delta_{\text{ss},j}^k = d_{\text{ss},j}(r_{\mathcal{I}}(t_k)) = -1$ for all $j \neq i$. Since the robot is uniquely inside $\mathcal{O}_i$, the exact SDF (142) is locally a linear function of $d_{\text{ss},i}$ and has a gradient $\partial d_{\text{ss}} / \partial d_{\text{ss},i} = 1$. Because the SCP subproblem should be an accurate local approximation of the nonconvex problem, the same behavior is expected for the approximate SDF (145a) for high values of σ. However, this may not be the case.

Figure 30 illustrates what happens to the approximate SDF gradient (160) as the slackness in (156) increases. First, note that when there is no slackness, increasing the σ parameter does indeed make the approximate gradient converge to the exact value of one. However, as slackness grows, there is a distinct cutoff value, below which (160) becomes zero. This is known as a *vanishing gradient* problem and was studied extensively for machine learning [242]. The core issue is that SCP relies heavily on gradient information to determine how to improve the feasibility and optimality of the subproblem solution. As an analogy, the gradient acts like a torchlight that illuminates the local surroundings in a dark room and allows one to take a step closer to a light switch. When the gradient vanishes, so does the torchlight, and SCP no longer has information about which direction is best to take. Unless the solution is already locally optimal, a vanished gradient most often either blocks SCP from finding more optimal solutions or forces it to use nonzero virtual control. The result is that the converged trajectory is either (heavily) suboptimal or even infeasible.

Looking at Figure 30, one may ask why SCP does not simply increase $\delta_{\text{ss},i}^k$ above the vanishing threshold to recover gradient information. Remember, however, it is the gradient that indicates that increasing $\delta_{\text{ss},i}^k$ is a good approach in the first place. The situation is much like focusing one's eyes on the flat region of the $\sigma = 50$ curve on the very left in Figure 30. If one saw only the part of the curve for $\delta_{\text{ss},i}^k / d_{\text{ss},i}(r_{\mathcal{I}}(t_k)) \in [-3, -2]$, then one would also not know whether it was best to increase or decrease $\delta_{\text{ss},i}^k$.

Fortunately, the remedy is quite simple. Because (134) is a necessary and sufficient condition, it is known that slackness in (156) cannot be used to make the trajectory more

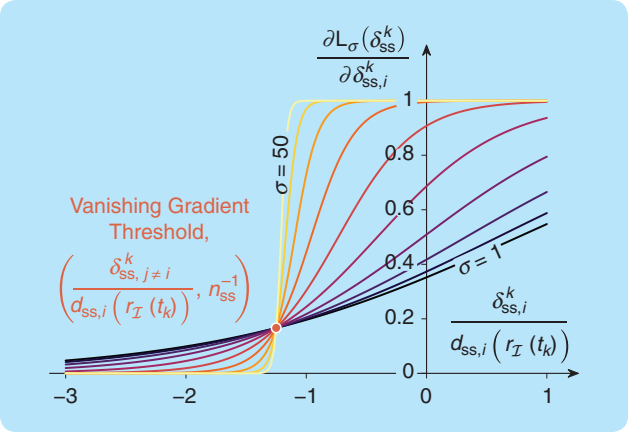


FIGURE 30 The effect of slackness in the signed distance function (SDF) lower-bound constraint (145b) on the gradient of the approximate SDF (145a). This plot is obtained by setting $n_{\text{ss}} = 6$ and $\delta_{\text{ss},j} = -1$ for all $j \neq i$. The individual curves are obtained by gradually reducing $\delta_{\text{ss},i}$ from its maximum value of $d_{\text{ss},i}$. As the sharpness parameter σ increases, a "cutoff" value appears, below which the approximate SDF becomes insensitive to changes in $\delta_{\text{ss},i}$.

TABLE 5 The algorithm parameters for the six-degrees-of-freedom free-flyer example.

Problem Parameters

t_f,min	60	s	Minimum final time
t_f,max	200	s	Maximum final time
m	7.2	kg	Free-flyer mass
J	$0.1083 \cdot I_3$	kg m^2	Free-flyer inertia matrix
$T_{\max}$	20	mN	Maximum two-norm of thrust
$M_{\max}$	100	μN·m	Maximum two-norm of torque
$v_{\max}$	0.4	ms^{-1}	Maximum free-flyer speed
$\omega_{\max}$	1	°s^{-1}	Maximum free-flyer angular velocity
c_1	(8.5, −0.15, 5)	m	Center of obstacle 1
c_2	(11.2, 1.84, 5)	m	Center of obstacle 2
c_3	(11.3, 3.8, 4.8)	m	Center of obstacle 3
H_1, H_2, H_3	$3.33 \cdot I_3$	m^{-1}	Shape of obstacles 1, 2, and 3
l_i^{ss}	See code	m	Room rear bottom-left corner
u_i^{ss}	See code	m	Room ahead top-right corner
r_0	(6.5, −0.2, 5)	m	Initial position vector
v_0	(0.035, 0.035, 0)	ms^{-1}	Initial velocity vector
q_0	−40(0, 1, 1)		Initial attitude quaternion
r_f	(11.3, 6, 4.5)	m	Final position vector
v_f	(0, 0, 0)	ms^{-1}	Final velocity vector
q_f	0(0, 0, 1)		Final attitude quaternion
σ	50		Softmax sharpness parameter
ε_{ss}	10^{-4}		Slack room signed distance function weight

Algorithm Parameters

	SCvx	GuSTO
N	50	30
q	∞	∞
$\hat{q}$	∞	∞
P	(48)	(48)
h_λ		(71)
λ	10^3	
$\lambda_0, \lambda_{\max}$		10^4, 10^9
η	1	1
η_0, η_1	10^{-3},10	10^{-3},10
ρ_0, ρ_1, ρ_2	0, 0.1, 0.7	0.1, 0.5, –
$\beta_{\text{sh}}, \beta_{\text{gr}}$	2, 2	2, 2
γ_{fail}		5
μ		0.8
k_*		6
ε	0	0
ε_r	0	0

optimal. In other words, a trajectory with nonzero slackness will not achieve a lower cost (148a). Hence, one needs a way to incentivize the convex subproblem optimizer to make (156) hold with equality. The simplest approach is to introduce a terminal cost that maximizes the concatenated slack SDF:

$$\phi(\Delta_{\text{ss}}) = -\varepsilon_{\text{ss}} \sum_{k=1}^{N} \sum_{i=1}^{n_{\text{ss}}} \Delta_{\text{ss},ik}, \tag{161}$$

where $\varepsilon_{\text{ss}} \in \mathbb{R}_{++}$ is any user-chosen positive number. To make sure that (161) does not interfere with the extra penalty terms introduced by SCvx, set ε_{ss} to a very small but numerically tolerable value, as shown in Table 5.

In summary, the investigation into (158) enabled the identification of a vanishing gradient issue. This resulted in a simple yet effective remedy in the form of a terminal cost (161). This discussion highlights three salient features of good modeling for SCP-based trajectory generation. First, SCP does not have equal performance for mathematically equivalent problem formulations [such as the free-flyer problem with and without (161)]. Second, favorable gradient behavior is instrumental for good performance. Third, remedies to recover good performance for difficult problems are often surprisingly simple.

GuSTO Formulation

As for the quadrotor example, the GuSTO formulation is very similar to SCvx. Equation (159) can be expressed as the quadratic running cost (67):

$$S(p) = \text{diag}(T_{\max}^{-2} I_3, M_{\max}^{-2} I_3), \tag{162a}$$

$$\ell(x,p) = 0, \tag{162b}$$

$$g(x,p) = 0. \tag{162c}$$

The dynamics (152) are also cast into the control affine form (68):

$$f_0(x,p) = \begin{bmatrix} v_{\mathcal{I}} \\ 0 \\ \frac{1}{2} q_{\mathcal{B}\leftarrow\mathcal{I}} \otimes \omega_{\mathcal{B}} \\ -J^{-1}(\omega_{\mathcal{B}}^{\times} J \omega_{\mathcal{B}}) \end{bmatrix}, \tag{163a}$$

$$f_i(x,p) = \begin{bmatrix} 0 \\ m^{-1} e_i \\ 0 \\ 0 \end{bmatrix}, \quad i = 1,2,3, \tag{163b}$$

$$f_i(x,p) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ J^{-1} e_i \end{bmatrix}, \quad i = 4,5,6, \tag{163c}$$

where $e_i \in \mathbb{R}^3$ is the ith standard basis vector. Like for the quadrotor example, the rest of the optimization model is exactly the same as for SCvx in the last section.

Initial Trajectory Guess

The initial trajectory guess is based on some simple intuition about what a feasible free-flyer trajectory might look like. Although this guess is more complicated than the straight-line initialization used for the quadrotor, it is based on purely kinematic considerations. This makes it quick to compute. However, it also means that the guess does not satisfy the dynamics and obstacle constraints. The fact that SCP readily morphs this coarse guess into a feasible and locally optimal trajectory corroborates the effectiveness of SCP methods for high-dimensional, nonconvex trajectory generation tasks.

To begin, the time dilation α_t is obtained by averaging the final time bounds as in (128). An "L-shape" path is then used for the position trajectory guess. In particular, recall that according to (131a), the free flyer has to go from r_0 to r_f. Define a constant velocity trajectory that closes the gap between r_0 and r_f along the first axis (then the second and, finally, the third) in a total time of α_t seconds. The trajectory thus consists of three straight legs with sharp 90° turns at the transition points, which is akin to the Manhattan or taxicab geometry of the one-norm [243]. The velocity is readily derived from the position trajectory and is a constant-norm vector whose direction changes twice to align with the appropriate axis in each trajectory leg. Furthermore, initialize the concatenated slack SDF parameter vector (150) by evaluating (140) along the position trajectory guess for each room and discrete-time grid node.

The attitude trajectory guess is slightly more involved, however, it can be obtained by a general procedure that is recommended for attitude trajectories. According to (131c), the free flyer must rotate between the attitudes encoded by q_0 and q_f. Since quaternions are not additive and must maintain a unit norm to represent rotation, straight-line interpolation from q_0 to q_f is not an option. Instead, the spherical linear interpolation (SLERP) procedure is used [189], [201]. This operation performs a continuous rotation from q_0 to q_f at a constant angular velocity around a fixed axis. To define the operation, introduce the logarithmic and exponential maps for unit quaternions:

$$\mathrm{Log}(q) \triangleq \alpha u, \quad \text{where } \alpha \in \mathbb{R},\ u \in \mathbb{R}^3, \tag{164a}$$

$$\mathrm{Exp}(\alpha u) \triangleq \begin{bmatrix} u \sin(\alpha/2) \\ \cos(\alpha/2) \end{bmatrix}. \tag{164b}$$

The exponential map converts a unit quaternion to its equivalent angle–axis representation. The logarithmic map converts an angle–axis rotation back to a quaternion, which is written here in the vectorized form used to implement $q_{\mathcal{B}\leftarrow\mathcal{I}}$ in (130c). SLERP for the attitude quaternion $q_{\mathcal{B}\leftarrow\mathcal{I}}$ can then be defined by leveraging (164):

$$q_e = q_0^* \otimes q_f, \tag{165a}$$

$$q_{\mathcal{B}\leftarrow\mathcal{I}}(t) = q_0 \otimes \mathrm{Exp}(t\,\mathrm{Log}(q_e)), \tag{165b}$$

where q_e is the error quaternion between q_f and q_0 and $t \in [0,1]$ is an interpolation parameter such that $q_{\mathcal{B}\leftarrow\mathcal{I}}(0) = q_0$ and $q_{\mathcal{B}\leftarrow\mathcal{I}}(1) = q_f$. The angular velocity trajectory guess is simple to derive since SLERP performs a constant velocity rotation around a fixed axis. Using (165a),

$$\omega_{\mathcal{B}}(t) = \alpha_t^{-1}\mathrm{Log}(q_e). \tag{166}$$

The free flyer is a very-low-thrust vehicle to begin with. By using (148a), the optimization is, in some sense, searching for the lowest of low thrust trajectories. Hence, it is expected that the control inputs $T_{\mathcal{I}}$ and $M_{\mathcal{B}}$ will be small. Without any further insight, it is hard to guess what the thrust and torque would look like for a 6-DoF vehicle in a microgravity environment. Thus, the initial control guess is simply set to zero.

