Robotics: Science and Systems 2026
Sydney, Australia, July 13–17, 2026

cuNRTO: GPU-Accelerated Nonlinear Robust Trajectory Optimization

Jiawei Wang[†§], Arshiya Taj Abdul[*§], Evangelos A. Theodorou[*]
[*]Georgia Institute of Technology, Atlanta [†]University of California, San Diego [‡]Deemos Corporation
[§]Equal Contribution

***Abstract*—Robust trajectory optimization enables autonomous systems to operate safely under uncertainty by computing control policies that satisfy the constraints for all bounded disturbances. However, these problems often lead to large Second Order Conic Programming (SOCP) constraints, which are computationally expensive. In this work, we propose the CUDA Nonlinear Robust Trajectory Optimization (cuNRTO) framework by introducing two dynamic optimization architectures that have direct application to robust decision-making and are implemented on CUDA. The first architecture, NRTO-DR, leverages the Douglas-Rachford (DR) splitting method to solve the SOCP inner subproblems of NRTO, thereby significantly reducing the computational burden through parallel SOCP projections and sparse direct solves. The second architecture, NRTO-FullADMM, is a novel variant that further exploits the problem structure to improve scalability using the Alternating Direction Method of Multipliers (ADMM). Finally, we provide GPU implementations of the proposed methodologies using custom CUDA kernels for SOC projection steps and cuBLAS GEMM chains for feedback gain updates. We validate the performance of cuNRTO through simulated experiments on unicycle, quadcopter, and Franka manipulator models, demonstrating speedups of up to 139.6×. More details are available at cunrto.github.io.**

I. Introduction

Trajectory optimization has become a foundational tool for motion planning and control of robotic systems, enabling robots to compute dynamically feasible paths that minimize objective cost while satisfying constraints [1, 2]. Applications span from manipulation and legged locomotion to aerial vehicles and autonomous driving [3, 4, 5]. However, real-world robotic systems inevitably face uncertainty. When constraints encode safety-critical requirements—such as obstacle avoidance or actuator limits—failing to account for these uncertainties can lead to catastrophic consequences [6].

Handling uncertainty is critical for real-world deployment. One common approach is to model uncertainty as a stochastic disturbance characterized by a probability distribution. Existing methods, ranging from chance-constrained optimization [6, 7] to covariance steering [8, 9], provide probabilistic guarantees on constraint satisfaction. However, stochastic models may not capture all uncertainty types; modeling inaccuracies or exogenous disturbances are often better represented as deterministic uncertainty, referring to disturbances assumed to lie within a bounded set. In many safety-critical applications, a guarantee is required for *all* possible realizations of such uncertainty. This work addresses trajectory optimization in the presence of these deterministic disturbances.

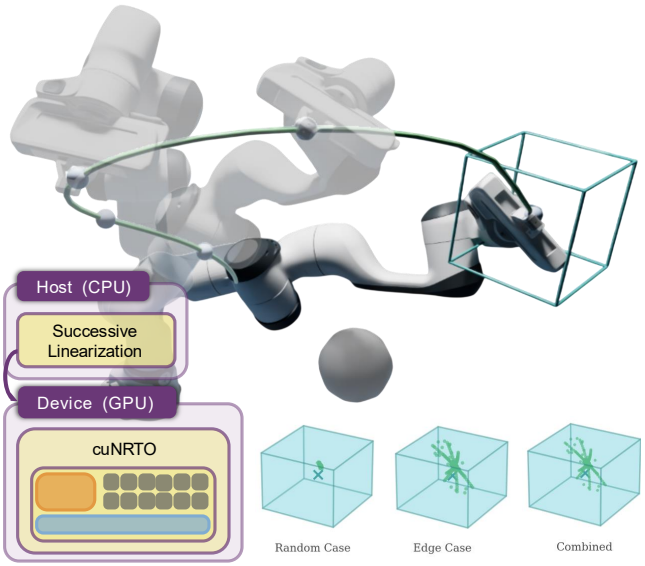


Fig. 1: **cuNRTO on a 7-DoF Franka manipulator:** cuNRTO involves an outer successive linearization (SL) loop run on the host CPU, with an inner loop executed on the GPU. Compared to NRTO, cuNRTO achieves a 25.9× wall-clock speedup on this setting with 100% constraint satisfaction. The three small boxes show the final state under Monte Carlo rollouts.

Standard approaches to address deterministic uncertainty involve providing worst-case guarantees over bounded uncertainty sets, a concept originating from the field of robust control [10]. This idea has been adapted for trajectory optimization in several forms, ranging from Tube-based and Robust Model Predictive Control (MPC) to min-max methods. Tube-based MPC computes a nominal trajectory and feedback gains to maintain the system within a safe tube; but can be overly conservative due to the decoupling between the nominal control and the feedback gains [11]. Robust MPC typically considers linear systems, and Min-Max DDP [12, 13] frame the problem as a game against an adversarial disturbance to optimize a robust policy, failing to address nonlinear state constraints. While DIRTREL [14] studies robust nonlinear direct transcription under ellipsoidal disturbances with an LQR tracking controller, it assumes uncorrelated uncertainty across the time-steps. To effectively handle both nonlinear dynamics and state constraints, recent research has shifted toward Robust Optimization (RO) [15, 16]. Some frameworks [17, 18] in this

category address nonlinear constraints but are limited to linear dynamical systems. Conversely, recent work [19] introduced the Nonlinear Robust Trajectory Optimization (NRTO) framework, a trajectory optimization framework that accounts for nonlinearity in both the dynamics and the constraints.

The NRTO framework comprises three core elements: successive linearization, robust constraint reformulation, and the resolution of linearized subproblem infeasibility via operator-splitting techniques [19]. Since the framework enforces robust constraints that must be satisfied for every possible realization of uncertainty within a bounded set, the problem is intractable in its original form due to the infinite number of constraints. To overcome this, the NRTO framework employs robust constraint reformulation, converting the problem into a tractable form that yields Second-Order Conic Programming (SOCP) constraints. Despite the effectiveness of this framework, the computational cost of the existing implementation based on Interior Point (IP) methods for solving the SOCP subproblems remains a primary barrier to its widespread adoption for high-dimensional systems with numerous constraints.

Leveraging GPU acceleration offers a powerful solution to overcome these computational barriers. As single-core CPU performance has largely plateaued, the robotics community has increasingly turned to massively parallel architectures to accelerate complex motion planning tasks. Early research in this domain focused primarily on parallelizing rigid body dynamics and their corresponding gradients [20, 21]. More recently, GPU acceleration has been pushed into the optimization loop itself. For example, MPCGPU [22] achieves real-time nonlinear MPC by accelerating the dominant sparse Newton solve (PCG) on the GPU. Similar GPU-parallelism has also been explored in constrained sampling-based planning [23].

Beyond nonlinear MPC, GPU implementations of first-order splitting methods for convex programs, particularly ADMM-based QP solvers such as OSQP [24, 25], achieve high throughput by mapping the repeated linear algebra and projection steps to GPU kernels while keeping iterates on-device. However, the NRTO framework in its current form is not inherently amenable to parallelization.

This paper presents cuNRTO: a GPU-accelerated nonlinear robust optimization framework by introducing two novel extensions to the original NRTO formulation. This enables efficient parallel execution. Fig. 1 provides a high-level overview of the cuNRTO execution flow. Specifically, the contributions of this paper are three-fold:

i) First, we introduce **NRTO-DR**, a minimal-change methodology to solve the computationally expensive sub-problems of the NRTO framework [19] using Douglas-Rachford splitting (DR) [26]. The proposed methodology improves the performance over Interior Point (IP) Methods through parallel SOC projections and sparse direct solves that reduce matrix operations.

ii) Second, we propose **NRTO-FullADMM**, a novel variant of the NRTO framework that could fully leverage GPU acceleration. By restructuring the inner ADMM update blocks, this framework integrates SOCP projections directly into the inner loop eliminating extra DR layer and host-device communication overhead.

iii) Third, we develop **cuNRTO**, which constitutes GPU implementation of the NRTO-DR and NRTO-FullADMM. Our approach utilizes cuBLAS GEMM chains for the feedback gain update and custom CUDA kernels for SOC projections. We validate our approach on unicycle, quadcopter, and Franka manipulator trajectory optimization, demonstrating a speedup of up to 139.60×.

A. Notations

The space of symmetric positive definite (semi-definite) matrices with dimension n is denoted as $\mathbb{S}_n^{++}$ ($\mathbb{S}_n^{+}$). The 2-norm of a vector $\boldsymbol{x}$ is denoted with $\|\boldsymbol{x}\|_2$, while the Frobenius norm of a matrix $\mathbf{X}$ is given by $\|\mathbf{X}\|_F$. The indicator function of a set X, $\mathcal{I}_X$ is defined as $\mathcal{I}(\mathbf{x}) = \{0 \text{ if } \boldsymbol{x} \in X \text{ or } +\infty \text{ if } \boldsymbol{x} \notin X\}$. With $[\![a, b]\!]$, we denote the integer set $\{[a, b] \cap \mathbb{Z}\}$.

