A Real-Time Algorithm for Non-Convex Powered Descent Guidance

Taylor P. Reynolds*, Danylo Malyuta*,
Mehran Mesbahi†, Behçet Açıkmeşe†,
Dept. of Aeronautics & Astronautics, University of Washington, Seattle, WA 98195, USA
and
John M. Carson III‡
NASA Johnson Space Center, Houston, TX 77058, USA

The on-board solution of constrained optimal control problems is a key technology for future entry, descent and landing systems. The constraints that must be satisfied to enable advanced navigation routines require powered descent guidance solutions that consider the coupled rotation and translation of the vehicle, leading to a non-convex 6-degree-of-freedom powered descent guidance problem. This paper builds on previous work and refines a successive convexification algorithm to be compatible with common flight code requirements. We highlight the aspects of each algorithmic step that are especially relevant for maximizing the computational performance. A case study is presented using the most general landing problem for which the optimal solution is theoretically known and that contains both rotational and translational states. We demonstrate that the real-time implementation achieves less than 1% sub-optimality with runtimes on the order of 100 ms on a single 3.2 GHz Intel i5 core with 8 GB of RAM. These results suggest that the same design methodology applied to the full 6-degree-of-freedom landing problem is capable of producing fast enough runtimes to be viable for future entry, descent and landing systems.

I. Introduction

Powered Descent Guidance (PDG) is the terminal phase of the entry, descent and landing sequence in which a lander uses its rocket engine(s) and, optionally, its reaction control system to maneuver from an initial state to a soft touchdown. Powered descent is a fundamental technology for safe and precise landing, and the search for a general and reliable PDG solution has been of increasing importance for extraterrestrial spaceflight. Space launch continues to be expensive, ranging from above \$10,000 kg^{-1} during the Space Shuttle era to a current low of \$3,000 kg^{-1} for the Falcon 9,[1] with high costs predicted for the Space Launch System.[2] To help reduce launch mass (and therefore mission cost), PDG solutions typically seek to minimize the use of propellant. For example, there is evidence that the (sub-optimal) polynomial guidance algorithm used during the Apollo program could come within 16 kg of the minimum fuel usage[3] for that problem.

Future space missions have a strong need for a PDG algorithm that satisfies the following requirements.[4] To account for lander pointing requirements imposed by sensing architectures, the algorithm must be capable of handling a large suite of continuous and discrete input and state constraints. Consequently, trajectory planning must at least consider the 6-degree-of-freedom (DoF) rigid body rotation and translation of the vehicle. Second, future robotic landers will have to navigate challenging environments such as volcanic vents and jagged blades of ice,[5,6] while human missions are likely to be preceded by cargo missions and will require that landings occur in close proximity.[7] To support hazard avoidance and real-time intelligent re-planning, the algorithm must have predictable convergence behaviour and be real-time capable on computationally constrained hardware. Lastly, the algorithm must function autonomously so that it can be used by both robotic and crewed missions. Motivated by the presence of water ice, future missions to the Moon in particular will target its south pole.[8] The low tilt

*Ph.D. Candidate, AIAA Student Member {tpr6,danylo}@uw.edu
†Professor, AIAA Associate Fellow, {mesbahi,behcet}@uw.edu
‡SPLICE Principal Investigator, AIAA Associate Fellow, john.m.carson@nasa.gov

of the Moon's spin axis creates extreme light-dark lighting conditions at the poles, further emphasizing the need for a fully automated sensor-based PDG sequence.[9]

The first step towards a real-time constrained 6-DoF PDG algorithm was taken during the Apollo era when the analytic solution to a fuel-optimal 1-DoF purely vertical descent was found by Meditch.[10] Around the same time, Lawden formulated the necessary conditions of optimality for a more general fuel-optimal 3-DoF problem that considers the mass and translational dynamics.[11,12] However, the computational resources required to solve the necessary conditions (e.g., using shooting methods) were not yet readily available. More recently, D'Souza found closed-form solutions to the necessary conditions for a mixed minimum-energy minimum-time unconstrained 3-DoF problem.[13] Using modern software, Topcu, Casoliva and Mease compared the attainable solution quality for the fuel-optimal 3-DoF problem to the necessary conditions of optimality derived by Lawden.[14,15] Their focus was not to develop a real-time PDG algorithm, as they used nonlinear programming methods that do not meet the previously mentioned computational requirements. A real-time capable algorithm that solves the 3-DoF necessary conditions using nonlinear root finding techniques has been developed by Lu.[16,17] However, this algorithm has limited ability to enforce state constraints that may be required by future missions.

Over the past 40 years, research in numerical optimization has developed a family of interior point methods that can solve optimization problems to global optimality in polynomial time and with guaranteed convergence.[18,19] These much-needed properties for an on-board PDG algorithm stand in stark contrast to both shooting and nonlinear programming. The only limitation, roughly speaking, is that problems must be convex.[20] However, Acikmese and Ploen recognized that the fuel-optimal 3-DoF problem with several important constraints like the approach angle and rocket engine thrust lower- and upper-bounds can be expressed as a convex optimization problem and solved to global optimality.[21] The method, termed lossless convexification, was the first to solve the 3-DoF problem with meaningful constraints in a fashion suitable for robust on-board implementation. Over the course of the next decade, the method was expanded to handle fairly general non-convex input sets,[22] minimum-error landing and thrust pointing constraints,[23–25] classes of affine and quadratic state constraints,[26–28] classes of nonlinear (mixed-integer) dynamics,[29] and even certain binary constraints without resorting to mixed-integer programming.[30,31] In principle, all lossless convexification results are real-time implementable as they require solving one or a (small) bounded sequence of problems via Second-Order Cone Programming (SOCP). Fast and efficient SOCP solvers that use only static memory allocation are available.[32–34] As an added benefit, the solvability of 3-DoF problem using lossless convexification is guaranteed anywhere within a well-defined convex set that can be computed during mission design.[35–37]

However, a major limitation inherent to any 3-DoF PDG algorithm is that the computed trajectory only respects translation dynamics and constraints related to either translation or the thrust vector. 3-DoF PDG methods do not model attitude dynamics, and instead assume that the attitude is controlled by a much faster inner control loop. More importantly, 3-DoF methods cannot produce solutions that are guaranteed to respect sensing requirements such as line-of-sight constraints for vehicle-mounted sensors. Even for missions where such constraints are "mild", an extensive simulation campaign is required to validate that the 3-DoF trajectories are executable by a fundamentally 6-DoF lander system.[4] For the aforementioned future missions where sensing and guidance are tightly coupled, 3-DoF solutions are likely out of the question without considerable backstage hand-tuning and edge-case handling efforts. Thus it is well motivated to seek a real-time algorithm for 6-DoF PDG that does not compromise the benefits of the lossless 3-DoF solution (i.e., speed, guaranteed convergence, and feasibility) while simultaneously providing the benefit of explicit feasibility with respect to the full 6-DoF dynamics and coupled guidance-navigation constraints.

Sequential Convex Programming (SCP) emerged as the most natural solution when transitioning from a fully convex 3-DoF problem to a 6-DoF problem in which some non-convexity is present. As a special case of a trust region method, SCP iteratively solves a convex approximation of the original optimal control problem, and updates the approximation as new solutions are obtained.[18] This permits the use of "heritage" SOCP solvers developed for 3-DoF PDG, and the balance of the algorithm design is used to ensure that the SCP iterations converge to a local optimum of the non-convex 6-DoF problem. In the context of fuel-optimal 6-DoF PDG, SCP methods have been used to handle: aerodynamic lift and drag; thrust slew-rate constraints;[38] free ignition time; state-triggered constraints that model discrete decisions such as velocity-based angle-of-attack constraints;[39] and compound (i.e., and/or) state-triggered constraints[40] such as line of sight constraints that are active only within pre-specified slant range bands.[41,42] Collectively, these works show that the constraint satisfaction requirements of future space flight missions can be handled via SCP-based 6-DoF PDG algorithms.

Several state-of-the-art SCP algorithms for trajectory generation have emerged in the past 5 years. These are: Penalized Trust Region (PTR),[39,41] successive convexification (SCvx),[43–45] TrajOpt,[46] GuSTO,[47,48] and ALTRO.[49] The trust region size in PTR is a solution variable whose magnitude is penalized in the cost, whereas

SCvx and GuSTO enforce hard trust regions with outer-loop update schemes. Similar work was done by Liu and Lu for the speci c case of concave state constraints, albeit with a trust region that is not updated.[50] TrajOpt updates the trust region size, but enforces only soft constraints and no dynamics. ALTRO is a new algorithm that alternates between augmented Lagrangian iterative LQR to deal with nonlinear dynamics directly, and active-set projection to satisfy constraints. Beyond these \generalized" solution methods, SCP has been used to solve a wide range of aerospace guidance problems.[51–54] In light of our history as the primary developers of the PTR method, this paper explores the real-time capability of the PTR method speci cally for PDG.

The real-time capability of the PTR method for other applications has been tested on several occasions. Szmuk et al. have shown that PTR is able to solve di cult quadrotor path planning problems (including obstacle avoidance and acrobatic ips) in less than 100 ms running on a 1.7 GHz Intel Atom processor of an embedded Intel Joule platform.[55,56] More di cult tasks such as ying through hoops and cooperatively transporting a beam while avoiding obstacles were solved in less than one second.[57] Beyond these real-world tests of on-board PTR, the real-time properties of the method and the e ect of discretization were also analyzed in simulation.[58,59] Ultimately, an SCP based method is a sequence of solutions to convex optimization problems and, in the case of PTR, these problems are SOCPs. In assessing the real-time capability of PTR it is therefore appropriate to consider existing work on the real-time solution of single-SOCP optimization problems. In the context of 3-DoF PDG, a 7- ight 3-year ight test campaign aboard the Masten Space Systems Xombie sounding rocket demonstrated that the on-board solution of an SOCP problem is feasible on space ight processors.[60–63] The tested algorithm, G-FOLD, is based on the theory of lossless convexi cation and was able to compute landing divert trajectories in 100 ms on a 1.4 GHz Intel Pentium M processor. Moreover, Dueri et al. have shown that the same algorithm can achieve runtimes of less than 700 ms on the radiation-hardened BAE RAD750 processor.[33]

The modern paradigm of real-time optimization is to use customized solvers, wherein problem structure is hard-coded into the solution process by an o ine code generator.[32,33,64] Well-known examples include CVXGEN and FORCES.[65–68] The majority of existing work on real-time optimization is found in the model predictive control (MPC) literature, where closed-loop feedback is achieved by (generally) calling an optimizer as fast as the desired execution frequency of the controller. In this context, execution frequencies for linear and quadratic optimization problems in the kHz-MHz range are possible, and have been applied to electrical drives and an atomic force microscope.[69,70] MPC has also been used on-board autonomous race cars, achieving re-solve rates of 50 Hz for miniature cars on an embedded 1.7 GHz ARM A9 chip,[71] and 20 Hz for a life-size car on a 2.1 GHz Intel i7-3612QE.[72]

MPC strategies have been proposed for 6-DoF PDG.[73] However, the MPC paradigm of constantly re-solving an optimization problem to make feedback control decisions is in con ict with the use of classical feedback control architectures in space missions and can pose constraint feasibility problems. To provide the required 6-DoF PDG capability while minimally impacting heritage guidance and control architectures, we propose that guidance algorithms solve for a single, complete, trajectory from the spacecraft's current state all the way to the landing site. The vehicle would subsequently track this trajectory using existing feedback control architectures. Given the real-time capability, new trajectories can be found as-needed (e.g., during landing site re-targeting for hazard avoidance), but re-solving is not a fundamental requirement of our approach. The objective of \real-time PDG" is therefore that a single solution for a complete 6-DoF PDG trajectory is computable in a short enough time span on space ight hardware so as to be implementable on-board a spacecraft.

