

CUSTOMIZED FIRST-ORDER METHOD FOR LOW-THRUST TRAJECTORY OPTIMIZATION USING CONVEX PROGRAMMING

Gonçalo Oliveira Pinho*, Carmine Giordano[†], and Francesco Topputo[‡]

This paper presents a customized Proportional-Integral Projected Gradient (PIPG) solver for onboard interplanetary low-thrust trajectory optimization. The solver targets the convex subproblems generated by sequential convex programming and aims to reduce the memory and computational costs associated with general-purpose conic solvers on embedded spacecraft hardware. The proposed approach exploits the structure of the discretized optimal-control problem through problem-specific matrix-vector products, closed-form projection operators, and warm starting. The customized solver is benchmarked against ECOS, CLARABEL, and a standard PIPG implementation on a ZCU102 embedded board over diverse interplanetary transfer scenarios. The results show that the customized PIPG implementation preserves solution quality and convergence robustness while reducing computation time over the considered benchmark set. In addition, a preliminary FPGA implementation of the inner PIPG loop is demonstrated for a representative transfer, showing numerical consistency with the software implementation and reducing the complete SCP runtime by $5.63\times$ with respect to ECOS. These results indicate that structure-exploiting first-order methods are promising candidates for embedded onboard low-thrust guidance.

INTRODUCTION

In recent years, the space industry has significantly shifted towards the miniaturization of satellites and their components, resulting in a proliferation of launches due to reduced costs. Nonetheless, the current paradigm of flight dynamic operations relies on expensive, dedicated ground-based teams and infrastructure. This operational paradigm becomes increasingly restrictive for deep-space small spacecraft, for which communication delays, limited contact windows, and reduced ground support motivate higher levels of onboard autonomy. To address such incongruity, autonomous deep-space guidance, navigation, and control (GNC) emerges as a viable solution for future small-body, interplanetary, and distributed space missions.^{1–3}

Focusing on deep-space onboard guidance algorithms, these must be reliable, robust, and computationally efficient given the limited resources embedded onboard systems present. At each guidance update, the spacecraft must compute a feasible trajectory that satisfies missions constraints, typically by solving an optimization problem. Among the current state-of-the-art approaches, convex optimization is ideal for onboard guidance because, once the original problem has been transformed

*PhD Student, Department of Aerospace Science and Technology, Politecnico di Milano, Milan, Italy, goncalo.oliveira@polimi.it.

[†]Assistant Professor, Department of Aerospace Science and Technology, Politecnico di Milano, Milan, Italy, carmine.giordano@polimi.it. AIAA Member.

[‡]Full Professor, Department of Aerospace Science and Technology, Politecnico di Milano, Milan, Italy, francesco.topputo@polimi.it. AIAA Senior Member.

into a convex form, the resulting problem can be solved reliably by means of polynomial-time algorithms.⁴ For low-thrust trajectory optimization, previous works have shown that this can be achieved through Sequential Convex Programming (SCP), where the nonlinear optimal control problem is approximated by a sequence of convex subproblems.^{4,5}

In the low-thrust case, the guidance problem is demanding since the spacecraft trajectory is shaped by continuous thrust arcs over long time horizons. As a result, accurate discretizations may require a large number of nodes, leading to optimization problems with many decision variables and constraints. Moreover, onboard guidance algorithms must be executed repeatedly during the mission under strict limits on computation time, memory usage, and power consumption. These requirements motivate the development of optimization algorithms that are not only reliable, but also tailored to the underlying structure of the problem.

After convexification, the subproblems arising in low-thrust trajectory optimization can often be formulated as quadratic programs (QPs) or second-order cone programs (SOCPs). State-of-the-art conic solvers such as ECOS⁶ and CLARABEL⁷ are highly effective for solving these problems and are based on interior-point methods (IPMs). However, their use in an onboard environment can be challenging because they typically require the repeated solution of large linear systems and rely on sparse linear algebra routines whose memory access patterns and factorization costs can become expensive on embedded hardware. Although well suited to desktop-class processors, these operations can limit real-time onboard implementation on resource-constrained platforms.

First-order optimization algorithms provide a practical alternative for resource-constrained applications. These methods rely on simpler linear algebra such as matrix-vector products, vector updates, and projection operations, avoiding the large matrix factorizations required by IPMs. In addition, first-order methods are naturally compatible with warm-starting, which is especially useful within SCP, where consecutive convex subproblems are closely related. Their main drawback is a stronger sensitivity to problem conditioning and the possible need for a larger number of iterations to achieve high accuracy, aspects that must be taken into account during implementation.

Among first-order methods, the Proportional-Integral Projected Gradient (PIPG) algorithm has recently been shown to be well suited for trajectory optimization problems.⁸ When applied to structured optimal control problems, the method can exploit the structured block-sparse dynamics and separable path constraints to completely eliminate large sparse matrix operations. Previous work has demonstrated the potential of PIPG in terms of computational efficiency, memory allocation, verification and validation, and its suitability as a conic optimization solver within SCP-based guidance algorithms.⁹

This paper investigates the use of a customized PIPG solver for interplanetary low-thrust trajectory optimization. The low-thrust minimum-fuel problem is transcribed into a sequence of convex subproblems using first-order-hold control discretization, virtual variables, and trust-region regularization. The resulting subproblem is then structured to be compatible with a projection-based first-order solver. By exploiting the sparsity and block structure of the discretized dynamics, a customized PIPG implementation is developed that avoids dynamic memory allocation and replaces generic sparse matrix operations with problem-specific linear algebra routines. The solver is benchmarked against ECOS, CLARABEL, and a standard PIPG implementation across multiple interplanetary transfer scenarios and discretization sizes. Finally, preliminary results toward an FPGA-oriented implementation are presented to assess the suitability of the proposed approach for embedded onboard guidance applications.

LOW-THRUST TRAJECTORY OPTIMIZATION PROBLEM FORMULATION

Continuous-Time Nonconvex Optimal Control Problem

This paper considers a fixed-time minimum-fuel interplanetary low-thrust trajectory optimization problem. The continuous-time nonconvex optimal control can be written as:⁴

$$\min_{\alpha(t), \sigma(t)} -m(t_f) \quad (1a)$$

$$\text{subject to: } \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \alpha(t), \sigma(t)) \quad (1b)$$

$$0 \leq \sigma(t) \leq 1 \quad (1c)$$

$$\|\alpha(t)\|_2 = 1 \quad (1d)$$

$$\mathbf{r}(0) = \mathbf{r}_i, \quad \mathbf{v}(0) = \mathbf{v}_i, \quad m(0) = m_i \quad (1e)$$

$$\mathbf{r}(t_f) = \mathbf{r}_f, \quad \mathbf{v}(t_f) = \mathbf{v}_f \quad (1f)$$

where vector $\mathbf{x}(t) = [\mathbf{r}(t), \mathbf{v}(t), m(t)]$, with $\mathbf{r}(t) \in \mathbb{R}^3$, $\mathbf{v}(t) \in \mathbb{R}^3$, and $m(t) \in \mathbb{R}$ denoting the position, velocity and mass of the spacecraft, respectively. $\sigma(t) \in [0, 1]$ is the throttle factor, and $\alpha(t) \in \mathbb{R}^3$ the thrust direction unit vector. Eqs. (1e) - (1f) impose the boundary conditions at the initial and final times respectively. The two-body dynamics imposed in Eq. (1b) are governed by:

$$\dot{\mathbf{r}}(t) = \mathbf{v}(t) \quad (2a)$$

$$\dot{\mathbf{v}}(t) = -\frac{\mu}{\|\mathbf{r}(t)\|_2^3} \mathbf{r}(t) + \sigma(t) \frac{T_{\max}}{m(t)} \alpha(t) \quad (2b)$$

$$\dot{m}(t) = -\sigma(t) \frac{T_{\max}}{g_0 I_{sp}} \quad (2c)$$

where μ denotes the gravitational parameter of the primary body, g_0 the gravitational acceleration at sea level, $T_{\max}$ the maximum available thrust provided by the propulsion system, and I_{sp} the specific impulse.