B. Organization of the paper

The remainder of this paper is organized as follows. Section II presents the problem statement and a detailed overview of the Nonlinear Robust Trajectory Optimization (NRTO) framework [19]. In Sections III and IV, we introduce our core architectural contributions: NRTO-DR and NRTO-FullADMM, respectively. Section V demonstrates the computational performance of these proposed frameworks through a series of simulation experiments involving unicycle, quadcopter, and Franka Emika manipulator models. Finally, Section VI provides concluding remarks and discusses future work.

II. Nonlinear Robust Trajectory Optimization Framework

In this section, we outline the nonlinear robust trajectory optimization framework (NRTO) [19] considered in this work. We start by presenting the problem statement followed by framework details.

A. Problem Statement

Consider the following discrete-time nonlinear dynamics

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{d}_k, \quad k \in [\![0, T-1]\!], \tag{1}$$

$$\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0 + \bar{\boldsymbol{d}}_0, \tag{2}$$

where $\boldsymbol{x}_k \in \mathbb{R}^{n_x}$ is the state, $\boldsymbol{u}_k \in \mathbb{R}^{n_u}$ is the control input, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is the known dynamics function and T is the time horizon. The initial state $\boldsymbol{x}_0$ consists of a known part $\bar{\boldsymbol{x}}_0$, as well as an uncertain part $\bar{\boldsymbol{d}}_0$. The terms $\bar{\boldsymbol{d}}_0, \boldsymbol{d}_k \in \mathbb{R}^{n_x}$ represent *unknown disturbances*. Considering $\boldsymbol{\zeta} = [\bar{\boldsymbol{d}}_0; \boldsymbol{d}_0; \ldots; \boldsymbol{d}_{T-1}] \in \mathbb{R}^{(T+1)n_x}$, the uncertainty vector $\boldsymbol{\zeta}$ is characterized to lie inside a bounded ellipsoidal set defined as follows

$$\begin{aligned}\mathcal{U}[\tau] = \{\boldsymbol{\zeta} \mid \exists (\boldsymbol{z} \in \mathbb{R}^{n_z}, \tau \in \mathbb{R}^{+}) : \\ \boldsymbol{\zeta} = \boldsymbol{\Gamma}\boldsymbol{z},\ \boldsymbol{z}^{\mathrm{T}}\mathbf{S}\boldsymbol{z} \leq \tau\},\end{aligned} \tag{3}$$

where $\boldsymbol{\Gamma} \in \mathbb{R}^{(T+1)n_x \times n_z}$, and $\mathbf{S} \in \mathbb{S}_{n_z}^{++}$. Further, the proposed methodology can be extended for other common types of uncertainty sets such as ellitopes, polytopes, etc. [15].

Consider affine control policies of the following form

$$\boldsymbol{u}_k = \bar{\boldsymbol{u}}_k + \mathbf{K}_k \boldsymbol{d}_{k-1} \tag{4}$$

where $\bar{\boldsymbol{u}}_k \in \mathbb{R}^{n_u}$ is feed-forward control and $\mathbf{K}_k \in \mathbb{R}^{n_u \times n_x}$ are feedback gain. By convention, we set $\boldsymbol{d}_{-1} = \bar{\boldsymbol{d}}_0$.

The system is subject to *robust* state and control constraints that should be satisfied for all possible disturbance realizations $\boldsymbol{\zeta}$ within the uncertainty set $\mathcal{U}[\tau]$, which are defined as

$$\boldsymbol{g}(\boldsymbol{x}; \boldsymbol{\zeta}) \leq 0, \quad \forall \boldsymbol{\zeta} \in \mathcal{U}[\tau], \tag{5}$$

$$\boldsymbol{h}(\boldsymbol{u}; \boldsymbol{\zeta}) \leq 0, \quad \forall \boldsymbol{\zeta} \in \mathcal{U}[\tau], \tag{6}$$

where $\boldsymbol{x} = [\boldsymbol{x}_0; \boldsymbol{x}_1; \ldots; \boldsymbol{x}_T]$, $\boldsymbol{u} = [\boldsymbol{u}_0; \boldsymbol{u}_1; \ldots; \boldsymbol{u}_{T-1}]$, $\boldsymbol{g} : \mathbb{R}^{(T+1)n_x} \to \mathbb{R}^{n_g}$, $\boldsymbol{h} : \mathbb{R}^{Tn_u} \to \mathbb{R}^{n_h}$. Further, g_i is concave or linear, and h_i is linear.

Subsequently, the nonlinear robust trajectory optimization problem addressed in this work is presented.

Problem 1 (Robust Trajectory Optimization Problem)**.** *Find the optimal control policy* $\{\bar{\boldsymbol{u}}_k, \mathbf{K}_k\}_{k=0}^{T-1}$ *such that*

$$\begin{aligned} \min_{\{\bar{\boldsymbol{u}}_k, \mathbf{K}_k\}_{k=0}^{T-1}} \quad & \sum_{k=0}^{T-1} \bar{\boldsymbol{u}}_k^{\mathrm{T}} \mathbf{R}_u^k \bar{\boldsymbol{u}}_k + \|\mathbf{R}_K^k \mathbf{K}_k\|_F^2 \\ \textit{s.t.} \quad & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{d}_k, \quad \boldsymbol{x}_0 = \bar{\boldsymbol{x}}_0 + \bar{\boldsymbol{d}}_0, \\ & \boldsymbol{g}(\boldsymbol{x}; \boldsymbol{\zeta}) \leq 0, \quad \boldsymbol{h}(\boldsymbol{u}; \boldsymbol{\zeta}) \leq 0, \quad \forall \boldsymbol{\zeta} \in \mathcal{U}[\tau]. \end{aligned}$$

B. NRTO Algorithm

In this section, we provide an outline of the NRTO framework [19], which integrates successive linearization, robust constraint reformulation, and Alternating Direction Method of Multipliers (ADMM) [24]. For the simplicity of analysis, we present the problem in terms of the variable $\boldsymbol{k}_v \in \mathbb{R}^{Tn_u n_x}$ defined as $\boldsymbol{k}_v = [\boldsymbol{k}_{v,0}; \boldsymbol{k}_{v,1}; \ldots; \boldsymbol{k}_{v,T-1}]$ with $\boldsymbol{k}_{v,k} = \mathrm{vec}(\mathbf{K}_k)$.

NRTO is a bi-level algorithm as shown in Fig. 2. In the outer loop of the algorithm, the problem is linearized around a nominal trajectory $\hat{\boldsymbol{u}}, \hat{\boldsymbol{x}}$ and converted into a tractable form. This results in Second Order Conic Programming (SOCP) constraints as shown below.

Problem 2 (Tractable Linearized Problem)**.** *Find the optimal decision variables* $\delta\hat{\boldsymbol{u}}, \boldsymbol{k}_v, \boldsymbol{p}$ *such that*

$$\begin{aligned} \min_{\delta\hat{\boldsymbol{u}}, \boldsymbol{k}_v, \boldsymbol{p}} \quad & \mathcal{Q}_{\hat{u}}(\delta\hat{\boldsymbol{u}}) + \tilde{\mathcal{Q}}(\boldsymbol{k}_v) \\ \textit{s.t.} \quad & \boldsymbol{g}^{lin,1}(\delta\hat{\boldsymbol{u}}, \boldsymbol{p}) \leq 0, \\ & \|\hat{\mathbf{A}}_j \boldsymbol{k}_v + \hat{\boldsymbol{b}}_j\|_2 \leq p_j \quad j \in [\![1, n_g]\!], \\ & \|\mathbf{F}_u \delta\hat{\boldsymbol{u}}\|_2 \leq r_{trust}. \end{aligned}$$

where the functions $\mathcal{Q}_{\hat{u}}(\delta\hat{\boldsymbol{u}})$, $\tilde{\mathcal{Q}}(\boldsymbol{k}_v)$, $\boldsymbol{g}^{lin,1}(\delta\hat{\boldsymbol{u}}, \boldsymbol{p})$, *the matrices* $\hat{\mathbf{A}}_j \in \mathbb{R}^{n_z \times Tn_u n_x}$, $\hat{\boldsymbol{b}}_j \in \mathbb{R}^{n_z}$, $\mathbf{F}_u \in \mathbb{R}^{(T+1)n_x \times Tn_u}$ *are defined in Section I of Supplementary Material (SM). Further,* $r_{trust} \in \mathbb{R}^+$ *is the trust region radius.*

This linearized problem can be infeasible due to linearization, especially during the initial iterations of the algorithm. Thus, the problem is solved indirectly using ADMM in the inner loop (refer to Algorithm 1 of [19] for details). For which,

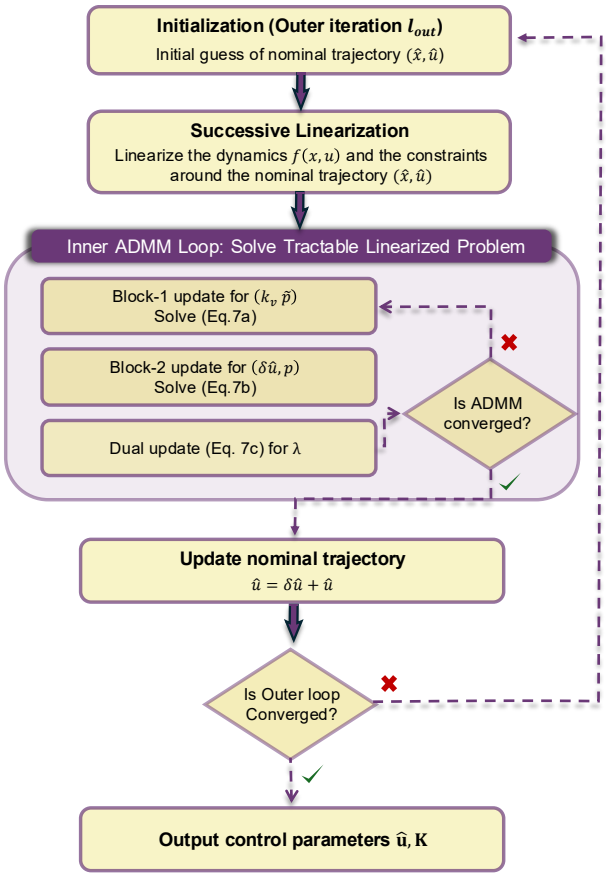


Fig. 2: **Overview of NRTO Framework:** A bi-level structure involving an outer successive linearization (SL) loop to generate tractable linearized problem, and an inner ADMM loop to solve the resulting linearized problem.