A. Primary Contribution

The primary contribution of this paper is to cover the major design aspects of a real-time capable PTR-based PDG algorithm. We discuss the analytic transformation of the desired optimization problem into a sequence of standard form SOCP problems, highlighting the in uence of certain parameters and each type of constraint on the resulting problem size and solution time. We present a hard-coded transcription of the optimization problem for use with a ight-proven SOCP solver referred to as BSOCP.[33] Unlike previous work on the 6-DoF PDG problem,[39–42] we only use static memory allocation, and ensure that our methodology and resulting code is compatible with common ight code requirements.[74] As a numerical demonstration, we use a simpli ed in-plane powered descent problem with independent thrust and torque inputs, for which a theoretically optimal solution is available.[75] The latter quali cation is key for assessing the optimality that is achieved by our real-time implementation, and is not available for any other PDG problem that considers the attitude of the vehicle. Despite having fewer states, the dynamics and constraints of this simpler problem capture important aspects of the dynamics present in 6-DoF PDG problems. Our secondary contribution is therefore to compare the solution obtained with the PTR method to this known optimum. This provides valuable insight into the level of sub-optimality in a PTR-based 6-DoF PDG solution. Our third contribution is to provide detailed timing information

Figure 1: Illustration of the steps in the PTR algorithm. The most time consuming step is highlighted in red and involves solving a convex SOCP problem.

as a function of the discretization density and a sensitivity analysis for three algorithm metrics as a function of key design parameters. Our findings support that the PTR-based 6-DoF PDG algorithm is a real-time capable algorithm for next generation entry, descent and landing systems.

This paper is organized as follows. First, Section II presents the steps to produce an algorithm that is both capable of solving non-convex optimization problems and is conducive to a real-time implementation. An overview of the algorithm shows that there are three primary steps: initialization, convexification, and the solve step that are outlined in Section II.B, Section II.C and Section II.D respectively. Next, Section III provides a planar landing case study that demonstrates how the algorithm is constructed using these steps, in addition to the computational and numerical performance that can be obtained. Lastly, Section IV offers concluding remarks.

II. Considerations for Real-Time Implementation of Non-Convex Problems

In this section, we present a general methodology for developing real-time implementations to solve non-convex optimal control problems of the form:

$$\begin{aligned} \min_{u(\cdot),p} \quad & \phi(x(t_f), p) \\ \text{subject to:} \quad & \dot{x}(t) = f(x, u, p), \\ & H_0 x(t_0) = x_{ic}, \quad H_f x(t_f) = x_f, \\ & g_i(x, u, p) \leq 0, \quad i \in \mathcal{I}_c := \{1, \ldots, n_c\}, \end{aligned} \tag{Problem 1}$$

where $x(t) \in \mathbb{R}^{n_x}$ is a continuous-time state vector, $u(t) \in \mathbb{R}^{n_u}$ is a continuous-time input vector and $p \in \mathbb{R}^{n_p}$ is a parameter vector. The scalar-valued functions $g_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}$ represent the path constraints imposed during the maneuver. We assume that the constraints indexed by $\mathcal{I}_{cvx} \subseteq \mathcal{I}_c$ are convex, and the constraints indexed by $\mathcal{I}_{ncvx} \subseteq \mathcal{I}_c$ are non-convex, so that $\mathcal{I}_c = \mathcal{I}_{cvx} \cup \mathcal{I}_{ncvx}$. The matrices H_0 and H_f represent the ability to constrain only a subset of the state vector at the initial and final time. We assume that $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$ is differentiable almost everywhere with respect to its arguments. Lastly, we assume without loss of generality that the cost function is convex and is given in Mayer form[76] using the function $\phi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}$. Any problem that nominally includes a running cost can be written in this form, and it is the simplest way to transcribe the continuous time problem into the standard form needed for real-time implementations. Note that free-final-time problems can be expressed in the form of Problem 1 by using the parameter vector p.

A. Successive Convexification: Overview

We begin our discussion on real-time solutions to Problem 1 with a high-level overview of the PTR algorithm, more generally referred to as successive convexification.[39,45,47] Figure 1 illustrates the major steps in the PTR algorithm. An initialization step is run first to define the initial solution guess (which may be very coarse), set the scaling matrices to improve the optimization numerical properties, and pre-parse fixed portions of the problem. Section II.B provides details on each of these aspects. Next, the main PTR loop is entered and the initial guess becomes the first reference solution. Throughout the paper, we make reference to "solver" and "PTR" iterations. Their meaning is highlighted in Figure 1, namely, the former are iterations performed internally by the SOCP solver, while the latter are the outer iterations around the main PTR loop defined by the convexification/test/solve steps.

Algorithm 1 Initilization step.

Input: Current vehicle state and desired terminal boundary conditions.

1: Compute initial guess for states, controls, and parameters using (1).
2: Compute S_x; S_u; S_p and c_x; c_u; c_p using (3). . Scaling matrices
3: Populate the A; b; c matrices with all constant non-zero entries. . Pre-parse step

The convexi cation step returns a convex approximation of Problem 1 about the current reference solution, wherein the dynamics have been approximated as a discrete linear time-varying system and the non-convex constraints have been linearized about the reference solution. xII.C provides details on both of these processes. The output of the convexi cation step allows us to check if the current reference solution satis es the desired feasibility and convergence requirements. This is done using the stopping criterion explained in xII.E. If passed, the algorithm terminates. Otherwise, the convex approximation is used to formulate an SOCP in the following general conic form:

$$\begin{aligned} \min_{z} \quad & c^{\top} z \\ \text{subject to:} \quad & Az = b; \\ & z \in \mathcal{C}_L \times \mathcal{C}_{Q_1} \times \cdots \times \mathcal{C}_{Q_m}; \end{aligned} \tag{Problem 2}$$

where $\mathcal{C}_L = \{w \in \mathbb{R}^l \mid w \geq 0\}$ is a linear cone of dimension l, and each $\mathcal{C}_{Q_i} = \{(w_0; w) \in \mathbb{R}^{d_i} \mid \|w\|_2 \leq w_0\}$ is a second order cone of dimension d_i. The problem data $A \in \mathbb{R}^{n_c \times n_z}$, $b \in \mathbb{R}^{n_c}$ and $c \in \mathbb{R}^{n_z}$ are then passed to the solver to compute a new reference solution. Note that careful parsing is required to express the optimization problem in the format of Problem 2. All previous work on successive convexi cation, and sequential convex programming in general, has relied on "modelling" interfaces (often called parsers) that automate the conversion.[39,42,47,54] However, these parsers introduce computational overhead, redundant constraints and dynamically allocated memory. Each of these characteristics are undesirable for real-time space ight implementations.[74] In this work, we o er a general methodology to "hand parse" the problem and remove these shortcomings by explicitly exploiting problem structure. Details are given in xII.D.

B. Initialization Step

This section details the initialization steps that are performed prior to calling the main PTR loop. The initialization step is summarized by Algorithm 1 and consists of three primary operations: generating the initial solution guess, computing the scaling matrices, and pre-parsing the problem data. These steps can all be done before calling the main PTR loop shown in Figure 1. If the initialization step is called at some time $t < t_0$, where t_0 is the time at which the guidance solution will begin to be executed by the vehicle, the initial state x_{ic} must be projected forward in time by an amount $t_0 - t$. The di erence $t_0 - t$ must (at least) account for the time spent computing the guidance solution. For the aforementioned test ights using the 3-DoF G-FOLD algorithm, this value was been chosen to be one second.[77]

1. Initial Solution Guess

Using the given initial state of the vehicle, we rst compute the state solution guess. The simplest way to initialize the state solution, and the one that is used here, is by using the straight line initialization method.[39] We take the given initial state of the vehicle, x_{ic}, and the desired boundary conditions, x_f, and compute a linear interpolation such that

$$x_{k;i} = \frac{N-k}{N-1} x_{ic;i} + \frac{k-1}{N-1} x_{f;i}, \quad i = 1, \ldots, n_x, \quad k \in \mathcal{N} := \{0, \ldots, N-1\}, \tag{1}$$

where N is the number of discrete nodes chosen to compute the solution (see xII.C). To initialize the control, we leverage the known pro le of the optimal solution whenever it is available. For the planar landing case study discussed in xIII, the optimal thrust is known to be bang-bang, and so the thrust input is initialized as such, with switching times guessed based on preliminary ground tests. For the constrained 6-DoF landing problem, the structure of the optimal thrust remains an open problem, and so we simply initialize with a thrust vector that opposes gravity. Any torque inputs are generally assumed to be zero. Note that neither the state nor control guess have to be feasible with respect to the dynamics or constraints. We have observed that di erent initial guesses can lead the algorithm to converge to di erent solutions,[42] and it should be clear that a more accurate initial guess will result in faster convergence.

With the exception of the final time, guesses for the parameter vector will always be application dependent, and thus no general statements can be made. However, for the final time specifically, we have observed that initializing the parameter t_f to be higher than the expected flight time produces slightly better convergence results. This is similar to an observation made for the 3-DoF problem in.[33] Dueri et al. found that as t_f approached the minimum feasible time of flight, more solver iterations were required to produce a solution. As t_f was increased, the solver had an easier time finding solutions. We have observed a similar trend here for more general landing problems, and this is discussed more in Sec. III.

2. Scaling Matrices

The scaling of solution variables can be done several ways. There is no general consensus on the best way to scale optimal control problems to produce numerically well-conditioned parameter optimization problems. While some authors argue that scaling (and balancing) the continuous time equations of motion is the most appropriate,[17,78] others argue that scaling the discrete parameter optimization problem is sufficient.[79,80] We have found the latter method to be acceptable in our implementations.[42] To this end we define the following affine transformations

$$x_k = S_x \hat{x}_k + c_x; \tag{2a}$$
$$u_k = S_u \hat{u}_k + c_u; \tag{2b}$$
$$p_k = S_p \hat{p}_k + c_p; \tag{2c}$$

where the S_x, S_u, S_p are diagonal matrices of commensurate dimension and c_x, c_u, c_p are vectors that center the state, control, and parameters. Throughout this paper, scaled quantities are referred to with the $\hat{\cdot}$ adornment. For the ith component of the state, control and parameter vectors, generically referred to using a_i, we can define two quantities: (i) the known range of the "true" component's value $[a_{i,\max}, a_{i,\min}]$ and (ii) an interval $[\hat{a}_{lb}, \hat{a}_{ub}]$ that we wish to scale each component of quantity a to. Using this information, we use

$$S_{a,ii} = \frac{a_{i,\max} - a_{i,\min}}{\hat{a}_{lb} - \hat{a}_{ub}}, \quad \text{and} \quad c_{a,i} = a_{i,\min} - S_{a,ii}\,\hat{a}_{lb}. \tag{3}$$

While the interval $[\hat{a}_{lb}, \hat{a}_{ub}]$ is in theory arbitrary, a judicious choice with respect to generating the standard form of Problem 2 is to use $[\hat{a}_{lb}, \hat{a}_{ub}] = [0, 1]$. This places the state, control, and parameter vectors into the linear cone by construction and eliminates the need to enforce the lower bound constraints explicitly. The computational savings afforded by this choice can be significant and are quantified in Theorem 3.

3. Pre-Parsing

Pre-parsing consists of initializing the matrix A, b and c used in Problem 2. There is a significant amount of structure to the non-convex problems that are solved using any successive convexification technique, and this structure is exploited to maximize the speed of the main PTR loop by performing as many computations as possible during the pre-parse step. For a fixed problem statement, the majority of the non-zero entries in these containers do not change across the PTR iterations, and the pre-parse step simply populates A, b, c with these constant non-zero entries.

For this step, an enumeration of the variables and constraints must be decided upon. A generic enumeration of the variable z and constraints is given in Figure 2. We define the block v_1 variables to be all those that appear in Problem 1 and/or contribute to the nonlinear equations of motion, block v_2 variables to be all linear slack variables added to write the problem in standard form, and block v_3 variables to be those used to form the trust region[a] in addition to any Second-Order Cone (SOC) slack variables added to write the problem in standard form. Similarly, we define block c_1 constraints to be those that represent the dynamics and boundary condition constraints, and define block c_2 constraints as all other constraints imposed. These definitions permit a block row/column decomposition of the A, b and c matrices that is used to guide the presentation in this paper. For PTR-type algorithms, the vector c can be fully populated during the pre-parse step, and only A and b will change across the PTR iterations. All block v_2 and v_3 slack variables added to write the convex approximation in the standard form of Problem 2 will have a corresponding entry in the A matrix of unit magnitude (-1) that appears in the row corresponding to the constraint the slack variable was added to. These are represented by the $-\mathbb{I}$ (for linear slack variables) and the $-I^{[A]}$ (for SOC slack variables) blocks shown in Figure 2. Since all non-zero entries

[a] The trust region is often implemented as a second order cone, though this is not necessary. Hence block v_3 may contain both linear and second-order cone variables.