Sequential Convex Programming

The continuous-time nonconvex optimal control problem is cast into a sequence of finite-dimensional convex subproblems by a specific change of variables, the introduction of auxiliary variables and linearization of a set of equations.⁵

Eq. (1b) is nonlinear and nonconvex as the state and control are coupled by the nonlinear term:

$$\sigma(t) \frac{T_{\max}}{m(t)} \alpha(t) \quad (3)$$

Exploiting the following change of variables:

$$\Gamma(t) := \sigma(t) \frac{T_{\max}}{m(t)} \quad (4a)$$

$$\boldsymbol{\tau}(t) := \sigma(t) \frac{T_{\max}}{m(t)} \boldsymbol{\alpha}(t) \quad (4b)$$

$$w(t) := \ln m(t) \quad (4c)$$

the states and controls are now defined as $\mathbf{x} := [\mathbf{r}(t), \mathbf{v}(t), w(t)]$ and $\mathbf{u} := [\boldsymbol{\tau}(t), \Gamma(t)]$, respectively.

Due to the change of variables, the constraint $\|\boldsymbol{\alpha}(t)\|_2 = 1$ changes and is relaxed to the second-order cone $\|\boldsymbol{\tau}(t)\|_2 \leq \Gamma(t)$. The constraint $0 \leq \sigma(t) \leq 1$ is updated and further linearized about the reference $\bar{w}$ as:

$$0 \leq \Gamma(t) \leq T_{\max} e^{-\bar{w}} [1 - w(t) + \bar{w}(t)] \quad (5)$$

The dynamics in Eq. (1b) are now written as:

$$\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) = \begin{bmatrix} \mathbf{v}(t) \\ -\frac{\mu}{\|\mathbf{r}(t)\|_2^3} \mathbf{r}(t) \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{3 \times 4} \\ \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 1} \\ \mathbf{0}_{1 \times 3} & -1/(g_0 I_{sp}) \end{bmatrix} \begin{bmatrix} \boldsymbol{\tau}(t) \\ \Gamma(t) \end{bmatrix} \quad (6)$$

To alleviate the nonconvexity, we perform a first-order Taylor series approximation of Eq. (6) about an arbitrary reference trajectory $(\bar{\mathbf{x}}(t), \bar{\mathbf{u}}(t))$ which yields a linear time-varying (LTV) system, given by:

$$\dot{\mathbf{x}}(t) \approx \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) + \mathbf{c}(t) \quad (7)$$

where

$$\mathbf{A}(t) := \left. \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right|_{\mathbf{x}=\bar{\mathbf{x}}(t), \mathbf{u}=\bar{\mathbf{u}}(t)} \quad (8a)$$

$$\mathbf{B}(t) := \left. \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \right|_{\mathbf{x}=\bar{\mathbf{x}}(t), \mathbf{u}=\bar{\mathbf{u}}(t)} \quad (8b)$$

$$\mathbf{c}(t) := \mathbf{f}(\bar{\mathbf{x}}(t), \bar{\mathbf{u}}(t)) - \mathbf{A}(t)\bar{\mathbf{x}}(t) - \mathbf{B}(t)\bar{\mathbf{u}}(t) \quad (8c)$$

The matrices $\mathbf{A}(t)$ and $\mathbf{B}(t)$, together with the affine term $\mathbf{c}(t)$, are fixed within each convex subproblem and recomputed at every SCP iteration from the updated reference trajectory. Depending on the thruster model considered, $\mathbf{B}(t)$ may be constant; however, the time-dependent notation is retained for consistency with the general LTV formulation.

Discretization. The continuous-time optimal control problem must be discretized as a finite-dimensional optimization problem. We start by discretizing the time grid $t \in [0, \text{ToF}]$ into $N - 1$ equally spaced segments:¹⁰

$$0 = t_1 < \dots < t_N = \text{ToF} \quad (9)$$

where N denotes the number of discretization points. Afterward, we adopt a first-order-hold (FOH) parameterization of the control input signal. This scheme is chosen because it provides a more accurate representation of the continuous control profile than zero-order hold while preserving the block-sparse structure of the discrete dynamics.^{10,11} With FOH, the continuous-time control input signal is obtained by linearly interpolating between the discrete values at successive nodes as:

$$\mathbf{u}(t) = \underbrace{\left(\frac{t_{k+1} - t}{t_{k+1} - t_k} \right) \mathbf{u}_k}_{\lambda_k^-(t)} + \underbrace{\left(\frac{t - t_k}{t_{k+1} - t_k} \right) \mathbf{u}_{k+1}}_{\lambda_k^+(t)}, \forall t \in [t_k, t_{k+1}] \quad (10)$$

with $k = 1, \dots, N - 1$.

The exact discretization of Eq. (7) can be computed as:

$$\mathbf{x}_{k+1} = \mathbf{A}_k \mathbf{x}_k + \mathbf{B}_k^- \mathbf{u}_k + \mathbf{B}_k^+ \mathbf{u}_{k+1} + \mathbf{c}_k \quad (11)$$

where

$$\mathbf{A}_k := \Phi(t_{k+1}, t_k) \quad (12a)$$

$$\mathbf{B}_k^- := \mathbf{A}_k \int_{t_k}^{t_{k+1}} \Phi^{-1}(t, t_k) \lambda_k^-(t) \mathbf{B}(t) dt \quad (12b)$$

$$\mathbf{B}_k^+ := \mathbf{A}_k \int_{t_k}^{t_{k+1}} \Phi^{-1}(t, t_k) \lambda_k^+(t) \mathbf{B}(t) dt \quad (12c)$$

$$\mathbf{c}_k := \mathbf{A}_k \int_{t_k}^{t_{k+1}} \Phi^{-1}(t, t_k) \mathbf{c}(t) dt \quad (12d)$$

and $\Phi(t, t_k)$ is the state transition matrix for the linear time-varying system in Eq. (7).

Each set of $N-1$ matrices given in Eq. (12) is computed by numerically integrating the integrands along with the state vector, for which a fixed-step RK8 integration routine was implemented.

Trust Region and Virtual Terms. As the nonconvex constraints are approximated using a first-order Taylor series expansion which is only accurate close to the reference trajectory, it is important to restrict the search space to a region where the linearization is sufficiently accurate, thus preserving the validity. In addition, linearization can render the cost unbounded from below when the solution is not enforced to lie in the neighborhood of the reference. Therefore, we impose a soft trust region by augmenting the objective function with a quadratic penalty term for the deviation between the solution state $\mathbf{x}$ and control $\mathbf{u}$ from the reference $\bar{\mathbf{x}}$ and $\bar{\mathbf{u}}$, respectively.¹²

Furthermore, when nonlinear constraints are linearized about a reference solution, we may encounter an infeasible convex subproblem even if there exists a feasible solution to the original non-convex problem. To mitigate this, an unconstrained virtual control variable is added to the linearized dynamical constraints in Eq. (11), therefore ensuring that the subproblem is feasible. While these terms maintain feasibility, they also result in constraint violations when active. To ensure these virtual control variables go to zero at convergence for a solution to be feasible with respect to the original nonconvex problem, we incorporate them in our objective function with a sufficiently large penalty term.¹²

In addition, auxiliary virtual state variables are introduced to decouple the variables appearing in the linearized dynamics from those used to enforce the boundary and path constraints. Specifically, we define the virtual state $\xi_k \in \mathbb{R}^7$ as a virtual copy of the state x_k , while $\beta_k \in \mathbb{R}$ is introduced as a virtual copy of the logarithmic mass w_k . The variable ξ_k is used to impose state bounds and boundary constraints, whereas β_k is used in the linearized upper bound on the thrust magnitude variable. To ensure that both the dynamics and the path constraints are satisfied at convergence, we penalize the squared deviations between the primary and virtual variables $x - \xi$ and $w - \beta$ directly within the objective function.