Numerical Results

There is now a specialized instance of (38) and an initialization strategy for the 6-DoF free-flyer problem. The trajectory solution is generated using SCvx and GuSTO, with temporal discretization performed using the FOH interpolating polynomial method in "Discretizing Continuous-Time Optimal Control Problems." The algorithm parameters are provided in Table 5, where the initial and final quaternion vectors are expressed in degrees using the angle-axis representation of (164a). The ECOS solver is used as the core numerical convex optimizer, and the full implementation is available in the code repository linked in Figure 2.

The convergence processes for SCvx and GuSTO are shown in Figure 31. Again, set $\varepsilon = \varepsilon_r = 0$ to observe the convergence process for exactly 15 iterations. At each iteration, the algorithms solve a convex subproblem whose size is documented in Table 6. Note that the subproblems of both algorithms are substantially larger than for the quadrotor example and represent a formidable increase in dimensionality for the numerical problem. However, modern IPMs easily handle problems of this size, and it will be shown that the increased variable and constraint count is of little concern. Further note that the larger number of variables and affine inequalities for GuSTO is due to how the implementation uses extra slack variables to encode the soft penalty function (69). Because GuSTO does not use a dynamics virtual control, it has no one-norm cones, while SCvx has several such cones to model the virtual control penalty (47). Due to its larger subproblem size and slightly more complicated code for including constraints as soft penalties, the "solve" and "formulate" times

per subproblem are slightly longer for GuSTO in this example. Nevertheless, both algorithms have roughly equal runtimes, and GuSTO has the advantage of converging to a given tolerance in slightly fewer iterations.

The converged trajectories are plotted in Figures 32 and 33. The left subplots in Figures 32(a) and 32(b) show a remarkably similar evolution from the initial guess to the converged trajectory. The final trajectories are visually identical, and both algorithms discover that the maximum allowed flight time of $t_{f,\max}$ minimizes the control power cost (148a), as expected. Finally, Figure 34 plots the evolution of the nonconvex flight space and obstacle avoidance inequalities (146) and (112). The first observation is that the constraints hold at the discrete-time nodes, and the free flyer approaches the ellipsoidal obstacles quite closely. This is similar to how the quadrotor brushes against the obstacles in Figure 25 and is a common feature of time- or energy-optimal trajectories in a cluttered environment.

The second observation concerns the sawtooth-like, nonsmooth nature of the SDF time history. Around $t = 25$ s and $t = 125$ s, the approximate SDF comes close to zero even though the position trajectory in Figure 32 is not near a wall at those times. This is a consequence of the modeling approach since the SDF is near zero at the room interfaces (see Figures 28 and 29) even though these are not physical "walls." However, around $t = 100$ s, the flight space constraint (146) is actually activated as the free flyer rounds a

TABLE 6 **A breakdown of the subproblem size for the six-degrees-of-freedom free-flyer example.**

	SCvx	GuSTO
Variables	2267	3352
Affine equalities	663	663
Affine inequalities	350	1650
One-norm cones	49	0
Infinity-norm cones	401	351
Second-order cones	200	200

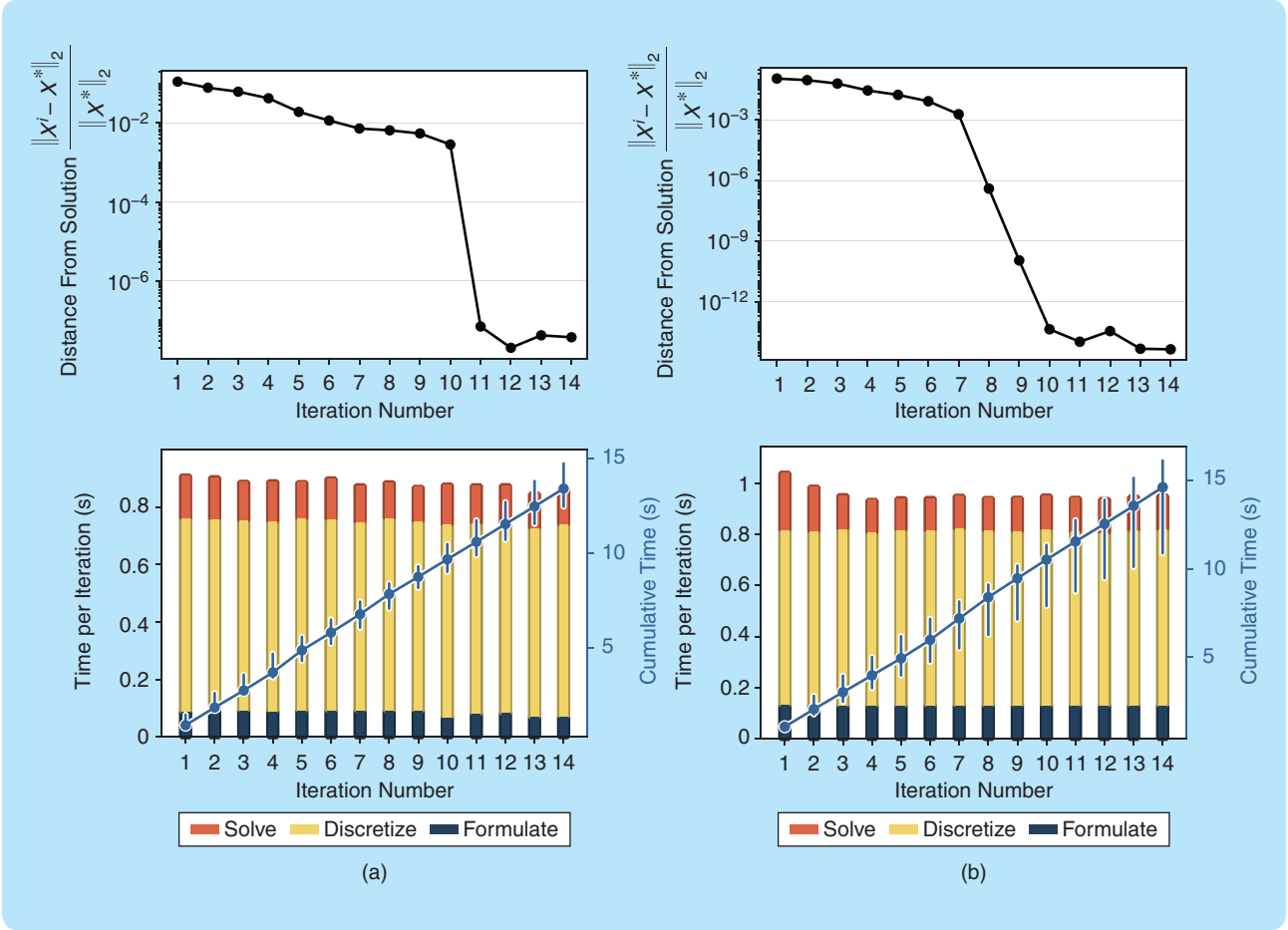


FIGURE 31 The convergence and runtime performance for the six-degrees-of-freedom free-flyer problem. Both algorithms take a similar amount of time to converge. The runtime subplots in the bottom row show statistics on algorithm performance over 50 executions. GuSTO converges slightly faster for this example and, although both algorithms reach numerical precision for practical purposes, GuSTO converges all the way down to a 10^{-14} tolerance. The plots are generated according to the description for Figure 24. (a) SCvx. (b) GuSTO.

corner. Roughly speaking, this is intuitively the optimal thing to do. Like a Formula One driver rounding a corner by following the racing line, the free flyer spends less control effort by following the shortest path. An unfortunate consequence is that this results in minor clipping of the continuous-time flight space constraint. The issue can be mitigated by the same strategies as proposed in the previous section for the quadrotor example [192], [193].

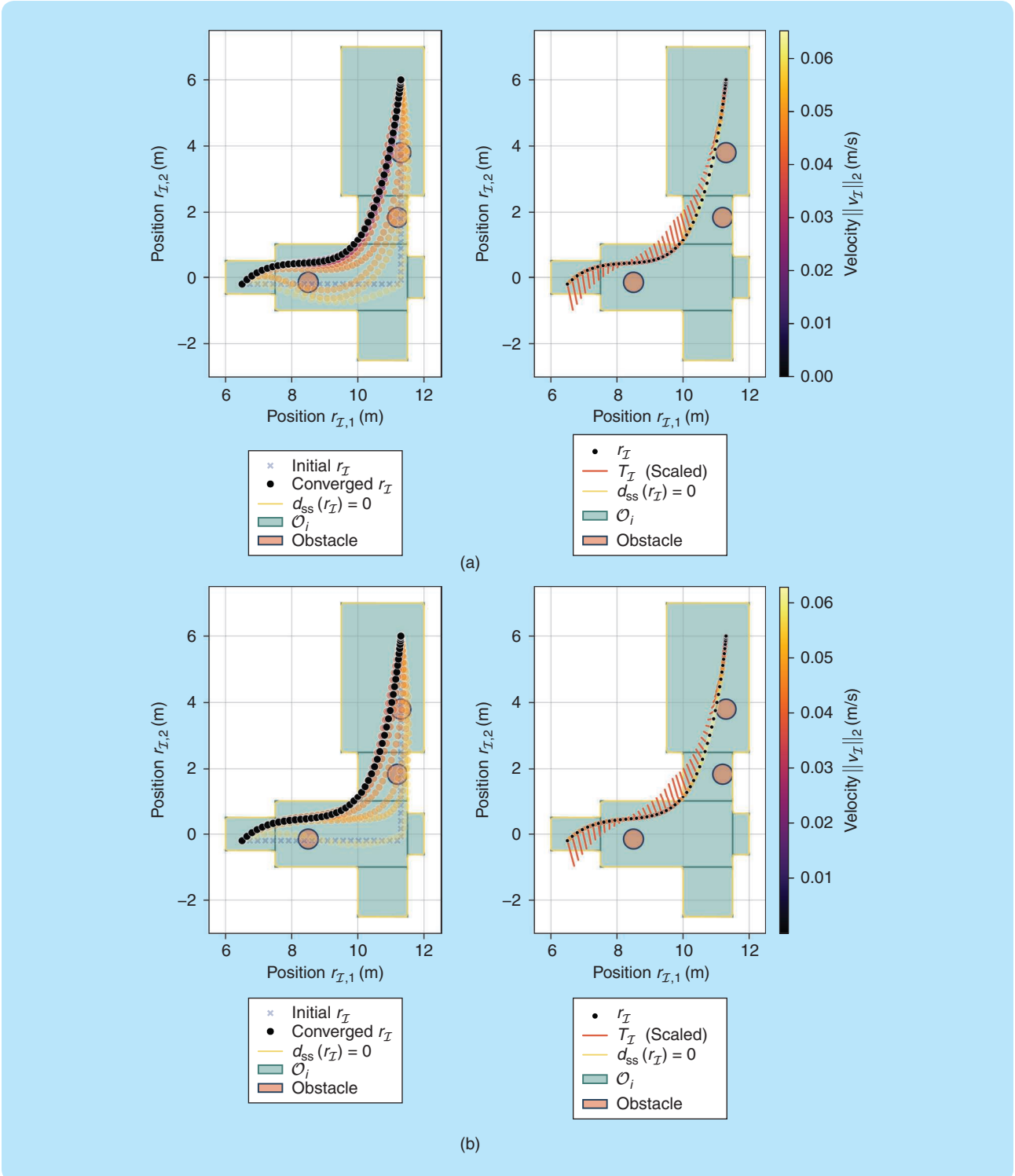


FIGURE 32 The position trajectory evolution (left) and final converged trajectory (right) for the six-degrees-of-freedom free-flyer problem. In the right plots for each algorithm, the continuous-time trajectory is obtained by numerically integrating the dynamics (130). The fact that this trajectory passes through the discrete-time subproblem solution confirms dynamic feasibility. (a) SCvx. (b) GuSTO.

