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

Safe Large-Scale Robust Nonlinear MPC in Milliseconds via Reachability-Constrained System Level Synthesis on the GPU

Jeffrey Fang and Glen Chou
Georgia Institute of Technology, Atlanta, GA 30308
Email: {jfang301, chou}@gatech.edu
Project Website: (Link) | **Code**: (Github) | **Video**: (YouTube)

***Abstract*—We present GPU-SLS, a GPU-parallelized framework for safe, robust nonlinear model predictive control (MPC) that scales to high-dimensional uncertain robotic systems and long planning horizons. Our method jointly optimizes an inequality-constrained, dynamically-feasible nominal trajectory, a tracking controller, and a closed-loop reachable set under disturbance, all in real-time. To efficiently compute nominal trajectories, we develop a sequential quadratic programming procedure with a novel GPU-accelerated quadratic program (QP) solver that uses parallel associative scans and adaptive caching within an alternating direction method of multipliers (ADMM) framework. The same GPU QP backend is used to optimize robust tracking controllers and closed-loop reachable sets via system level synthesis (SLS), enabling reachability-constrained control in both fixed- and receding-horizon settings. We achieve substantial performance gains, reducing nominal trajectory solve times by 97.7% relative to state-of-the-art CPU solvers and 71.8% compared to GPU solvers, while accelerating SLS-based control and reachability by 237×. Despite large problem scales, our method achieves 100% empirical safety, unlike high-dimensional learning-based reachability baselines. We validate our approach on complex nonlinear systems, including whole-body quadrupeds (61D) and humanoids (75D), synthesizing robust control policies online on the GPU in 20 milliseconds on average and scaling to problems with 2×10^5 decision variables and 8×10^4 constraints.**

I. Introduction

Safe real-time control is essential for long-duration robotic operation. This requires methods to both *synthesize* trajectories and controllers that satisfy safety and task constraints, and *verify* that the resulting closed-loop behavior is robust to *disturbances*. Model-based *trajectory optimization* methods such as nonlinear model predictive control (NMPC) [47] address synthesis, while *reachability analysis* [4], which overapproximates the set of closed-loop reachable states, enables verification.

Despite their success, constrained trajectory optimization and reachability methods struggle to scale to large problems. For legged robots (>60 states), NMPC is often too slow for real-time, while nonlinear reachability becomes intractable beyond ≈10 states [10, 36, 40]. Data-driven reachability scales better [9, 31, 21], but the resulting estimates are often inaccurate and compromise safety. System-level synthesis (SLS) [25, 7] is a scalable alternative, enabling reachability analysis and the co-design of robust control policies [38, 39]. However, existing SLS methods are too slow for real-time control of high-dimensional robotic systems. While GPU acceleration can enable real-time NMPC [5, 49, 2], existing methods largely ignore inequality constraints and reachability, and therefore cannot guarantee safety under uncertainty.

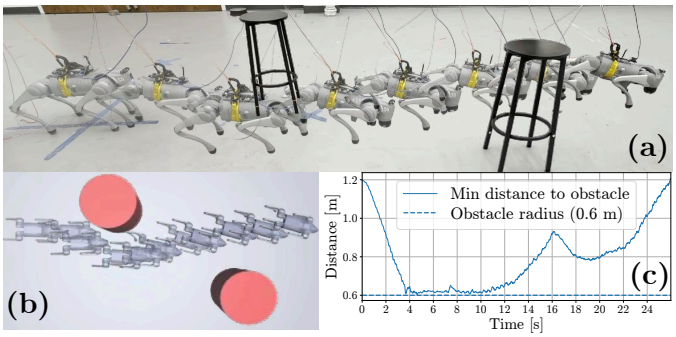

Fig. 1. **(a)**: GPU-accelerated nonlinear constrained whole-body control (61 states, 12 controls) executed on hardware with a Unitree Go2 EDU quadruped, navigating through an obstacle field in real-time at 50 Hz. **(b)**: Overhead view of simulated positions reconstructed from hardware-experiment data showing successful navigation around the obstacles. **(c)**: Graph illustrating minimum distance to any obstacle, demonstrating zero obstacle collisions.

To close these gaps, we propose a GPU-parallelized method for robust NMPC ensuring constraint satisfaction under disturbance. *Our key insight is that local dynamics linearizations reduce constrained trajectory planning and reachability analysis to GPU-friendly matrix operations, enabling robust and rapid large-scale robotic control.* We use the alternating direction method of multipliers (ADMM) and the temporal structure of optimal control problems to jointly optimize a constrained nominal trajectory and a robust tracking controller. We achieve *computational efficiency* by caching ADMM iteration-invariant computations and applying logarithmic-depth horizon splitting via parallel associative scans. We ensure *robustness* by computing forward reachable sets of the optimized closed-loop dynamics using SLS on GPU, with an analogous logarithmic-depth parallelization, yielding margins that are used to tighten constraints in the nominal trajectory planning. For a horizon of length N with state and control dimensions n_x and n_u, we achieve a runtime of $\mathcal{O}(\log N \log^2 n_x + \log^2 n_u)$, improving upon the prior state-of-the-art $\mathcal{O}(N(n_x^3 + n_u^3))$ [38], and enabling real-time, large-scale robotic control. Our contributions are:

- An ADMM-based, GPU-accelerated inequality-constrained linear quadratic regulator (LQR) solver leveraging factorization caching and parallel associative scans.
- A real-time NMPC method embedding the LQR solver in a sequential quadratic programming (SQP) loop, enabling nonlinear inequality-constrained trajectory optimization at a cost of $\mathcal{O}(\log N \log^2 n_x + \log^2 n_u)$ per SQP iteration.
- A unified method for real-time reachability, trajectory op-

timization, and disturbance-feedback synthesis via SLS, with per-iteration complexity $\mathcal{O}(\log N \log^2 n_x + \log^2 n_u)$.
- Evaluation on large-scale problems, including humanoid (75D) control and hardware validation of whole-body quadruped (61D) control, with up to 3×10^3 horizon, 2×10^5 decision variables, and 8×10^4 constraints, surpassing baselines in safety rate and solve speed.

II. Related Work

A. Robust Control, Reachability Analysis, and SLS

Safety verification is commonly performed via reachability analysis [4], which computes forward invariant sets. Hamilton–Jacobi (HJ) reachability [10] provides strong guarantees but scales poorly due to high-dimensional PDEs, limiting it to low-dimensional systems. Control barrier functions (CBFs) [6] offer an alternative, yet synthesizing valid CBFs remains difficult [20]. Data-driven variants scale further [48, 50, 53, 21] but are often heuristic and may violate safety in practice [18, 43]; learning-based HJ methods face similar reliability concerns [9]. Robust and tube-based MPC [47, 41, 34, 37, 42, 1] enforce constraint satisfaction by tightening nominal constraints using reachable sets of the closed-loop system. However, directly enforcing closed-loop constraints is typically nonconvex [25], leading many methods to rely on conservative overapproximations, often by fixing a feedback policy and computing invariant tubes around nominal trajectories [35] via sums-of-squares [51, 40], HJ reachability [26], and contraction [16, 33, 17, 52, 32]. While effective, this can be overly conservative. In contrast, SLS, also known as disturbance-feedback MPC [25, 8], offers a convex parameterization of closed-loop responses for linear time-varying (LTV) systems, enabling robust constraint satisfaction under disturbance [15, 11]. Recent work extends SLS to nonlinear dynamics and constraints [39, 57]. We build on the SLS framework and leverage GPU parallelism to scale reachability and NMPC to large-scale systems, while maintaining sound reachable-set overapproximations under bounded disturbance.

B. CPU and GPU Parallelization for MPC

Recent work has focused on accelerating MPC solvers via parallelization and first-order methods like ADMM. TinyMPC [44] caches factorizations for fast linear solves on embedded platforms, while ReLU-QP [12] unrolls ADMM iterations as a neural-network-like forward pass for GPU speed. However, these methodologies do not extend to NMPC. MPCGPU [2, 3, 30] accelerates NMPC on GPUs but lacks native inequality constraint support, and naive penalty approaches converge slower than our ADMM-based method (Sec. V). [28] reduces horizon complexity by a constant factor through a limited number of parallel trajectory segments, however, does not fully exploit the massive parallelism of GPUs. [46] performs cyclic reductions on the KKT system, introducing increased numerical sensitivity, especially under ill-conditioned systems. [29] uses an augmented Lagrangian without operator splitting, making convergence sensitive to inexact solves and the penalty parameter [22, 23]. ADMM-based OSQP [55] uses a direct $LDL^\top$ factorization of the KKT matrix, limiting scalability and parallelization.

Interior-point methods handle inequalities but are hard to parallelize. HPIPM [24] exploits dense CPU linear algebra, yet its Riccati recursions create sequential horizon dependencies. Just-In-Time Newton [27] adapts interior-point ideas to GPUs using associative scans, but dense floating point computations and multiple scans often underperform optimized CPU baselines. Scan-based trajectory optimization, e.g., Temporal-in-Time [49] and Primal-Dual iLQR [5], enables GPU acceleration, yet constraint handling is limited and robust satisfaction under uncertainty is not addressed. Existing GPU methods mainly target linear or LTI systems, lack nonlinear inequality support, or lack robustness guarantees. Furthermore, second-order methods such as HPIPM are sensitive to ill-conditioning due to their reliance on multiple factorization stages and condensing routines. Limited GPU precision, especially with single-precision arithmetic can degrade stability and convergence behavior. In contrast, first-order methods such as ADMM empirically are more robust to ill-conditioning. Motivated by this, we design a GPU-parallel ADMM method to efficiently solve robust nonlinear MPC with formal guarantees.