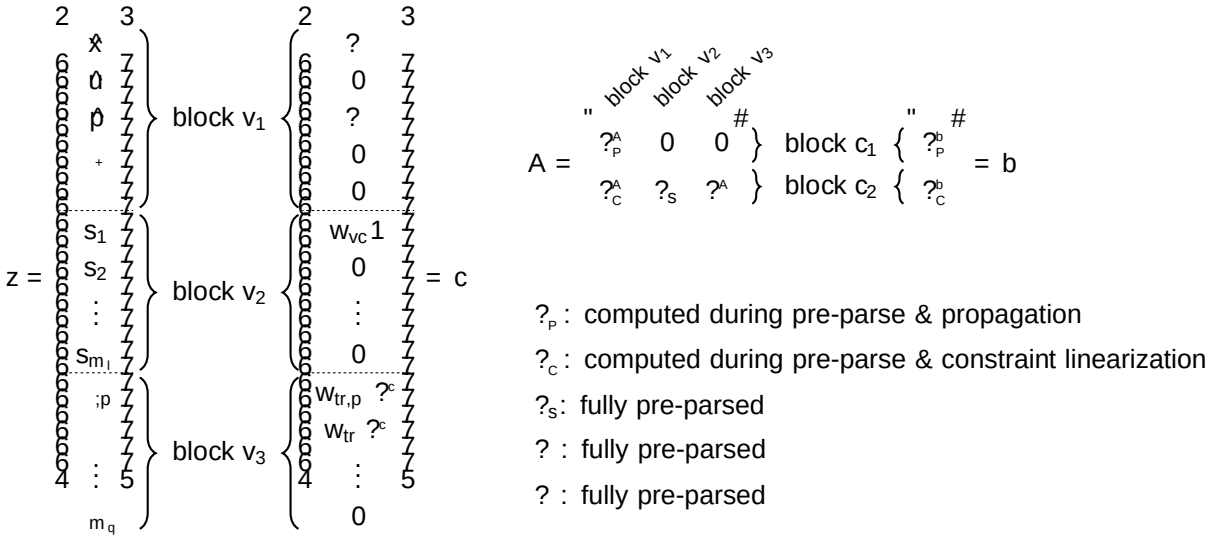

Figure 2: Generic enumeration of the variables and constraints for the standard form of a trajectory optimization problem. The constraint block c_1 corresponds to the dynamics and boundary conditions, while constraint block c_2 corresponds to all path constraints enforced as inequalities. Block v_1 variables are used either directly in Problem 1 or to impose the dynamics and boundary conditions. Block v_2 variables represent linear slack variables of arbitrary dimension, and there are m_l such variables. Block v_3 variables represent the trust region implementation and SOC variables of arbitrary dimension, and there are m_q such variables.

are 1, and occur in user-speci ed rows and columns, the entire $?_S$ and $?^A$ blocks can be populated during the pre-parse step and do not change throughout the iterations.

The $?_P$ blocks correspond to the equations of motion and boundary conditions, and can be partially populated during the pre-parsing step. The remaining portions can only be populated after the convexi cation step. The $?_C$ blocks corresponds to the path constraints enforced as inequality constraints. These blocks can also only be partially populated during the pre-parse step, and the remaining portions added only after the convexi cation step. The degree to which the $?_P$ and $?_C$ blocks can be pre-populated is application dependent and will vary.

C. Convexi cation Step

The convexi cation step is responsible for computing a convex approximation to the nonlinear equations of motion and all non-convex constraints g_i for $i \in I_{ncvx}$. This is achieved using two separate steps that can be performed in parallel: propagation and constraint linearization. Note that in implementation, all matrix computations are performed using 1D \ at" arrays. This avoids both the use of 3D arrays to provide a temporal index, and maintaining large 2D arrays that contain mostly zeros.[59]

1. Propagation

To enforce the dynamic equations of motion as a set of convex equality constraints, they must be mapped to discrete-time a ne functions of the state, control, and parameters. The technique to do this has been outlined previously in[39,42,59] and so only the key steps required to discuss the implications for real-time implementation are included here. An archetypal algorithm is given in Algorithm 2 to provide an outline of the steps. Using the current reference solution, the nonlinear dynamics are rst normalized with respect to time, and then expanded in a rst order Taylor series about the reference triplet $\{x(t), u(t), p\}$ to yield:

$$\underline{x}(\tau) = A(\tau)x(\tau) + B(\tau)u(\tau) + S(\tau)p + R(\tau), \tag{4}$$

where $\tau \in [0, 1]$ is a normalized time variable and $A(\tau), B(\tau), S(\tau)$ are the partial derivatives of f with respect to state, control, and the parameters respectively evaluated along the reference trajectory. Based on the results presented in,[59] we recommend using an a ne interpolation of the control to discretize (4). To do so, we select

N time nodes between the current initial and final times

$$t_0 = \underbrace{t_0 < t_1 < \cdots < t_{N-1}}_{\text{real time nodes}} = t_f \qquad 0 = \underbrace{\tau_0 < \tau_1 < \cdots < \tau_{N-1}}_{\text{propagation time nodes}} = 1: \tag{5}$$

The temporal nodes do not need to be distributed equally between their end points. We now use the discrete variable $k \in \mathbb{N} := \{0, \ldots, N-1\}$ to index the state and control vectors. The control vector is assumed to vary linearly between the values of u_k and u_{k+1} at each node $k \in \mathbb{N}$ producing the relation

$$u(\tau) = \alpha_k u_k + \beta_k^+ u_{k+1}, \qquad \tau \in [\tau_k, \tau_{k+1}], \tag{6}$$

where $\alpha_k = \frac{\tau_{k+1} - \tau}{\tau_{k+1} - \tau_k}$ and $\beta_k^+ = 1 - \alpha_k$. The exact discretization of (4) can then be computed as

$$x_{k+1} = A_k x_k + B_k u_k + B_k^+ u_{k+1} + S_k p + R_k, \quad k \in \{0, \ldots, N-2\}, \tag{7}$$

where for each $k \in \{0, \ldots, N-2\}$,

$$A_k := \Phi(\tau_{k+1}, \tau_k); \tag{8a}$$

$$B_k := A_k \int_{\tau_k}^{\tau_{k+1}} \Phi^{-1}(\tau, \tau_k) \alpha_k(\tau) B(\tau) d\tau; \tag{8b}$$

$$B_k^+ := A_k \int_{\tau_k}^{\tau_{k+1}} \Phi^{-1}(\tau, \tau_k) \beta_k^+(\tau) B(\tau) d\tau; \tag{8c}$$

$$S_k := A_k \int_{\tau_k}^{\tau_{k+1}} \Phi^{-1}(\tau, \tau_k) S(\tau) d\tau; \tag{8d}$$

$$R_k := A_k \int_{\tau_k}^{\tau_{k+1}} \Phi^{-1}(\tau, \tau_k) R(\tau) d\tau: \tag{8e}$$

where $\Phi(\tau, \tau_k)$ is the zero input state transition matrix for the linear time varying system (4).

In implementation, we must compute $N-2$ sets of the matrices given in (8). Each set of matrices is computed by numerically integrating the integrands in (8) along with the state vector. We use a fixed-step RK4 integration routine, with N_{sub} points. A flat 1D array of dimension $n_x(n_x + 2n_u + 3)$ is needed for this operation. For each $k = 0, \ldots, N-2$ we integrate the following differential equation over the interval $[\tau_k, \tau_{k+1}]$ using N_{sub} nodes

$$\dot{P}(\tau) = \begin{bmatrix} f(P_x(\tau), u(\tau), p) \\ A(\tau) P(\tau) \\ P(\tau)^{-1} \alpha_k(\tau) B(\tau) \\ P(\tau)^{-1} \beta_k^+(\tau) B(\tau) \\ P(\tau)^{-1} S(\tau) \\ P(\tau)^{-1} R(\tau) \end{bmatrix}; \quad P(\tau_k) = \begin{bmatrix} x_k \\ \text{flat}(I_{n_x}) \\ 0_{n_x n_u \times 1} \\ 0_{n_x n_u \times 1} \\ 0_{n_x \times 1} \\ 0_{n_x \times 1} \end{bmatrix}; \quad \text{where } P(\tau) = \begin{bmatrix} P_x(\tau) \\ P(\tau) \\ P_B(\tau) \\ P_{B^+}(\tau) \\ P_S(\tau) \\ P_R(\tau) \end{bmatrix}: \tag{9}$$

Here, the $\text{flat}(\cdot)$ operation maps a 2D array to a 1D array using a column major representation.[b] Note that the initial condition $P(\tau_k)$ is reset at each iteration to the value of the reference state trajectory at the kth node. The benefits of this resetting strategy are significant and have been discussed in previous work.[39,42,59] The derivative in (9) must be evaluated four times per subinterval, for a total of $4N_{sub}$ times per propagation step. Our results suggest that the PTR algorithm's convergence behaviour and quality of the final solution is not strongly dependent on the choice of N_{sub}, and a value of $N_{sub} \in [5, 15]$ is typically sufficient for rigid body PDG dynamics. To obtain the final values of the matrices in (8), we must multiply each of $P_B(\tau_{k+1})$ through $P_R(\tau_{k+1})$ by the value of $P(\tau_{k+1})$. The final matrices in (8) can either be stored as "flattened" vectors in an output data structure or placed directly into the A and b matrices used to construct the standard form SOCP.

Computing the Matrix Inverse One of the key steps that drives the computational complexity of the propagation step is the matrix inverse required to evaluate the derivative of $P(\tau)$ in (9). It is customary to use Gaussian elimination with partial pivoting to compute this inverse.[81] This involves computing an LU decomposition and inverting the two factors. For powered descent problems that use Cartesian variables to represent the state vector and have the mass variable in the first position, we have observed that the entries

[b] In the code, no explicit conversions between 1D and 2D arrays are needed, and so the use of column major representation is not in conflict with the C/C++ row major array storage.

below the diagonal in $P(\ )$ do not exceed 1 under the condition that the ratio $(\ )=m(\ )^2 < 1$, where $(\ )$ is the thrust magnitude and $m(\ )$ is the vehicle mass. Moreover, the equations of motion imply that the (1,1) entry in $P(\ )$ is constant and equal to 1. This means that partial pivoting may not be theoretically necessary to ensure numeric stability in this case. However, for powered descent problems stated using dual quaternions the same statement does not hold,[42] and partial pivoting must be retained in this case. To maintain a general solution method, we therefore retain partial pivoting in our implementations.

Computational Complexity For each $k = 0; : : : ; N \quad 2$, the propagation step can be performed in

$$O \quad 4N_{sub} \quad \frac{5}{3}n_x^3 + 2n_x^2 n_u + 6n_x^2$$

oating point operations. Powered descent problems typically have $n_x \ 2 \ [7; 15]$ and $n_u \ 2 \ [2; 6]$. As demonstrated in xIII, the propagation step can be performed in a near-negligible amount of time if it is implemented properly.

Measuring Dynamic Feasibility One of the advantages of the propagation step is the ability to estimate the dynamic feasibility of the current reference solution $f x_k ; u_k ; p g_{k2N}$. What we mean by this is that, given the nonlinear dynamics of Problem 1, if the discrete control and parameter solution $f u_k ; p g_{k2N}$ are used to integrate the nonlinear equations of motion using the interpolation scheme (6), then the resulting (open-loop) state will pass through the discrete points $f x_k g_{k2N}$ to within some user de ned tolerance. This notion of dynamic feasibility, coupled with an analogous measure of feasibility with respect to the constraints, provides a simple measure of the quality of a solution before it is executed.

To measure dynamic feasibility, we leverage the availability of the propagated state $P_x(\ _{k+1})$. Ideally, this vector would match the value predicted by the reference trajectory, x_{k+1}. We compute the defect at the $(k+1)$th node using

$$_{k+1} \ = \ kP_x(\ _{k+1}) \quad x_{k+1} k_2: \tag{10}$$

If each $_{k+1}$ is less than a prescribed tolerance $_{feasible}$, then we say that the reference trajectory is dynamically feasible. Typically, choosing $_{feasible} \quad 10^{\ 2}$ provides acceptable results, but this value is dependent on the function f that describes the dynamic equations of motion.