CONCLUSIONS

Modern vehicle engineering is moving in the direction of increased autonomy. This includes aerospace, automotive, and marine transport as well as robots on land, in the air, and in domestic environments. No matter the application, a common feature across autonomous systems is the basic requirement to generate trajectories. In a general sense, trajectories serve like plans to be executed for the system to complete its task. Due to the large scale of deployment and/or the safety-critical nature of the system, reliable real-time onboard trajectory generation has never been more important.

This article took the stance that convex optimization is a prime contender for the job, thanks to more than 40 years of optimization research having produced a remarkable suite of numerical methods for quickly and reliably solving convex problems [26], [29], [34]. Many of these methods are now packaged as either commercial or open source off-the-shelf codes [229], [244]. This makes the injection of convex optimization into an autonomous system easier than ever before, provided that the right high-level algorithms exist to leverage it.

To leverage convex optimization for the difficult task of nonconvex trajectory generation, this article provided an expansive tutorial on three algorithms. First, the LCvx algorithm was introduced to remove acute nonconvexities in the control input constraints. This provides a method supported by optimal control theory to transform certain families of nonconvex trajectory generation tasks into ones that can be solved in one shot by a convex optimizer. A variable-mass rocket landing example at the end of the article illustrated a real-world application of the LCvx method.

Not stopping there, the article then motivated an entire family of optimization methods: sequential convex programming. These methods use a linearize–solve loop, whereby a convex optimizer is called several times until a locally optimal trajectory is obtained. SCP strikes a compelling middle

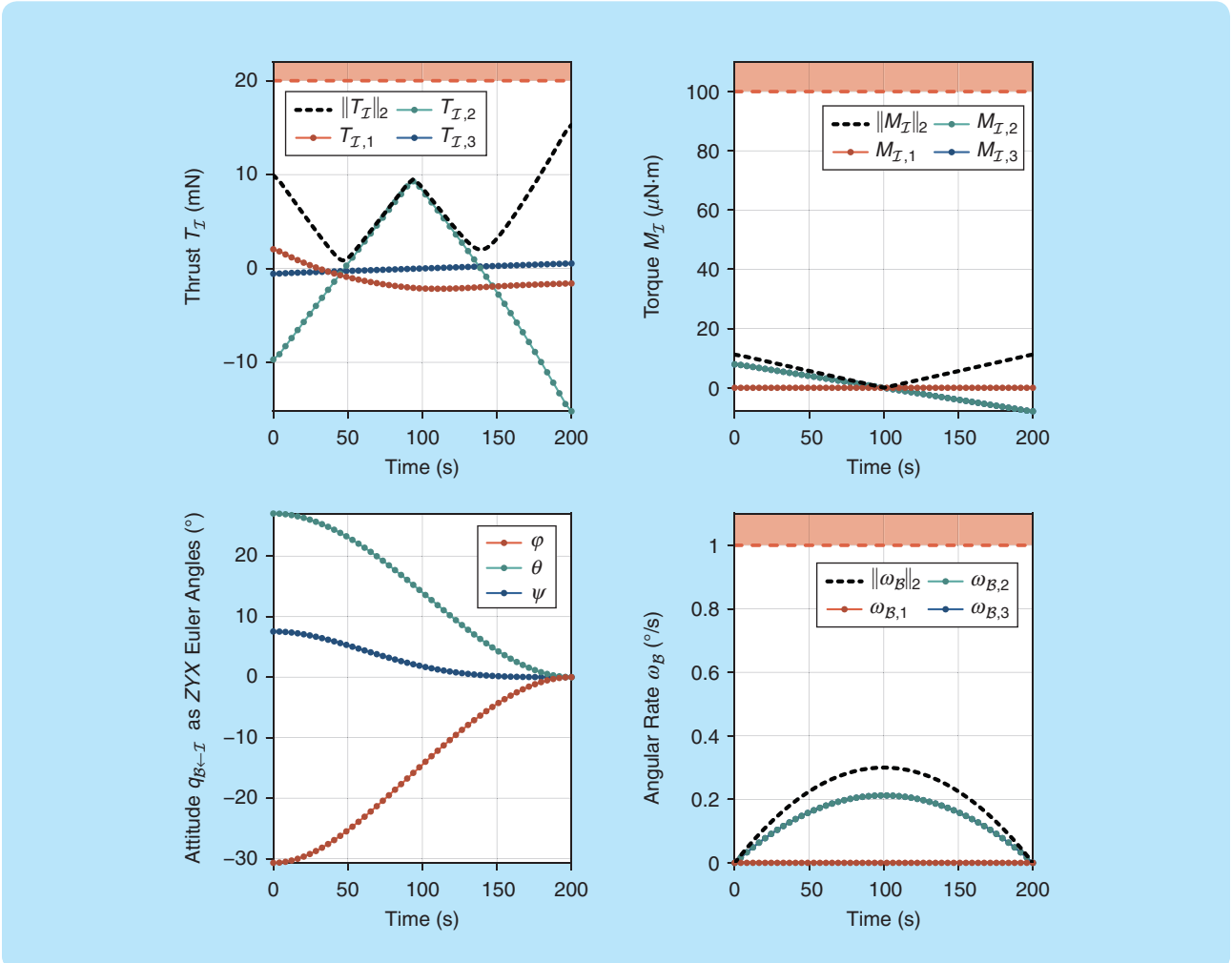


FIGURE 33 The state and control time histories for the converged trajectory of the six-degrees-of-freedom free-flyer problem. These are visually identical for SCvx and GuSTO, so only a single plot is shown. Euler angles using the intrinsic Tait–Bryan convention are shown in place of the quaternion attitude. As in Figure 26, the dots represent the discrete-time solution, while the continuous lines are obtained by propagating the solution through the actual continuous-time dynamics (38b).

No method has yet attained the full potential of its capabilities, and this presents the reader with an exciting opportunity to contribute to the research and development effort.

ground between "what is possible" and "what is acceptable" for safety-critical, real-time trajectory generation. In particular, SCP inherits much from trust region methods in numerical optimization, and its performance is amenable to theoretical analysis using standard tools of analysis, constrained optimization, and optimal control theory [11], [29], [47]. This article provided a detailed overview of two specific and closely related SCP algorithms: SCvx and GuSTO [49], [50]. To corroborate their effectiveness for difficult trajectory generation tasks, two numerical examples were presented based on a quadrotor and a space station free-flyer maintenance robot.

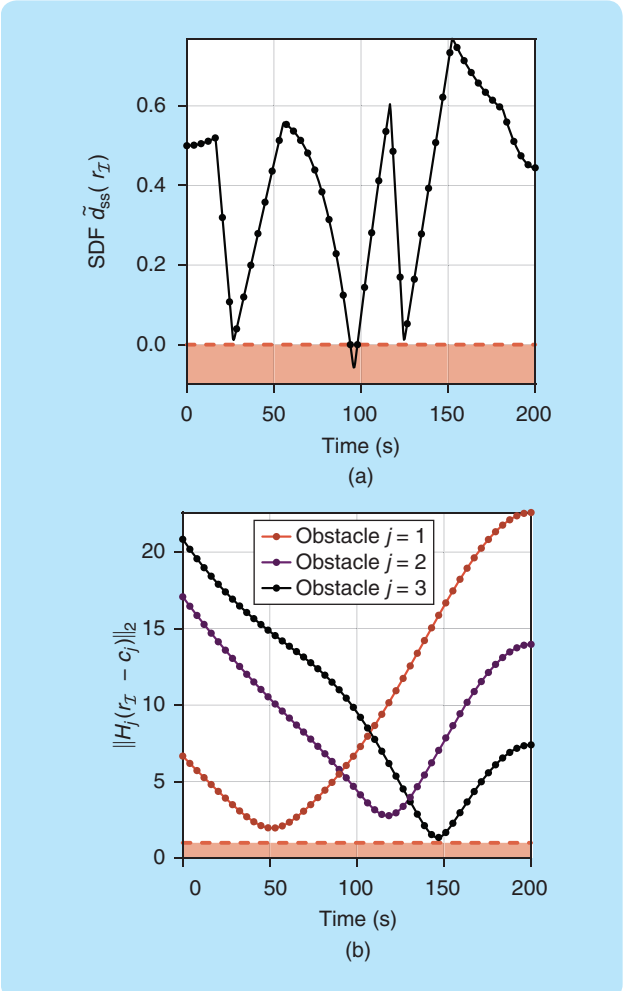


FIGURE 34 The (a) signed distance function (SDF) and (b) obstacle avoidance time histories for the converged trajectory of the six-degrees-of-freedom free-flyer problem. As for Figure 33, these are visually identical for SCvx and GuSTO, so only a single plot is shown. Note the highly nonlinear nature of the SDF, whose time history exhibits sharp corners as the robot traverses the feasible flight space. Although the SDF constraint (158) is satisfied at the discrete-time nodes, minor intersample constraint clipping occurs around 100 s as the robot rounds a turn in the middle of its trajectory (see Figure 32).

The theory behind LCvx, SCvx, and GuSTO is relatively new and under active research, with the oldest method in this article (that is, classical LCvx [85]) being just 17 years old. No method has yet attained the full potential of its capabilities, and this presents the reader with an exciting opportunity to contribute to the research and development effort. It is clear that convex optimization has a role to play in the present and future of advanced trajectory generation. With the help of this article and the associated source code, the reader now has the knowledge and tools to join the adventure.

ACKNOWLEDGMENTS

The University of Washington authors were supported, in part, by the National Science Foundation (NSF) Cyber-Physical Systems (CPS) program, under award 1931744; NSF CAREER grant CNS-1619729; Office of Naval Research (ONR) grant N00014-17-1-2433; Air Force Office of Scientific Research grant FA9550-20-1-0053; and NASA cooperative agreement NNX17AH02A. The Stanford University authors were supported, in part, by NSF CPS program award 1931815, ONR Young Investigator Program contract N00014-17-1-2433, and King Abdulaziz City for Science and Technology. The authors would like to extend their gratitude to the following collaborators, in particular: Jonathan P. How, for his encouragement to write this article; Matthew W. Harris, for his expert commentary on the nuances of LCvx; Yuanqi Mao, for his invaluable inputs on SCP algorithms; and Abhinav G. Kamath, for his meticulous review of the full article. The authors are also indebted to the control and optimization research communities at large, whose efforts have resulted in the theory and software that is instrumental to our work.

AUTHOR INFORMATION

Danylo Malyuta (danylo@malyuta.name) received the B.Sc. degree in mechanical engineering from the Swiss Federal Institute of Technology Lausanne (EPFL); M.Sc. degree in robotics, systems, and control from ETH Zürich;

and Ph.D. degree in aerospace engineering from the University of Washington. His doctoral research was carried out at the Autonomous Controls Lab, Department of Aeronautics and Astronautics, primarily focusing on computationally efficient optimization-based control of dynamical systems. He has interned at the NASA Jet Propulsion Laboratory, the NASA Johnson Space Center, and Amazon Prime Air. His present research lies at the intersection of software engineering, computer science, and mathematics, with the goal of creating fast, reliable, and scalable decision-making systems. He currently works at SpaceX in Redmond, Washington, 98053, USA on the Starlink satellite Internet constellation.