Discretized Convex Subproblem. The discretized convex subproblem is therefore written as:

$$\begin{aligned} \min_z \quad & -w_N + W_{TR} \left(\sum_{k=1}^N \|x_k - \bar{x}_k\|_2^2 + \sum_{k=1}^N \|u_k - \bar{u}_k\|_2^2 \right) \\ & + W_{VS} \sum_{k=1}^N \|x_k - \xi_k\|_2^2 + W_{VS} \sum_{k=1}^N (w_k - \beta_k)^2 + W_{VC} \sum_{k=1}^{N-1} \|\nu_k\|_2^2 \end{aligned} \quad (13a)$$

$$\text{subject to: } x_{k+1} = A_k x_k + B_k^- u_k + B_k^+ u_{k+1} + c_k + \nu_k, \quad \forall k = 1, \dots, N-1 \quad (13b)$$

$$\Gamma_k \leq T_{\max} e^{-\bar{w}_k} (1 - \beta_k + \bar{w}_k), \quad \forall k = 1, \dots, N \quad (13c)$$

$$\|\tau_k\|_2 \leq \Gamma_k, \quad \forall k = 1, \dots, N \quad (13d)$$

$$\xi_1 = [r_i, v_i, w_i] \quad (13e)$$

$$r_{\xi, N} = r_f, v_{\xi, N} = v_f \quad (13f)$$

$$x_l \leq \xi_k \leq x_u, \quad u_l \leq u_k \leq u_u, \quad \forall k = 1, \dots, N \quad (13g)$$

where $z = [x, \xi, \beta, u, \nu]$ is a single high-dimensional vector where the decision variables are stacked, $\nu_k \in \mathbb{R}^{n_x}$ is the virtual control variable associated with the dynamical constraint, W_{TR} , W_{VS} and W_{VC} are the large penalty terms for the trust region, virtual states and virtual control terms, respectively. For the initial guess of the SCP framework, a shape-based cubic interpolation or a sub-optimal trajectory from a modified Q-law guidance algorithm are used.⁴

The resulting optimization problem is a second-order cone program with affine equality constraints, box constraints, second-order cone constraints, singleton constraints, and a problem-specific affine upper bound on the thrust magnitude variable. This structure is suitable for a projection-based first-order method because the equality constraints correspond to discrete dynamics while the remaining constraints are separable across the adopted time grid and admit closed-form projection operators. This particular structure motivates the use of the solver described in the following section.

PIPG

The convex subproblems generated at each SCP iteration are solved using the Proportional-Integral Projected Gradient (PIPG) method.¹³ The method is well suited to the present problem because each iteration requires only matrix-vector products and projections onto simple convex sets, avoiding the matrix factorizations required by second-order conic solvers. The algorithm has been used effectively to solve convex subproblems for powered-descent guidance problems, spacecraft rendezvous guidance problems and multi-phase landing problems.⁸

The convex subproblem is therefore written in the following PIPG standard form:

$$\min_z \quad \frac{1}{2} \mathbf{z}^T \mathbf{Q} \mathbf{z} + \mathbf{q}^T \mathbf{z} \quad (14a)$$

$$\text{subject to} \quad \mathbf{H} \mathbf{z} - \mathbf{h} = \mathbf{0} \quad (14b)$$

$$\mathbf{z} \in \mathbb{D} \quad (14c)$$

where $\mathbf{z}$ is the decision vector previously defined, $\mathbf{Q}$ is a positive definite matrix, and the set $\mathbb{D}$ is a Cartesian product of simple sets that admits a closed-form projection operation.¹⁴

Following the formulation strategy adopted in Reference 8, all constraints that admit efficient closed-form projections are assigned to the set $\mathbb{D}$ rather than being imposed through the affine equality constraint $\mathbf{H} \mathbf{z} - \mathbf{h} = \mathbf{0}$. This is done since the constraints included in $\mathbb{D}$ are enforced at every PIPG iteration through the projection step, whereas the equality constraints associated with the discretized dynamics are only recovered asymptotically as the primal-dual iterations converge. Additionally, keeping as many constraints as possible in projected form avoids the introduction of additional auxiliary variables that would otherwise be required by conic reformulation.

Therefore, matrix $\mathbf{Q}$ and vector $\mathbf{q}$ will define the quadratic and linear part of the cost function in Eq. (13a). Matrix $\mathbf{H}$ enforces the first-order-hold discretized dynamics in Eq. (13b), while $\mathbf{h}$ contains the corresponding constant terms. The set $\mathbb{D}$ contains the remaining constraints satisfied via the following closed-form projections:¹⁵

$$\text{Combined thrust vector projection} \quad \left\{ \begin{array}{l} \Gamma_k \leq T_{\max} e^{-\bar{w}_k} (1 - \beta_k + \bar{w}_k) \\ \|\boldsymbol{\tau}_k\|_2 \leq \Gamma_k \end{array} \right. \quad (15a)$$

$$\quad \quad \quad (15b)$$

$$\text{Singleton} \quad \boldsymbol{\xi}_1 = [\mathbf{r}_i, \mathbf{v}_i, w_i] \quad (15c)$$

$$\text{Singleton} \quad \mathbf{r}_{\xi, N} = \mathbf{r}_f, \quad \mathbf{v}_{\xi, N} = \mathbf{v}_f \quad (15d)$$

$$\text{Box} \quad \mathbf{x}_l \leq \boldsymbol{\xi}_k \leq \mathbf{x}_u, \quad \mathbf{u}_l \leq \mathbf{u}_k \leq \mathbf{u}_u \quad (15e)$$

The singleton and box projections in Eqs. (15c) - (15e) admit direct closed-form expressions.¹⁵ The two thrust constraints in Eqs. (15a) - (15b), however, share the variable Γ_k and are therefore projected jointly at each discretization node. Defining:

$$a_k = T_{\max} \exp(-\bar{w}_k), \quad b_k = a_k(1 + \bar{w}_k) \quad (16)$$

the local thrust projection set is:

$$\mathbb{T}_k = \{(\beta_k, \boldsymbol{\tau}_k, \Gamma_k) : \|\boldsymbol{\tau}_k\|_2 \leq \Gamma_k, \quad \Gamma_k + a_k \beta_k \leq b_k\} \quad (17)$$

This set combines the second-order cone thrust-magnitude constraint with the affine upper bound obtained by linearizing $T_{\max} e^{-w_k}$ about the SCP reference $\bar{w}_k$. For this paper, the implementation uses the exact euclidean projection onto $\mathbb{T}_k$, rather than alternating projections between the cone and the affine halfspace. A sequential projection onto the second-order cone and the affine halfspace would not, in general, be equivalent to the euclidean projection onto their intersection. Alternating projections are mainly feasibility methods, and computing the exact metric projection onto an intersection of convex sets generally requires additional correction schemes.¹⁶ For this reason, a closed-form node-wise projection is derived for the local thrust set $\mathbb{T}_k$.