2. Constraint Linearization

The next part of the convexi cation step is to compute a linear approximation to all non-convex path constraints. These are constraints of the form

$$g_i(x; u; p) \quad 0; \quad i \ 2 \ I_{ncvx}; \tag{11}$$

for some functions g_i that are di erentiable almost everywhere. This step is performed as part of the convexi cation step so that it always precedes the parsing step discussed in xII.D. Using the reference trajectory $f x_k ; u_k ; p g_{k2N}$ we compute the linearization of (11) to be

$$g_i(x_k; u_k; p) + r \ g_i \ x_k; u_k; p \ > \begin{bmatrix} x_k & x_k \\ u_k & u_k \\ p & p \end{bmatrix} \quad 0; \quad k \ 2 \ N; \tag{12}$$

where $r \ g_i \ x_k; u_k; p \ 2 \ R^{(n_x+n_u+n_p) \ 1}$ is gradient of g_i evaluated at the kth reference node. Each non-convex constraint is then approximated by N linear inequalities. Note that similar to the storage of output $:A_d$ in the propagation step of Algorithm 2, we store the data for the linear approximation of the ith non-convex constraint in a single 2D array where the rst entry in the kth column is the value of

$$g_i(x_k; u_k; p) \quad r \quad g_i \ x_k; u_k; p \ > \begin{bmatrix} x_k \\ u_k \\ p \end{bmatrix}; \tag{13}$$

and the remaining rows are used to store the value of the gradient of g_i evaluated at the reference solution's kth node. Note that if a constraint g_i a ects only a subset of state, control, or parameter, then the size of this 2D array is shrunk accordingly.

As $g_i(x_k; u_k; p)$ is computed for each $i \ 2 \ I_{ncvx}$ and $k \ 2 \ N$, the feasibility with respect to each constraint at the solution nodes can be checked. If any constraints are found to be violated, then the value of output.feasible

Algorithm 2 The convexification step. All matrix operations are performed as 1D array operations.

Input: Reference trajectory $\{x_k, u_k, p\}$ for $k \in 0:N$ and feasibility tolerance $\varepsilon_{feasible}$.

Output: Data structure, output, containing the discretized dynamics matrices, linearized constraint data, and feasibility indicator.

```
function convexify
    Set output.feasible = true
    Call the propagate function
    Call the linearize function
end function

function propagate
    P_0 = vec(I_{n_x}) 0_{1×n_x n_u} 0_{1×n_x n_u} 0_{1×n_x} 0_{1×n_x}^T.
    for k = 0, ..., N − 2 do
        P = x_k^T P_0^T                          . reset using reference state, not using the previous P_x
        h = (t_{k+1} − t_k)/N_sub                . the rk4 step size using N_sub steps
        Call rk4 on the derivs function to update P
        Compute the defect ν_{k+1} using (10)
        if ν_{k+1} > ε_feasible then
            output.feasible = false
        end if
        Set outputs:
            output: A_d[:][k] = P                . [:] means "all rows"
            output: B_{d,−}[:][k] = P ∘ P_B
            output: B_{d,+}[:][k] = P ∘ P_{B+}
            output: S[:][k] = P ∘ P_S
            output: R[:][k] = P ∘ P_R
    end for
end function

function linearize
    for i ∈ I_ncvx do
        for k = 0, ..., N − 1 do
            output: G_i[1:][k] = ∇g_i(x_k, u_k, p)   . [1:] means "all rows starting from 1"
            output: G_i[0][k] = g_i(x_k, u_k, p) − ∇g_i(x_k, u_k, p)^T [x_k^T, u_k^T, p^T]^T
            if g_i(x_k, u_k, p) > ε_feasible then
                output.feasible = false
            end if
        end for
    end for
end function

function derivs(τ, P)
    Compute P^{-1} using an LU decomposition with partial pivoting
    Compute α_k(τ) and α_k^+(τ) and interpolate u(τ) using (6)
    Compute partials:
        f = f(P_x, u(τ), p)
        A = ∇_x f(P_x, u(τ), p)
        B = ∇_u f(P_x, u(τ), p)
        S = ∇_p f(P_x, u(τ), p)
        R = AP_x + Bu(τ)
    Set outputs:
        P_{x,−} = f
        P_− = AP
        P_{B,−} = α_k(τ)B
        P_{B+} = α_k^+(τ)B
        P_S = S
        P_R = R
end function
```

is set to false . If no constraints are violated, this does not necessarily imply that the trajectory will satisfy the constraints at times between the solution nodes, a phenomenon referred to as *clipping*. This is therefore a necessary but not sufficient assessment of feasibility with respect to path constraints. In contrast, the previously defined assessment of dynamic feasibility is both necessary and sufficient.

D. Solve Step

The convexification step computes all of the data that is needed to formulate a convex approximation to Problem 1 and to obtain a new reference solution. Prior to calling the solver, the remaining data that was not placed in $\hat{A}$ and $\hat{b}$ during the pre-parsing step must be added, a function that we refer to as *parsing*. Upon completion of the parsing step, we have fully defined a problem in the format of Problem 2, and the solver is then called. The PTR method solves each convex approximation to full optimality, rather than settling for a sub-optimal solution and iterating again. Since calling the solver tends to dominate the algorithm runtime (even for a small number of solver iterations), reducing the number of calls to the solver is paramount to obtaining fast guidance trajectories.

1. Parsing

The parsing step adds the data from the convexification step to $\hat{A}$ and $\hat{b}$. These data form the remaining portions of $\mathcal{P}_P$ and $\mathcal{P}_C$ blocks shown in Figure 2. Since the standard form solution vector z contains the scaled state, control and parameters as 1D stacked vectors, the discretized dynamics (7) are written in block form as

$$\hat{R} = \hat{A}x + \hat{B}u + \hat{S}p + \nu_+ \quad ; \tag{14}$$

where $\nu = \nu_+$ is a virtual control term expressed using two variables in the linear cone[39,43–45] and

$$\hat{A} = \begin{bmatrix} A_0 S_x & S_x & 0 & \cdots & 0 \\ 0 & A_1 S_x & S_x & & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & A_{N-2} S_x & S_x \end{bmatrix}; \tag{15a}$$

$$\hat{B} = \begin{bmatrix} B_0 S_u & B_0^+ S_u & 0 & \cdots & 0 \\ 0 & B_1 S_u & B_1^+ S_u & & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & B_{N-2} S_u & B_{N-2}^+ S_u \end{bmatrix}; \tag{15b}$$

$$\hat{S} = \begin{bmatrix} S_0 S_p \\ S_1 S_p \\ \vdots \\ S_{N-2} S_p \end{bmatrix}; \tag{15c}$$

$$\hat{R} = \begin{bmatrix} c_x - R_0 - A_0 c_x - B_0 c_u - B_0^+ c_u - S_0 c_p \\ \vdots \\ c_x - R_{N-2} - A_{N-2} c_x - B_{N-2} c_u - B_{N-2}^+ c_u - S_{N-2} c_p \end{bmatrix}. \tag{15d}$$

If $H_0 \in \mathbb{R}^{n_0 \times n_x}$ and $H_f \in \mathbb{R}^{n_f \times n_x}$ are the matrices in Problem 1 corresponding to the boundary conditions, the $\mathcal{P}_P$ block can then be filled in as

$$\mathcal{P}_P^A = \begin{bmatrix} H_0 S_x & 0_{n_0 \times n_x(N-1)} & 0_{n_0 \times n_u N} & 0_{n_0 \times n_p} & 0_{n_0 \times n_x N} & 0_{n_0 \times n_x N} \\ \hat{A} & & \hat{B} & \hat{S} & I_{n_x N} & I_{n_x N} \\ 0_{n_f \times n_x(N-1)} & H_f S_x & 0_{n_f \times n_u N} & 0_{n_f \times n_p} & 0_{n_f \times n_x N} & 0_{n_f \times n_x N} \end{bmatrix} \quad \mathcal{P}_P^b = \begin{bmatrix} x_0 - H_o c_x \\ \hat{R} \\ x_f - H_f c_x \end{bmatrix}. \tag{16}$$

The entries in (15) and (16) that are independent of the matrices computed in (8) will have been populated during the pre-parse step outlined in §II.B.3.

Remark 1 The notation adopted in some previous work has used an identity block in the upper left corner of the $\hat{A}$ matrix in (15a).[59] Including this block in addition to the initial condition can create an A matrix that does not have full row rank and lead to potential numerical issues in the solver.

While every trajectory optimization problem will have the same basic structure for the $\mathcal{P}$ blocks, the $\mathcal{C}$ blocks are heavily application dependent. A trajectory optimization problem stated with two different sets of constraints will have the same $\mathcal{P}$ blocks, but will have different $\mathcal{C}$ blocks. As such, we shall only treat two generic classes of path constraints here: convex linear or second-order cone constraints and general non-convex constraints. Since the non-convex constraints are simply linearized, they are effectively the same as the linear constraints. Hence we need only consider linear constraints and second-order cone constraints.

Linear Constraints There are several linear constraints that will form part of all successive convexification algorithms. For example, the use of 1-norm penalty functions for the virtual control and box-type constraints on the state, control, and parameter vectors each produce linear constraints. In addition, there may be several other linear constraints that arise either directly in Problem 1 or from linearizing non-convex constraints. To illustrate how these are parsed, consider a linear state constraint of the form $g_k^\top x_k \leq h_k$ for each $k \in \mathbb{N}$ and some $g_k \in \mathbb{R}^{n_x}$ and $h_k \in \mathbb{R}$. This constraint is rewritten in standard form by introducing the slack variable $s \in \mathbb{R}_+^N$ and writing

$$G^\top x + s = h, \quad s \geq 0, \tag{17}$$

where

$$G := \begin{bmatrix} g_0^\top & 0 & \cdots & 0 \\ 0 & g_1^\top & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g_{N-1}^\top \end{bmatrix} \in \mathbb{R}^{N \times n_x N} \quad \text{and} \quad h := \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{N-1} \end{bmatrix} \in \mathbb{R}^N. \tag{18}$$

The matrix G is then added to the $\mathcal{C}_c^A$ block in the appropriate location based on the definition of block v_1 in z and block c_2 in A (see Figure 2). Similarly, h is added to $\mathcal{C}_c^b$ in the corresponding rows of block c_2. The same procedure applies to linear control and parameter constraints.

Remark 2 If the linear constraint comes from a non-convex path constraint, then G and h will in general be functions of the reference trajectory $\{x_k, u_k, p\}_{k \in \mathbb{N}}$, and must be re-evaluated during each convexification step when a new reference is available. The same can be true even for constraints that are linear in their original form in Problem 1. As an example of the latter, consider a trust region imposed using the ℓ_1-norm, which is equivalent to two linear constraints on each of the state, control, and parameter vectors. For each linear constraint, h must be re-evaluated at each PTR iteration as the reference solution changes.

Theorem 3 To impose a linear path constraint on the state or control in Problem 1 requires adding N slack variables to Problem 2 and adding N rows to both A and b.

As mentioned in §II.B.2, the state, control, and parameter vectors are scaled so that the variables being solved for are already in the linear cone $\mathbb{R}_+$. Hence no additional constraints are needed to enforce the scaled variable's lower bound. Rather, they are enforced implicitly by the solver. Theorem 3 tells us that scaling the solution vectors in this way saves us $n_x N$ slack variables and $n_x N$ rows in the matrices A and b. Even for the simplest powered descent guidance problems, which have $n_x = 7$, this represents a savings of at least 70 variables and rows of A and b for typical implementations.

Second-Order Cone Constraints Not all successive convexification algorithms designed for trajectory optimization will make use of second order cone constraints. For powered descent problems specifically, however, there are several constraints that are naturally expressed in this form, such as an upper bound on thrust magnitude, thrust pointing constraints, approach cone constraints, and more. The ability to obtain trajectories that explicitly enforce such constraints is important for space flight applications. Moreover, we have found that 2-norm-based trust regions are an efficient method to guide the convergence process for the PTR algorithm. To illustrate how these constraints are handled during the parsing step, consider a generic second order cone path constraint imposed on the state vector of the form $\|S_q x_k + c_q\|_2 \leq g_q^\top x_k + h_q$, for some $S_q \in \mathbb{R}^{d \times n_x}$, $c_q \in \mathbb{R}^d$, $g_q \in \mathbb{R}^{n_x}$ and $h_q \in \mathbb{R}$ and each $k \in \mathbb{N}$. Note that for the trajectory optimization problems considered here, it is rare for the data describing a second order cone constraint to vary in time (i.e., change with k). This constraint is rewritten in standard form by introducing three slack variables $\nu_k \in \mathbb{R}^d$, $\rho_k \in \mathbb{R}_+$ and $s_k \in \mathbb{R}_+$ for each $k \in \mathbb{N}$ and writing

$$\nu_k = S_q x_k + c_q, \quad \rho_k + s_k = g_q^\top x_k + h_q \quad \Big) \quad \begin{bmatrix} \rho_k \\ \nu_k \end{bmatrix} \in \mathcal{C}_{Q_{d+1}}. \tag{19}$$

Algorithm 3 The solve step.