Taylor P. Reynolds received the B.Sc. degree in mathematics and engineering from Queen's University in 2016. He received the Ph.D. degree from the Department of Aeronautics and Astronautics, University of Washington, in 2020, under the supervision of Mehran Mesbahi. During his Ph.D., he worked with the NASA Johnson Space Center and Draper Laboratories to develop advanced guidance algorithms for planetary landing in the Safe and Precise Landing—Integrated Capability Evolution project, and he cofounded the Aeronautics and Astronautics CubeSat Team at the University of Washington. He now works as a research scientist at Amazon Prime Air in Seattle, Washington, USA.

Michael Szmuk received the B.S. and M.S. degrees in aerospace engineering from the University of Texas at Austin. In 2019, he received the Ph.D. degree while working in the Autonomous Controls Lab, Department of Aeronautics and Astronautics, University of Washington, under the supervision of Behçet Açıkmeşe. During his academic career, he completed internships at NASA, the Air Force Research Laboratory, Emergent Space, Blue Origin, and Amazon Prime Air. He now works as a research scientist at Amazon Prime Air, in Seattle, Washington, USA, specializing in the design of flight control algorithms for autonomous air delivery vehicles.

Thomas Lew is a Ph.D. candidate in aeronautics and astronautics at Stanford University, Stanford, California, 94305, USA. He received the B.Sc. degree in microengineering from École Polytechnique Fédérale de Lausanne in 2017 and the M.Sc. degree in robotics from ETH Zürich in 2019. His research focuses on the intersection between optimal control and machine learning techniques for robotics and aerospace applications.

Riccardo Bonalli obtained the M.Sc. degree in mathematical engineering from Politecnico di Milano in 2014 and the Ph.D. degree in applied mathematics from Sorbonne Université in 2018, in collaboration with the French National Office for Aerospace Studies and Research (ONERA). He is the recipient of the 2018 ONERA Department of Information Processing and Systems Best Ph.D. Student Award. He was a postdoctoral researcher in the Department of Aeronautics and Astronautics, Stanford University. Currently, he is a tenured CNRS researcher with the Laboratory of Signals and Systems (L2S), Université Paris-Saclay, National Center for Scientific Research (CNRS), CentraleSupélec, France. His research interests include theoretical and numerical robust optimal control with techniques from differential geometry, statistical analysis, and machine learning, and applications in aerospace systems and robotics.

Marco Pavone is an associate professor of aeronautics and astronautics at Stanford University, Stanford, California, 94305, USA, where he is the director of the Autonomous Systems Laboratory. Before joining Stanford, he was a research technologist within the Robotics Section at the NASA Jet Propulsion Laboratory. He received the Ph.D. degree in aeronautics and astronautics from the Massachusetts Institute of Technology in 2010. His main research interests are in the development of methodologies for the analysis, design, and control of autonomous systems, with an emphasis on self-driving cars, autonomous aerospace vehicles, and future mobility systems. He is a recipient of a number of awards, including a Presidential Early Career Award for Scientists and Engineers, an Office of Naval Research Young Investigator Program Award, a National Science Foundation CAREER Award, and a NASA Early Career Faculty Award. He was identified by the American Society for Engineering Education as one of America's 20 most highly promising investigators under the age of 40. He is an associate editor of *IEEE Control Systems Magazine*.

Behçet Açıkmeşe is a professor at the University of Washington, Seattle, Washington, 98195, USA. He received the Ph.D. degree in aerospace engineering from Purdue University. He was a senior technologist at the NASA Jet Propulsion Laboratory (JPL) and a lecturer at the California Institute of Technology. At the JPL, he developed control algorithms for planetary landing, spacecraft formation flying, and asteroid and comet sample return missions. He developed the "flyaway" control algorithms used successfully in NASA's Mars Science Laboratory and Mars 2020 missions during the landings of the *Curiosity* and *Perseverance* rovers. He is a recipient of the National Science Foundation CAREER Award, the IEEE Award for Technical Excellence in Aerospace Control, and numerous NASA Achievement Awards for his contributions to NASA missions and technology development. His research interests include optimization-based control, nonlinear and robust control, and stochastic control. He is a Fellow of IEEE.

REFERENCES

[1] R. D'Andrea, "Guest editorial can drones deliver?" *IEEE Trans. Autom. Sci. Eng. (from July 2004)*, vol. 11, no. 3, pp. 647–648, Jul. 2014, doi: 10.1109/tase.2014.2326952.

[2] A. San Martin, E. Lee, and S. W. Wong, "The development of the MSL guidance, navigation, and control system for entry, descent, and landing," in *Proc. 23rd Space Flight Mech. Meeting*, 2013, pp. 1–20.

[3] A. D. Steltzner, A. M. S. Martin, T. P. Rivellini, A. Chen, and D. Kipp, "Mars science laboratory entry, descent, and landing system development challenges," *J. Spacecraft Rockets*, vol. 51, no. 4, pp. 994–1003, Jul. 2014, doi: 10.2514/1.a32866.

[4] D. W. Way *et al*., "Mars science laboratory: Entry, descent, and landing system performance," in *Proc. 2007 IEEE Aerosp. Conf*., pp. 1–19, doi: 10.1109/aero.2007.352821.
[5] G. Stein, "Respect the unstable," *IEEE Control Syst. Mag. (through 2019)*, vol. 23, no. 4, pp. 12–25, Aug. 2003, doi: 10.1109/mcs.2003.1213600.
[6] D. Malyuta, Y. Yu, P. Elango, and B. Açikmeşe, "Advances in trajectory optimization for space vehicle control," *Annu. Rev. Contr*., vol. 52, pp. 282–315, Dec. 2021, doi: 10.1016/j.arcontrol.2021.04.013.
[7] D. A. Mindell, *Digital Apollo*. Cambridge, MA, USA: MIT Press, 2008.
[8] J. M. Carson III *et al*., "The SPLICE project: Continuing NASA development of GN&C technologies for safe and precise landing," in *Proc. Amer. Inst. Aeronaut. Astronaut. Scitech Forum*, Jan. 2019, p. 1, doi: 10.2514/6.2019-0660.
[9] D. Dueri, B. Açikmeşe, D. P. Scharf, and M. W. Harris, "Customized real-time interior-point methods for onboard powered-descent guidance," *J. Guid., Contr., Dyn*., vol. 40, no. 2, pp. 197–212, Feb. 2017, doi: 10.2514/1.g001480.
[10] "The Mars 2020 rover's 'Brains,'" National Aeronautics and Space Administration, Washington, DC, USA, 2020. [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/brains/
[11] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*. Montreux, Switzerland: Gordon & Breach, 1986.
[12] L. D. Berkovitz, *Optimal Control Theory*. New York: Springer-Verlag, 1974.
[13] J. T. Betts, *Practical Methods for Optimal Control and Estimation using Nonlinear Programming*. Philadelphia, PA, USA: SIAM, 2010.
[14] D. Lawden, *Optimal Trajectories for Space Navigation*. London, U.K.: Butterworth, 1963.
[15] J.-P. Marec, *Optimal Space Trajectories*. Amsterdam, The Netherlands: Elsevier, 1979.
[16] A. E. Bryson Jr. and Y.-C. Ho, *Applied Optimal Control*. Washington, DC, USA: Hemisphere, 1975.
[17] D. E. Kirk, *Optimal Control Theory: An Introduction*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1970.
[18] J. M. Longuski, J. J. Guzmán, and J. E. Prussing, *Optimal Control with Aerospace Applications*. New York: Springer-Verlag, 2014.
[19] J. Meditch, "On the problem of optimal thrust programming for a lunar soft landing," *IEEE Trans. Autom. Control*, vol. 9, no. 4, pp. 477–484, Oct. 1964, doi: 10.1109/tac.1964.1105758.
[20] M. Morari and J. H. Lee, "Model predictive control: Past, present and future," *Comput. Chem. Eng*., vol. 23, nos. 4–5, pp. 667–682, 1999, doi: 10.1016/S0098-1354(98)00301-9.
[21] J. W. Eaton and J. B. Rawlings, "Model-predictive control of chemical processes," *Chem. Eng. Sci*., vol. 47, no. 4, pp. 705–720, 1992, doi: 10.1016/0009-2509(92)80263-C.
[22] P. J. Campo and M. Morari, "Robust model predictive control," in *Proc. 1987 Amer. Contr. Conf*., pp. 1021–1026, doi: 10.23919/ACC.1987.4789462.
[23] F. Oldewurtel *et al*., "Use of model predictive control and weather forecasts for energy efficient building climate control," *Energy Buildings*, vol. 45, pp. 15–27, Feb. 2012, doi: 10.1016/j.enbuild.2011.09.022.
[24] Y. Ma, F. Borrelli, B. Hencey, B. Coffey, S. Bengea, and P. Haves, "Model predictive control for the operation of building cooling systems," *IEEE Trans. Control Syst. Technol*., vol. 20, no. 3, pp. 796–803, 2011, doi: 10.1109/TCST.2011.2124461.
[25] S. C. Bengea, A. D. Kelman, F. Borrelli, R. Taylor, and S. Narayanan, "Implementation of model predictive control for an HVAC system in a mid-size commercial building," *HVAC&R Res*., vol. 20, no. 1, pp. 121–135, 2014, doi: 10.1080/10789669.2013.834781.
[26] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[27] J. T. Betts, "Survey of numerical methods for trajectory optimization," *J. Guid., Contr., Dyn*., vol. 21, no. 2, pp. 193–207, 1998, doi: 10.2514/2.4231.
[28] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Rev*., vol. 59, no. 4, pp. 849–904, Jan. 2017, doi: 10.1137/16m1062569.
[29] J. Nocedal and S. Wright, *Numerical Optimization*. New York: Springer-Verlag, 1999.
[30] R. T. Rockafellar, "Lagrange multipliers and optimality," *SIAM Rev*., vol. 35, no. 2, pp. 183–238, Jun. 1993, doi: 10.1137/1035044.
[31] S. J. Wright, *Primal-Dual Interior-Point Methods*. Philadelphia, PA, USA: SIAM, Jan. 1997.
[32] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. Philadelphia, PA, USA: SIAM, Jan. 2019.
[33] C. Roos, T. Terlaky, and J.-P. Vial, *Theory and Algorithms for Linear Optimization: An Interior Point Approach*. Hoboken, NJ, USA: Wiley, Oct. 2000.
[34] M. H. Wright, "The interior-point revolution in optimization: History, recent developments, and lasting consequences," *Bull. Amer. Math. Soc*., vol. 42, no. 1, pp. 39–56, 2005, doi: 10.1090/S0273-0979-04-01040-7.
[35] J. Peng, C. Roos, and T. Terlaky, "Primal-dual interior-point methods for second-order conic optimization based on self-regular proximities," *SIAM J. Optim*., vol. 13, no. 1, pp. 179–203, Jan. 2002, doi: 10.1137/s1052623401383236.
[36] L. Blackmore, "Autonomous precision landing of space rockets," *Bridge*, vol. 4, no. 46, pp. 15–20, Dec. 2016.
[37] D. P. Scharf, B. Açikmeşe, D. Dueri, J. Benito, and J. Casoliva, "Implementation and experimental demonstration of onboard powered-descent guidance," *J. Guid., Contr., Dyn*., vol. 40, no. 2, pp. 213–229, Feb. 2017, doi: 10.2514/1.g000399.
[38] JPL and Masten Space Systems. *500 Meter Divert Xombie Test Flight for G-FOLD, Guidance for Fuel Optimal Large Divert*. (Aug. 2012). [Online Video]. Available: http://www.youtube.com/watch?v=1GRwimo1AwY
[39] JPL and Masten Space Systems. *750 Meter Divert Xombie Test Flight for G-FOLD, Guidance for Fuel Optimal Large Divert*. (Jul. 2012). [Online Video]. Available: http://www.youtube.com/watch?v=jl6pw2oossU
[40] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh*., vol. 1, no. 1, pp. 33–55, Mar. 2016, doi: 10.1109/tiv.2016.2578706.
[41] M. Buehler, K. Iagnemma, and S. Singh, Eds., *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Berlin, Heidelberg: Springer-Verlag, 2009.
[42] R. Buchanan, L. Wellhausen, M. Bjelonic, T. Bandyopadhyay, N. Kottege, and M. Hutter, "Perceptive whole-body planning for multilegged robots in confined spaces," *J. Field Robot*., vol. 38, no. 1, pp. 68–84, Jun. 2020, doi: 10.1002/rob.21974.
[43] J.-P. Sleiman, F. Farshidian, M. V. Minniti, and M. Hutter, "A unified MPC framework for whole-body dynamic locomotion and manipulation," *IEEE Robot. Automat. Lett*., vol. 6, no. 3, pp. 4688–4695, Jul. 2021, doi: 10.1109/lra.2021.3068908.
[44] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," in *Proc. 2nd Conf. Robot Learn*., A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds. Oct. 2018, vol. 87, pp. 133–145.
[45] B. Açikmeşe and S. R. Ploen, "Convex programming approach to powered descent guidance for Mars landing," *J. Guid., Contr., Dyn*., vol. 30, no. 5, pp. 1353–1366, Sep. 2007, doi: 10.2514/1.27553.
[46] L. Blackmore, B. Açikmeşe, and J. M. Carson III, "Lossless convexification of control constraints for a class of nonlinear optimal control problems," *Syst. Contr. Lett*., vol. 61, no. 8, pp. 863–870, Aug. 2012, doi: 10.1016/j.sysconle.2012.04.010.
[47] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust Region Methods*. Philadelphia, PA, USA: SIAM, 2000.
[48] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. Cambridge, MA, USA: MIT Press, 2019.
[49] Y. Mao, M. Szmuk, and B. Acikmese, "Successive convexification: A superlinearly convergent algorithm for non-convex optimal control problems," 2018, *arXiv:1804.06539*.
[50] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming," in *Proc. IEEE Int. Conf. Robot. Automat*., 2019, pp. 6741–6747, doi: 10.1109/ICRA.2019.8794205.
[51] M. W. Harris and B. Açikmeşe, "Maximum divert for planetary landing using convex optimization," *J. Optim. Theory Appl*., vol. 162, no. 3, pp. 975–995, Dec. 2013, doi: 10.1007/s10957-013-0501-7.
[52] M. Szmuk, C. A. Pascucci, D. Dueri, and B. Açikmeşe, "Convexification and real-time on-board optimization for agile quad-rotor maneuvering and obstacle avoidance," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst*., Vancouver, CA, USA, 2017, pp. 4862–4868, doi: 10.1109/IROS.2017.8206363.
[53] X. Liu, Z. Shen, and P. Lu, "Entry trajectory optimization by second-order cone programming," *J. Guid., Contr., Dyn*., vol. 39, no. 2, pp. 227–241, Feb. 2016, doi: 10.2514/1.g001210.
[54] Z. Wang and M. J. Grant, "Constrained trajectory optimization for planetary entry via sequential convex programming," *J. Guid., Contr., Dyn*., vol. 40, no. 10, pp. 2603–2615, Oct. 2017, doi: 10.2514/1.g002150.
[55] X. Liu, "Fuel-optimal rocket landing with aerodynamic controls," *J. Guid., Contr., Dyn*., vol. 42, no. 1, pp. 65–77, Jan. 2019, doi: 10.2514/1.g003537.
[56] M. W. Harris and B. Açikmeşe, "Minimum time rendezvous of multiple spacecraft using differential drag," *J. Guid., Contr., Dyn*., vol. 37, no. 2, pp. 365–373, Mar. 2014, doi: 10.2514/1.61505.