III. Preliminaries and Problem Statement

Notation: We use subscripts (e.g. x_k) to denote the time index k, superscripts (s) for the SQP outer-iteration index and (t) for the ADMM inner-iteration index. We denote the Frobenius norm of a matrix $A \in \mathbb{R}^{m\times n}$ as $\|A\|_F^2 := \text{Trace}(A^\top A)$, $\mathbb{S}^n_{++}$ as the set of symmetric positive definite matrices of size $n \times n$, and $[N] := \{0, ..., N-1\}$ and $[M, N] := \{M, ..., N\}$ for $M, N \in \mathbb{N}$. For a matrix $A \in \mathbb{R}^{n\times p}$ we define the row-wise Euclidean norm $\|A\|_{2,\text{row}} \in \mathbb{R}^n$ by $\|A\|_{2,\text{row}} := [\|A_{1,:}\|_2, \ldots, \|A_{n,:}\|_2]^\top$.

A. Problem Statement

We consider uncertain discrete-time nonlinear dynamics

$$x_{k+1} = f(x_k, u_k) + E(x_k)w_k, \tag{1}$$

where $x_k \in \mathbb{R}^{n_x}$ denotes the system state at time step k, $u_k \in \mathbb{R}^{n_u}$ denotes the control input, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ the dynamics function, $E : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x \times n_x}$ the disturbance scaling function, and $w_k \in \mathcal{E}_{n_x} := \{w \in \mathbb{R}^{n_x}, \|w\|_2 \le 1\}$ the disturbance, normalized to be contained in a unit ball.

We consider the problem of designing an optimal controller $\pi(\cdot)$ for the following *robust* nonlinear control problem:

$$\begin{aligned}
\min_{\pi(\cdot)} \quad & J\left(\bar{x}, \pi(\cdot)\right) && \text{(2a)}\\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k) + E(x_k)w_k, \quad \forall k \in [N], && \text{(2b)}\\
& x_0 = \bar{x}_0, && \text{(2c)}\\
& u_k = \pi_k(x_{0:k}), \quad \forall k \in [N], && \text{(2d)}\\
& g(x_k, u_k) \le 0, \quad \forall k \in [N], \quad \forall w_k \in \mathcal{E}_{n_x}, && \text{(2e)}\\
& g^f(x_N) \le 0, \quad \forall w_N \in \mathcal{E}_{n_x}, && \text{(2f)}
\end{aligned}$$

where $\pi = (\pi_0, \dots, \pi_N)$ is a sequence of causal control policies, $\bar{x}_0 \in \mathbb{R}^{n_x}$ is the initial state, $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_c}$ denotes the stagewise state-input constraints, and $g^f : \mathbb{R}^{n_x} \to \mathbb{R}^{n_f}$ the terminal constraint. As (2) is an intractable infinite-dimensional policy optimization, approximate methods typically decompose it by solving for (A) a nominal state-input trajectory and (B) a feedback controller around this nominal solution. Our solution uses the formalism of NMPC for (A) (Sec. III-B) and system level synthesis (SLS) for (B) (Sec. III-D).

B. Nonlinear Model Predictive Control (NMPC)

NMPC is a strategy that repeatedly solves the following optimal control problem given an initial state $\bar{x}_0 \in \mathbb{R}^{n_x}$

$$\begin{aligned}
\min_{X,\,U} \quad & J(X,U) := \ell_f(x_N) + \textstyle\sum_{k=0}^{N-1} \ell(x_k, u_k) && \text{(3a)}\\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k), \qquad \forall k \in [N], && \text{(3b)}\\
& x_0 = \bar{x}_0, && \text{(3c)}\\
& g(x_k, u_k) \le 0,\ \forall k \in [N], \quad g^f(x_N) \le 0, && \text{(3d)}
\end{aligned}$$

yielding an *open-loop nominal* state trajectory $X := \{x_k\}_{k=0}^{N}$ and control sequence $U := \{u_k\}_{k=0}^{N-1}$ over a horizon of length N that minimizes a nonlinear cost function $J : \mathbb{R}^{n_x(N+1)} \times \mathbb{R}^{n_u N}$ with running $\ell : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ and terminal costs $\ell_f : \mathbb{R}^{n_x} \to \mathbb{R}$, subject to the *nominal* dynamics (3b), and the constraints (3d). It is commonly solved using sequential quadratic programming (SQP) [45], which linearizes the nonlinear dynamics and constraints and forms a quadratic approximation of the Lagrangian around a nominal trajectory. At each SQP iteration, we form the associated Lagrangian

$$\begin{aligned}
\mathcal{L}(X,U,\mu,\gamma,\nu) = J(X,U) &+ \mu_0^\top(\bar{x}_0 - x_0)\\
&+ \textstyle\sum_{k=0}^{N-1} \mu_{k+1}^\top (f(x_k,u_k) - x_{k+1})\\
&+ \textstyle\sum_{k=0}^{N-1} \gamma_k^\top g(x_k,u_k) + \nu^\top g^f(x_N),
\end{aligned} \tag{4}$$

where $\mu_k \in \mathbb{R}^{n_x}$ are the Lagrange multipliers for the dynamics constraints, $\gamma_k \in \mathbb{R}^{n_c}_{\ge 0}$ for the stage inequality constraints, and $\nu \in \mathbb{R}^{n_f}_{\ge 0}$ for the terminal inequality constraint. Linearizing the dynamics and constraints around the current nominal trajectory $\xi^{(s)} := (x^{(s)}, u^{(s)})$ and forming a quadratic approximation of the Lagrangian (4) yields the structured LTV-QP (5):