Input: Matrices A, b, c and the output data structure of the convexification step.
Output: The data structure output with the reference trajectory overwritten.

```
function solve_socp
    Call parse to update A, b, c
    z = bsocp(A, b, c)                    . Call solver BSOCP with pre-defined tolerance values
    Compute output.delta_x using (21)
    Set outputs:                          . x̂, û, p̂ are retrieved from z.
        output.x = (I_N ⊗ S_x) x̂ + (1_N ⊗ c_x)
        output.u = (I_N ⊗ S_u) û + (1_N ⊗ c_u)
        output.p = S_p p̂ + c_p
end function
```

The expressions in (19) now form two linear constraints, and can be written in standard form as follows. First, define a vector $\hat{z} := \hat{x}_0^\top \hat{u}_0^\top \cdots \hat{x}_{N-1}^\top \hat{u}_{N-1}^\top \hat{x}_N^\top \in \mathbb{R}^{N(d+1)}$. Then,

$$I_N \otimes S_q \hat{x} + I_N \otimes [0\ I_d] = 1_N \otimes c_q \quad \text{and} \quad I_N \otimes g_q \hat{x} + I_N \otimes [1\ 0_{1\times d}] + I_N s = h_q 1_N \tag{20}$$

where $\otimes$ is the Kronecker product, are equivalent to (19) with the additional distinction that each pair $(\eta_k, \zeta_k) \in \mathcal{C}_{Q_{d+1}}$.

Remark 4 The representation of the SOC constraint (20) can be entirely added to the matrices A and b during the pre-parse step, unless it represents a trust region constraint. For a trust region constraint stated as an SOC constraint, the matrices S_q, g_q, h_q are constant and may be pre-parsed, while c_q changes across the iterations and is part of the $\begin{smallmatrix} b \\ c \end{smallmatrix}$ block.

Theorem 5 To impose a $d+1$ dimensional second order cone path constraint in Problem 1 requires adding at most $N(d+2)$ variables to Problem 2 and an additional $N(d+1)$ rows to both A and b.

The at most qualifier in Theorem 5 is due to the structure of variable-width trust regions in PTR methods, where the right hand side of the SOC path constraint is already a solution variable and an additional slack variable does not need to be added to the problem. Contrasting Theorem 5 with Theorem 3 allows us to quantify the possible difference in problem size available by replacing second order cone variables with linear variables and vice versa.

2. Calling the Solver

After the parsing step, the data A, b, c are passed to the BSOCP solver that in turn provides an optimal solution, see Algorithm 3. The BSOCP solver uses interior point methods, is written in C, and was designed with the express goal of solving powered descent guidance algorithms for flight implementations. All technical details relating to the solver are summarized in [64] and we do not attempt to cover them here. The solver's capabilities were demonstrated during the aforementioned G-FOLD flight test campaign as part of the Autonomous Ascent and Descent Powered-Flight Testbed (ADAPT) project led by NASA's Jet Propulsion Laboratory in collaboration with Masten Space Systems.[61–63, 77]

The solver is called by passing a data structure that contains A, b and c, along with the sizes of each cone $\mathcal{C}_L \times \mathcal{C}_{Q_1} \times \cdots \times \mathcal{C}_{Q_{m_q}}$ and the numbers l, m_q, where l is the dimension of the cone $\mathcal{C}_L$. The parameters used to define numerical tolerances and convergence, listed in [64, Table A.1], must be tailored to the specific problem instance being implemented. If z is the solution returned by the solver, we note that not all of z needs to be retained. The new reference trajectory $\{x_k, u_k, p\}_{k\in\mathbb{N}}$ can be extracted by unscaling the entries of z that correspond to $\hat{x}, \hat{u}, \hat{p}$. Prior to terminating the solve step, we leverage the fact that the scaled state vector $\hat{x}$ is already available on the stack and compute the maximum temporal difference for later use:

$$\Delta_x := \max_{k\in\mathbb{N}} \| \hat{x}_k - S_x^{-1}(x_k - c_x) \|_1 : \tag{21}$$

Table 1: Possible exit conditions for the successive convexi cation algorithm.

Converged	Feasible	Description
3	3	Converged and feasible.
7	3	Reached maximum PTR iterations before $_x <$ $_{solve}$: Safe but sub-optimal.
3	7	Converged but not feasible. Do not use.
7	7	Not converged and not feasible. Do not use.

E. Stopping Criteria

The nal component of the algorithm is the stopping criteria, shown as the \Test" block in Figure 1. The possible exit conditions are given in Table 1. As noted in[42] we have found for powered descent problems that relatively large changes in the control variables (in particular the thrust) can have little impact on the state variables. Moreover, since the cost function is assumed to be in Mayer form, small changes to the state vector imply small changes in the optimal cost. The converse is not necessarily true, and therefore we use a state-based convergence tolerance that is based on $_x$ computed in (21). Once $_x <$ $_{solve}$, the iterations are terminated provided that output.feasible is true. Note that a trajectory cannot be dynamically feasible unless the virtual control norm is small enough to allow this, and hence checkingoutput.feasible is a more robust exit criterion than checking the norm of the virtual control, since it also includes checking the feasibility with respect to non-convex constraints at each temporal node.

III. Case Study: Planar Landing Problem

To illustrate the design methodology for real-time implementable algorithms, we present a case study of the fuel-optimal planar landing problem introduced in.[75] This problem has nonlinear dynamics that serve as the sole source of non-convexity in the problem. While non-convex constraints can be added and the design process remains unchanged, we retain the convex state and control constraints for this case study so that we may compare our results to the analytically optimal solution structure proven in.[75] Planar landing problems have $n_x = 7$ states, $n_u = 2$ controls and $n_p = 1$ parameter (the nal time) for which to solve. The state vector, $x(t) \in \mathbb{R}^{n_x}$, is comprised of the mass $m(t) \in \mathbb{R}_{++}$, the inertial position $r_I(t) \in \mathbb{R}^2$, the inertial velocity, $v_I(t) \in \mathbb{R}^2$, a single attitude variable $(t) \in \mathbb{R}$ and the angular velocity $!(t) \in \mathbb{R}$. The control, $u(t) \in \mathbb{R}^{n_u}$, is assumed to come from a body- xed rocket engine capable of generating variable thrust $(t) \in \mathbb{R}_{++}$ and a separate actuation mechanism capable of independently providing a torque $(t) \in \mathbb{R}$ (e.g., an RCS system, grid ns, etc.). The equations of motion are given for $t \in [t_0; t_f]$ by

$$\dot{m}(t) = \quad (t); \quad \dot{r}_I(t) = v_I(t); \quad \dot{v}_I(t) = \frac{(t)}{m(t)}d(t) + g_I; \quad \dot{}(t) = !(t); \quad \dot{!}(t) = \frac{(t)}{J}; \tag{22}$$

where is the reciprocal of the e ective exhaust velocity, $d(t) := \begin{bmatrix} \sin(t) & \cos(t) \end{bmatrix}^\top$, g_I is the constant inertial gravitational acceleration, and J is the constant inertia about the body axis orthogonal to the landing plane. The initial and nal states are constrained according to

$$H_0 x(t_0) = \begin{bmatrix} 1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1 \end{bmatrix} x(t_0) = \begin{bmatrix} m_{ic}\\ r_{I;ic}\\ v_{I;ic}\\ !_{ic} \end{bmatrix}; \quad H_f x(t_f) = \begin{bmatrix} 0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1 \end{bmatrix} x(t_f) = \begin{bmatrix} r_{I;f}\\ v_{I;f}\\ _f\\ !_f \end{bmatrix} \tag{23}$$

where $x_{ic} \in \mathbb{R}^{n_0}$ and $x_f \in \mathbb{R}^{n_f}$ represent the right hand sides of the prescribed boundary conditions.

For this problem, since the control inputs are scalar and decoupled, we formulate bounds on their magnitude as the box constraints

$$u_{min} \quad u(t) \quad u_{max} \qquad \begin{cases} _{min} \quad (t) \quad _{max}; \\ _{max} \quad (t) \quad _{max}: \end{cases} \tag{24}$$

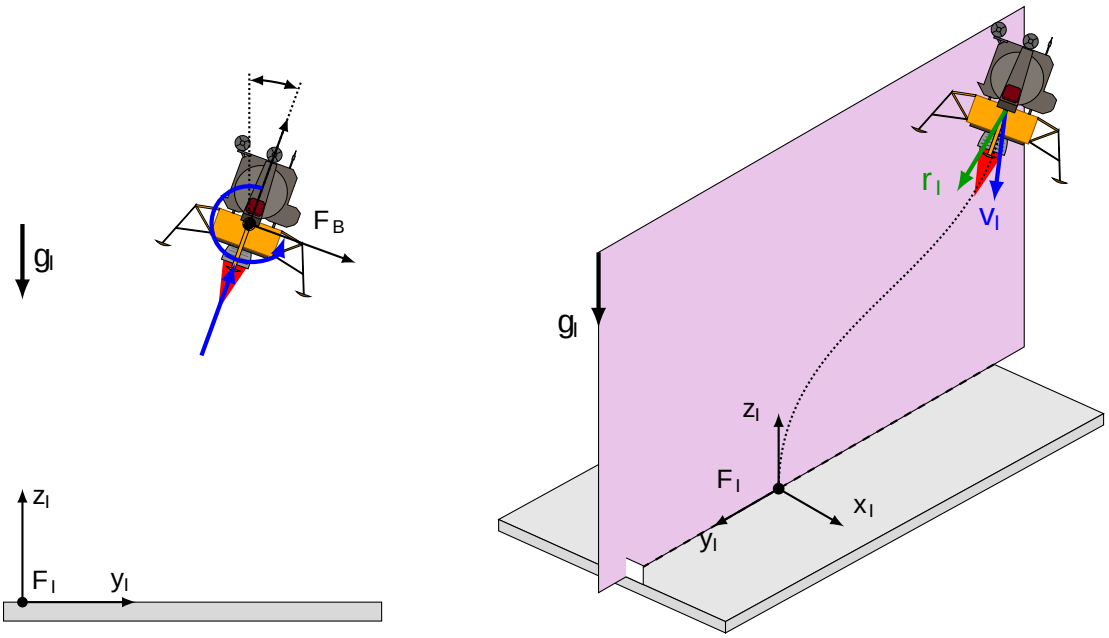

Figure 3: A depiction of the planar landing scenario used for the real-time powered descent guidance case study.

where $_{min}$; $_{max}$; $_{max}$ 2 R_{++} . For numerical solutions, we bound the state variables using

$$x_{min} \quad x(t) \quad x_{max}; \tag{25}$$

for some x_{min} ; x_{max} 2 R^{n_x} . In this case study, we take x_{min} and x_{max} to be just large enough so that none of the bounds are active during the descent, thereby using their numerical values for scaling purposes only.

The minimum fuel cost function can be expressed in Mayer form by using the nal mass of the vehicle. Maximizing the nal mass is equivalent to minimizing the fuel consumed, and so we take $`^{>} x(t_f) := m(t_f)$ as the cost. The continuous time optimal control problem that we wish to solve is given by

$$\begin{aligned}
\min_{t_f\,;u(\,)} \quad & `^{>} x(t_f) \\
\text{subject to:} \quad & \underline{x} = f(x;u;t_f); && (22)\\
& H_0 x(t_0) = x_{ic} \quad H_f x(t_f) = x_f; && (23)\\
& u_{min} \quad u(t) \quad u_{max}; && (24)\\
& x_{min} \quad x(t) \quad x_{max}: && (25)
\end{aligned} \qquad \text{(Problem 3)}$$

Following the procedures discussed in xII and,[39,41] for each PTR iteration we solve the convex approximation given by Problem 4, posed for clarity using unscaled variables. This case study uses N equally spaced temporal nodes. Note that we elect to use a 2-norm trust region that requires the use of several SOC variables. Use of a di erent norm, such as the in nity norm, would in contrast allow the trust region to be written using only linear constraints. However, Theorems 3 and 5 indicate that using the in nity norm trust region in place of the 2-norm trust region would in fact require $N(n_x + n_u \quad 1)$ additional variables and $N(n_x + n_u)$ additional rows in the A and b matrices. Empirical results for this particular problem reveal that this increase in problem size has a signi cant impact on the resulting solve times, even though the use of linear trust regions leads to a linear program. Over 100 trials using both methods on the exact same problem, the 2-norm trust region provided an average decrease of 11 ms in solver time per PTR iteration compared to the in nity norm trust region. This speed-up may be speci c to this particular problem. As such, we proceed by using the 2-norm trust region on

both the state and control.