[57] D. Malyuta and B. Açikmeşe, "Lossless convexification of non-convex optimal control problems with disjoint semi-continuous inputs," Nov. 2019, *arXiv:1902.02726*.
[58] M. Szmuk, B. Acikmese, and A. W. Berning, "Successive convexification for fuel-optimal powered landing with aerodynamic drag and non-convex constraints," in *Proc. Amer. Inst. Aeronaut. Astronaut. Guid., Navig., Contr. Conf.*, Jan. 2016, pp. 1–16, doi: 10.2514/6.2016-0378.
[59] M. Szmuk, T. P. Reynolds, and B. Açikmeşe, "Successive convexification for real-time 6-dof powered descent guidance with state-triggered constraints," *J. Guid., Contr., Dyn.*, vol. 48, no. 8, pp. 1399–1413, 2020, doi: 10.2514/1.G004549.
[60] M. Szmuk and B. Açikmeşe, "Successive convexification for 6-DoF Mars rocket powered landing with free-final-time," in *Proc. Amer. Inst. Aeronaut. Astronaut. Guid., Navig., Contr. Conf.*, Jan. 2018, pp. 1–14, doi: 10.2514/6.2018-0617.
[61] M. Szmuk, T. Reynolds, B. Açikmeşe, M. Mesbahi, and J. M. Carson III, "Successive convexification for 6-DoF powered descent guidance with compound state-triggered constraints," in *Proc. Amer. Inst. Aeronaut. Astronaut. Scitech Forum*, Jan. 2019, pp. 1–16, doi: 10.2514/6.2019-0926.
[62] T. P. Reynolds, M. Szmuk, D. Malyuta, M. Mesbahi, B. Açikmeşe, and J. M. Carson, "Dual quaternion-based powered descent guidance with state-triggered constraints," *J. Guid., Contr., Dyn.*, vol. 43, no. 9, pp. 1584–1599, Sep. 2020, doi: 10.2514/1.g004536.
[63] T. P. Reynolds, D. Malyuta, M. Mesbahi, B. Açikmeşe, and J. M. Carson III, "A real-time algorithm for non-convex powered descent guidance," in *Proc. Amer. Inst. Aeronaut. Astronaut. Scitech Forum*, 2020, pp. 1–24, doi: 10.2514/6.2020-0844.
[64] Y. Mao, D. Dueri, M. Szmuk, and B. Açikmeşe, "Successive convexification of non-convex optimal control problems with state constraints," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4063–4069, 2017, doi: 10.1016/j.ifacol.2017.08.789.
[65] M. Szmuk, D. Malyuta, T. P. Reynolds, M. S. Mceowen, and B. Açikmeşe, "Real-time quad-rotor path planning using convex optimization and compound state-triggered constraints," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 7666–7673, doi: 10.1109/iros40897.2019.8967706.
[66] D. Malyuta, T. Reynolds, M. Szmuk, B. Açikmeşe, and M. Mesbahi, "Fast trajectory optimization via successive convexification for spacecraft rendezvous with integer constraints," in *Proc. Amer. Inst. Aeronaut. Astronaut. Scitech Forum*, Jan. 2020, pp. 1–24, doi: 10.2514/6.2020-0616.
[67] T. P. Reynolds *et al.*, "SOC-i: A CubeSat demonstration of optimization-based real-time constrained attitude control," in *Proc. IEEE Aerosp. Conf.*, Mar. 2021, pp. 1–18, doi: 10.1109/AERO50100.2021.9438540.
[68] "NASA tipping point partnership with Blue Origin to test precision lunar landing technologies," National Aeronautics and Space Administration, Washington, DC, USA, Sep. 2020. [Online]. Available: https://www.nasa.gov/directorates/spacetech/NASA_Tipping_Point_Partnership_to_Test_Precision_Lunar_Landing_Tech/
[69] R. Bonalli, A. Bylard, A. Cauligi, T. Lew, and M. Pavone, "Trajectory optimization on manifolds: A theoretically-guaranteed embedded sequential convex programming approach," in *Proc. Robot., Sci. Syst.*, 2019, pp. 1–10, doi: 10.15607/rss.2019.xv.078.
[70] S. Banerjee *et al.*, "Learning-based warm-starting for fast sequential convex programming and trajectory optimization," in *Proc. IEEE Aerosp. Conf.*, Big Sky, MT, USA, Mar. 2020, pp. 1–8, doi: 10.1109/AERO47225.2020.9172293.
[71] T. Lew, R. Bonalli, and M. Pavone, "Chance-constrained sequential convex programming for robust trajectory optimization," in *Proc. Eur. Contr. Conf. (ECC)*, May 2020, pp. 1871–1878, doi: 10.23919/ecc51009.2020.9143595.
[72] R. Bonalli, T. Lew, and M. Pavone, "Analysis of theoretical and numerical properties of sequential convex programming for continuous-time optimal control," *IEEE Trans. Autom. Control*, submitted for publication. [Online]. Available: https://arxiv.org/abs/2009.05038
[73] R. Bonalli, T. Lew, and M. Pavone, "Sequential convex programming for non-linear stochastic optimal control," *ESAIM Control Optim. Calc. Var.*, submitted for publication. [Online]. Available: https://arxiv.org/abs/2009.05182
[74] Y. Mao, M. Szmuk, and B. Açikmeşe, "A tutorial on real-time convex optimization based guidance and control for aerospace applications," in *Proc. Annu. Amer. Contr. Conf. (ACC)*, Jun. 2018, pp. 2410–2416, doi: 10.23919/acc.2018.8430984.
[75] Y. Mao, D. Dueri, M. Szmuk, and B. Açikmeşe, "Convexification and real-time optimization for MPC with aerospace applications," in *Handbook of Model Predictive Control*, S. Raković and W. Levine, Eds. Cham: Springer International Publishing, Sep. 2018, pp. 335–358, doi: 10.1007/978-3-319-77489-3_15.
[76] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, Dec. 2014, doi: 10.1016/j.automatica.2014.10.128.
[77] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," *Automatica*, vol. 25, no. 3, pp. 335–348, May 1989, doi: 10.1016/0005-1098(89)90002-2.
[78] U. Eren, A. Prach, B. B. Koçer, S. V. Raković, E. Kayacan, and B. Açikmeşe, "Model predictive control in aerospace systems: Current state and opportunities," *J. Guid., Contr., Dyn.*, vol. 40, no. 7, pp. 1541–1566, Jul. 2017, doi: 10.2514/1.g002507.
[79] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation and Design*, 2nd ed. Madison, WI, USA: Nob Hill Publishing, 2017.
[80] M. H. Wright. "Fast times in linear programming: Early success, revolutions, and mysteries." Documents.pub. https://documents.pub/document/fast-times-in-linear-programming-university-of-washington-2011-10-24-fast-times.html (Accessed: Mar. 20, 2020).
[81] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Rev.*, vol. 59, no. 1, pp. 65–98, 2017, doi: 10.1137/141000671.
[82] R. T. Rockafellar, *Convex Analysis*. Princeton, NJ, USA: Princeton Univ. Press, 1970.
[83] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, Dec. 1984, doi: 10.1007/bf02579150.
[84] J. Peng, C. Roos, and T. Terlaky, *Self-Regularity: A New Paradigm for Primal-Dual Interior-Point Algorithms*. Princeton, NJ, USA: Princeton Univ. Press, 2002.
[85] B. Açikmeşe and S. Ploen, "A powered descent guidance algorithm for Mars pinpoint landing," in *Proc. Amer. Inst. Aeronaut. Astronaut. Guid., Navig., Contr. Conf. Exhib.*, Aug. 2005, p. 1, doi: 10.2514/6.2005-6288.
[86] B. Açikmeşe 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, Feb. 2011, doi: 10.1016/j.automatica.2010.10.037.
[87] H. D'Angelo, *Linear Time-Varying Systems: Analysis and Synthesis*. Boston, MA, USA: Allyn & Bacon, 1970.
[88] P. J. Antsaklis and A. N. Michel, *Linear Systems*. Basel, Switzerland: Birkhauser, 2006.
[89] J. M. Carson III, B. Açikmeşe, and L. Blackmore, "Lossless convexification of powered-descent guidance with non-convex thrust bound and pointing constraints," in *Proc. Amer. Contr. Conf.*, Jun. 2011, pp. 2651–2656, doi: 10.1109/acc.2011.5990959.
[90] B. Açikmeşe, J. M. Carson III, and L. Blackmore, "Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem," *IEEE Trans. Control Syst. Technol.*, vol. 21, no. 6, pp. 2104–2113, Nov. 2013, doi: 10.1109/tcst.2012.2237346.
[91] R. F. Hartl, S. P. Sethi, and R. G. Vickson, "A survey of the maximum principles for optimal control problems with state constraints," *SIAM Rev.*, vol. 37, no. 2, pp. 181–218, Jun. 1995, doi: 10.1137/1037043.
[92] M. W. Harris and B. Açikmeşe, "Lossless convexification of non-convex optimal control problems for state constrained linear systems," *Automatica*, vol. 50, no. 9, pp. 2304–2311, Sep. 2014, doi: 10.1016/j.automatica.2014.06.008.
[93] M. W. Harris and B. Açikmeşe, "Lossless convexification for a class of optimal control problems with linear state constraints," in *Proc. 52nd IEEE Conf. Decis. Contr.*, Dec. 2013, pp. 7113–7118, doi: 10.1109/cdc.2013.6761017.
[94] M. W. Harris, "Lossless convexification of optimal control problems," Ph.D. dissertation, Univ. Texas, Austin, TX, USA, 2014.
[95] L. D. Landau and E. M. Lifshitz, *Mechanics*, 2nd ed. Bristol, U.K.: Pergamon, 1969.
[96] W. F. Phillips, *Mechanics of Flight*. Hoboken, NJ, USA: Wiley, 2010.
[97] A. H. J. de Ruiter, C. J. Damaren, and J. R. Forbes, *Spacecraft Dynamics and Control: An Introduction*. Hoboken, NJ, USA: Wiley, 2013.
[98] H. L. Trentelman, A. A. Stoorvogel, and M. Hautus, *Control Theory for Linear Systems*. London, U.K.: Springer-Verlag, 2001.
[99] M. W. Harris and B. Açikmeşe, "Lossless convexification for a class of optimal control problems with quadratic state constraints," in *Proc. Amer. Contr. Conf.*, Jun. 2013, pp. 3415–3420, doi: 10.1109/acc.2013.6580359.
[100] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*, 2nd ed. Hoboken, NJ, USA: Wiley, 2005.
[101] A. Milyutin and N. Osmolovskii, *Calculus of Variations and Optimal Control*. Providence, RI, USA: American Mathematical Society, 1998.
[102] L. Blackmore, B. Açikmeşe, and D. P. Scharf, "Minimum-landing-error powered-descent guidance for Mars landing using convex optimization," *J. Guid., Contr., Dyn.*, vol. 33, no. 4, pp. 1161–1171, Jul. 2010, doi: 10.2514/1.47202.