Problem 2 is converted to the following form solvable using ADMM, by introducing a slack variable $\tilde{p}$,

Problem 3 (Tractable Linearized Problem - ADMM form)**.** *Find the optimal decision variables* $\delta\hat{\boldsymbol{u}}, \boldsymbol{k}_v, \boldsymbol{p}, \tilde{\boldsymbol{p}}$ *such that*

$$\begin{aligned} \min_{\delta\hat{\boldsymbol{u}}, \mathbf{K}, \boldsymbol{p}, \tilde{\boldsymbol{p}}} \quad & \mathcal{H}_p(\delta\hat{\boldsymbol{u}}, \boldsymbol{p}) + \mathcal{H}_{\tilde{p}}(\boldsymbol{k}_v, \tilde{\boldsymbol{p}}) \\ \textit{s.t.} \quad & \boldsymbol{p} = \tilde{\boldsymbol{p}} \end{aligned}$$

where $\mathcal{H}_p(\delta\hat{\boldsymbol{u}}, \boldsymbol{p})$, $\mathcal{H}_{\tilde{p}}(\boldsymbol{k}_v, \tilde{\boldsymbol{p}})$ *can be referred from [19].*

The above problem is solved using ADMM by considering the variables $\{\boldsymbol{k}_v, \tilde{\boldsymbol{p}}\}$ as the first block, and $\{\delta\hat{\boldsymbol{u}}, \boldsymbol{p}\}$ as the second block of ADMM. Each iteration of the ADMM (indexed by l_{in}) involves sequential minimization of the Augmented Lagrangian (AL) of the problem with respect to each block variables, followed by a dual update [24], given as follows -

$$\{\boldsymbol{k}_v, \tilde{\boldsymbol{p}}\}^{l_{\text{in}}} = \underset{\boldsymbol{k}_v, \tilde{\boldsymbol{p}}}{\operatorname{argmin}} \, \mathcal{L}_\rho(\{\delta\hat{\boldsymbol{u}}, \boldsymbol{p}\}^{l_{\text{in}}-1}, \boldsymbol{k}_v, \tilde{\boldsymbol{p}}; \boldsymbol{\lambda}^{l_{\text{in}}-1}) \tag{7a}$$

$$\{\delta\hat{\boldsymbol{u}}, \boldsymbol{p}\}^{l_{\text{in}}} = \underset{\delta\hat{\boldsymbol{u}}, \boldsymbol{p}}{\operatorname{argmin}} \, \mathcal{L}_\rho(\delta\hat{\boldsymbol{u}}, \boldsymbol{p}, \{\boldsymbol{k}_v, \tilde{\boldsymbol{p}}\}^{l_{\text{in}}}; \boldsymbol{\lambda}^{l_{\text{in}}-1}) \tag{7b}$$

$$\boldsymbol{\lambda}^{l_{\text{in}}} = \boldsymbol{\lambda}^{l_{\text{in}}-1} + \rho(\boldsymbol{p}^{l_{\text{in}}-1} - \tilde{\boldsymbol{p}}^{l_{\text{in}}-1}) \tag{7c}$$

where the AL of the problem $\mathcal{L}_\rho(\delta\hat{\boldsymbol{u}}, \boldsymbol{p}, \boldsymbol{k}_v, \tilde{\boldsymbol{p}}; \boldsymbol{\lambda}) = \mathcal{H}_p(\delta\hat{\boldsymbol{u}}, \boldsymbol{p}) + \mathcal{H}_{\tilde{p}}(\boldsymbol{k}_v, \tilde{\boldsymbol{p}}) + \boldsymbol{\lambda}^{\mathrm{T}}(\boldsymbol{p} - \tilde{\boldsymbol{p}}) + \frac{\rho}{2}\|\boldsymbol{p} - \tilde{\boldsymbol{p}}\|_2^2$, with $\boldsymbol{\lambda} \in \mathbb{R}^{n_g}$

being the dual variable for the constraint $\boldsymbol{p} = \tilde{\boldsymbol{p}}$ and $\rho > 0$ as the penalty parameter. For the complete details of the NRTO algorithm, the reader is referred to [19].

C. NRTO-LE Algorithm

This section outlines NRTO-LE [19], an extension of the NRTO framework developed to address linearization error. While the standard NRTO algorithm ensures that a linearized trajectory satisfies all the constraints, it may not guarantee the safety of the actual trajectory when the linearization error is significant. To address this, NRTO-LE models the linearization error at each time step as a bounded uncertainty lying inside an ellipsoidal set. For the comprehensive derivation of the modification, the reader is referred to [19].

III. NRTO-DR Framework

In this section, we introduce a framework for accelerating the solving of the sub-problem (7a) which is the most computationally expensive update in the inner loop of the NRTO. Subsequently, we provide a GPU version of the framework that could further enhance the computational speed. The design goal of NRTO-DR is to make the smallest algorithmic change to NRTO while replacing the expensive interior-point SOCP subsolve in (7a). By rewriting this block with Douglas-Rachford splitting, the update decomposes into reusable sparse linear solves and independent SOC projections, exposing the parallelism needed for GPU acceleration. We begin by explicitly writing the ADMM update step (7a) as follows

$$\begin{aligned} \min_{\boldsymbol{k}_v, \tilde{\boldsymbol{p}}} \quad & \tilde{\mathcal{Q}}(\boldsymbol{k}_v) + \tfrac{\rho}{2}\left\|\tilde{\boldsymbol{p}} - \boldsymbol{p}^{l_{\text{in}}-1} - \boldsymbol{\lambda}^{l_{\text{in}}-1}/\rho\right\|_2^2 \\ \text{s.t.} \quad & \|\hat{\mathbf{A}}_j \boldsymbol{k}_v + \hat{\boldsymbol{b}}_j\|_2 \le \tilde{p}_j, \qquad j \in [\![1, n_g]\!] \end{aligned} \tag{8}$$

A. Framework

We introduce a framework leveraging Douglas-Rachford Splitting [27, 26] (DR) to reduce the computational complexity of solving the above problem. To achieve this, the problem needs to be transformed to a form solvable by DR method. Let us define $\boldsymbol{\chi} = [\boldsymbol{k}_v; \tilde{\boldsymbol{p}}]$ and a slack variable $\boldsymbol{s}$ such that (8) can be equivalently given as follows -

$$\begin{aligned} \min_{\boldsymbol{\chi}, \boldsymbol{s}} \quad & \tfrac{1}{2}\boldsymbol{\chi}^{\mathrm{T}}\mathbf{P}\boldsymbol{\chi} + \boldsymbol{q}^{\mathrm{T}}\boldsymbol{\chi} \\ \text{s.t.} \quad & \mathbf{A}\boldsymbol{\chi} + \boldsymbol{s} = \mathbf{b}, \qquad \boldsymbol{s} \in \mathcal{K}, \end{aligned} \tag{9}$$

with $\mathbf{P} = \text{blkdiag}(\mathbf{Q}_v, \rho\mathbf{I})$ and a product of SOCs $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_{n_g}$. We provide the explicit construction of $\mathbf{A}, \mathbf{b}$ and $\mathcal{K}_i$ in Section II of SM. The above problem can be rewritten in a form solvable by the DR method as follows -

$$\min_{\boldsymbol{\chi}, \boldsymbol{s}} \quad \mathcal{J}(\boldsymbol{\chi}, \boldsymbol{s}) + \hat{\mathcal{J}}(\boldsymbol{s}) \tag{10}$$

where $\mathcal{J}(\boldsymbol{\chi}, \boldsymbol{s}) = (1/2)\boldsymbol{\chi}^{\mathrm{T}}\mathbf{P}\boldsymbol{\chi} + \boldsymbol{q}^{\mathrm{T}}\boldsymbol{\chi} + \mathcal{I}_{\mathbf{A}\boldsymbol{\chi}+\boldsymbol{s}=\mathbf{b}}(\boldsymbol{\chi}, \boldsymbol{s})$ and $\hat{\mathcal{J}}(\boldsymbol{s}) = \mathcal{I}_{\mathcal{K}}(\boldsymbol{s})$.