For each node k , given a tentative point $(\boldsymbol{\tau}_k^*, \Gamma_k^*, \beta_k^*)$, the closest point satisfying Eq. (17) is found by solving the projection problem:

$$\min_{\boldsymbol{\tau}_k, \Gamma_k, \beta_k} \quad \|\boldsymbol{\tau}_k - \boldsymbol{\tau}_k^*\|_2^2 + (\Gamma_k - \Gamma_k^*)^2 + (\beta_k - \beta_k^*)^2 \quad (18a)$$

$$\text{subject to} \quad \|\boldsymbol{\tau}_k\|_2 \leq \Gamma_k \quad (18b)$$

$$\Gamma_k + a_k \beta_k \leq b_k \quad (18c)$$

The projection problem is simplified by using Γ_k as the coupling variable between the two thrust constraints. For a fixed value of Γ_k , the second-order cone constraint reduces to the projection of $\boldsymbol{\tau}_k^*$ onto a ball of radius Γ_k , while the affine thrust bound reduces to the projection of β_k^* onto a scalar upper bound. Therefore, once Γ_k is fixed, the optimal values of $\boldsymbol{\tau}_k$ and β_k are known in closed form:¹⁵

$$\boldsymbol{\tau}_k(\Gamma_k) = \begin{cases} \boldsymbol{\tau}_k^*, & \|\boldsymbol{\tau}_k^*\|_2 \leq \Gamma_k \\ \frac{\Gamma_k}{\|\boldsymbol{\tau}_k^*\|_2} \boldsymbol{\tau}_k^*, & \text{otherwise} \end{cases} \quad \beta_k(\Gamma_k) = \begin{cases} \beta_k^*, & \Gamma_k \leq b_k - a_k \beta_k^* \\ \frac{b_k - \Gamma_k}{a_k}, & \text{otherwise} \end{cases} \quad (19)$$

Substituting Eq. (19) into Eq. (18) reduces the combined projection to a scalar minimization in Γ . Thus, the closed-form projection is obtained by evaluating the candidate minimizers over well-defined intervals and then recovering $\boldsymbol{\tau}$ and β from Eq. (19).

After defining the projection operators, the scaling of the quadratic conic subproblem is crucial for efficient PIPG convergence. Being a first-order method, the problem is sensitive to conditioning and typically performs poorly on ill-conditioned problems such as the convex subproblem in Eq. (13) due to the presence of heavy weights on the objective function. The hypersphere preconditioning procedure presented in Reference 17 is applied since it minimizes the condition number of the resulting Karush-Kuhn-Tucker (KKT) matrix by optimally scaling the objective function while preserving the structure of the preconditioned problem.

Since $\mathbf{Q}$ is positive definite, it can be expressed as $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is the lower-triangular Cholesky factor of $\mathbf{Q}$. Applying a change of variables, the transformed primal variable is defined as:

$$\hat{\mathbf{z}} = \mathbf{L}^T \mathbf{z}, \quad \mathbf{z} = \mathbf{L}^{-T} \hat{\mathbf{z}} \quad (20)$$

for which are defined the following preconditioned equality matrix and vector, and the preconditioned linear cost term:

$$\hat{\mathbf{H}} = \mathbf{D}_r \mathbf{H} \mathbf{L}^{-T}, \quad \hat{\mathbf{h}} = \mathbf{D}_r \mathbf{h}, \quad \hat{\mathbf{q}} = \mathbf{L}^{-1} \mathbf{q} \quad (21)$$

where $\mathbf{D}_r$ is the diagonal matrix that normalizes each row of the original preconditioned matrix $\mathbf{H} \mathbf{L}^{-T}$ to unit norms.

A new set $\hat{\mathbb{D}} = \{\hat{\mathbf{z}} : \mathbf{L}^{-T} \hat{\mathbf{z}} \in \mathbb{D}\}$ is also defined. For the implemented algorithm, the constraints are projected in the preconditioned coordinates apart from the combined thrust vector projection. This projection is performed on the corresponding segment of the primal vector, after temporarily mapping the variables back to their original scaling.

The resulting objective is then scaled by the factor $\lambda = \sqrt{\sigma_{\min}/2}$, where $\sigma_{\min}$ is the square of the smallest singular value of $\hat{\mathbf{H}}$.¹⁷ This value can be estimated using the shifted power iteration method.¹⁸ The choice of λ minimizes the condition number of the resulting KKT matrix and is therefore critical to the convergence behavior of the algorithm.¹⁷ Further details about this preconditioning procedure can be found in Reference 17.

The resulting preconditioned version of the conic subproblem described in Problem (14) is:

$$\min_{\hat{\mathbf{z}}} \quad \frac{\lambda}{2} \hat{\mathbf{z}}^T \hat{\mathbf{z}} + \hat{\mathbf{q}}^T \hat{\mathbf{z}} \quad (22a)$$

$$\text{subject to} \quad \hat{\mathbf{H}} \hat{\mathbf{z}} - \hat{\mathbf{h}} = \mathbf{0} \quad (22b)$$

$$\hat{\mathbf{z}} \in \hat{\mathbb{D}} \quad (22c)$$

In this form, the scaled problem (22) has the same structure as the original PIPG standard form, but with improved conditioning. It is solved using the preconditioned extrapolated PIPG algorithm described in Algorithm 1. The inputs $\rho \in [1.5, 1.9]$ and $\omega \in \mathbb{R}_{++}$ are hyperparameters used in the extrapolation factor and choice of primal and dual step sizes, respectively.¹⁹ The input σ corresponds to the maximum eigenvalue of $\hat{\mathbf{H}}^T \hat{\mathbf{H}}$, computed through a power iteration algorithm.⁸

The solver terminates either upon reaching a maximum number of iterations $j_{\max}$ or when convergence is achieved. Convergence occurs when the difference between two consecutive iterates becomes sufficiently small given an absolute accuracy tolerance ϵ_{abs} and a relative accuracy tolerance ϵ_{rel} , a condition typically used in first-order solvers.⁸ The stop condition is checked every j_{check} iterations.

Algorithm 1 Preconditioned extrapolated PIPG

Inputs: $\hat{H}, \hat{h}, \hat{q}, \mathbb{D}, L, L^{-T}, \lambda, \sigma, \rho, \omega, j_{\max}, z^0, \hat{y}^0$

```
1:  $\zeta^0 \leftarrow L^T z^0$  ▷ initialize transformed primal variable
2:  $\eta^0 \leftarrow \hat{y}^0$  ▷ initialize transformed dual variable
3:  $\alpha \leftarrow \frac{2}{\lambda + \sqrt{\lambda^2 + 4\sigma\omega^2}}$  ▷ primal step-size
4:  $\gamma \leftarrow \omega^2 \alpha$  ▷ dual step-size

5: for  $j = 0, \dots, j_{\max} - 1$  do
6:    $\hat{z}^{j+1} \leftarrow \Pi_{\mathbb{D}}(\zeta^j - \alpha(\lambda\zeta^j + \hat{q} + \hat{H}^T \eta^j))$  ▷ primal update
7:    $\hat{y}^{j+1} \leftarrow \eta^j + \gamma(\hat{H}(2\hat{z}^{j+1} - \zeta^j) - \hat{h})$  ▷ dual update
8:    $\zeta^{j+1} \leftarrow (1 - \rho)\zeta^j + \rho\hat{z}^{j+1}$  ▷ extrapolate primal variable
9:    $\eta^{j+1} \leftarrow (1 - \rho)\eta^j + \rho\hat{y}^{j+1}$  ▷ extrapolate dual variable

10:  if  $(j + 1) \bmod j_{\text{check}} = 0$  then ▷ check stopping criterion
11:    if  $\|\hat{z}^{j+1} - \hat{z}^j\|_{\infty} \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\{\|\hat{z}^{j+1}\|_{\infty}, \|\hat{z}^j\|_{\infty}\}$  and then
12:       $\|\hat{y}^{j+1} - \hat{y}^j\|_{\infty} \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\{\|\hat{y}^{j+1}\|_{\infty}, \|\hat{y}^j\|_{\infty}\}$ 
13:      break
14:    end if
15:  end for
```

Return: $z = L^{-T} \hat{z}^{j+1}, \hat{y}^{j+1}$

Customized Solver

Algorithm 1 identifies the operations required at each iteration: projections, products with $\hat{H}$ and $\hat{H}^T$, and transformations involving the Cholesky factor. The customized implementation focuses on evaluating these operations by exploiting the structure of the discretized low-thrust trajectory optimization problem to replace generic sparse linear algebra routines. Although the PIPG algorithm can be written compactly in terms of the stacked matrices Q and H , explicitly assembling and manipulating these matrices as generic sparse objects introduces memory overhead and indirect access patterns that are undesirable for embedded implementations.²⁰ Instead, the customized solver differs from a generic sparse PIPG implementation in how the products $\hat{H}\hat{z}$ and $\hat{H}^T\eta$ are evaluated.