$$
\begin{aligned}
\min_{x,u,p,\tau,\gamma,\rho} \quad & \ell^\top x_{N-1} + w_{vc}\|\tau\|_1 + w_{tr}\mathbf{1}_N^\top \rho + w_{tr,p}\rho_p \\
\text{subject to:} \quad & H_0 x_0 = x_{ic}, \\
& x_{k+1} = A_k x_k + B_k u_k + B_k^+ u_{k+1} + S_k p + R_k + \tau_k, \qquad k = 0,\ldots,N-2 \\
& H_f x_{N-1} = x_f, \\
& x_{min} - \rho_k \le x_k \le x_{max} + \rho_k, \qquad k = 0,\ldots,N-1 \\
& u_{min} - \rho_k \le u_k \le u_{max} + \rho_k, \qquad k = 0,\ldots,N-1 \\
& p_{min} - \rho_p \le p \le p_{max} + \rho_p, \\
& \|p - p_j\|_2 \le \rho_p, \\
& \|x_k - x_{k,2}\| + \|u_k - u_{k,2}\|_2 \le \rho_k, \qquad k = 0,\ldots,N-1.
\end{aligned}
\tag{Problem 4}
$$

We then scale the solution variables in Problem 4 by substituting $x_k \leftarrow S_x\hat{x}_k + c_x$, $u_k \leftarrow S_u\hat{u}_k + c_u$ and $p \leftarrow S_p\hat{p} + c_p$ wherever they appear, except in the trust regions. Following [41, Remark IV.1], we scale the trust region constraints using

$$\|\hat{p} - S_p^{-1}(p_j - c_p)\|_2 \le \rho_p, \tag{26a}$$

$$\|\hat{x}_k - S_x^{-1}(x_k - c_x)\|_2 + \|\hat{u}_k - S_u^{-1}(u_k - c_u)\|_2 \le \rho_k, \quad k = 0,\ldots,N-1. \tag{26b}$$

Next, we introduce slack variables so that all problem variables reside either in a linear cone or in a second-order cone, and all constraints are expressed as equalities. This follows the general procedures outlined in [41, Appendix D.1]. Note that slack variables are also introduced to express the cost function as a linear function of the solution variables using equivalent problem transformations.[20] Each of these steps are straightforward, except the implementation of the 2-norm trust region, which we highlight here. The state and control trust region is implemented by introducing N variables $(\nu_{x,k}, \gamma_{x,k})$ and $(\nu_{u,k}, \gamma_{u,k})$ such that

$$\nu_{x,k} = \hat{x}_k - S_x^{-1}(x_k - c_x) \quad \text{and} \quad \nu_{u,k} = \hat{u}_k - S_u^{-1}(u_k - c_u), \tag{27}$$

whereby $\|\nu_{x,k}\|_2 \le \gamma_{x,k}$ and $\|\nu_{u,k}\|_2 \le \gamma_{u,k}$ are now equivalent to (26b). Note that since $\gamma_{x,k}$ and $\gamma_{u,k}$ are solution variables, we do not need to add another slack variable to this formulation as in (19). We can reconstruct the original trust region ρ_k if needed as $\rho_k = \gamma_{x,k} + \gamma_{u,k}$. Define then the two variables

$$\gamma_x := \begin{bmatrix} \gamma_{x,0} \\ \gamma_{x,0} \\ \vdots \\ \gamma_{x,N-1} \\ \gamma_{x,N-1} \end{bmatrix} \in \mathbb{R}^{N(n_x+1)}, \quad \text{and} \quad \gamma_u := \begin{bmatrix} \gamma_{u,0} \\ \gamma_{u,0} \\ \vdots \\ \gamma_{u,N-1} \\ \gamma_{u,N-1} \end{bmatrix} \in \mathbb{R}^{N(n_u+1)}. \tag{28}$$

The trust region constraint (26) can then be expressed as

$$\hat{p} - \nu_p + s_p = b_p, \qquad \hat{p} + \nu_p - s_p^0 = b_p, \tag{29a}$$

$$\hat{x} + H_{:,x}\nu_x = b_x, \qquad H_{:,x} := I_N \otimes 0_{n_x \times 1} \; I_{n_x}, \tag{29b}$$

$$\hat{u} + H_{:,u}\nu_u = b_u, \qquad H_{:,u} := I_N \otimes 0_{n_u \times 1} \; I_{n_u}. \tag{29c}$$

where, $s_p, s_p^0 \in \mathbb{R}_+$, $(\nu_{x,k}, \gamma_{x,k}) \in \mathcal{C}_{Q_{n_x+1}}$ and $(\nu_{u,k}, \gamma_{u,k}) \in \mathcal{C}_{Q_{n_u+1}}$ for each $k \in \mathbb{N}$ and

$$b_x = \begin{bmatrix} S_x^{-1}(x_0 - c_x) \\ \vdots \\ S_x^{-1}(x_{N-1} - c_x) \end{bmatrix}, \quad b_u = \begin{bmatrix} S_u^{-1}(u_0 - c_u) \\ \vdots \\ S_u^{-1}(u_{N-1} - c_u) \end{bmatrix}, \quad \text{and} \quad b_p = S_p^{-1}(p - c_p). \tag{30}$$

Note that since there is only a single parameter in this problem, the trust region $\|\hat{p} - b_p\|_2 \le \rho_p$ is simply a linear constraint and both $\hat{p}$ and ρ_p remain in the linear cone.

To write the convex Problem 4 in standard form, we follow Figure 2 to order the solution variables using the block v_1, v_2 and v_3 structure. Recall that block v_1 contains all variables used to write the (nonlinear) equations

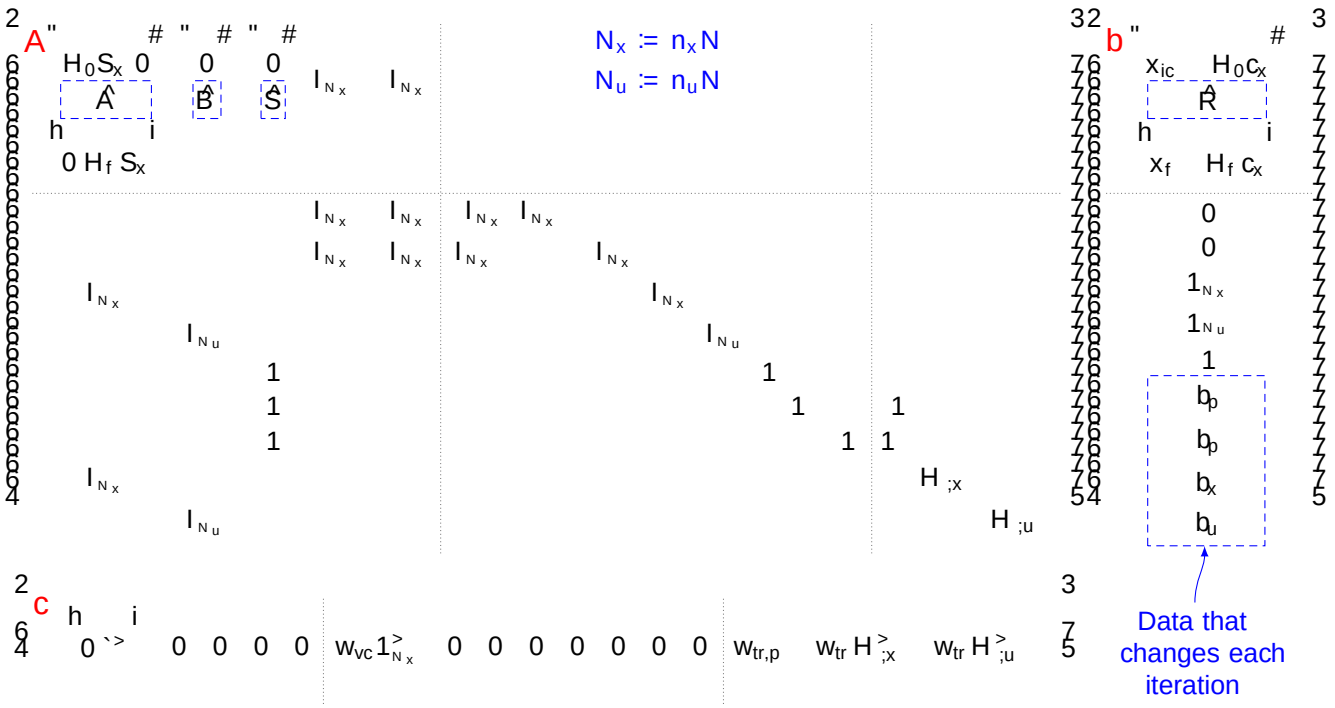

Figure 4: The A; b and c matrices for the planar landing case study. All empty blocks are zero, and the data that is not pre-parsed are highlighted by blue boxes.

of motion, block v_2 contains the linear slack variables added to write the problem in standard form, and block v_3 contains all variables relating to the trust region implementations and all SOC variables. The A, b and c matrices can then be written as shown in Figure 4, with the block structure indicated by the gray lines. Note that in Figure 4 we have de ned

$$H_{;x} := 1_N \quad 1\,0_{1\ n_x} \quad \text{and} \quad H_{;u} := 1_N \quad 1\,0_{1\ n_u} : \tag{31}$$

The problem dimensions can be computed as a function of the base units N; n_x; n_u and n_p to be:

Variables:	$N(8n_x + 3n_u + 2) + 5\ n_p,$
Equality constraints:	$N(5n_x + 2n_u) + 3\ n_p + (\ n_x \quad 1),$
Linear cones:	$N(7n_x + 2n_u) + 5\ n_p,$
Second order cones:	$N.$

A. Initialization Step

Using (1), the initial guess was computed by linearly interpolating between the vector $m_{ic}\ r_{I;ic}^{>}\ v_{I;ic}^{>}\ 0\ !_{ic}\ ^{>}$ and the vector $m_f\ r_{I;f}^{>}\ v_{I;f}^{>}\ _f\ !_f\ ^{>}$ where $m_f = \frac{m_{ic} + m_{min}}{2}$. Given that the optimal thrust solution is always on the boundary of the interval (24),[75] we initialize the thrust input channel using

$$_k = \begin{cases} _{min}; & k = 0; 1; : : : ; m \\ _{max}; & k = m + 1; : : : ; N: \end{cases} \tag{32}$$

The e ect of the parameter m on the computed solution for a xed N is studied in Figure 8. The optimal torque input does not necessarily activate either of its constraints (24), and so we initialize the torque to be zero at each temporal node.

The pre-parse step was implemented after writing down the matrices in Figure 4. Each non-zero entry that is not encompassed by a blue box was set during the pre-parse step of the initialization function, along with a

Table 2: Percentage of the data de ned during the pre-parse step for Problem 4 for a discretization density of N = 20. Note that A is 99:7% sparse.

	Total Size	Non-Zero Entries	De ned at Pre-parse	Pct. %
A	789 1285	2919	2410	7:3
b	789	506	193	38:1
c	1285	182	182	100

Table 3: Nominal parameters used in the planar landing case study.

Parameter	Value	Units
$t_{f;\,guess}$	8:0	s
m_{ic}	5:0	kg
$r_{I\,;ic}$	$[6:0\ 24:0]^{>}$	m
$v_{I\,;ic}$	$[\ 4:0\quad 2:0]^{>}$	m=s
$!_{ic}$	0	rad=s
$_0$	2 [;]	rad
$_{min}$	1:5	N
$_{max}$	0:1	Nm
I_{sp}	3	$s^{\ 1}$
w_{vc}	$10^{\ 2}$	
$w_{tr,p}$	$10^{\ 2}$	
$_{feasible}$	$10^{\ 2}$	

Parameter	Value	Units
$t_{f;\,max\,=min}$	4:0=12:0	s
m_{min}	2:0	kg
$r_{I\,;f}$	$[0\ 0]^{>}$	m
$v_{I\,;f}$	$[0\ 0]^{>}$	m=s
$!_{f}$	0	rad=s
$_{f}$	0	rad
$_{max}$	6:5	N
J	0:5	$kg\,m^2$
g	1	$m=s^2$
w_{tr}	$5^{\ 2}$	
N_{sub}	15	
$_{solve}$	$10^{\ 3}$	

subset of the data within the blue boxes. With regards to the planar landing problem considered here, Table 2 indicates the number of non-zero entries that can be populated at this time. Decoupling the pre-parse and parse steps allows signi cant computational savings, since only about 800 non-zero values need to be updated during each SCP iteration of the successive convexi cation algorithm. Implementations that use a generic parser must construct the entire A, b and c matrices each iteration and do not leverage the fact that the same optimization problem is being solved repeatedly.