For simplicity, let us also define $\boldsymbol{\xi} = [\boldsymbol{\chi}; \boldsymbol{s}]$. The update steps in each iteration of relaxed DR method (indexed as l_{dr}) to solve the above problem can be given as follows [27, 26]

$$\boldsymbol{\xi}^{l_{\text{dr}}} = \text{Prox}_{\mathcal{J}}(\tilde{\boldsymbol{\xi}}^{l_{\text{dr}}-1}) \tag{11a}$$

$$\boldsymbol{\xi}_{\text{ref}}^{l_{\text{dr}}} = 2\boldsymbol{\xi}^{l_{\text{dr}}} - \tilde{\boldsymbol{\xi}}^{l_{\text{dr}}-1} \tag{11b}$$

$$\tilde{\boldsymbol{\xi}}^{l_{\text{dr}}} = \tilde{\boldsymbol{\xi}}^{l_{\text{dr}}-1} + \alpha(\text{Prox}_{\hat{\mathcal{J}}}(\boldsymbol{\xi}_{\text{ref}}^{l_{\text{dr}}}) - \boldsymbol{\xi}^{l_{\text{dr}}}) \tag{11c}$$

where $\alpha \in (0, 1)$ is the relaxation parameter. We will now simplify the above update steps. The update (11a) is a quadratic programming (QP) problem, which can be solved by solving the KKT conditions. This involves solving the following

$$\begin{bmatrix} \boldsymbol{\chi}^{l_{\text{dr}}} \\ \boldsymbol{y}^{l_{\text{dr}}} \end{bmatrix} = (\mathbf{L}\mathbf{L}^{\mathrm{T}})^{-1} \begin{bmatrix} \mathbf{R}_{\chi}\tilde{\boldsymbol{\chi}}^{l_{\text{dr}}-1} - \boldsymbol{q} \\ \mathbf{b} - \tilde{\boldsymbol{s}}^{l_{\text{dr}}-1} \end{bmatrix} \tag{12a}$$

$$\boldsymbol{s}^{l_{\text{dr}}} = \tilde{\boldsymbol{s}}^{l_{\text{dr}}-1} - \mathbf{R}_s^{-1}\boldsymbol{y}^{l_{\text{dr}}} \tag{12b}$$

where $\mathbf{R}_{\chi}, \mathbf{R}_s \succ 0$ are fixed proximal scalings, $\mathbf{L}$ is obtained from the Cholesky decomposition of $\mathbf{K}_{\text{KKT}}$. The matrix $\mathbf{K}_{\text{KKT}}$ and the derivation of above update are provided in Section II of the SM. Since $\mathbf{K}_{\text{KKT}}$ is constant within each iteration of the inner ADMM loop, we factor it once and reuse triangular solves throughout DR.

Now, the update (11c) involves

$$\tilde{\boldsymbol{\chi}}^{l_{\text{dr}}} = \tilde{\boldsymbol{\chi}}^{l_{\text{dr}}-1} + \alpha(\boldsymbol{\chi}_{\text{ref}}^{l_{\text{dr}}} - \boldsymbol{\chi}^{l_{\text{dr}}}) \tag{13a}$$

$$\tilde{\boldsymbol{s}}_j^{l_{\text{dr}}} = \tilde{\boldsymbol{s}}_j^{l_{\text{dr}}-1} + \alpha(\Pi_{\mathcal{K}_j}(\boldsymbol{s}_{\text{ref},j}^{l_{\text{dr}}}) - \boldsymbol{s}^{l_{\text{dr}}}) \quad \forall j \in [\![1, n_g]\!] \tag{13b}$$

where the projection step $\Pi_{\mathcal{K}_j}(\boldsymbol{s}_{\text{ref},j}^{l_{\text{dr}}})$ is defined in Section II of SM, a visual interpretation is provided in Fig. 3.

The DR iterations are terminated when the fixed-point residual (r_{dr}), defined as $r_{\text{dr}}^{l_{\text{dr}}} := \|\tilde{\boldsymbol{s}}^{l_{\text{dr}}} - \tilde{\boldsymbol{s}}^{l_{\text{dr}}-1}\|_2$, is below a specified threshold ε_{dr} or when the maximum iteration count $L_{\max}$ is reached.

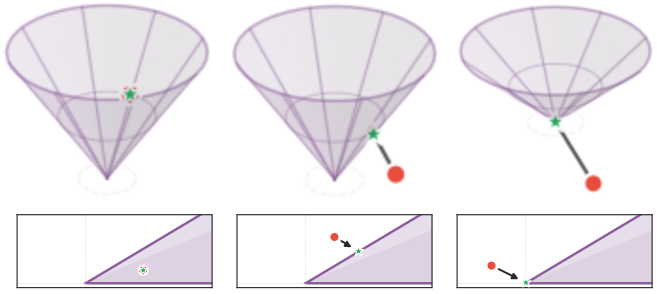


Fig. 3: Three cases of projecting a point $(\hat{t}, \hat{\boldsymbol{y}})$ (red) onto a SOC, illustrated in the $(t, \|\boldsymbol{y}\|_2)$ plane. **Left:** if $\|\hat{\boldsymbol{y}}\|_2 \le \hat{t}$, the point is already feasible and remains unchanged after projection (green). **Middle:** if $\|\hat{\boldsymbol{y}}\|_2 > |\hat{t}|$, the point lies outside the cone and is projected onto the cone boundary, preserving the direction of $\hat{\boldsymbol{y}}$. **Right:** if $\|\hat{\boldsymbol{y}}\|_2 \le -\hat{t}$, the point lies in the opposite cone and the projection collapses to the origin.

B. GPU implementation

The GPU implementation of NRTO-DR (Fig. 4) follows the relaxed DR updates (11a)-(11c). The affine set proximal step (11a) reduces to solving a fixed sparse KKT system with

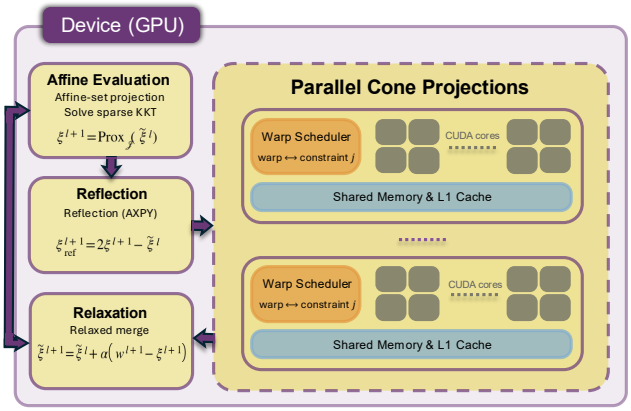


Fig. 4: **GPU execution of one relaxed DR iteration:** Each iteration involves an affine-set projection (11a), a reflection step (11b), and massively parallel second-order cone projections (11c) that are separable across constraints. The right panel illustrates the GPU mapping: each SOCP constraint block is handled by one warp, the warp scheduler dispatches these warps across streaming multiprocessors, and the grey blocks depict the execution lanes (CUDA cores) that run the warp instructions, with shared memory cache supporting fast projection and vector updates.

coefficient matrix $\mathbf{K}_{\text{KKT}}$, which is factored once across DR iterations. The details are as follows:

i) The one-time sparse factorization of $\mathbf{K}_{\text{KKT}}$ and per-iteration triangular solves are performed by cuDSS.
ii) SOC projections are computed by a CUDA kernel. The cone-projection case logic is adapted from the open-source SCS implementation [28].
iii) All vector operations, such as reflection, updates, and residuals, are handled via cuBLAS and kept on-device.

IV. NRTO-FullADMM Framework

In this section, we present a framework that accelerates the solving of Problem 2. Specifically, we propose a novel architecture by modifying the inner ADMM loop of the NRTO framework. We start by rewriting Problem 2 by introducing a slack variable $\boldsymbol{\nu}$, as follows

$$\begin{aligned}\min_{\delta\hat{\boldsymbol{u}},\boldsymbol{k}_v,\boldsymbol{\nu},\boldsymbol{p},\tilde{\boldsymbol{p}}} \quad & \mathcal{Q}_{\hat{u}}(\delta\hat{\boldsymbol{u}}) + \tilde{\mathcal{Q}}(\boldsymbol{k}_v) \\ \text{s.t.} \quad & \boldsymbol{g}^{\text{lin},1}(\delta\hat{\boldsymbol{u}},\boldsymbol{p}) \le 0, \quad \|\mathbf{F}_u\delta\hat{\boldsymbol{u}}\|_2 \le r_{\text{trust}} && \text{(14a)} \\ & \|\boldsymbol{\nu}_j\|_2 \le \tilde{p}_j \quad j \in [\![1, n_g]\!], && \text{(14b)} \\ & \boldsymbol{p} = \tilde{\boldsymbol{p}}, \quad \hat{\mathbf{A}}_j\boldsymbol{k}_v + \hat{\boldsymbol{b}}_j = \boldsymbol{\nu}_j, \ j \in [\![1, n_g]\!] && \text{(14c)}\end{aligned}$$