The matrix $\hat{H}$ inherits the block-banded structure of the discretized dynamics for which the implementation stores the preconditioned equality matrix and its transpose as node-local dense blocks. Therefore, the forward product is evaluated by sweeping over the nodes and assembling each dynamics residual block directly considering the corresponding components of $\hat{z}$. Similarly, the transpose product considers the node-local matrices multiplied by the respective block of the dual variable and stored in the corresponding primal-variable segments. This process avoids indirect sparse-matrix access inside the algorithm loop and replaces generic matrix-vector operations with fixed-size dense operations.

Furthermore, the Cholesky factor of matrix $\mathbf{Q}$ is assembled analytically from the known block structure obtained by expanding the quadratic components of the objective function in Eq. (13a), since the penalty weights are fixed within the considered SCP framework.⁸ With the decision vector $\mathbf{z}$ defined above, the quadratic cost contains only local couplings between variables associated with the same discretization node. For notational clarity, the following block representation is written in a grouped ordering of the decision variables, obtained by collecting the locally coupled components together.

For each node k , the non-mass state components form repeated 2×2 blocks involving the pairs $(x_{k,j}, \xi_{k,j})$, with $j = 1, \dots, 6$:

$$\mathbf{Q}_{x\xi} = 2 \begin{bmatrix} W_{\text{TR}} + W_{\text{VS}} & -W_{\text{VS}} \\ -W_{\text{VS}} & W_{\text{VS}} \end{bmatrix} \quad (23)$$

while for the logarithmic mass component, the coupled variables are $(w_k, \xi_{k,7}, \beta_k)$, where $\xi_{k,7}$ denotes the seventh component of the virtual state ξ_k . These variables form the local 3×3 block:

$$\mathbf{Q}_{w\xi\beta} = 2 \begin{bmatrix} W_{\text{TR}} + 2W_{\text{VS}} & -W_{\text{VS}} & -W_{\text{VS}} \\ -W_{\text{VS}} & W_{\text{VS}} & 0 \\ -W_{\text{VS}} & 0 & W_{\text{VS}} \end{bmatrix} \quad (24)$$

The blocks associated with the control $\mathbf{u}$ and virtual control $\mathbf{v}$ are diagonal. Therefore, in the grouped notation, the quadratic matrix can be written as:

$$\mathbf{Q} = \text{blkdiag}(\mathbf{I}_{6N} \otimes \mathbf{Q}_{x\xi}, \mathbf{I}_N \otimes \mathbf{Q}_{w\xi\beta}, 2W_{\text{TR}}\mathbf{I}_{4N}, 2W_{\text{VC}}\mathbf{I}_{7(N-1)}) \quad (25)$$

Being block diagonal in the defined local components, the Cholesky factor of $\mathbf{Q}$ can be obtained by factorizing only the small local blocks:

$$\mathbf{Q}_{x\xi} = \mathbf{L}_{x\xi}\mathbf{L}_{x\xi}^\top, \quad \mathbf{Q}_{w\xi\beta} = \mathbf{L}_{w\xi\beta}\mathbf{L}_{w\xi\beta}^\top \quad (26)$$

Thus, in the same grouped ordering:

$$\mathbf{L} = \text{blkdiag}(\mathbf{I}_{6N} \otimes \mathbf{L}_{x\xi}, \mathbf{I}_N \otimes \mathbf{L}_{w\xi\beta}, \sqrt{2W_{\text{TR}}}\mathbf{I}_{4N}, \sqrt{2W_{\text{VC}}}\mathbf{I}_{7(N-1)}) \quad (27)$$

where $\mathbf{L}_{x\xi}$ and $\mathbf{L}_{w\xi\beta}$ are the Cholesky factors of the local blocks defined in Eqs. (23) - (24). The Cholesky factors of the control and virtual-control blocks are directly obtained from their diagonal structure. Since $\mathbf{L}$ is block diagonal in this grouped representation, its inverse can be evaluated through the inverses of the corresponding local blocks.

Figure 1 presents the sparsity pattern of the equality matrix before and after preconditioning. The original matrix $\mathbf{H}$ contains nonzero blocks associated with the discretized dynamics constraints. Following Eq. (21), the matrix $\hat{\mathbf{H}}$ presents additional local nonzero blocks due to the structure of the customized Cholesky factor in Eq. (27). The row normalization matrix $\mathbf{D}_r$ is a left diagonal scaling and therefore does not alter the sparsity pattern. Despite the modified sparsity pattern, the preconditioned equality matrix retains a node-local structure that can still be exploited by the customized implementation.

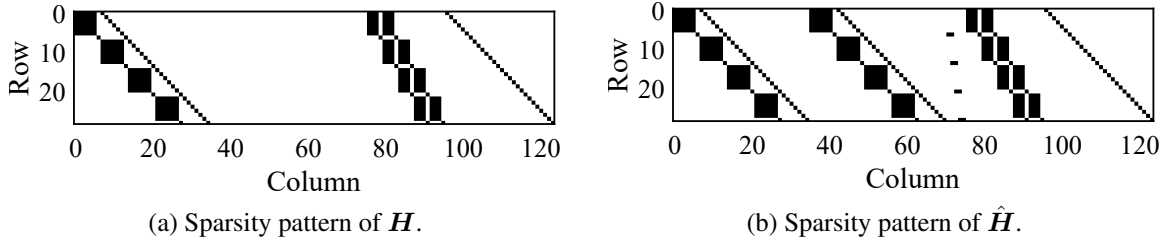


Figure 1: Equality matrix sparsity for the low-thrust interplanetary problem with $N = 5$ nodes.

The same customization principle is applied to the power and shifted power methods. These routines use the same matrix-free products adopted inside the PIPG loop, avoiding the need for generic sparse matrix-vector multiplications during both solver setup and online iterations.

Additionally, to reduce the number of iterations required to solve each SCP subproblem, the primal-dual solution of the last-accepted subproblem is used as an initial guess to warm-start the algorithm. The power and shifted-power iterations used to estimate σ and $\sigma_{\min}$ are warm-started in the same way using the eigenvector estimates from the previous SCP iteration.

All work vectors used by the solver are allocated at compile time, avoiding dynamic memory allocation during the solver iterations and making the implementation suitable for embedded deployment.

FPGA Implementation

The FPGA implementation targets the inner PIPG solve performed at each SCP iteration. This is the most computationally expensive and repetitive part of the SCP algorithm: once the convex subproblem has been formed and preconditioned, the PIPG iterations require products with the preconditioned equality matrix, simple vector updates, and projections onto the sets described previously. This structure is well suited for FPGA acceleration as the inner solver does not require large matrix factorizations and is composed mainly of regular, bounded operations.