$$\begin{aligned}
\min_{\delta x,\delta u} \quad & J_{\text{QP}}(\delta x,\delta u) := \sum_{k=0}^{N-1} \frac{1}{2}\begin{bmatrix}\delta x_k\\ \delta u_k\end{bmatrix}^\top \begin{bmatrix}Q_k & S_k^\top\\ S_k & R_k\end{bmatrix}\begin{bmatrix}\delta x_k\\ \delta u_k\end{bmatrix}\\
& + \begin{bmatrix}q_k\\ r_k\end{bmatrix}^\top \begin{bmatrix}\delta x_k\\ \delta u_k\end{bmatrix} + \frac{1}{2}\delta x_N^\top Q_N \delta x_N + q_N^\top \delta x_N && \text{(5a)}\\
\text{s.t.} \quad & \delta x_{k+1} = A_k \delta x_k + B_k \delta u_k + b_k, \quad \forall k \in [N], && \text{(5b)}\\
& \delta x_0 = \bar{x}_0 - x_0^{(s)}, && \text{(5c)}\\
& C_k \delta x_k + D_k \delta u_k \le f_k, \quad \forall k \in [N], && \text{(5d)}\\
& C_N \delta x_N \le f_N, && \text{(5e)}
\end{aligned}$$

where

$$\begin{aligned}
&\delta x_k = x_k - x_k^{(s)} \qquad \delta u_k = u_k - u_k^{(s)} && \text{(6a)}\\
&A_k = \nabla_x f\,|_{\xi_k^{(s)}}, \qquad B_k = \nabla_u f\,|_{\xi_k^{(s)}}, && \text{(6b)}\\
&C_k = \nabla_x g\,|_{\xi_k^{(s)}}, \qquad D_k = \nabla_u g\,|_{\xi_k^{(s)}}, && \text{(6c)}\\
&C_N = \nabla_x g_f\,|_{x_N^{(s)}}, && \text{(6d)}\\
&f_k = -g(x_k^{(s)}, u_k^{(s)}), \quad f_N = -g_f(x_N^{(s)}). && \text{(6e)}
\end{aligned}$$

The quadratic terms are approximated by

$$\begin{aligned}
&\begin{bmatrix}Q_k & S_k^\top\\ S_k & R_k\end{bmatrix} = \nabla^2_{(x_k,u_k)}\mathcal{L}_k\,|_{\xi_k^{(s)}}, \quad Q_N = \nabla^2_{x_k}\mathcal{L}_N\,|_{x_N^{(s)}}, && \text{(7a)}\\
&q_k = \nabla_{x_k}\mathcal{L}_k|_{\xi_k^{(s)}} \qquad r_k = \nabla_{u_k}\mathcal{L}_k|_{\xi_k^{(s)}} && \text{(7b)}\\
&q_N = \nabla_{x_N}\mathcal{L}_N|_{x_N^{(s)}}. && \text{(7c)}
\end{aligned}$$

The solution of the LTV-QP (5) yields search direction $\delta\xi := (\delta x, \delta u)$, which updates the nominal iterate as $x^{(s+1)} = x^{(s)} + \alpha\delta x$ and $u^{(s+1)} = u^{(s)} + \alpha\delta u$, with step size $\alpha \in (0, 1]$. This reduces NMPC to a sequence of constrained optimal control problems with linear time-varying (LTV) dynamics, posed as quadratic programs (QPs), whose solutions iteratively update the nominal trajectories [47].

C. Alternating Direction Method of Multipliers (ADMM)

The ADMM [14] is a first-order method to efficiently solve constrained QPs. We consider a generic QP of the form

$$\min_{x \in \mathcal{C}} \quad f(x) \tag{8}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex and $\mathcal{C} \subseteq \mathbb{R}^n$ is a convex set. ADMM absorbs the set constraints $x \in \mathcal{C}$ into the objective via a split variable z and an indicator function, reformulating (8) into

$$\begin{aligned}
&\min_{x,z} \quad f(x) + I_{\mathcal{C}}(z) \qquad \text{s.t.} \quad x = z, && \text{(9a)}\\
&I_{\mathcal{C}}(z) = \begin{cases}0 & z \in \mathcal{C}\\ +\infty & \text{otherwise,}\end{cases} && \text{(9b)}
\end{aligned}$$

and solves it using an augmented Lagrangian

$$\mathcal{L}_A(x,z,\lambda) := f(x) + I_{\mathcal{C}}(z) + \lambda^\top(x - z) + \tfrac{\rho}{2}\|x - z\|_2^2, \tag{10}$$

where $\lambda \in \mathbb{R}^n$ is the Lagrange multiplier and $\rho \in \mathbb{R}_{>0}$ is a scalar penalty parameter. ADMM proceeds by alternating between 1) minimizing (10) with respect to the primal variables x and z, and 2) performing a gradient ascent step on the dual variable λ, i.e., at iteration t, ADMM performs the updates

$$\begin{aligned}
x^{(t+1)} &:= \arg\min_x \mathcal{L}_A(x, z^{(t)}, \lambda^{(t)}), && \text{(11)}\\
z^{(t+1)} &:= \arg\min_z \mathcal{L}_A(x^{(t+1)}, z, \lambda^{(t)}), && \text{(12)}\\
\lambda^{(t+1)} &:= \lambda^{(t)} + \rho(x^{(t+1)} - z^{(t+1)}). && \text{(13)}
\end{aligned}$$