B. Performance Results

Prior to discussing the computational performance, we brie y compare the real-time computed trajectory to one that is (locally) optimal. The problem parameters used for this comparison are given in Table 3. We use GPOPS-II to compute the locally optimal solution for comparison.[82] Figure 5 shows the resulting trajectories in the y_I z_I landing plane, alongside the thrust and torque trajectories. For the real-time solution, the circles represent the last reference solution $f x_k g_{k2N}$ obtained by the solve step prior to exiting the main iterative loop, whereas the solid line corresponds to the continuous state trajectory obtained by integrating the solution $f u_k ; p g_{k2N}$ through the nonlinear dynamics using (6). The attitude of the vehicle is depicted by the red lines in Figure 5(a) at identical temporal points in each trajectory, and it can be seen to match well at each point. Both the state and control trajectories show some key di erences; namely the real-time control solution does not have the minimum thrust arc seen in the solution computed by GPOPS-II and the torque solution follows a di erent albeit smoother trajectory. The torque jitter in the GPOPS-II solution is an artifact of that software's known di culty in nding singular solutions when encompassed by non-singular arcs.[82] These control di erences do not greatly impact the PTR algorithm's ability to nd an acceptable sub-optimal solution, as the cost is not signi cantly a ected by the presence of the minimum thrust arc. As seen in Figure 6, the nal mass is very close between the two trajectories. In fact, the real-time solution nds a solution that is roughly 0:6% sub-optimal and with a nal time that is within 1 :3% of the optimal solution. This sub-optimality is traded-o for the ability to compute the solutions in real-time in an architecture that is appropriate for a ight code implementation.

(a) The computed inertial position trajectories (b) The computed control trajectories

Figure 5: The real-time solution using $N = 20$ compared to the (locally) optimal planar landing solution. In (a), the circles represent solution nodes at each $k \in \mathcal{N}$ and the vehicle attitude is shown for both solutions at the same time instances.

We now study the computational performance obtained by the real-time PTR algorithm in addition to the sensitivity of key outputs to user-defined algorithm parameters. The solution method was implemented in C++ and run using a single 3.2 GHz Intel i5 processor with 8GB of RAM. All code was compiled using the `-O3` flag. Figure 7 shows the computational times obtained as a function of the discretization density N. Each data point represents the mean value of 100 trials using the specified value of N and all other parameters as in Table 3.[c] The error bars that are shown indicate one standard deviation. The contributions of the pre-parse, convexification, parse and solve steps are shown individually, in addition to the total solution time. The top plot in Figure 7 shows that the total solution time and the solver time per PTR iteration follow the same trend as N increases. It is expected that the solver time is cubic in N since the BSOCP solver performs a Cholesky factorization each time it is called to obtain the Newton search direction,[32] a function whose operation count scales with the cube of the matrix dimension.[81] The cubic polynomial fit to the total time (shown by the red line) supports the hypothesis that the total solution time is dominated by the internal operations of the BSOCP solver, and in particular the Cholesky factorization. The time spent outside the solver consumes on average 2% of the total solution time. The desirable conclusion is that the PTR algorithm adds little computational overhead to what is needed by the solver. However, caution should be used when extrapolating this result to other problem instances, as the ratio of time spent in the solver to time spent in other functions can vary between different problems and implementations.

The bottom four plots in Figure 7 show that the percent sub-optimality is roughly consistent across all tested values of N, and that the computation time of the pre-parse, convexification and parse steps scale linearly with N. For the convexification step, this is expected based on the computational complexity discussion presented in §II.C. Both the pre-parse and parsing steps can be done very quickly, and do not impact the overall solution time of the PTR algorithm.

Lastly, Figure 8 shows the variation in the final time, the final mass (i.e., the cost) and the total solution time versus several parameters that are selected by the user. For each plot, all nominal values given in Table 3 were held constant (with $N = 15$) and only the variable shown on the x-axis was changed. The final time guess is seen to strongly factor into the total solution time, and a minimum is observed around the optimal value of 9s, as expected. As previously mentioned, the final time guesses that are much shorter than the optimal one lead to higher solution times compared to those that are longer. The values chosen for the virtual control and trust region weights can affect the dependent variables tested if they are chosen improperly; the results suggest there is a rather large range for each parameter where the computed solution remains nearly the same. The solution using $w_{tr} = 10^{-3}$ produced results that did not converge and are marked with a red (the total solve time is off of the chart). This unsurprisingly indicates that if state and control deviations are not penalized enough,

[c] Since the results shown are on a per iteration basis, we hypothesize that varying the initial conditions in a more traditional Monte Carlo test would not change the results as presented. There are no additional constraints to activate and no additional non-zero entries in A or b to make the problem more challenging to solve.

(a) Translational states (b) Rotational states

Figure 6: Individual state trajectories for the planar landing case study. The real-time results use $N = 20$.

the algorithm can step far enough from the reference trajectory that the linearizations are no longer valid, and convergence suffers accordingly. The variations observed for the solution that uses $w_{vc} = 10^3$ are due to scaling issues that start to arise when this parameter is too large.

The fourth row of Figure 8 shows that the values of the inertia and specific impulse can affect the total solution time, but do not strongly affect the final time or mass for this case study. This implies that algorithm parameters tuned for nominal values do not need to be changed if the actual values change by up to $\pm 10 = 20\%$. Beyond this range, the algorithm parameters may need to be changed slightly to obtain the best computational performance. The increased solution time for large I_{sp} is caused by the increasing optimal final time that eventually requires an additional PTR iteration to converge to. Caution must be exercised when extrapolating these latter two results to other problem instances, since different dynamics, boundary conditions and/or constraints can change the relationship between J; I_{sp} and the dependent variables studied here. A similar sensitivity analysis should be carried out for each new implementation. Lastly, the fifth row of Figure 8 shows the effect of varying m in (32). While the final mass and final time are largely unaffected, it can be seen that lower values of m produce faster convergence. The optimal choice in this case is $m = 1$, the nominal value that is used.

IV. Conclusions

Solving challenging non-convex optimization problems in real-time on-board spacecraft has been identified as a key technology required for future entry, descent and landing systems. In this paper, we have presented a methodology for designing real-time algorithms that can be implemented in adherence to the standards of space flight code. A set of general algorithmic steps were presented, followed by a case study to illustrate how these steps are applied to a powered descent guidance problem. The planar landing problem that was studied constitutes the most general landing problem with both rotational and translational states for which the optimal control structure is theoretically understood. The case study indicates that solutions that are less than 1% sub-optimal can be computed on the order of 100 ms using a 3.2 GHz Intel i5 processor with 8 GB of RAM. While these results are specific to the planar landing problem that was studied, they offer a promising indication that the algorithm and methodology are suitable for the full scale 6-DoF problem.

Acknowledgements

This research has been supported by NASA grant NNX17AH02A and government sponsorship is acknowledged. This paper is dedicated to the memory of John Alexander Hansuld.

Figure 7: The computational times and percent sub-optmality obtained for each primary step in the algorithm versus the discretization density N for the planar landing case study.

References

[1] J. W. Jones, \The Recent Large Reduction in Space Launch Cost," in 48th International Conference on Environmental Systems, (Albuquerque, NM), 2018.

[2] R. T. Vought, \Commerce, Justice, Science, and Related Agencies Appropriations Act, 2020." Executive O ce of the President, O ce of Management and Budget, 2019.

[3] A. R. Klumpp, Apollo Lunar-Descent Guidance. Charles Stark Draper Laboratory, 1971.

[4] J. M. Carson III, M. M. Munk, R. R. Sostaric, J. N. Estes, F. Amzajerdian, J. B. Blair, D. K. Rutishauser, C. I. Restrepo, A. Dwyer-Cianciolo, G. T. Chen, and T. Tse, \The SPLICE Project: Continuing NASA Development of GN&C Technologies for Safe and Precise Landing," in AIAA SciTech Forum, (San Diego, CA), 2019.

[5] E. A. Robertson, \Synopsis of Precision Landing and Hazard Avoidance (PL&HA) Capabilities for Space Exploration," in AIAA Guidance, Navigation, and Control Conference, American Institute of Aeronautics and Astronautics, 2017.

[6] E. S. Team, \Europa Study 2012 Report: Europa Lander Mission," Tech. Rep. Task Order NMO711062 Outer Planets Flagship Mission, Jet Propulsion Laboratory, 2012.

[7] A. M. Dwyer-Cianciolo, C. D. Karlgaard, D. Wo nden, R. A. Lugo, J. Tynis, R. R. Sostaric, S. Striepe, R. Powell, and J. M. Carson, \De ning Navigation Requirements for Future Missions," in AIAA Scitech 2019 Forum, American Institute of Aeronautics and Astronautics, 2019.

[8] NASA Science, \Moon's South Pole in NASA's Landing Sites." https://solarsystem.nasa.gov/news/907/moons-south-pole-in-nasas-landing-sites/, 2019. Accessed: 11/26/2019.

[9] M. Robinson, \Lunar Reconnaissance Orbiter Camera (LROC)." http://lroc.sese.asu.edu/posts/993, 2018. Accessed: 11/26/2019.

[10] J. S. Meditch, \On the Problem of Optimal Thrust Programming For a Lunar Soft Landing," IEEE Transactions on Automatic Control, vol. 9, no. 4, pp. 477{484, 1964.

[11] D. Lawden, \Optimal Trajectories for Space Navigation," in Optimal Trajectories for Space Navigation, London: Butterworths, 1963.

[12] J.-P. Marec, Optimal Space Trajectories. Amsterdam: Elsevier Scienti c Publishing Company, 1979.

[13] C. N. D'Souza, \An Optimal Guidance Law for Planetary Landing," in AIAA Guidance, Navigation, and Control Conference, (New Orleans, LA), 1997.

[14] U. Topcu, J. Casoliva, and K. D. Mease, \Fuel E cient Powered Descent Guidance for Mars Landing," in AIAA Guidance, Navigation, and Control Conference, (San Francisco, CA), 2005.