A. Framework

We propose a framework to solve the above problem using a scaled form of ADMM [24]. The variables $(\boldsymbol{\nu}, \tilde{\boldsymbol{p}})$ are considered as the first block, and $(\delta\hat{\boldsymbol{u}}, \boldsymbol{p}, \boldsymbol{k}_v)$ are as the second block of the inner ADMM loop, with (14c) being the coupling constraints. The AL of the above problem is disclosed in Section III of SM, with the penalty parameter as ρ, and $\rho\boldsymbol{\lambda}_p, \rho\boldsymbol{\lambda}_\nu$ as the dual variables. Subsequently, we present the

Algorithm 1 Inner ADMM Loop - NRTO-FullADMM

Require: $\{\hat{\mathbf{A}}_j, \hat{\boldsymbol{b}}_j\}_{j=1}^{n_g}$, tol ε, max iters $L_{\max}$
Form $\boldsymbol{q}, \mathcal{M}, \{\mathcal{M}_j\}_{j=1}^{n_g}$
Initialize $\boldsymbol{\lambda}_p^0, \boldsymbol{\lambda}_\nu^0, \boldsymbol{k}_v^0, \boldsymbol{p}^0 \leftarrow \mathbf{0}$
for $l_{\text{in}} = 0$ to L_{max} **do**
  Sequentially run (15), (16), (17)
  if ($r_p \le \varepsilon_p$ and $r_d \le \varepsilon_d$) **then break**
end for
return $(\delta\hat{\boldsymbol{u}}, \boldsymbol{k}_v, \boldsymbol{\nu}, \boldsymbol{p}, \tilde{\boldsymbol{p}})$

update steps in the (l_{in}^{th}) iteration, with the detailed derivation disclosed in Section III of SM.

a) Block-1 update: This update can be decoupled with respect to the variables $\{\boldsymbol{\nu}_j, \tilde{p}_j\}_{j=1}^{n_g}$, and is given as follows

$$\begin{aligned}&(\boldsymbol{\nu}_j^{l_{\text{in}}}, \tilde{p}_j^{l_{\text{in}}}) \\ &\quad = \Pi_{\text{SOC}}(\hat{\mathbf{A}}_j\boldsymbol{k}_v^{l_{\text{in}}-1} + \hat{\boldsymbol{b}}_j + \boldsymbol{\lambda}_{\nu,j}^{l_{\text{in}}-1}, p_j^{l_{\text{in}}-1} + \lambda_{p,j}^{l_{\text{in}}-1})\end{aligned} \tag{15}$$

where $\Pi_{\text{SOC}}(\hat{\boldsymbol{y}}, \hat{t})$ represents projection of $(\hat{\boldsymbol{y}}, \hat{t})$ onto the cone $\|\boldsymbol{y}\|_2 \le t$ (shown in Fig. 3) . This projection step is similar to the projection involved in (13b) of NRTO-DR framework.

b) Block-2 update: The variables $(\delta\hat{\boldsymbol{u}}, \boldsymbol{p})$ and $\boldsymbol{k}_v$ are decoupled, with the ADMM update steps given as

$$(\delta\hat{\boldsymbol{u}}^{l_{\text{in}}}, \boldsymbol{p}^{l_{\text{in}}}) = \underset{\delta\hat{\boldsymbol{u}},\boldsymbol{p}}{\text{argmin}}\ \mathcal{Q}_{\hat{u}}(\delta\hat{\boldsymbol{u}}) + \frac{\rho}{2}\|\boldsymbol{p} - \tilde{\boldsymbol{p}}^{l_{\text{in}}} + \boldsymbol{\lambda}_p^{l_{\text{in}}-1}\|_2^2$$

$$\text{s.t. (14a)} \tag{16a}$$

$$\boldsymbol{k}_v^{l_{\text{in}}} = \boldsymbol{q} + \mathcal{M}\boldsymbol{k}_v^{l_{\text{in}}-1} + \bar{\mathcal{M}}\boldsymbol{\nu}^{l_{\text{in}}} \tag{16b}$$

where $\boldsymbol{q}, \mathcal{M}, \bar{\mathcal{M}}$, disclosed in Section III of SM, remain constant throughout the inner ADMM loop. The update (16a) is similar to the update step (7b) of NRTO.

c) Dual update: The update of the regularized dual variables is given as

$$\boldsymbol{\lambda}_p^{l_{\text{in}}} = \boldsymbol{\lambda}_p^{l_{\text{in}}-1} + (\boldsymbol{p}^{l_{\text{in}}} - \tilde{\boldsymbol{p}}^{l_{\text{in}}}) \tag{17a}$$

$$\boldsymbol{\lambda}_{\nu,j}^{l_{\text{in}}} = \boldsymbol{\lambda}_{\nu,j}^{l_{\text{in}}-1} + (\hat{\mathbf{A}}_j\boldsymbol{k}_v^{l_{\text{in}}} + \hat{\boldsymbol{b}}_j - \boldsymbol{\nu}_j^{l_{\text{in}}}) \tag{17b}$$

The inner ADMM loop is terminated when the primal residual $r_p := \|\mathbf{p}^{l_{\text{in}}} - \tilde{\mathbf{p}}^{l_{\text{in}}}\|_2$ and the dual residual $r_d := \rho\|\tilde{\mathbf{p}}^{l_{\text{in}}} - \tilde{\mathbf{p}}^{l_{\text{in}}-1}\|_2$ fall below their respective thresholds, ε_p and ε_d, or when max iterations $L_{\max}$ are reached.

B. GPU Implementation

An overview of the end-to-end on-device inner ADMM loop pipeline for NRTO-FullADMM is shown in Fig. 5.

The details are as follows

1) On-device data layout: We store $\boldsymbol{\nu} \in \mathbb{R}^{n_g \times n_z}$ and $\boldsymbol{\lambda}_\nu \in \mathbb{R}^{n_g \times n_z}$ as contiguous row-major blocks with each row corresponding to one constraint j, and $\boldsymbol{p}, \tilde{\boldsymbol{p}}, \boldsymbol{\lambda}_p \in \mathbb{R}^{n_g}$ as contiguous vectors. All constant matrices $\{\hat{\mathbf{A}}_j, \hat{\boldsymbol{b}}_j\}_{j=1}^{n_g}$ as well as $\boldsymbol{q}, \mathcal{M}, \bar{\mathcal{M}}$ are uploaded once per outer iteration.

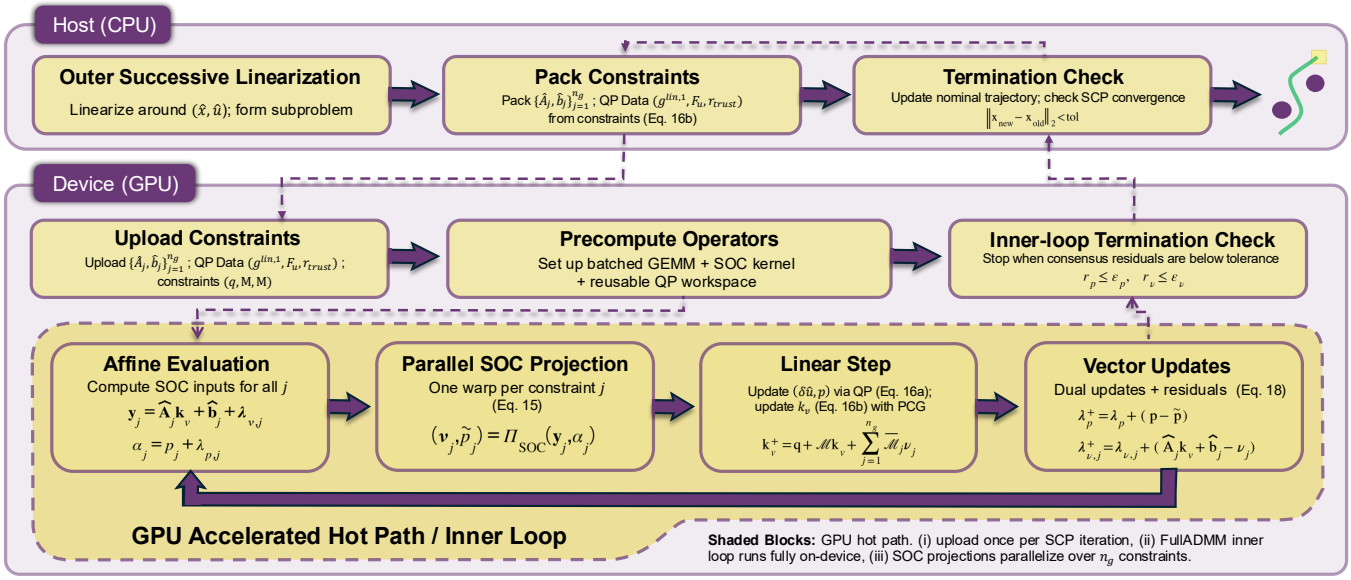


Fig. 5: **cuNRTO pipeline for NRTO-FullADMM:** Each outer SL iteration on the host CPU linearizes the problem, packs the SOCP and QP data, and uploads constants to the GPU once. The FullADMM inner loop runs entirely on-device: (i) batched affine evaluation forms per-constraint inputs, (ii) SOC projections are computed in parallel over j, (iii) Block-2 updates solve the QP and update $\boldsymbol{k}_v$ using prepacked operators, and (iv) dual updates and residual checks determine termination.