The hardware kernel does not implement the complete SCP algorithm due to the limited resources available. Instead, the workload is divided such that the CPU performs the SCP-level operations while the FPGA executes the repeated inner PIPG iterations. This division keeps the hardware kernel simple and deterministic, limiting the FPGA implementation to the operations that dominate the repeated online solve.

Despite the PIPG iterations being sequential, as each update depends on the previous primal and dual iterates, the operations within each iteration can be exploited for computational efficiency. Therefore, the FPGA implementation exploits parallelism inside the matrix-vector products, vector updates, and node-wise projections, rather than attempting to parallelize across PIPG iterations.²¹ Moreover, instead of implementing generic sparse linear algebra routines, the kernel exploits the fixed sparsity pattern of the low-thrust optimal-control problem. In particular, the products involving $\hat{H}$ and $\hat{H}^T$ are evaluated using the same node-local structure adopted in the customized software implementation.

The main differences with respect to the software implementation are therefore implementation-related rather than algorithmic. The FPGA kernel uses fixed-size arrays and static loop bounds whenever possible. In the current implementation, the bitstream is generated for a fixed number of

discretization nodes, $N = 150$. This choice enables compile-time optimization of loop structures and memory layouts, although it also means that changing the number of nodes requires regenerating the hardware kernel.

To distinguish between implementation and precision effects, the current FPGA version uses double-precision floating-point arithmetic, matching the software implementation. The problem data are packed by the host and transferred to the FPGA before the inner PIPG solve, while the final primal-dual iterate is returned after the kernel terminates. Additionally, the data layout and packaging are organized to reduce host-kernel transfer overhead. Mixed-precision implementations could further reduce resource usage and improve throughput, but their impact on the convergence of both the inner PIPG solve and the outer SCP loop is left for future work. The resulting kernel is used in the following section to compare the generic solver, the customized software implementation, and the FPGA-accelerated implementation in terms of runtime and solution consistency.

RESULTS

We benchmark the SCP algorithm using `ECOS`, `CLARABEL`, a standard PIPG implementation, and the proposed customized PIPG implementation as solvers for the recursively generated convex subproblems.^{6,7} All algorithms were implemented in C++ and profiled on the AMD Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit.²² Since this board is representative of the type of embedded hardware used in CubeSat-class computers, it provides a suitable platform for assessing the feasibility of the proposed solver for onboard optimization.

The benchmark set consists of more than 50 low-thrust interplanetary scenarios, including transfers from Earth to Mars, Venus, and asteroid targets. The scenarios differ in time of flight, initial spacecraft mass, and initial and final boundary conditions. Across all scenarios, a constant low-thrust propulsion model is considered, with $T_{\max} = 2$ mN and $I_{\text{sp}} = 2000$ s. To evaluate problem scaling, the comparison is performed for three different discretization sizes, namely $N \in \{100, 200, 300\}$.

Figure 2 shows a representative Earth-to-Mars solution returned by the SCP algorithm using $N = 300$, where the arrows indicate the thrust direction along the transfer trajectory. The corresponding thrust profile is also shown.

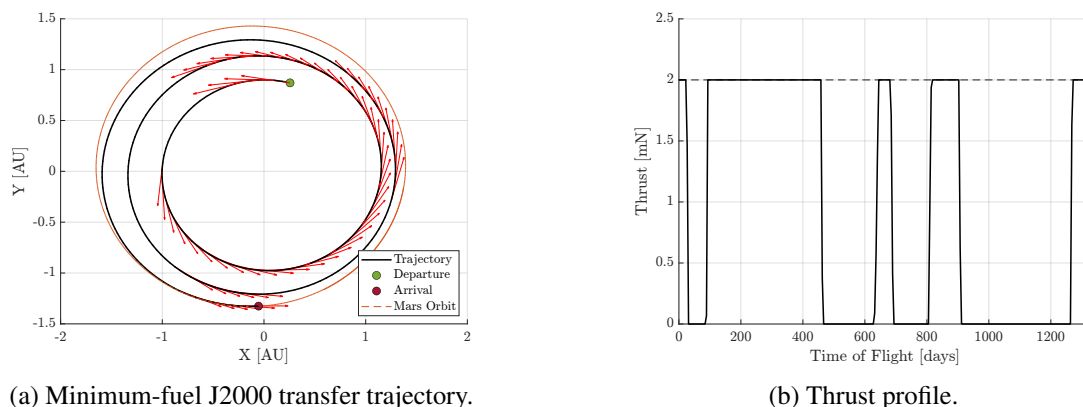


Figure 2: Transfer trajectory and thrust profile for a representative Earth-to-Mars case.

Figure 3 summarizes the aggregate performance of the tested solvers across different problem sizes. The benchmark set used in this comparison includes scenarios for which at least one solver converged for at least one tested discretization. The convergence rate in Figure 3a is computed over this filtered benchmark set, while the propellant mass, SCP iteration count, and computation time are reported only for converged runs.

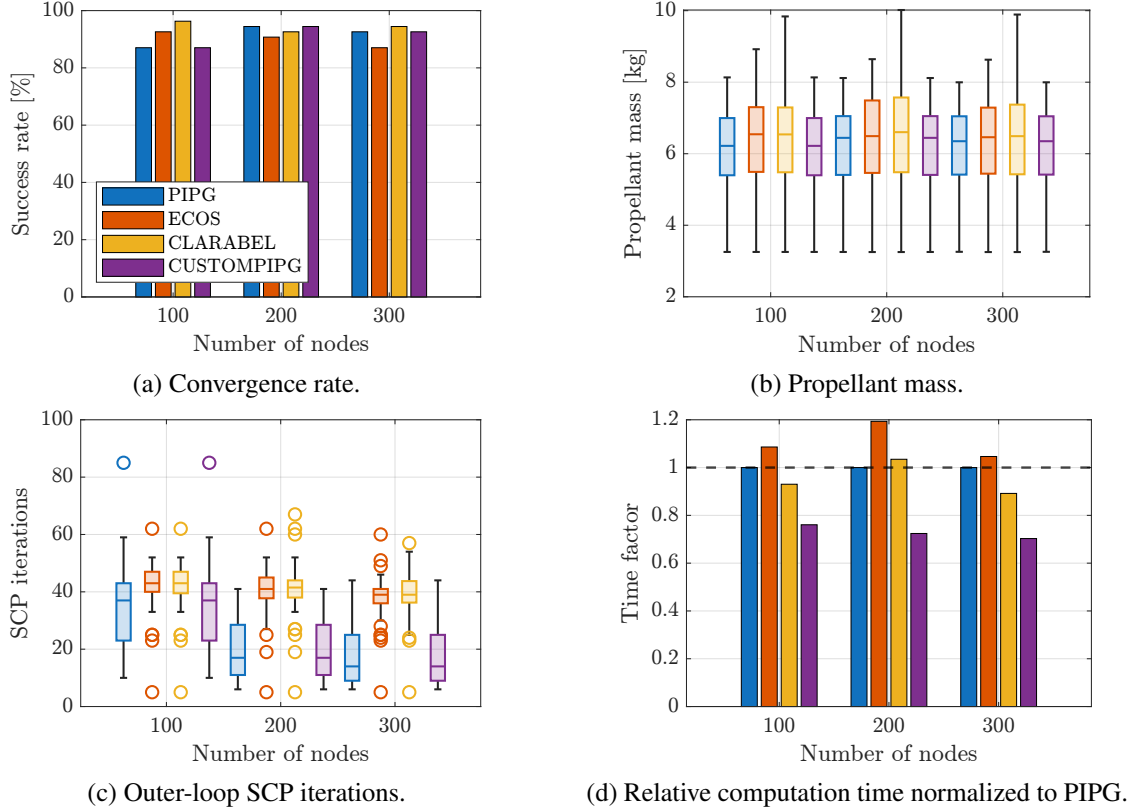


Figure 3: Solver comparison on the ZCU102 board for the considered interplanetary scenarios.