In ADMM formulations tailored for QPs, (11) corresponds to the solution of an unconstrained QP, (12) reduces to a projection onto $\mathcal{C}$, and the dual variable λ is updated via a simple ascent step (13). By iterating over updates (11)-(13), ADMM is guaranteed to converge to an optimal solution of (8) [14].

D. System Level Synthesis

NMPC alone does not robustly guarantee safety under disturbance. To do so, we unify NMPC with SLS, which optimizes over causal *disturbance feedback* controllers

$$u_k = v_k + \sum_{j=0}^{k-1} \mathbf{\Phi}^u_{k,j} w_j, \tag{14}$$

with nominal control $v_k \in \mathbb{R}^{n_u}$, i.e., we assign a distinct disturbance feedback matrix $\mathbf{\Phi}^u_{k,j} \in \mathbb{R}^{n_u \times n_x}$ for each disturbance w_j and control u_k for $k > j$. For uncertain LTV dynamics

$$x_{k+1} = A_k x_k + B_k u_k + E_k w_k, \tag{15}$$

it can be shown using standard algebraic manipulations [8] that the resulting closed-loop state sequence can be expressed as

$$x_k = z_k + \sum_{j=0}^{k-1} \mathbf{\Phi}^x_{k,j} w_j, \;\; z_0 = \bar{x}_0, \tag{16}$$

where $z_k \in \mathbb{R}^{n_x}$ is the nominal state and $\mathbf{\Phi}^x_{k,j} \in \mathbb{R}^{n_x \times n_x}$ captures the influence of disturbance w_j on state x_k. Starting with $\mathbf{\Phi}^x_{j+1,j} = E_j$, SLS propagates the disturbance via

$$\mathbf{\Phi}^x_{k+1,j} = A_k \mathbf{\Phi}^x_{k,j} + B_k \mathbf{\Phi}^u_{k,j}, \tag{17}$$

for all $j \in [N]$ and $k \in [j+1, N-1]$. Since (16) and (14) describe the *true* closed-loop trajectory under a disturbance sequence, enforcing constraints over a worst-case disturbance set ensures robustness. Using this idea, SLS can be extended to nonlinear systems by planning a nominal trajectory and modeling tracking errors as an LTV system (15), where E_k can be chosen to bound linearization error [57, 39]. We note that we do not formally consider linearization error, however, such bounds can be formally incorporated following [39], albeit at the cost of increased conservativeness. This yields an approximate solution to the robust NMPC problem (2):

$$\min_{X,U,\mathbf{\Phi}} \quad J(X,U) + \tilde{H}_0(\mathbf{\Phi}) \tag{18a}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k), \quad \forall k \in [N], \quad x_0 = \bar{x}_0, \tag{18b}$$

$$\mathbf{\Phi}^{\mathrm{x}}_{k+1,j} = A^{(s)}_k \mathbf{\Phi}^{\mathrm{x}}_{k,j} + B^{(s)}_k \mathbf{\Phi}^{\mathrm{u}}_{k,j}, \tag{18c}$$

$$\forall j \in [N], \;\; \forall k \in [j+1, N-1],$$

$$\mathbf{\Phi}^{\mathrm{x}}_{j+1,j} = E(x_j), \tag{18d}$$

$$g(x_k, u_k) + h_k(\mathbf{\Phi}) \leq 0, \qquad \forall k \in [N], \tag{18e}$$

$$g^f(x_N) + h^f(\mathbf{\Phi}) \leq 0, \tag{18f}$$

where $A^{(s)}_k, B^{(s)}_k$ are the linearized dynamics (5b) at stage k. We define the stacked constraint tightenings as

$$h_k(\mathbf{\Phi}) = \sum_{j=0}^{k-1} \left\| (C^{(s)}_k) \mathbf{\Phi}^{\mathrm{x}}_{k,j} + (D^{(s)}_k) \mathbf{\Phi}^{\mathrm{u}}_{k,j} \right\|_{2,\mathrm{row}} \tag{19a}$$

$$h^f(\mathbf{\Phi}) = \sum_{j=0}^{N-1} \left\| (C^{(s)}_N) \mathbf{\Phi}^{\mathrm{x}}_{N,j} \right\|_{2,\mathrm{row}}. \tag{19b}$$

where $C^{(s)}_k, D^{(s)}_k, C^{(s)}_N$ are the linearized constraints (5d)-(5e). The vectors $h_k(\mathbf{\Phi})$ and $h^f(\mathbf{\Phi})$ quantify conservative margins induced by the propagation of disturbances through the closed-loop dynamics, and are used to robustly enforce the constraints. We define