2) Block-1: The Block-1 update requires projecting

$$(\hat{\boldsymbol{y}}_j, \hat{\alpha}_j) \;=\; (\hat{\mathbf{A}}_j \boldsymbol{k}_v + \hat{\boldsymbol{b}}_j + \boldsymbol{\lambda}_{\nu,j},\; p_j + \boldsymbol{\lambda}_{p,j}) \;\text{ onto }\; \|\boldsymbol{\nu}_j\|_2 \le \tilde{p}_j.$$

We implement this in two GPU steps. First, affine evaluation. We compute all $\hat{\boldsymbol{y}}_j$ via a sequence of cuBLAS GEMM calls that exploit the structured factorization of $\hat{\mathbf{A}}_j$ through $\mathbf{M}$, $\mathbf{U}_k$, and $\mathbf{B}_k$, avoiding explicit dense matrix formation. Second, SOC projection. We launch a CUDA kernel that performs the SOC projection for each constraint block in parallel. The projection routine follows the standard SOC projection cases and is adapted from the SCS reference implementation [28], with modifications for warp-level execution and our batched memory layout. This step is bandwidth-bound and scales linearly with n_g, it achieves near-ideal parallel efficiency.

3) Block-2: The update (16a) is a convex QP with fixed quadratic term and fixed constraints (14a). We solve it using preconditioned conjugate gradient (PCG) with a Jacobi preconditioner. Crucially, its linear algebra is reusable across inner iterations, so any preconditioner is computed once per outer iteration and reused thereafter. The $\boldsymbol{k}_v$ update (16b) is solved via PCG with matrix-free Hessian-vector products, avoiding explicit formation of the dense Hessian matrix.

4) Dual updates: Dual updates are vector axpy-style operations. We compute the primal residual r_p and dual residual r_d on the GPU and terminate when both fall below the tolerance.

Remark 1. *NRTO-FullADMM scales effectively over NRTO and NRTO-DR through two key innovations. First, it replaces the nested structure of NRTO-DR by directly integrating the parallel SOCP block projections into the inner ADMM updates. Second, it eliminates CPU-GPU data transfer bottlenecks by executing the entire inner ADMM loop on the GPU.*

V. Simulation

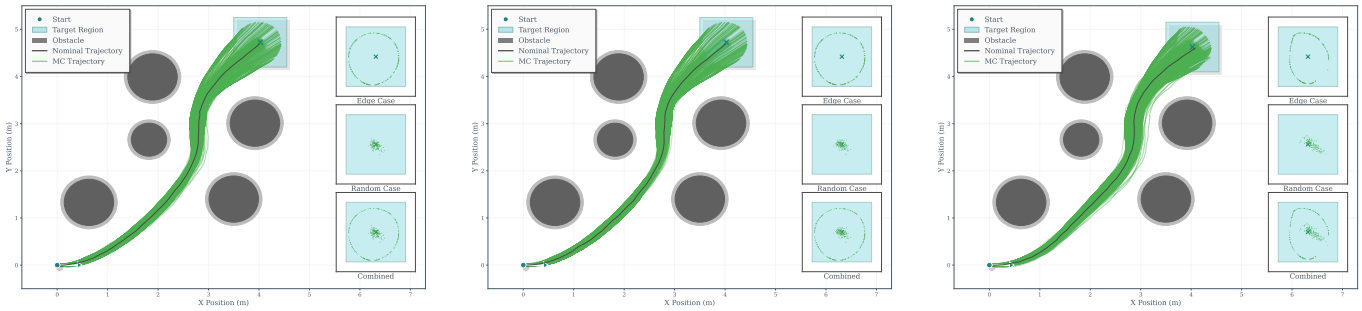
We evaluate cuNRTO, a suite comprising the GPU-accelerated versions of NRTO-DR and NRTO-FullADMM, on unicycle, quadcopter, and Franka manipulator tasks. We provide a comprehensive analysis of the impact of key system parameters, such as time horizon (T), uncertainty level (τ), and the number of obstacles, on both constraint satisfaction and wall-clock time in comparison to the baseline NRTO. Furthermore, we highlight the effectiveness of the proposed architectures in addressing complex dynamical systems through a high-dimensional Franka manipulator task.

A. Methodology

Results were collected on a high-performance workstation with a 3.06 GHz 60-core Intel Xeon W-3500 CPU and an NVIDIA A100 80GB GPU, running Ubuntu 22.04 and CUDA 12.9. Code was compiled with g++ 11.4.0. We use the same outer-loop parameters for all methods. The baseline NRTO solves subproblems using MOSEK [29] interior-point optimizer with multi-threading enabled. Solver-specific hyperparameters were tuned independently for best performance, and its complete list is provided in Section IV of SM.

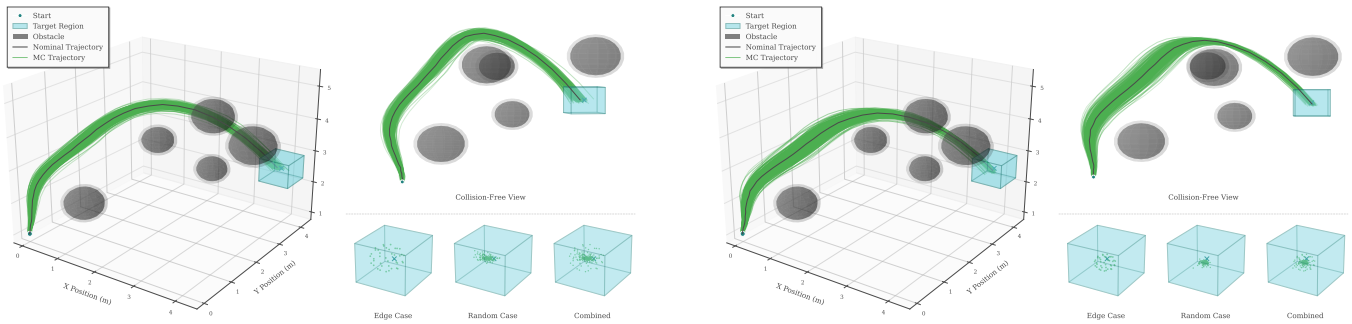
The performance is evaluated based on the following:

- **Constraint satisfaction:** Constraint satisfaction was verified using Monte Carlo sampling with 1,000 i.i.d. disturbance drawn uniformly from the interior of the uncertainty set, combined with 1,000 reproducible edge



(a) NRTO (b) NRTO-DR (c) NRTO-FullADMM

Fig. 6: **Performance comparison on Unicycle model with five obstacles:** We compare (a) the baseline NRTO solver, (b) NRTO-DR, and (c) NRTO-FullADMM. All produce collision-free trajectories that satisfy the robust constraints for 2,000 disturbance realizations. The right insets represent the distribution of terminal state realizations for random, edge, and combined rollouts, demonstrating robustness.



(a) NRTO-DR (b) NRTO-FullADMM

Fig. 7: **Performance comparison on Quadcopter model with five obstacles:** We compare (a) NRTO-DR and (b) NRTO-FullADMM; both produce collision-free trajectories for 2,000 disturbance realizations. Insets report the distribution of terminal state realizations under random Monte Carlo, boundary edge-case, and combined disturbances.

cases that probe the boundary of the uncertainty set via convex combinations of worst-case constraint directions. We set fixed seed to ensure the consistency of the sampling across all experiments. The reported satisfaction probability is the fraction of successful rollouts.

- **Wall-clock time:** We report the wall-clock time required for each solver to converge to a feasible solution. Runtimes were measured using `C++ std::chrono::high_resolution_clock`.

B. Performance Evaluation

Trajectory comparisons between the proposed frameworks and the baseline NRTO for the unicycle and quadcopter models are presented in Fig. 6 and Fig. 7, respectively. Further, the computational efficiency is demonstrated in Table I. We report three sweep studies over horizon length T, uncertainty level τ, and the number of obstacles for both unicycle and quadcopter. We denote the nominal setting as $(T_0, \tau_0, O_0) = (30, 0.05, 0)$ and vary one parameter at a time. All experiments use time-step $\Delta k = 25$ ms for both optimization and rollout.

1) Horizon Sweep (T): At the nominal horizon $T_0 = 30$, NRTO-FullADMM reduces solve time from 32.137 s to 6.231 s for the unicycle, which is 5.16× faster, and from 1327.390 s to 98.585 s for the quadcopter, which is 13.46× faster. At $T = 45$, NRTO-FullADMM reaches 6.387 s for the unicycle and 132.484 s for the quadcopter, corresponding to 6.94× and 13.21× speedups over NRTO, respectively.

2) Obstacle Sweep (Obs): As the number of obstacles increases, the number of robust constraints and associated SOC projections grow, and the GPU implementations benefit from the resulting parallelism. At $Obs = 5$, NRTO-FullADMM reduces solve time from 30423.523 s to 217.932 s for the unicycle, which is 139.60× faster, and from 13374.375 s to 199.935 s for the quadcopter, which is 66.89× faster.