[15] U. Topcu, J. Casoliva, and K. D. Mease, \Minimum-Fuel Powered Descent for Mars Pinpoint Landing," Journal of Spacecraft and Rockets, vol. 44, no. 2, pp. 324{331, 2007.
[16] P. Lu, S. Forbes, and M. Baldwin, \A Versatile Powered Guidance Algorithm," in AIAA Guidance, Navigation ,and Control Conference, (Minneapolis, MN), 2012.
[17] P. Lu, \Propellant-Optimal Powered Descent Guidance," Journal of Guidance, Control, and Dynamics , vol. 41, no. 4, pp. 813{826, 2018.
[18] J. Nocedal and S. J. Wright, Numerical Optimization . New York, NY: Springer Science & Business Media, 2006.
[19] A. Forsgren, P. E. Gill, and M. H. Wright, \Interior Methods for Nonlinear Optimization," SIAM Review , vol. 44, no. 4, pp. 525{597, 2002.
[20] S. Boyd and L. Vandenberghe, Convex Optimization . Cambridge, UK: Cambridge University Press, 2004.
[21] B. Ac kmese and S. Ploen, \Convex Programming Approach to Powered Descent Guidance for Mars Landing," Journal of Guidance, Control, and Dynamics , vol. 30, no. 5, pp. 1353{1366, 2007.
[22] B. Ac kmese and L. Blackmore, \Lossless Convexi cation of a Class of Optimal Control Problems with Non-Convex Control Constraints," Automatica , vol. 47, no. 2, pp. 341{347, 2011.
[23] L. Blackmore, B. Ac kmese, and D. P. Scharf, \Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization," Journal of Guidance, Control, and Dynamics , vol. 33, no. 4, pp. 1161{1171, 2010.
[24] J. M. Carson III, B. Ac kmese, and L. Blackmore, \Lossless Convexi cation of Powered-Descent Guidance with Non-Convex Thrust Bound and Pointing Constraints," in IEEE American Control Conference , (San Francisco, CA), 2011.
[25] B. Ac kmese, J. M. Carson III, and L. Blackmore, \Lossless Convexi cation of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," IEEE Transactions on Control Systems Technology , vol. 21, no. 6, pp. 2104{2113, 2013.
[26] M. W. Harris and B. Ac kmese, \Lossless convexi cation for a class of optimal control problems with linear state constraints," in 52nd IEEE Conference on Decision and Control , IEEE, 2013.
[27] M. W. Harris and B. Ac kmese, \Lossless convexi cation for a class of optimal control problems with quadratic state constraints," in 2013 American Control Conference , IEEE, 2013.
[28] M. W. Harris and B. Ac kmese, \Lossless Convexi cation of Non-Convex Optimal Control Problems for State Constrained Linear Systems," Automatica , vol. 50, no. 9, pp. 2304{2311, 2014.
[29] L. Blackmore, B. Ac kmese, and J. M. Carson III, \Lossless Convexi cation of Control Constraints for a Class of Nonlinear Optimal Control Problems," Systems and Control Letters , vol. 61, no. 8, pp. 863{870, 2012.
[30] D. Malyuta and B. Acikmese, \Lossless Convexi cation of Optimal Control Problems with Semi-Continuous Inputs," arXiv e-prints , p. arXiv:1911.09013, 2019.
[31] D. Malyuta, M. Szmuk, and B. Acikmese, \Lossless Convexi cation of Non-Convex Optimal Control Problems with Disjoint Semi-Continuous Inputs," arXiv e-prints , p. arXiv:1902.02726, 2019.
[32] D. Dueri, J. Zhang, and B. Ac kmese, \Automated Custom Code Generation for Embedded, Real-time Second Order Cone Programming," IFAC Proceedings Volumes , vol. 47, no. 3, pp. 1605{1612, 2014.
[33] D. Dueri, B. Ac kmese, D. P. Scharf, and M. W. Harris, \Customized Real-Time Interior-Point Methods for Onboard Powered-Descent Guidance," Journal of Guidance, Control, and Dynamics , vol. 40, no. 2, pp. 197{212, 2016.
[34] A. Domahidi, E. Chu, and S. Boyd, \ECOS: An SOCP Solver for Embedded Systems," in European Control Conference , pp. 3071{3076, 2013.
[35] D. Dueri, B. Ac kmese, M. Baldwin, and R. S. Erwin, \Finite-Horizon Controllability and Reachability for Deterministic and Stochastic Linear Control Systems with Convex Constraints," in IEEE American Control Conference , 2014.
[36] U. Eren, D. Dueri, and B. Ac kmese, \Constrained Reachability and Controllability Sets for Planetary Precision Landing via Convex Optimization," Journal of Guidance, Control, and Dynamics , vol. 38, no. 11, pp. 2067{2083, 2015.
[37] D. Dueri, S. V. Rakovic, and B. Acikmese, \Consistently Improving Approximations for Constrained Controllability and Reachability," in 2016 European Control Conference (ECC) , IEEE, 2016.
[38] M. Szmuk, B. Ac kmese, and A. W. Berning, \Successive Convexi cation for Fuel-Optimal Powered Landing with Aerodynamic Drag and Non-Convex Constraints," in AIAA Guidance, Navigation, and Control Conference , (San Diego, CA), 2016.
[39] M. Szmuk, T. P. Reynolds, and B. Ac kmese, \Successive Convexi cation for Real-Time 6-DoF Powered Descent Guidance with State-Triggered Constraints," arXiv e-prints , 2018. arXiv:1811.10803.
[40] M. Szmuk, T. P. Reynolds, B. Ac kmese, M. Mesbahi, and J. M. Carson III, \Successive Convexi cation for Real-Time Rocket Landing Guidance with Compound State-Triggered Constraints," in AIAA SciTech Forum , (San Diego, CA), 2019.
[41] T. P. Reynolds, M. Szmuk, D. Malyuta, M. Mesbahi, B. Ac kmese, and J. M. Carson III, \A State-Triggered Line of Sight Constraint for 6-DoF Powered Descent Guidance Problems," in AIAA SciTech Forum , (San Diego, CA), 2019.
[42] T. P. Reynolds, M. Szmuk, D. Malyuta, M. Mesbahi, B. Ac kmese, and J. M. Carson III, \Dual Quaternion Based 6-DoF Powered Descent Guidance with State-Triggered Constraints," arXiv e-prints , 2019. arXiv:1904.09248.
[43] Y. Mao, M. Szmuk, and B. Ac kmese, \Successive Convexi cation of Non-Convex Optimal Control Problems and its Convergence Properties," in IEEE Conference on Decision and Control , (Las Vegas, NV), 2016.
[44] Y. Mao, D. Dueri, M. Szmuk, and B. Ac kmese, \Successive Convexi cation of Non-Convex Optimal Control Problems with State Constraints," IFAC-PapersOnLine , vol. 50, no. 1, pp. 4063{4069, 2017.
[45] Y. Mao, M. Szmuk, and B. Ac kmese, \Successive Convexi cation: A Superlinearly Convergent Algorithm for Non-convex Optimal Control Problems," arXiv e-prints , 2018. arXiv:1804.06539.
[46] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, \Motion Planning with Sequential Convex Optimization and Convex Collision Checking," The International Journal of Robotics Research , vol. 33, no. 9, pp. 1251{1270, 2014.
[47] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, \GuSTO: Guaranteed Sequential Trajectory Optimization via Sequential Convex Programming," in 2019 IEEE International Conference on Robotics and Automation (ICRA) , IEEE, 2019.
[48] R. Bonalli, A. Bylard, A. Cauligi, T. Lew, and M. Pavone, \Trajectory Optimization on Manifolds: A Theoretically-Guaranteed Embedded Sequential Convex Programming Approach," in Robotics: Science and Systems, 2019.
[49] T. A. Howell, B. E. Jackson, and Z. Manchester, \ALTRO: A Fast Solver for Constrained Trajectory Optimization," in IEEE/RSJ International Conference on Intelligent Robots and Systems , 2019.

[50] X. Liu and P. Lu, \Solving Nonconvex Optimal Control Problems by Convex Optimization," Journal of Guidance, Control, and Dynamics , vol. 37, no. 3, pp. 750{765, 2014.
[51] X. Liu, Z. Shen, and P. Lu, \Solving the Maximum-Crossrange Problem via Successive Second-Order Cone Programming With a Line Search," Aerospace Science and Technology , vol. 47, pp. 10{20, 2015.
[52] X. Liu, Z. Shen, and P. Lu, \Entry Trajectory Optimization by Second-Order Cone Programming," Journal of Guidance, Control, and Dynamics , vol. 39, no. 2, pp. 227{241, 2016.
[53] Z. Wang and M. J. Grant, \Constrained Trajectory Optimization for Planetary Entry via Sequential Convex Programming," in AIAA Atmospheric Flight Mechanics Conference , American Institute of Aeronautics and Astronautics, 2016.
[54] Z. Wang and M. J. Grant, \Improved Sequential Convex Programming Algorithms for Entry Trajectory Optimization," in AIAA Scitech 2019 Forum , American Institute of Aeronautics and Astronautics, 2019.
[55] M. Szmuk, C. A. Pascucci, D. Dueri, and B. Acikmese, \Convexi cation and Real-Time On-board Optimization for Agile Quad-Rotor Maneuvering and Obstacle Avoidance," in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) , IEEE, 2017.
[56] M. Szmuk, C. A. Pascucci, and B. Acikmese, \Real-Time Quad-Rotor Path Planning for Mobile Obstacle Avoidance Using Convex Optimization," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) , IEEE, 2018.
[57] M. Szmuk, D. Malyuta, T. P. Reynolds, M. S. Mceowen, and B. Ac kmese, \Real-Time Quad-Rotor Path Planning Using Convex Optimization and Compound State-Triggered Constraints," in IEEE/RSJ International Conference on Intelligent Robots and Systems, p. arXiv:1902.09149, IEEE, 2019.
[58] Y. Mao, M. Szmuk, and B. Acikmese, \A Tutorial on Real-time Convex Optimization Based Guidance and Control for Aerospace Applications," in 2018 Annual American Control Conference (ACC) , IEEE, 2018.
[59] D. Malyuta, T. Reynolds, M. Szmuk, M. Mesbahi, B. Acikmese, and J. M. Carson, \Discretization Performance and Accuracy Analysis for the Rocket Powered Descent Guidance Problem," in AIAA Scitech 2019 Forum , American Institute of Aeronautics and Astronautics, 2019.
[60] B. Ac kmese, M. Aung, J. Casoliva, S. Mohan, A. Johnson, Scharf, M. D., S. D., A. Wolf, and M. Regehr, \Flight Testing of Trajectories Computed by G-FOLD: Fuel Optimal Large Divert Guidance Algorithm for Planetary Landing," in 23rd AAS/AIAA Space Flight Mechanics Meeting, Kauai, HI, 2013 , AAS/AIAA, 2017.
[61] D. P. Scharf, M. W. Regehr, G. M. Vaughan, J. Benito, H. Ansari, M. Aung, A. Johnson, J. Casoliva, S. Mohan, D. Dueri, B. Acikmese, D. Masten, and S. Nietfeld, \ADAPT Demonstrations of Onboard Large-Divert Guidance with a VTVL Rocket," in 2014 IEEE Aerospace Conference , IEEE, 2014.
[62] JPL and M. S. Systems, \750 Meter Divert Xombie Test Flight for G-FOLD, Guidance for Fuel Optimal Large Divert, Validation," http://www.youtube.com/watch?v=jl6pw2oossU , 2012.
[63] JPL and M. S. Systems, \500 meter divert Xombie test ight for G-FOLD, Guidance for Fuel Optimal Large Divert, validation," http://www.youtube.com/watch?v=1GRwimo1AwY , 2012.
[64] D. Dueri, Real-time Optimization in Aerospace Systems . Ph.D., University of Washington, 2018.
[65] J. Mattingley and S. Boyd, \CVXGEN: A Code Generator for Embedded Convex Optimization," Optimization and Engineering , vol. 13, no. 1, pp. 1{27, 2011.
[66] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, \FORCES NLP: An E cient Implementation of Interior-Point Methods for Multistage Nonlinear Nonconvex Programs," International Journal of Control , pp. 1{17, 2017.
[67] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, \E cient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control," in 2012 IEEE 51st IEEE Conference on Decision and Control (CDC) , IEEE, 2012.
[68] A. Domahidi, Methods and Tools for Embedded Optimization and Control . PhD thesis, ETH Zurich, 2013. Diss., Eidgenossische Technische Hochschule ETH Zurich, Nr. 21366, 2013.
[69] S. Mariethoz, A. Domahidi, and M. Morari, \High-Bandwidth Explicit Model Predictive Control of Electrical Drives," IEEE Transactions on Industry Applications , vol. 48, no. 6, pp. 1980{1992, 2012.
[70] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, \Embedded Predictive Control on an FPGA using the Fast Gradient Method," in 2013 European Control Conference (ECC) , IEEE, 2013.
[71] A. Liniger, A. Domahidi, and M. Morari, \Optimization-Based Autonomous Racing of 1:43 Scale RC Cars," arXiv e-prints , p. arXiv:1711.07300, 2017.
[72] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, \Learning-Based Model Predictive Control for Autonomous Racing," IEEE Robotics and Automation Letters , vol. 4, no. 4, pp. 3363{3370, 2019.
[73] U. Lee and M. Mesbahi, \Constrained Autonomous Precision Landing via Dual Quaternions and Model Predictive Control," Journal of Guidance, Control, and Dynamics , vol. 40, no. 2, pp. 292{308, 2017.
[74] G. J. Holzmann, \The Power of Ten { Rules for Developing Safety Critical Code," IEEE Computer , 2006. https://spinroot.com/p10/.
[75] T. P. Reynolds and M. Mesbahi, \Optimal Planar Powered Descent with Independent Thrust and Torque," Journal of Guidance, Control, and Dynamics (Submitted) , 2019.
[76] L. D. Berkovitz, Optimal Control Theory . New York, NY: Springer-Verlag, 1974.
[77] D. P. Scharf, B. Ac kmese, D. Dueri, J. Benito, and J. Casoliva, \Implementation and Experimental Demonstration of Onboard Powered-Descent Guidance," Journal of Guidance, Control, and Dynamics , vol. 40, no. 2, pp. 213{229, 2017.
[78] M. I. Ross, Q. Gong, M. Karpenko, and R. J. Proulx, \Scaling and Balancing for High-Performance Computation of Optimal Controls," Journal of Guidance, Control, and Dynamics , vol. 41, no. 10, pp. 2086{2097, 2018.
[79] J. T. Betts, Practical Methods for Optimal Control Using Nonlinear Programming . Philadelphia, PA: SIAM, 2001.
[80] P. E. Gill, W. Murray, and M. H. Wright, Practical Optimization . Academic Press, Inc., 1981.
[81] L. N. Trefethen and D. Bau, Numerical Linear Algebra . Philadelphia, PA: SIAM, 1997.
[82] M. A. Patterson and A. V. Rao, \GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming," ACM Transactions on Mathematical Software, vol. 41, no. 1, pp. 1{37, 2014.