[103] T. Achterberg and R. Wunderling, "Mixed integer programming: Analyzing 12 years of progress," in *Facets of Combinatorial Optimization*, M. Jünger and G. Reinelt, Eds. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 449–481, doi: 10.1007/978-3-642-38189-8_18.
[104] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton, NJ, USA: Princeton Univ. Press, 2012.
[105] T. Achterberg, "Constrained integer programming," Ph.D. dissertation, Technische Universität Berlin, Berlin, Germany, 2007.
[106] T. Schouwenaars, B. D. Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *Proc. Eur. Contr. Conf. (ECC)*, Sep. 2001, pp. 2603–2608, doi: 10.23919/ecc.2001.7076321.
[107] Z. Zhang, J. Wang, and J. Li, "Lossless convexification of nonconvex MINLP on the UAV path-planning problem," *Optimal Contr. Appl. Methods*, vol. 39, no. 2, pp. 845–859, Dec. 2017, doi: 10.1002/oca.2380.
[108] X. Liu, P. Lu, and B. Pan, "Survey of convex optimization for aerospace applications," *Astrodynamics*, vol. 1, no. 1, pp. 23–40, Sep. 2017, doi: 10.1007/s42064-017-0003-8.
[109] S. Kunhippurayil, M. W. Harris, and O. Jansson, "Lossless convexification of optimal control problems with annular control constraints," *Automatica*, vol. 133, p. 109,848, Nov. 2021, doi: 10.1016/j.automatica.2021.109848.
[110] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, Mar. 1999, doi: 10.1016/s0005-1098(98)00178-2.
[111] D. Malyuta and B. Açikmeşe, "Fast homotopy for spacecraft rendezvous trajectory optimization with discrete logic," *J. Guid., Contr., Dyn.*, to be published, doi: 10.48550/arXiv:2107.07001.
[112] T. Schouwenaars, "Safe trajectory planning of autonomous vehicles," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, USA, 2006.
[113] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 69–81, Oct. 2013, doi: 10.1177/0278364913506757.
[114] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 895–902, Apr. 2018, doi: 10.1109/lra.2018.2792536.
[115] K. Yunt and C. Glocker, "Trajectory optimization of mechanical hybrid systems using SUMT," in *Proc. 9th IEEE Int. Workshop Adv. Motion Contr.*, May 2006, pp. 665–671, doi: 10.1109/amc.2006.1631739.
[116] R. Goebel, R. G. Sanfelice, and A. R. Teel, "Hybrid dynamical systems," *IEEE Control Syst. Mag. (through 2019)*, vol. 29, no. 2, pp. 28–93, Apr. 2009, doi: 10.1109/mcs.2008.931718.
[117] R. Tedrake, "MIT's entry into the DARPA robotics challenge," in *Brains, Minds and Machines Summer Course*. Cambridge MA, USA: MIT OpenCourseWare, 2015.
[118] D. Malyuta and B. Açikmeşe, "Lossless convexification of optimal control problems with semi-continuous inputs," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6843–6850, 2020, doi: 10.1016/j.ifacol.2020.12.341.
[119] M. W. Harris, "Optimal control on disconnected sets using extreme point relaxations and normality approximations," *IEEE Trans. Autom. Control*, vol. 66, no. 12, pp. 6063–6070, 2021, doi: 10.1109/tac.2021.3059682.
[120] H. Hermes and J. P. LaSalle, *Functional Analysis and Time Optimal Control*. Amsterdam, The Netherlands: Elsevier, 1969.
[121] R. Vinter, *Optimal Control*. Cambridge, MA, USA: Birkhauser, 2000.
[122] F. Clarke, "The Pontryagin maximum principle and a unified theory of dynamic optimization," *Proc. Steklov Inst. Math.*, vol. 268, no. 1, pp. 58–69, Apr. 2010, doi: 10.1134/s0081543810010062.
[123] X. Liu, Z. Shen, and P. Lu, "Solving the maximum-crossrange problem via successive second-order cone programming with a line search," *Aerosp. Sci. Technol.*, vol. 47, pp. 10–20, Dec. 2015, doi: 10.1016/j.ast.2015.09.008.
[124] X. Liu and P. Lu, "Solving nonconvex optimal control problems by convex optimization," *J. Guid., Contr., Dyn.*, vol. 37, no. 3, pp. 750–765, 2014, doi: 10.2514/1.62110.
[125] U. Lee and M. Mesbahi, "Constrained autonomous precision landing via dual quaternions and model predictive control," *J. Guid., Contr., Dyn.*, vol. 40, no. 2, pp. 292–308, 2017, doi: 10.2514/1.G001879.
[126] R. Bonalli, B. Hérissé, and E. Trélat, "Optimal control of endo-atmospheric launch vehicle systems: Geometric and computational issues," *IEEE Trans. Autom. Control*, vol. 65, no. 6, pp. 2418–2433, 2020, doi: 10.1109/TAC.2019.2929099.
[127] M. Kočvara, "On the modelling and solving of the truss design problem with global stability constraints," *Structural Multidisciplinary Optim.*, vol. 23, no. 3, pp. 189–203, 2002, doi: 10.1007/s00158-002-0177-3.
[128] A. Beck, A. Ben-Tal, and L. Tetruashvili, "A sequential parametric convex approximation method with applications to nonconvex truss topology design problems," *J. Global Optim.*, vol. 47, no. 1, pp. 29–51, 2010, doi: 10.1007/s10898-009-9456-5.
[129] W. Wei, J. Wang, N. Li, and S. Mei, "Optimal power flow of radial networks and its variations: A sequential convex optimization approach," *IEEE Trans. Smart Grid*, vol. 8, no. 6, pp. 2974–2987, 2017, doi: 10.1109/TSG.2017.2684183.
[130] L. T. Biegler, "Recent advances in chemical process optimization," *Chemie Ingenieur Technik*, vol. 86, no. 7, pp. 943–952, 2014, doi: 10.1002/cite.201400033.
[131] H. Jiang, M. S. Drew, and Z.-N. Li, "Matching by linear programming and successive convexification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 959–975, 2007, doi: 10.1109/TPAMI.2007.1048.
[132] T. F. Chan, S. Esedoğlu, and M. Nikolova, "Algorithms for finding global minimizers of image segmentation and denoising models," *SIAM J. Appl. Math.*, vol. 66, no. 5, pp. 1632–1648, 2006, doi: 10.1137/040615286.
[133] S. E. T. Corp., *Starship — SN10 — High-Altitude Flight Recap*. (Mar. 2021). [Online Video]. Available: https://www.youtube.com/watch?v=gA6ppby3JC8
[134] D. Malyuta *et al.* "Starship landing SCP example." GitHub. https://github.com/UW-ACL/SCPToolbox.jl/tree/master/test/examples/starship_flip (Accessed: Aug. 8, 2022).
[135] D. Rocha, C. J. Silva, and D. F. M. Torres, "Stability and optimal control of a delayed HIV model," *Math. Methods Appl. Sci.*, vol. 41, no. 6, pp. 2251–2260, 2018, doi: 10.1002/mma.4207.
[136] C. J. Silva, H. Maurer, and D. F. M. Torres, "Optimal control of a tuberculosis model with state and control delays," *Math. Biosci. Eng.*, vol. 14, no. 1, pp. 321–337, 2017, doi: 10.3934/mbe.2017021.
[137] R. Dorfman, "An economic interpretation of optimal control theory," *Amer. Econ. Rev.*, vol. 59, no. 5, pp. 817–831, 1969. [Online]. Available: https://www.jstor.org/stable/1810679.
[138] M. Caputo, *Foundations of Dynamic Economic Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
[139] J. R. Banga, "Optimization in computational systems biology," *BMC Syst. Biol.*, vol. 2, no. 1, 2008, Art. no. 47, doi: 10.1186/1752-0509-2-47.
[140] A. Goelzer, V. Fromion, and G. Scorletti, "Cell design in bacteria as a convex optimization problem," *Automatica*, vol. 47, no. 6, pp. 1210–1218, 2011, doi: 10.1016/j.automatica.2011.02.038.
[141] M. Liski, P. M. Kort, and A. Novak, "Increasing returns and cycles in fishing," *Resour. Energy Econ.*, vol. 23, no. 3, pp. 241–258, 2001, doi: 10.1016/S0928-7655(01)00038-0.
[142] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*. Philadelphia, PA, USA: SIAM, Jan. 1994.
[143] D. Bertsekas, *Dynamic Programming and Optimal Control*, 4th ed. Nashua, NH, USA: Athena Scientific, 2017.
[144] D. Bertsekas, *Convex Optimization Algorithms*. Nashua, NH, USA: Athena Scientific, 2015.
[145] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1996.
[146] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
[147] J. E. Falk and R. M. Soland, "An algorithm for separable nonconvex programming problems," *Manage. Sci.*, vol. 15, no. 9, pp. 550–569, 1969, doi: 10.1287/mnsc.15.9.550.
[148] R. M. Soland, "An algorithm for separable nonconvex programming problems II: Nonconvex constraints," *Manage. Sci.*, vol. 17, no. 11, pp. 759–773, 1971, doi: 10.1287/mnsc.17.11.759.
[149] R. Horst, "An algorithm for nonconvex programming problems," *Math. Program.*, vol. 10, no. 1, pp. 312–321, 1976, doi: 10.1007/BF01580678.
[150] R. Horst, "On the convexification of nonlinear programming problems: An applications-oriented survey," *Eur. J. Oper. Res.*, vol. 15, no. 3, pp. 382–392, 1984, doi: 10.1016/0377-2217(84)90107-3.
[151] G. P. McCormick, "Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems," *Math. Program.*, vol. 10, no. 1, pp. 147–175, 1976, doi: 10.1007/BF01580665.
[152] A. Mitsos, B. Chachuat, and P. I. Barton, "Mccormick-based relaxations of algorithms," *SIAM J. Optim.*, vol. 20, no. 2, pp. 573–601, 2009, doi: 10.1137/080717341.
[153] A. Tsoukalas and A. Mitsos, "Multivariate McCormick relaxations," *J. Global Optim.*, vol. 59, nos. 2–3, pp. 633–662, 2014, doi: 10.1007/s10898-014-0176-0.
[154] A. B. Singer and P. I. Barton, "Global solution of optimization problems with parameter-embedded linear dynamic systems," *J. Optim. Theory Appl.*, vol. 121, no. 3, pp. 613–646, 2004, doi: 10.1023/B:JOTA.0000037606.79050.a7.
[155] A. B. Singer and P. I. Barton, "Bounding the solutions of parameter dependent nonlinear ordinary differential equations," *SIAM J. Sci. Comput.*, vol. 27, no. 6, pp. 2167–2182, 2006, doi: 10.1137/040604388.