$$\tilde{H}_0(\mathbf{\Phi}) = \sum_{j=0}^{N-1} \Bigg(\sum_{k=j}^{N-1} \left(\|\bar{Q}^{\frac{1}{2}} \mathbf{\Phi}^{\mathrm{x}}_{k,j}\|^2_{\mathcal{F}} + \|\bar{R}^{\frac{1}{2}} \mathbf{\Phi}^{\mathrm{u}}_{k,j}\|^2_{\mathcal{F}} \right) + \|\bar{Q}_N^{\frac{1}{2}} \mathbf{\Phi}^{\mathrm{x}}_{N,j}\|^2_{\mathcal{F}} \Bigg), \tag{20}$$

where $\mathbf{\Phi}$ collects all $\mathbf{\Phi}^{\mathrm{x}}, \mathbf{\Phi}^{\mathrm{u}}$ and $\bar{Q} \in \mathbb{S}^{n_x}_{++}, \bar{R} \in \mathbb{S}^{n_u}_{++}, \bar{Q}_N \in \mathbb{S}^{n_x}_{++}$ define a quadratic cost on the system-level responses $\mathbf{\Phi}$. The state-of-the-art solver for (18) is FastSLS [38], which decomposes (18) into an iterative procedure that alternates between optimizing a (A) nominal trajectory and a (B) robust controller. The nominal trajectory (X, U) is computed by solving a constraint-tightened NMPC,

$$\min_{X,U} \quad J(X,U) \tag{21a}$$

$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k), \; \forall k \in [N], \qquad x_0 = \bar{x}_0, \tag{21b}$$

$$g(x_k, u_k) + h_k(\mathbf{\Phi}) \leq 0, \qquad \forall k \in [N], \tag{21c}$$

$$g^f(x_N) + h^f(\mathbf{\Phi}) \leq 0. \tag{21d}$$

By defining $g^*(x_k, u_k) := g(x_k, u_k) + h_k(\mathbf{\Phi})$ and $g^{f^*}(x_N) := g^f(x_N) + h^f(\mathbf{\Phi})$, (21) mirrors the structure of (2) and can be solved using the method described in Sec. III-B and Sec. IV-A. After the nominal trajectory update, FastSLS solves for the robust controller update to obtain $\mathbf{\Phi}$ using:

$$\min_{\mathbf{\Phi}^{\mathrm{x}}, \mathbf{\Phi}^{\mathrm{u}}} \quad \sum_{j=0}^{N-1} \left(\sum_{k=j}^{N-1} \|\mathcal{Q}_{k,j} \mathbf{\Phi}_{k,j}\|^2_{\mathcal{F}} + \|\mathcal{Q}_{N,j} \mathbf{\Phi}^{\mathrm{x}}_{N,j}\|^2_{\mathcal{F}} \right) \tag{22a}$$

$$\text{s.t.} \; \mathbf{\Phi}^{\mathrm{x}}_{k+1,j} = A^{(s)}_k \mathbf{\Phi}^x_{k,j} + B^{(s)}_k \mathbf{\Phi}^{\mathrm{u}}_{k,j}, \tag{22b}$$

$$\mathbf{\Phi}^{\mathrm{x}}_{j+1,j} = E_j, \;\; \forall j \in [N], \;\; \forall k \in [j+1, N-1], \tag{22c}$$

where $\mathbf{\Phi}_{k,j} := \begin{bmatrix} \mathbf{\Phi}^{\mathrm{x}}_{k,j}{}^\top & \mathbf{\Phi}^{\mathrm{u}}_{k,j}{}^\top \end{bmatrix}^\top$. We define the concatenation and block decomposition

$$\begin{aligned} \mathcal{Q}_{k,j} &= \left(\mathrm{diag}\left(\sqrt{\tau_{k,j}}\right) \begin{bmatrix} C^{(s)}_k & D^{(s)}_k \end{bmatrix}, \begin{bmatrix} \bar{Q}^{\frac{1}{2}} & 0 \\ 0 & \bar{R}^{\frac{1}{2}} \end{bmatrix} \right), \\ \mathcal{Q}_{N,j} &= \left(\mathrm{diag}\left(\sqrt{\tau_{N,j}}\right) C^{(s)}_N, \bar{Q}^{\frac{1}{2}}_N \right), \\ \begin{bmatrix} \mathcal{Q}^{\mathrm{x}}_{k,j} & \mathcal{Q}^{\mathrm{xu}}_{k,j} \\ \mathcal{Q}^{\mathrm{ux}}_{k,j} & \mathcal{Q}^{\mathrm{u}}_{k,j} \end{bmatrix} &:= \mathcal{Q}^\top_{k,j} \mathcal{Q}_{k,j}. \end{aligned} \tag{23}$$

where τ is the Lagrange multiplier that enforces consistency between the nominal optimization and controller synthesis and $\mathcal{Q}^{\mathrm{x}}_{k,j} \in \mathbb{S}^{n_x}_{++}, \mathcal{Q}^{\mathrm{u}}_{k,j} \in \mathbb{S}^{n_u}_{++}, \mathcal{Q}^{\mathrm{xu}}_{k,j} \in \mathbb{R}^{n_x \times n_u}, \mathcal{Q}^{\mathrm{ux}}_{k,j} \in \mathbb{R}^{n_u \times n_x}$.