3) Uncertainty sweep (τ): As τ increases, the NRTO success rate decreases, reflecting the increased conservativeness and difficulty of the robust constraints. In contrast, when accounting for linearization error using NRTO-LE [19], we observe 100% constraint satisfaction across all reported τ values for both dynamics. Across the sweep, NRTO-FullADMM consistently yields the lowest wall-clock times among the compared methods.

Sweep	Setting	Controls	Framework	Unicycle			Quadcopter		
				NRTO Succ. (%)	NRTO LE Succ. (%)	Time (s)	NRTO Succ. (%)	NRTO LE Succ. (%)	Time (s)
Horizon T	$T = 15$	$\tau = \tau_0$ $Obs = O_0$	NRTO	**100**	**100**	12.013	**100**	**100**	952.603
			NRTO-DR	**100**	**100**	**5.324**	**100**	**100**	197.171
			NRTO-FullADMM	99	**100**	5.983	99	**100**	**43.543**
	$T = 30$	$\tau = \tau_0$ $Obs = O_0$	NRTO	**91**	**100**	32.137	98	**100**	1327.390
			NRTO-DR	**91**	**100**	7.328	98	**100**	218.259
			NRTO-FullADMM	90	**100**	**6.231**	**99**	**100**	**98.585**
	$T = 45$	$\tau = \tau_0$ $Obs = O_0$	NRTO	82	**100**	44.324	97.5	**100**	1749.458
			NRTO-DR	82	**100**	7.847	97	**100**	238.313
			NRTO-FullADMM	**84**	**100**	**6.387**	**98**	**100**	**132.484**
Uncertainty τ	$\tau = 0.01$	$T = T_0$ $Obs = O_0$	NRTO	**100**	**100**	29.843	**100**	**100**	1132.324
			NRTO-DR	**100**	**100**	7.671	99	**100**	65.454
			NRTO-FullADMM	**100**	**100**	**6.988**	99	**100**	**19.452**
	$\tau = 0.05$	$T = T_0$ $Obs = O_0$	NRTO	**91**	**100**	31.543	**99**	**100**	1332.384
			NRTO-DR	**91**	**100**	8.031	98	**100**	248.473
			NRTO-FullADMM	90	**100**	**6.593**	98	**100**	**97.482**
	$\tau = 0.075$	$T = T_0$ $Obs = O_0$	NRTO	**82**	**100**	33.314	81	**100**	1371.342
			NRTO-DR	**82**	**100**	7.623	81	**100**	739.546
			NRTO-FullADMM	81	**100**	**7.013**	**82**	**100**	**211.483**
Obstacles	$Obs = 0$	$T = T_0$ $\tau = \tau_0$	NRTO	**91**	**100**	31.153	**99**	**100**	1320.487
			NRTO-DR	**91**	**100**	8.352	98	**100**	213.384
			NRTO-FullADMM	90	**100**	**6.324**	98	**100**	**98.345**
	$Obs = 3$	$T = T_0$ $\tau = \tau_0$	NRTO	**90**	**100**	898.342	77	**100**	9142.582
			NRTO-DR	87	**100**	45.532	**77.5**	**100**	6384.378
			NRTO-FullADMM	87	**100**	**43.483**	77	**100**	**121.392**
	$Obs = 5$	$T = T_0$ $\tau = \tau_0$	NRTO	**90**	**100**	30423.523	**74**	**100**	13374.375
			NRTO-DR	85	**100**	1934.314	72	**100**	9836.486
			NRTO-FullADMM	85	**100**	**217.932**	69	**100**	**199.935**

TABLE I: Unicycle and quadcopter dynamics. Three independent sweeps vary one factor while holding the others fixed at (T_0, τ_0, O_0). **Bold**: best; Underline: second-best.

C. Source of speedup

For the unicycle task with five obstacles in Fig. 6, NRTO-DR spends most of its runtime on host–device synchronization and linear solves (43.5% and 32.6%), while SOC projections account for only 11.0%. A detailed runtime breakdown is provided in Section IV of the SM. By keeping the inner ADMM loop on-device, NRTO-FullADMM reduces synchronization overhead and increases average GPU utilization from 34.9% to 86.5%.

D. Solution quality

The proposed reformulations achieve significant speedup without compromising solution quality. The visual variations in terminal-state distributions in Fig. 6-7 and the linearization error differences in Table I are due to convergence toward distinct local minima. These variations arise from the sensitivity of successive linearization towards initialization, hyperparameters, and solver tolerances, rather than a degradation of robust feasibility. Consequently, Table I reveals no distinct pattern in linearization errors across the solvers. Furthermore, for the unicycle task (Fig. 6), the objective components (J_u, J_{kv}) for NRTO, NRTO-DR, and NRTO-FullADMM are $(14.624, 204.914)$, $(14.932, 204.912)$, and $(13.117, 207.425)$, respectively (refer to Table II), with less than 0.5% difference across solvers.

Objective Component	NRTO	NRTO-DR	NRTO-FullADMM
$J_u = 0.05\|\bar{u}\|_2^2$	14.624	14.932	**13.117**
$J_{K_v} = 0.1\|K_v\|_2^2$	204.914	**204.912**	207.425
Total	**219.538**	219.844	220.542

TABLE II: Solution quality comparison for unicycle case (Fig. 6). **Bold**: best; Underline: second-best.

E. Real-World Robotarium Experiment

To further demonstrate that the policy can compensate for real-world model mismatch, we deploy NRTO on a Robotarium unicycle robot. The disturbance set is estimated from open-loop rollout residuals, and the resulting NRTO policy applies the affine feedback $u_k = \bar{u}_k + K_k d_k$, where d_k is estimated online from consecutive state-transition residuals. As shown in Fig. 8–9, the policy substantially reduces accumulated tracking error and steers the robot into the target region, whereas executing the nominal control sequence alone leads to visible drift. Additional details on disturbance estimation are provided in Section V of SM.

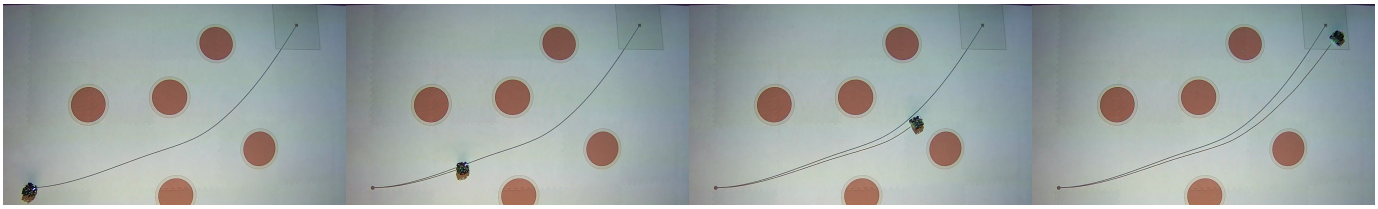


Fig. 8: **Real-world Robotarium rollout with NRTO disturbance-feedback.** Four snapshots over time show the executed trajectory using the affine disturbance-feedback policy. The feedback gain significantly reduces accumulated tracking error and steers the robot into the target region despite real-world disturbances and actuation imperfections.

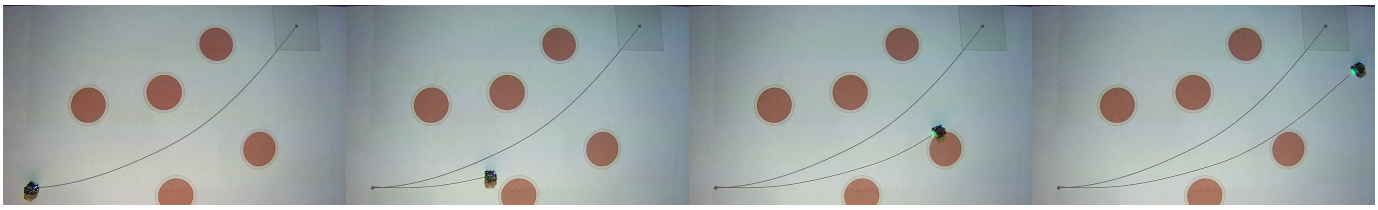


Fig. 9: **Real-world Robotarium rollout with nominal control only.** Using only the nominal sequence leads to substantial drift from the planned path. The resulting tracking error accumulates over time, and the robot fails to reliably satisfy constraints.

Framework	NRTO Succ. (%)	NRTO-LE Succ. (%)	Time (s)
NRTO	**96**	**100**	3948.523
NRTO-DR	**96**	**100**	<u>1342.455</u>
NRTO-FullADMM	<u>95</u>	**100**	**152.384**

TABLE III: Franka manipulator dynamics with fixed setting $(T, \tau, \text{Obs}) = (30, 0.01, 1)$. **best**; <u>second-best</u>.

F. Franka Manipulator Experiment

We evaluate our framework on a 7-DOF Franka Emika Panda manipulator, a high-dimensional system with $n_x = 14$ joint positions and velocities and $n_u = 7$ joint torques, as shown in Fig. 10. The task is to drive the end-effector from a nominal configuration to a goal region while satisfying joint position limits, joint velocity limits, torque bounds, and collision avoidance constraints with respect to spherical obstacles in the workspace. Collision constraints are evaluated using Isaac Sim collision spheres. The feedback gain variables scale as $\mathcal{O}(Tn_un_x)$, yielding $\mathbf{K}_k \in \mathbb{R}^{7\times 14}$ per timestep, and obstacle avoidance constraints are enforced at every knot point via the end-effector position obtained from forward kinematics. We report experiments across fixed horizon length $T = 30$, number of obstacles $n_{\text{obs}} = 1$, and robustness level $\tau = 0.01$. As shown in Table III, NRTO-FullADMM achieves a 25.9× wall-clock speedup over NRTO on this Franka setting, indicating the scalability of NRTO-FullADMM.