[156] R. Horst and N. V. Thoai, "DC programming: Overview," *J. Optim. Theory Appl.*, vol. 103, no. 1, pp. 1–43, 1999, doi: 10.1023/a:1021765131316.
[157] T. Lipp and S. Boyd, "Variations and extension of the convex–concave procedure," *Optim. Eng.*, vol. 17, no. 2, pp. 263–287, 2016, doi: 10.1007/s11081-015-9294-x.
[158] A. L. Yuille and A. Rangarajan, "The concave-convex procedure," *Neural Comput.*, vol. 15, no. 4, pp. 915–936, 2003, doi: 10.1162/08997660360581958.
[159] G. R. Lanckriet and B. K. Sriperumbudur, "On the convergence of the concave-convex procedure," in *Proc. 22nd Adv. Neural Inf. Process. Syst.*, Curran Associates, Inc., 2009, pp. 1759–1767.
[160] F. Palacios-Gomez, L. Lasdon, and M. Engquist, "Nonlinear optimization by successive linear programming," *Manage. Sci.*, vol. 28, no. 10, pp. 1106–1120, 1982, doi: 10.1287/mnsc.28.10.1106.
[161] 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," *Math. Program., B*, vol. 100, pp. 27–48, Nov. 2003, doi: 10.1007/s10107-003-0485-4.
[162] S. P. Han, "A globally convergent method for nonlinear programming," *J. Optim. Theory Appl.*, vol. 22, no. 3, pp. 297–309, 1977, doi: 10.1007/BF00932858.
[163] S. P. Han and O. L. Mangasarian, "Exact penalty functions in nonlinear programming," *Math. Program.*, vol. 17, no. 1, pp. 251–269, 1979, doi: 10.1007/BF01588250.
[164] M. J. D. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in *Numerical Analysis*, G. A. Watson, Ed. Berlin, Heidelberg: Springer-Verlag, 1978, pp. 144–157, doi: 10.1007/BFb0067703.
[165] M. J. D. Powell, "Algorithms for nonlinear constraints that use Lagrangian functions," *Math. Program.*, vol. 14, pp. 224–248, Dec. 1978, doi: 10.1007/BF01588967.
[166] M. J. D. Powell and Y. Yuan, "A recursive quadratic programming algorithm that uses differentiable exact penalty functions," *Math. Program.*, vol. 35, no. 3, pp. 265–278, 1986, doi: 10.1007/BF01580880.
[167] P. T. Boggs, J. W. Tolle, and P. Wang, "On the local convergence of quasi-newton methods for constrained optimization," *SIAM J. Control Optim.*, vol. 20, no. 2, pp. 161–171, 1982, doi: 10.1137/0320014.
[168] P. T. Boggs and J. W. Tolle, "A family of descent functions for constrained optimization," *SIAM J. Numer. Anal.*, vol. 21, no. 6, pp. 1146–1161, 1984, doi: 10.1137/0721071.
[169] P. T. Boggs and W. J. Tolle, "A strategy for global convergence in a sequential quadratic programming algorithm," *SIAM J. Numer. Anal.*, vol. 26, no. 3, pp. 600–623, 1989, doi: 10.1137/0726036.
[170] M. Fukushima, "A successive quadratic programming algorithm with global and superlinear convergence properties," *Math. Program.*, vol. 35, no. 3, pp. 253–264, 1986, doi: 10.1007/BF01580879.
[171] P. T. Boggs and W. J. Tolle, "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–52, Jan. 1995, doi: 10.1017/S0962492900002518.
[172] R. H. Byrd, R. B. Schnabel, and G. A. Shultz, "Approximate solution of the trust region problem by minimization over two-dimensional subspaces," *Math. Program.*, vol. 40, nos. 1–3, pp. 247–263, Jan. 1988, doi: 10.1007/bf01580735.
[173] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Rev.*, vol. 47, no. 1, pp. 99–131, 2005, doi: 10.1137/s0036144504446096.
[174] J. T. Betts and W. P. Huffman, "Path-constrained trajectory optimization using sparse sequential quadratic programming," *J. Guid., Contr., Dyn.*, vol. 16, no. 1, pp. 59–68, Feb. 1993, doi: 10.2514/3.11428.
[175] C. T. Lawrence and A. L. Tits, "A computationally efficient feasible sequential quadratic programming algorithm," *SIAM J. Optim.*, vol. 11, no. 4, pp. 1092–1118, 2001, doi: 10.1137/s1052623498344562.
[176] P. E. Gill and E. Wong, "Sequential quadratic programming methods," in *Mixed Integer Nonlinear Programming*, J. Lee and S. Leyffer, Eds. New York: Springer-Verlag, 2012, pp. 147–224, doi: 10.1007/978-1-4614-1927-3_6.
[177] R. Fletcher, *Practical Methods of Optimization*. Hoboken, NJ, USA: Wiley, 2013.
[178] B. Fares, D. Noll, and P. Apkarian, "Robust control via sequential semidefinite programming," *SIAM J. Control Optim.*, vol. 40, no. 6, pp. 1791–1820, 2002, doi: 10.1137/S0363012900373483.
[179] S. Boyd, L. E. Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Philadelphia, PA, USA: SIAM, 1994.
[180] T. Reynolds, D. Malyuta, M. Mesbahi, B. Acikmese, and J. M. Carson, "Funnel synthesis for the 6-DOF powered descent guidance problem," in *Proc. Amer. Inst. Aeronaut. Astronaut. Scitech Forum*, Jan. 2021, pp. 1–21, doi: 10.2514/6.2021-0504.
[181] T. P. Reynolds, "Computation guidance and control for aerospace systems," Ph.D. dissertation, Univ. of Washington, Seattle, WA, USA, 2021.
[182] J. Schulman *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1251–1270, 2014, doi: 10.1177/0278364914528132.
[183] M. Sagliano, "Pseudospectral convex optimization for powered descent and landing," *J. Guid., Contr., Dyn.*, vol. 41, no. 2, pp. 320–334, 2017, doi: 10.2514/1.G002818.
[184] P. Simplício, A. Marcos, and S. Bennani, "Guidance of reusable launchers: Improving descent and landing performance," *J. Guid., Contr., Dyn.*, vol. 42, no. 10, pp. 2206–2219, 2019, doi: 10.2514/1.G004155.
[185] Y. Mao, M. Szmuk, and B. Açikmeşe, "Successive convexification of nonconvex optimal control problems and its convergence properties," in *Proc. IEEE 5th Conf. Decis. Contr.*, 2016, pp. 3636–3641, doi: 10.1109/CDC.2016.7798816.
[186] M. Szmuk, "Successive convexification & high performance feedback control for agile flight," Ph.D. dissertation, Univ. of Washington, Seattle, WA, USA, 2019.
[187] H. Saranathan and M. J. Grant, "Relaxed autonomously switched hybrid system approach to indirect multiphase aerospace trajectory optimization," *J. Spacecraft Rockets*, vol. 55, no. 3, pp. 611–621, May 2018, doi: 10.2514/1.a34012.
[188] E. Taheri, J. L. Junkins, I. Kolmanovsky, and A. Girard, "A novel approach for optimal trajectory design with multiple operation modes of propulsion system, part 1," *Acta Astronaut.*, vol. 172, pp. 151–165, Jul. 2020, doi: 10.1016/j.actaastro.2020.02.042.
[189] K. Shoemake, "Animating rotation with quaternion curves," in *Proc. 1985 12th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, pp. 245–254, doi: 10.1145/325334.325242.
[190] T. P. Reynolds and M. Mesbahi, "Optimal planar powered descent with independent thrust and torque," *J. Guid., Contr., Dyn.*, vol. 43, no. 7, pp. 1225–1231, Jul. 2020, doi: 10.2514/1.g004701.
[191] D. Malyuta, T. P. Reynolds, M. Szmuk, M. Mesbahi, B. Açikmeşe, and J. M. Carson III, "Discretization performance and accuracy analysis for the rocket powered descent guidance problem," in *Proc. Amer. Inst. Aeronaut. Astronaut. SciTech Forum*, 2019, pp. 1–20, doi: 10.2514/6.2019-0925.
[192] B. Açikmeşe, D. Scharf, L. Blackmore, and A. Wolf, "Enhancements on the convex programming based powered descent guidance algorithm for Mars landing," in *Proc. Amer. Inst. Aeronaut. Astronaut. Amer. Astronaut. Soc. Astrodyn. Specialist Conf. Exhib.*, Aug. 2008, pp. 1–16, doi: 10.2514/6.2008-6426.
[193] D. Dueri, Y. Mao, Z. Mian, J. Ding, and B. Acikmese, "Trajectory optimization with inter-sample obstacle avoidance via successive convexification," in *Proc. IEEE 56th Annu. Conf. Decis. Contr.*, Dec. 2017, pp. 1150–1156, doi: 10.1109/cdc.2017.8263811.
[194] M. W. Hirsch, S. Smale, and R. L. Devaney, *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Amsterdam, The Netherlands: Elsevier, 2013.
[195] F. Bullo and A. D. Lewis, *Geometric Control of Mechanical Systems*. New York: Springer-Verlag, 2005.
[196] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing," in *Proc. 13th Int. Conf. Neural Inf. Process. Syst.*, MIT Press, Cambridge, MA, USA, 2000, pp. 451–457.
[197] P. A. Absil, R. Mahony, and B. Andrews, "Convergence of the iterates of descent methods for analytic cost functions," *SIAM J. Optim.*, vol. 16, no. 2, pp. 531–547, Jan. 2005, doi: 10.1137/040605266.
[198] N. Schorghofer, *Lessons in Scientific Computing: Numerical Mathematics, Computer Technology, and Scientific Discovery*. Boca Raton, FL, USA: CRC Press, 2018.
[199] I. M. Ross, Q. Gong, M. Karpenko, and R. J. Proulx, "Scaling and balancing for high-performance computation of optimal controls," *J. Guid., Contr., Dyn.*, vol. 41, no. 10, pp. 2086–2097, 2018, doi: 10.2514/1.G003382.
[200] D. Malyuta, "An optimal endurance power limiter for an electric race car developed for the AMZ racing team," Semester Project, ETH Zurich, Institute for Dynamic Systems and Control, Zurich, Switzerland, Sep. 2016.
[201] J. Sola, "Quaternion kinematics for the error-state Kalman filter," 2017, *arXiv:1711.02508*.
[202] J. Renegar, *A Mathematical View of Interior-Point Methods in Convex Optimization*. Philadelphia, PA, USA: SIAM, Jan. 2001.
[203] T. P. Reynolds and M. Mesbahi, "The crawling phenomenon in sequential convex programming," in *Proc. Amer. Contr. Conf.*, Denver, CO, USA, 2020, pp. 3613–3618, doi: 10.23919/ACC45564.2020.9147550.
[204] A. G. Kamath, "Robust thrust vector control for precision rocket-landing," Ph.D. dissertation, Univ. of California Davis, Davis, CA, USA, 2021.
[205] A. G. Kamath, F. F. Assadian, and S. K. Robinson, "Multivariable robust control for the powered-descent of a multibody lunar landing system,"