Figure 3a shows that all solvers converge for a large fraction of the tested cases. The PIPG-based solvers maintain high convergence rates as the number of discretization nodes increases, indicating that the first-order approach remains effective for the larger problem sizes considered. Additionally, the customized PIPG implementation does not introduce a loss of robustness with respect to the standard PIPG implementation, since both methods solve the same preconditioned subproblems with the same underlying algorithmic structure.

The final propellant-mass solution distributions in Figure 3b largely overlap across the different solvers and discretization sizes. This indicates that, for the cases where the SCP algorithm converged, the different solvers lead to comparable local solutions of the original nonlinear trajectory optimization problem. Therefore, the computational advantage of the customized PIPG implementation is not obtained at the cost of a visibly different propellant-optimal solution.

Figure 3c reports the number of outer SCP iterations required for convergence. For larger discretizations, the PIPG-based solvers generally require fewer outer SCP iterations than ECOS and CLARABEL. Therefore, for the selected tolerances and SCP acceptance criteria, the PIPG-based updates appear to provide effective corrections within the outer loop, allowing the nonlinear problem

to converge in fewer SCP iterations.

This behavior directly affects the full SCP runtime reported in Figure 3d. The computation time shown in this figure corresponds to the complete SCP solve, including all outer iterations, rather than to a single convex subproblem. Therefore, the reduction in the number of outer SCP iterations contributes to the lower total runtime observed for the PIPG-based solvers. For each discretization size, the computation time of each solver is normalized by the computation time of the standard PIPG implementation. Therefore, values below one indicate a faster complete SCP solve than standard PIPG, while values above one indicate a slower solve time.

The customized PIPG implementation consistently reduces the computation time with respect to the standard PIPG implementation, presenting a speedup of roughly $1.25\times$ – $1.4\times$ compared to the standard PIPG runtime over the tested discretization sizes and scenarios. This reduction is obtained without modifying the mathematical formulation of the convex subproblem or the outer SCP strategy. Instead, it results from exploiting the fixed sparsity and block structure of the discretized optimal-control problem in the matrix-vector products and projection operations.

Therefore, the comparison with `ECOS` and `CLARABEL` reflects both effects: the cost of each convex subproblem solve and the number of outer SCP iterations required to reach convergence. Although `CLARABEL` remains competitive and can be faster than the standard PIPG implementation for some discretization sizes, the customized PIPG implementation provides the lowest average relative computation time across the considered benchmark set.

Overall, these results show that the customized PIPG solver preserves the solution quality and convergence robustness of the reference implementations while providing a consistent reduction in computation time compared to the other off-the-shelf solvers. This makes it a promising candidate for embedded implementation and motivates the FPGA deployment discussed next.

FPGA Deployment Results

The previous results compared software implementations running on the processing system of the ZCU102 board. To assess the feasibility of deploying the customized PIPG solver on the chosen hardware, the inner PIPG loop was also implemented as an FPGA kernel. This implementation is evaluated on a representative Earth-to-Mars transfer with $N = 150$ nodes, which corresponds to the fixed problem size used to generate the current bitstream. The resource cost of the current design is listed in Table 1.

Table 1: Resource utilization of the customized PIPG FPGA kernel on the ZCU102.

Resource	Used	Utilization (%)
LUT	135043	49.27
FF	91975	16.78
BRAM	534.5	58.61
DSP	355	14.09
Clock frequency		100 MHz

Two tests are considered. First, the inner PIPG solve is isolated by forcing the solver to execute the maximum number of PIPG iterations, $j_{\max} = 20000$, on the same convex subproblem. This test removes the outer SCP logic and directly compares the cost of the repeated PIPG operations. The

results are reported in Table 2 where the fixed-iteration test shows that the FPGA implementation significantly reduces the cost of the repeated PIPG loop. The FPGA implementation is approximately $4.91\times$ faster than standard PIPG and approximately $3.89\times$ faster than the custom software implementation. This indicates that, despite the overhead associated with the data transfer between the processing system and the FPGA, the kernel is significantly faster than the software implementation.

Table 2: Fixed-iteration timing comparison for the inner PIPG solve on a representative convex subproblem. All implementations are forced to run for $j_{\max} = 20000$ PIPG iterations.

Implementation	PIPG iterations	Time (ms)	Speedup vs. PIPG
PIPG	20000	14371	$1.00\times$
Custom PIPG	20000	11365	$1.26\times$
Custom PIPG FPGA	20000	2924	$4.91\times$

To verify that the FPGA kernel reproduces the customized software implementation, the final primal and dual iterates obtained after the fixed 20000 PIPG iterations were compared against the custom software PIPG solution. The infinity-norm differences were

$$\Delta z_{\infty} = \|z_{\text{FPGA}} - z_{\text{SW}}\|_{\infty} = 5.00 \times 10^{-15}, \quad \Delta y_{\infty} = \|y_{\text{FPGA}} - y_{\text{SW}}\|_{\infty} = 3.54 \times 10^{-13}$$

These quantities correspond to the largest entry-wise differences between the final FPGA and software iterates. Their magnitude is close to double-precision round-off level, indicating that the FPGA implementation reproduces the custom PIPG iteration with negligible numerical discrepancy for this fixed-iteration test.

The second test embeds the FPGA kernel inside the complete SCP loop. In this case, the reported computation time includes the complete nonlinear solve, including all outer SCP iterations and the repeated calls to the solver. Table 3 reports the results for the representative Earth-to-Mars scenario.

Table 3: Full SCP performance for the representative Earth-to-Mars scenario with $N = 150$ nodes. The reported time corresponds to the complete SCP solve.

Solver	SCP iterations	Time (s)	Propellant (kg)	Speedup vs. ECOS
ECOS	39	75.903	5.08320	$1.00\times$
PIPG	11	45.054	4.92782	$1.68\times$
Custom PIPG	11	35.358	4.92782	$2.15\times$
Custom PIPG FPGA	11	13.489	4.92782	$5.63\times$

When integrated within the full SCP loop, the FPGA implementation converges in the same number of outer SCP iterations as the software PIPG implementations and returns the same final propellant mass. This indicates that the FPGA kernel preserves the convergence behavior of the customized PIPG solver for the considered representative case. The total SCP runtime is reduced from 45.054 s for standard PIPG and 35.358 s for custom software PIPG to 13.489 s with the FPGA implementation. This corresponds to a speedup of approximately $3.34\times$ with respect to standard PIPG and $2.62\times$ with respect to the custom software implementation.

The comparison with ECOS further highlights the combined effect of the FPGA acceleration on the customized PIPG implementation. For this scenario, the FPGA implementation achieves a speedup of approximately $5.63\times$ with respect to ECOS. In this case, the FPGA implementation benefits both from the lower number of SCP iterations observed for the PIPG-based solvers and accelerates the repeated inner PIPG operations through hardware execution.

These results demonstrate that the customized PIPG solver can be deployed on FPGA hardware while preserving the numerical behavior of the software implementation for a representative trajectory-optimization problem. The current implementation uses double-precision arithmetic and a bitstream generated for a fixed discretization size, $N = 150$. Therefore, the result should be interpreted as a feasibility demonstration rather than a complete FPGA benchmark. Further improvements are expected from reducing host-kernel transfer overhead, increasing the level of parallelism inside the hardware kernel, and investigating mixed-precision implementations.