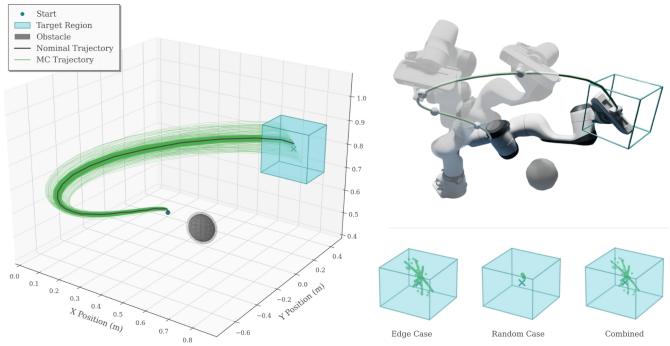


Fig. 10: **NRTO-FullADMM on Franka Manipulator Task:** Left: end-effector trajectory with disturbance rollouts (green) toward the goal region (cyan) while avoiding obstacles (gray). Right: qualitative visualization of the motion in Isaac Sim [30].

VI. Conclusion

We introduced cuNRTO, a GPU-accelerated implementation of nonlinear robust trajectory optimization (NRTO) [19]. By exposing fine-grained parallelism in second-order cone (SOC) projections and reusing constant linear operators across inner iterations, cuNRTO enables two accelerated inner solvers, NRTO-DR and NRTO-FullADMM. Across unicycle, quadcopter, and Franka manipulator tasks, the proposed methods achieve substantial wall-clock speedups up to 139.6× over the baseline while maintaining robust constraint satisfaction.

In future work, we will extend cuNRTO to contact-rich manipulation and locomotion settings involving larger conic programs. We aim to further enhance computational speed by leveraging learning-to-optimize frameworks [31, 32], ranging from learning-to-warmstart architectures [33, 34, 35] to deep-unfolding techniques [36, 37, 38]. Another promising direction is to extend the framework's applicability to multi-agent swarms and to address heterogeneous forms of uncertainty [18, 17].

Acknowledgments

Arshiya Taj Abdul and Evangelos A. Theodorou are supported by the ARO Award #W911NF2010151.

References

[1] T. A. Howell, B. E. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 7674–7679.

[2] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2536–2542.

[3] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6741–6747.

[4] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 5332–5339.

[5] A. D. Saravanos, Y. Aoyama, H. Zhu, and E. A. Theodorou, "Distributed differential dynamic programming architectures for large-scale multiagent control," *IEEE Transactions on Robotics*, vol. 39, no. 6, pp. 4387–4407, 2023.

[6] T. Lew, R. Bonalli, and M. Pavone, "Chance-constrained sequential convex programming for robust trajectory optimization," in *2020 European Control Conference (ECC)*, 2020, pp. 1871–1878.

[7] Y. K. Nakka and S.-J. Chung, "Trajectory optimization of chance-constrained nonlinear stochastic systems for motion planning under uncertainty," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 203–222, 2023.

[8] A. Tsolovikos and E. Bakolas, "Cautious nonlinear covariance steering using variational gaussian process predictive models," *IFAC-PapersOnLine*, vol. 54, no. 20, pp. 59–64, 2021, modeling, Estimation and Control Conference MECC 2021.

[9] A. Ratheesh, V. Pacelli, A. D. Saravanos, and E. A. Theodorou, "Operator splitting covariance steering for safe stochastic nonlinear control," in *2025 IEEE 64th Conference on Decision and Control (CDC)*. IEEE, 2025, pp. 3552–3559.

[10] M. G. Safonov, "Origins of robust control: Early history and future speculations," *IFAC Proceedings Volumes*, vol. 45, no. 13, pp. 1–8, 2012.

[11] D. Q. Mayne, E. C. Kerrigan, and P. Falugi, "Robust model predictive control: Advantages and disadvantages of tube-based methods," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 191–196, 2011.

[12] W. Sun, Y. Pan, J. Lim, E. Theodorou, and P. Tsiotras, "Min-max differential dynamic programming: Continuous and discrete time formulations," vol. 41, 11 2018, pp. 1–13.

[13] W. Sun, Y. Pan, J. Lim, E. A. Theodorou, and P. Tsiotras, "Min-max differential dynamic programming: Continuous and discrete time formulations," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 12, pp. 2568–2580, 2018.

[14] Z. Manchester and S. Kuindersma, "Dirtrel: Robust nonlinear direct transcription with ellipsoidal disturbances and lqr feedback," *Robotics, Sciences and Systems (RSS)*, 2017.

[15] A. Ben-Tal, L. Ghaoui, and A. Nemirovski, *Robust Optimization*, 08 2009.

[16] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM Review*, vol. 53, no. 3, pp. 464–501, 2011.

[17] A. T. Abdul, A. D. Saravanos, and E. A. Theodorou, "Scalable robust optimization for safe multi-agent control under unknown deterministic uncertainty," in *2025 American Control Conference (ACC)*, 2025, pp. 3666–3673.

[18] G. Kotsalis, G. Lan, and A. S. Nemirovski, "Convex optimization for finite-horizon robust covariance control of linear stochastic systems," *SIAM Journal on Control and Optimization*, vol. 59, no. 1, pp. 296–319, 2021.

[19] A. T. Abdul, A. D. Saravanos, and E. A. Theodorou, "Nonlinear robust optimization for planning and control," in *2025 IEEE 64th Conference on Decision and Control (CDC)*, 2025, pp. 3383–3390.

[20] B. Plancher, S. M. Neuman, R. Ghosal, S. Kuindersma, and V. J. Reddi, "Grid: Gpu-accelerated rigid body dynamics with analytical gradients," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 6253–6260.

[21] B. Plancher, S. M. Neuman, T. Bourgeat, S. Kuindersma, S. Devadas, and V. J. Reddi, "Accelerating robot dynamics gradients on a cpu, gpu, and fpga," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2335–2342, 2021.

[22] E. Adabag, M. Atal, W. Gerard, and B. Plancher, "MPCGPU: Real-time nonlinear model predictive control through preconditioned conjugate gradient on the gpu," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 9787–9794.

[23] J. Hu, J. Wang, and H. Christensen, "cpRRTC: GPU-Parallel RRT-Connect for constrained motion planning," in *2025 RSS Workshop on Fast Motion Planning and Control in the Era of Parallelism*.

[24] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, p. 1–122, Jan. 2011.

[25] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," pp. 339–339, 2018.

[26] J. Eckstein and D. P. Bertsekas, "On the douglas-rachford splitting method and the proximal point algorithm for maximal monotone operators," *Math. Program.*, vol. 55,

no. 3, p. 293–318, Jun. 1992.
[27] P. L. Lions and B. Mercier, "Splitting algorithms for the sum of two nonlinear operators," vol. 16, no. 6, p. 964–979, Dec. 1979.
[28] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, June 2016.
[29] M. ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019.
[30] NVIDIA, "Isaac Sim."
[31] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin, "Learning to optimize: A primer and a benchmark," *Journal of Machine Learning Research*, vol. 23, no. 189, pp. 1–59, 2022.
[32] K. Tang and X. Yao, "Learn to optimize—a brief overview," *National Science Review*, vol. 11, no. 8, p. nwae132, 2024.
[33] S. Banerjee, T. Lew, R. Bonalli, A. Alfaadhel, I. A. Alomar, H. M. Shageer, and M. Pavone, "Learning-based warm-starting for fast sequential convex programming and trajectory optimization," in *2020 IEEE Aerospace Conference*. IEEE, 2020, pp. 1–8.
[34] D. Celestini, D. Gammelli, T. Guffanti, S. D'Amico, E. Capello, and M. Pavone, "Transformer-based model predictive control: Trajectory optimization via sequence modeling," *IEEE Robotics and Automation Letters*, 2024.
[35] V. Zinage, A. Khalil, and E. Bakolas, "Transformermpc: Accelerating model predictive control via transformers," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 9221–9227.
[36] A. D. Saravanos, H. Kuperman, A. Oshin, A. T. Abdul, V. Pacelli, and E. A. Theodorou, "Deep distributed optimization for large-scale quadratic programming," *arXiv preprint arXiv:2412.12156*, 2024.
[37] J. Timaná, S. Merino, A. Basarab, R. J. G. Van Sloun, and R. Lavarello, "Acs-net: A deep unfolded admm framework for ultrasound attenuation imaging," in *2025 IEEE International Ultrasonics Symposium (IUS)*, 2025, pp. 1–4.
[38] A. Oshin, R. V. Ghosh, A. D. Saravanos, and E. A. Theodorou, "Deep flexqp: Accelerated nonlinear programming via deep unfolding," *arXiv preprint arXiv:2512.01565*, 2025.