IV. Method

We present an efficient framework to solve (2) using NMPC and SLS on the GPU. First, Sec. IV-A introduces a GPU accelerated ADMM-based method for efficiently solving the structured LTV-QPs in each SQP iteration, accelerating nominal NMPC. Second, Sec. IV-B details our parallelized solution of the robust SLS control synthesis problem, accelerating control design and reachability analysis. Sec. IV-C describes the Real-Time Iteration (RTI) approach used to enable online simultaneous reachability analysis and trajectory optimization.

A. GPU ADMM QP Solver

We solve LTV-QPs of the form (5) using a GPU-parallelized ADMM QP solver. We introduce the split variable $z := (z_0, \ldots, z_N)$ that tracks the left-hand side of the linearized inequality constraints (5d)-(5e), i.e., we define

$$z_k = C_k \delta x_k + D_k \delta u_k, \quad \forall k \in [N]; \quad z_N = C_N \delta x_N. \tag{24}$$

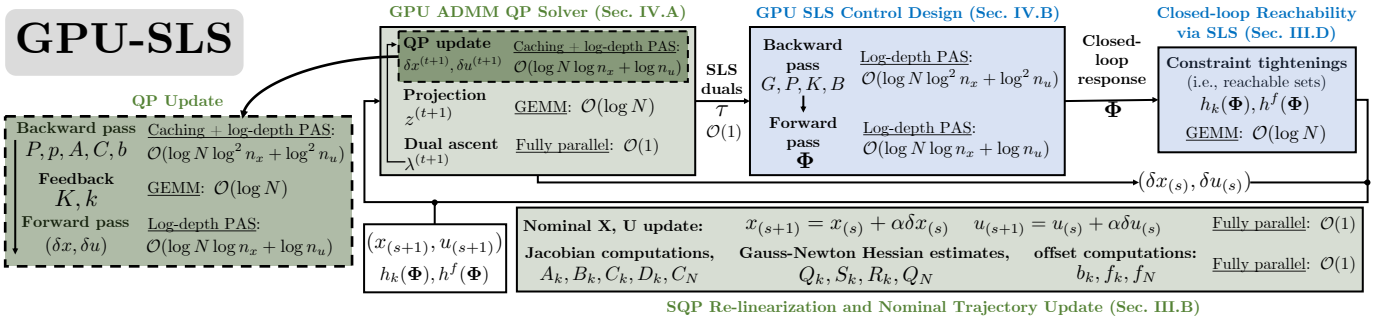

Fig. 2. **Schematic of our method, GPU-SLS.** At each SQP iteration (s), we compute a nominal trajectory update $(\delta x, \delta u)$ using a GPU-parallelized ADMM-based QP solver that exploits caching and parallel associative scans (PASs) for acceleration. The resulting dual variables are post-processed into SLS-compatible duals τ (App. B), which inform the objective in the SLS controller optimization problem (18a). This optimization is likewise GPU-parallelized via PASs, yielding closed-loop response matrices $\mathbf{\Phi}$ that implicitly define the controller. These matrices define constraint tightenings (18e) and (18f), i.e., closed-loop reachable sets, which are used to tighten the constraints of the subsequent nominal trajectory solve, ensuring robust feasibility under closed-loop control. The next SQP iteration then proceeds after fully parallelizable Jacobian and Hessian updates, along with state, input, and function evaluations. Green denotes nominal trajectory computations; blue denotes feedback controller and reachable set computations. GEMM is a $\mathcal{O}(\log N)$ matrix multiplication routine.

Let $f := (f_0, \dots, f_N)$ denote the stacked constraint offsets. The inequality constraints in (5), i.e., (5d)-(5e), can then equivalently be written as $z \leq f$ (elementwise). For compactness, we define $\delta\xi_k := (\delta x_k, \delta u_k)$ and the linear mapping $G : \mathbb{R}^{n_x(N+1)} \times \mathbb{R}^{n_u N} \to \mathbb{R}^{n_c N + n_f}$,

$$\begin{aligned} G(\delta\xi) &:= G(\delta x, \delta u) = (G_0, \dots, G_N), \quad \text{with} \\ G_k(\delta\xi_k) &:= C_k \delta x_k + D_k \delta u_k, \quad G_N(\delta x_N) := C_N \delta_N. \end{aligned} \tag{25}$$

The split constraints (9a) can be then written as $G(\delta x, \delta u) = z$. Define the convex set $\mathcal{C} := \{z \in \mathbb{R}^{n_c N + n_f} : z \leq f\}$, we can equivalently form (5) as

$$\begin{aligned} \min_{\delta x, \delta u} \quad & J_{\text{QP}}(\delta x, \delta u) + I_{\mathcal{C}}(z) \\ \text{s.t.} \quad & \delta x_{k+1} = A_k \delta x_k + B_k \delta u_k + b_k, \quad \forall k \in [N], \\ & G(\delta x, \delta u) = z. \end{aligned} \tag{26}$$