CONCLUSIONS

This paper presented a customized PIPG-based solver for the solution of convex subproblems arising in sequential convex programming for low-thrust interplanetary trajectory optimization. The formulation exploits the structure of the discretized optimal-control problem, including the block sparsity of the dynamics constraints and the simple projection structure of the admissible set.

The proposed solver was compared against ECOS, CLARABEL, and a standard PIPG implementation over a benchmark set. The results show that the customized PIPG implementation preserves the solution quality of the reference solvers, with similar propellant-mass distributions for converged runs. The PIPG-based methods also required fewer outer SCP iterations for the larger discretizations considered. When computation time was compared, the customized PIPG solver achieved the lowest average runtime over the considered benchmark set and solvers.

Finally, the feasibility of hardware deployment was demonstrated by implementing the customized PIPG iteration as an FPGA kernel on the ZCU102 board. For a representative Earth-to-Mars transfer with $N = 150$ nodes, the FPGA implementation reproduced the custom software PIPG iterates with negligible numerical discrepancy and reduced the isolated fixed-iteration PIPG runtime by $4.91\times$ with respect to standard PIPG. When embedded in the complete SCP loop, the FPGA implementation presented a speedup factor of approximately $5.63\times$ when compared to ECOS. These results indicate that structure-exploiting first-order methods are promising candidates for embedded onboard guidance, although further work is required to extend the FPGA benchmark set, reduce host-kernel transfer overhead, and investigate mixed-precision implementations.

ACKNOWLEDGMENT

This research is part of EXTREMA, a project that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (Grant Agreement No. 864697).

REFERENCES

- [1] J. Schoolcraft, A. Klesh, and T. Werne, “MarCO: interplanetary mission development on a CubeSat scale,” *Space Operations: Contributions from the Global Community*, pp. 221–231, Springer, 2017.
- [2] F. Topputo, Y. Wang, C. Giordano, V. Franzese, H. Goldberg, F. Perez-Lissi, and R. Walker, “Envelop of reachable asteroids by M-ARGO CubeSat,” *Advances in Space Research*, Vol. 67, No. 12, 2021, pp. 4193–4221.
- [3] V. Fogliano, R. Walker, S. Simonetti, F. d. S. P. Cabral, G. Ambrosio, O. Karatekin, B. Ritter, C. Güttler, and K. Soons, “SATIS: a mission study for a deep-space CubeSat to observe (99942) Apophis, a near-Earth potential hazardous asteroid, before, during, and after the Earth fly-by,” *Small Satellites Systems and Services Symposium (4S 2024)*, Vol. 13546, SPIE, 2025, pp. 692–706.
- [4] A. C. Morelli, C. Hofmann, and F. Topputo, “Robust Low-Thrust Trajectory Optimization Using Convex Programming and a Homotopic Approach,” *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 58, No. 3, 2022, pp. 2103–2116, 10.1109/TAES.2021.3128869.
- [5] Z. Wang and M. J. Grant, “Minimum-Fuel Low-Thrust Transfers for Spacecraft: A Convex Approach,” *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 54, No. 5, 2018, pp. 2274–2290, 10.1109/TAES.2018.2812558.
- [6] A. Domahidi, E. Chu, and S. Boyd, “ECOS: An SOCP solver for embedded systems,” *2013 European Control Conference (ECC)*, 2013, pp. 3071–3076, 10.23919/ECC.2013.6669541.
- [7] P. J. Goulart and Y. Chen, “Clarabel: An interior-point solver for conic programs with quadratic objectives,” 2024.
- [8] A. G. Kamath, P. Elango, T. Kim, S. Mceowen, Y. Yu, J. M. Carson, M. Mesbahi, and B. Acikmese, “Customized Real-Time First-Order Methods for Onboard Dual Quaternion-based 6-DoF Powered-Descent Guidance,” *AIAA SCITECH 2023 Forum*, 01 2023, 10.2514/6.2023-2003.
- [9] A. G. Kamath, P. Elango, Y. Yu, S. Mceowen, G. M. Chari, J. M. Carson III, and B. Açıkmeşe, “Real-Time Sequential Conic Optimization for Multi-Phase Rocket Landing Guidance,” *IFAC-PapersOnLine*, Vol. 56, No. 2, 2023, pp. 3118–3125. 22nd IFAC World Congress, <https://doi.org/10.1016/j.ifacol.2023.10.1444>.
- [10] C. Hofmann, A. C. Morelli, and F. Topputo, “Performance assessment of convex low-thrust trajectory optimization methods,” *Journal of Spacecraft and Rockets*, Vol. 60, No. 1, 2023, pp. 299–314.
- [11] T. Reynolds, D. Malyuta, M. Mesbahi, B. Acikmese, and J. M. Carson, “A Real-Time Algorithm for Non-Convex Powered Descent Guidance,” *AIAA Scitech 2020 Forum*, 2020, 10.2514/6.2020-0844.
- [12] D. Malyuta, T. P. Reynolds, M. Szmuk, T. Lew, R. Bonalli, M. Pavone, and B. Açıkmeşe, “Convex Optimization for Trajectory Generation: A Tutorial on Generating Dynamically Feasible Trajectories Reliably and Efficiently,” *IEEE Control Systems Magazine*, Vol. 42, No. 5, 2022, pp. 40–113, 10.1109/MCS.2022.3187542.
- [13] Y. Yu, P. Elango, and B. Açıkmeşe, “Proportional-Integral Projected Gradient Method for Model Predictive Control,” *IEEE Control Systems Letters*, Vol. PP, 12 2020, pp. 1–1, 10.1109/LCSYS.2020.3044977.
- [14] Y. Yu, K. Nagpal, S. Mceowen, B. Açıkmeşe, and U. Topcu, “Real-time quadrotor trajectory optimization with time-triggered corridor constraints,” *Journal of Guidance, Control, and Dynamics*, Vol. 46, No. 6, 2023, pp. 1197–1205.
- [15] H. Bauschke and P. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Space*. Springer, 01 2011, 10.1007/978-1-4419-9467-7.
- [16] H. H. Bauschke and J. M. Borwein, “On projection algorithms for solving convex feasibility problems,” *SIAM review*, Vol. 38, No. 3, 1996, pp. 367–426.
- [17] A. G. Kamath, P. Elango, and B. Açıkmeşe, “Optimal Preconditioning for Online Quadratic Cone Programming,” *IEEE Control Systems Letters*, Vol. 9, 2025, pp. 108–113, 10.1109/LCSYS.2025.3563219.
- [18] J. Wilkinson, “The algebraic eigenvalue problem,” *Handbook for automatic computation*, Vol. 2, 1971, p. 191.
- [19] Y. Yu, P. Elango, B. Açıkmeşe, and U. Topcu, “Extrapolated Proportional-Integral Projected Gradient Method for Conic Optimization,” *IEEE Control Systems Letters*, Vol. 7, 2023, pp. 73–78, 10.1109/LC-SYS.2022.3186647.
- [20] D. Dueri, B. Açıkmeşe, D. P. Scharf, and M. W. Harris, “Customized real-time interior-point methods for onboard powered-descent guidance,” *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2017, pp. 197–212.
- [21] Z. Wu, B. Plancher, I. McInerney, H. K.-H. So, M. Wang, and K.-T. Cheng, “MPC Solver Hardware Generation Framework with Model-Specific Operation Fusion and Pruning,” *2025 International Conference on Field Programmable Technology (ICFPT)*, IEEE, 2025, pp. 239–247.
- [22] Advanced Micro Devices, Inc., *Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit*. AMD, San Jose, CA, 2026. Part No. EK-U1-ZCU102-G.