in *Proc. Amer. Inst. Aeronaut. Astronaut. Amer. Astronaut. Soc. Astrodyn. Specialist Conf.*, Aug. 2020, pp. 1–19.
[206] M. Szmuk, C. A. Pascucci, and B. Acikmese, "Real-time quad-rotor path planning for mobile obstacle avoidance using convex optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 1–9, doi: 10.1109/IROS.2018.8594351.
[207] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *Int. J. Robot. Res.*, vol. 36, no. 8, pp. 947–982, 2017, doi: 10.1177/0278364917712421.
[208] T. P. Reynolds, D. Malyuta, M. Mesbahi, and B. Açikmeşe, "Temporally-interpolated funnel synthesis for nonlinear systems," in *Proc. RSS Workshop Robust Autonomy*, Jul. 2020, pp. 1–4.
[209] B. Açikmeşe, J. M. Carson III, and D. S. Bayard, "A robust model predictive control algorithm for incrementally conic uncertain/nonlinear systems," *Int. J. Robust Nonlinear Contr.*, vol. 21, no. 5, pp. 563–590, Jul. 2010, doi: 10.1002/rnc.1613.
[210] T. P. Reynolds and M. Mesbahi, "Small body precision landing via convex model predictive control," in *Proc. Amer. Inst. Aeronaut. Astronaut. SPACE Astronaut. Forum Expo.*, Sep. 2017, pp. 1–13, doi: 10.2514/6.2017-5179.
[211] C. A. Pascucci, S. Bennani, and A. Bemporad, "Model predictive control for powered descent guidance and control," in *Proc. Eur. Contr. Conf. (ECC)*, Jul. 2015, pp. 1388–1393, doi: 10.1109/ecc.2015.7330732.
[212] Q. T. Dinh, C. Savorgnan, and M. Diehl, "Real-time sequential convex programming for nonlinear model predictive control and application to a hydro-power plant," in *Proc. IEEE Conf. Decis. Contr. Eur. Contr. Conf*, Dec. 2011, pp. 5905–5910, doi: 10.1109/cdc.2011.6160919.
[213] D. Lee, K. Turitsyn, and J.-J. Slotine, "Robust model predictive control for nonlinear systems using convex restriction," Apr. 2021, *arXiv:2003.00345*.
[214] D. Morgan, S.-J. Chung, and F. Y. Hadaegh, "Model predictive control of swarms of spacecraft using sequential convex programming," *J. Guid., Contr., Dyn.*, vol. 37, no. 6, pp. 1725–1740, Nov. 2014, doi: 10.2514/1.g000218.
[215] D. Morgan, G. P. Subramanian, S.-J. Chung, and F. Y. Hadaegh, "Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming," *Int. J. Robot. Res.*, vol. 35, no. 10, pp. 1261–1285, Feb. 2016, doi: 10.1177/0278364916632065.
[216] M. J. Tenny, S. J. Wright, and J. B. Rawlings, "Nonlinear model predictive control via feasibility-perturbed sequential quadratic programming," *Comput. Optim. Appl.*, vol. 28, no. 1, pp. 87–121, Apr. 2004, doi: 10.1023/b:coap.0000018880.63497.eb.
[217] R. H. Goddard, "A method of reaching extreme altitudes," *Nature*, vol. 105, pp. 809–811, Aug. 1920, doi: 10.1038/105809a0.
[218] P. Lu, "Entry guidance and trajectory control for reusable launch vehicle," *J. Guid., Contr., Dyn.*, vol. 20, no. 1, pp. 143–149, 1997, doi: 10.2514/2.4008.
[219] B. Bonnard, L. Faubourg, G. Launay, and E. Trélat, "Optimal control with state constraints and the space shuttle re-entry problem," *J. Dyn. Contr. Syst.*, vol. 9, no. 2, pp. 155–199, 2003, doi: 10.1023/A:1023289721398.
[220] N. Ratliff, M. Zucker, J. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 489–494, doi: 10.1109/ROBOT.2009.5152817.
[221] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 4569–4574, doi: 10.1109/ICRA.2011.5980280.
[222] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom. (1989–June 2004)*, vol. 12, no. 4, pp. 566–580, 1996, doi: 10.1109/70.508439.
[223] S. M. LaValle and J. J. Kuffner Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001, doi: 10.1177/02783640122067453.
[224] S. Ploen, B. Acikmese, and A. Wolf, "A comparison of powered descent guidance laws for Mars pinpoint landing," in *Proc. Amer. Inst. Aeronaut. Astronaut. Amer. Astronaut. Soc. Astrodyn. Specialist Conf. Exhib.*, Aug. 2006, pp. 1–16, doi: 10.2514/6.2006-6676.
[225] J. M. Carson, B. Acikmese, L. Blackmore, and A. A. Wolf, "Capabilities of convex powered-descent guidance algorithms for pinpoint and precision landing," in *Proc. Aerosp. Conf.*, Mar. 2011, pp. 1–8, doi: 10.1109/aero.2011.5747244.
[226] A. Wolf, J. Tooley, S. Ploen, M. Ivanov, B. Acikmese, and K. Gromov, "Performance trades for Mars pinpoint landing," in *Proc. 2006 IEEE Aerosp. Conf.*, p. 16, doi: 10.1109/aero.2006.1655793.
[227] A. A. Wolf, J. Casoliva, J. B. Manrique, B. Acikmese, and S. Ploen, "Improving the landing precision of an MSL-class vehicle," in *Proc. IEEE Aerosp. Conf.*, Mar. 2012, pp. 1–10, doi: 10.1109/aero.2012.6187005.
[228] A. B. Mandalia and R. D. Braun, "Supersonic retropropulsion thrust vectoring for Mars precision landing," *J. Spacecraft Rockets*, vol. 52, no. 3, pp. 827–835, May 2015, doi: 10.2514/1.a33119.
[229] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *Proc. Eur. Contr. Conf. (ECC)*, Jul. 2013, pp. 3071–3076, doi: 10.23919/ecc.2013.6669541.
[230] M. A. Estrada, B. Hockman, A. Bylard, E. W. Hawkes, M. R. Cutkosky, and M. Pavone, "Free-flyer acquisition of spinning objects with gecko-inspired adhesives," in *Proc. IEEE Int. Conf. Robot. Automat.*, Stockholm, Sweden, 2016, pp. 4907–4913, doi: 10.1109/ICRA.2016.7487696.
[231] M. Mote, M. Egerstedt, E. Feron, A. Bylard, and M. Pavone, "Collision-inclusive trajectory optimization for free-flying spacecraft," *J. Guid., Contr., Dyn.*, vol. 43, no. 7, pp. 1247–1258, 2020, doi: 10.2514/1.G004788.
[232] T. Smith *et al.*, "ASTROBEE: A new platform for free-flying robotics on the international space station," in *Proc. Int. Symp. Artif. Intell., Robot. Automat. Space*, Beijing, China, 2016, pp. 1–8.
[233] "Meet CIMON-2, a new and improved AI robot astronaut." Discover Magazine. https://www.discovermagazine.com/technology/meet-cimon-2-a-new-and-improved-ai-robot-astronaut (Accessed: Aug. 8, 2022).
[234] "What is Astrobee?" National Aeronautics and Space Administration, Washington, DC, USA, Nov. 2020. [Online]. Available: https://www.nasa.gov/astrobee
[235] J. Barlow *et al.*, "Astrobee: A new platform for free-flying robotics on the international space station," in *Proc. Int. Symp. Artif. Intell., Robot., Automat. Space (i-SAIRAS)*, 2016, pp. 1–8.
[236] D. Szafir, B. Mutlu, and T. Fong, "Communicating directionality in flying robots," in *Proc. 10th Annu. ACM/IEEE Int. Conf. Hum.-Robot Interact.*, Mar. 2015, pp. 19–26, doi: 10.1145/2696454.2696475.
[237] M. Bualat, J. Barlow, T. Fong, C. Provencher, and T. Smith, "Astrobee: Developing a free-flying robot for the international space station," in *Proc. Amer. Inst. Aeronaut. Astronaut. SPACE Conf. Expo.*, Aug. 2015, pp. 1–10, doi: 10.2514/6.2015-4643.
[238] JAXA. *Ultra-Compact Triaxial Attitude Control Module-Application to 'Kibo' Inboard Drone (Int-Ball).* (Jul. 2017). [Online Video]. Available: https://youtu.be/ZtIARUS7Lqc
[239] P. Roque and R. Ventura, "Space CoBot: Modular design of an holonomic aerial robot for indoor microgravity environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 4383–4390, doi: 10.1109/IROS.2016.7759645.
[240] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, and S. Hillaire, *Real-Time Rendering*, 4th ed. Boca Raton, FL, USA: CRC Press, Jul. 2018.
[241] E. Coumans and Y. Bai. "Pybullet, A python module for physics simulation for games, robotics and machine learning." GitHub. http://pybullet.org (Accessed: Aug. 8, 2022).
[242] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[243] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[244] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "FORCES NLP: An efficient implementation of interior-point methods for multistage nonlinear nonconvex programs," *Int. J. Control*, vol. 93, no. 1, pp. 13–29, May 2017, doi: 10.1080/00207179.2017.1316017.
[245] S. Mceowen, D. Sullivan, B. Chasnov, D. Calderone, and M. Szmuk, "Visual modeling system for real-time optimal trajectory planning for autonomous aerial drones," in *Proc. IEEE Aerospace Conf.*, 2022.
[246] Autonomous Control Laboratory. *Scenario 5: Quadrotor Vehicle Tracking Performance*. (Oct. 2020). [Online Video]. Available: https://www.youtube.com/watch?v=Ou7ZfyiXeyw
[247] "First disclosure of images taken by the Kibo's internal drone 'Int-Ball'." Japan Aerospace Exploration Agency. https://iss.jaxa.jp/en/kiboexp/news/170714_int_ball_en.html (Accessed: Aug. 8, 2022).
[248] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming," CVX Research, Version 2.1, Mar. 2014. [Online]. Available: http://cvxr.com/cvxr
[249] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control* (Lecture Notes in Control and Information Sciences), V. Blondel, S. Boyd, and H. Kimura, Eds., London, U.K.: Springer-Verlag Ltd., 2008, pp. 95–110. [Online]. Available: http://stanford.edu/~boyd/graph_dcp.html