Now, we define the augmented Lagrangian of (26).

$$\begin{aligned} \mathcal{L}_A(\delta\xi, \lambda, \rho) &= J_{\text{QP}}(\delta\xi) + I_{\mathcal{C}}(z) + \lambda^T (G(\delta\xi) - z) \\ &\quad + \tfrac{\rho}{2} \|G(\delta\xi) - z\|_2^2 \\ \text{s.t.} \quad & \delta x_{k+1} = A_k \delta x_k + B_k \delta u_k + b_k, \quad \delta x_0 = \bar{x}_0 - x_0^{(s)}, \end{aligned} \tag{27}$$

where λ is the Lagrange multiplier associated with the consensus constraint and ρ is the penalty parameter. ADMM proceeds by (11), (12), and (13) to solve this optimal control problem.

First, the primal update (equivalent of (11)) reduces to solving an equality-constrained QP of the form

$$\begin{aligned} \min_{\delta x, \delta u} \quad & \sum_{k=0}^{N-1} \frac{1}{2} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix}^\top \begin{bmatrix} \hat{Q}_k & \hat{S}_k^\top \\ \hat{S}_k & \hat{R}_k \end{bmatrix} \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} + \begin{bmatrix} \hat{q}_k \\ \hat{r}_k \end{bmatrix}^\top \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} \\ & + \frac{1}{2} \delta x_N^\top \hat{Q}_N \delta x_N + \hat{q}_N^\top \delta x_N \qquad (28a) \\ \text{s.t.} \quad & \delta x_{k+1} = A_k \delta x_k + B_k \delta u_k + b_k, \ \forall k \in [N], \qquad (28b) \\ & \delta x_0 = \bar{x}_0 - x_0^{(s)}, \qquad (28c) \end{aligned}$$

where the augmented costs are defined as

$$\begin{aligned} \hat{Q}_k &= Q_k + \rho C_k^\top C_k, & \hat{q}_k &= q_k + C_k^\top (\lambda_k - \rho z_k), \\ \hat{R}_k &= R_k + \rho D_k^\top D_k, & \hat{r}_k &= r_k + D_k^\top (\lambda_k - \rho z_k), \\ \hat{S}_k &= S_k + \rho C_k^\top D_k, & & \forall k \in [N], \\ \hat{Q}_N &= Q_N + \rho C_N^\top C_N, & \hat{q}_N &= q_N + C_N^\top (\lambda_N - \rho z_N). \end{aligned} \tag{29}$$

As in common ADMM practice, we reformulate (29) by introducing the scaled dual variables $y := (y_0, \dots, y_N)$, where $y_k := \lambda_k / \rho$ for all $k \in [N]$ and $y_N := \lambda_N / \rho$:

$$\begin{aligned} \hat{q}_k &= q_k + \rho C_k^\top (y_k - z_k), \quad \hat{r}_k = r_k + \rho D_k^\top (y_k - z_k), \\ \hat{q}_N &= q_N + \rho C_N^\top (y_N - z_N). \end{aligned} \tag{30}$$

Next, the z-update (12) becomes a simple linear projection onto the convex set $\mathcal{C}$, after which the dual ascent step (13) is performed as vector operations:

$$z^{(t+1)} = \min\left(G(\delta x^{(t+1)}, \delta u^{(t+1)}) + y^{(t)}, f\right), \tag{31a}$$

$$\lambda^{(t+1)} = \lambda^{(t)} + \rho\left(G(\delta x^{(t+1)}, \delta u^{(t+1)}) - z^{(t+1)}\right). \tag{31b}$$

Note that z- and λ-updates are fast and fully component-wise parallelizable across time and constraints. Thus, the dominant computational cost of our ADMM formulation is in the primal update (28). To accelerate this structured equality-constrained solve on GPUs, we build on recent GPU-parallel optimal control solvers that exploit temporal parallelism via associative scans [5].

1) Efficiently Solving (28) via Associative Scans: Note that (28) is a linear quadratic regulator (LQR) problem, which is traditionally solved via Riccati recursions [47], yielding an optimal solution of the form $(\delta x, \delta u := K \delta x + k)$ [56]. However, Riccati recursions are solved sequentially across time-steps, leading to a computational bottleneck. To accelerate the solution of (28), we use *reverse parallel associative scan* operations to *temporally parallelize* the solution of the primal update (28). Given elements $a_1, \dots, a_N$ and an associative binary operator $\otimes$, a reverse parallel associative scan computes a sequence of suffix reductions $(s_1, s_2, \dots, s_N) := s_{1:N}$ where

$$s_{1:N} := (a_1 \otimes a_2 \otimes \cdots \otimes a_N, \ a_2 \otimes \cdots \otimes a_N, \ \dots \ , a_N), \tag{32}$$

in $\mathcal{O}(\log N)$ depth on parallel hardware [13]. A more detailed overview is given in App. A. To exploit this associative scan to solve (28), we first define the conditional value function (CVF) of (28), building on the result of [5] and extending it to solve the modified LQR problem given by ADMM (28):

$$\begin{aligned} V_{i \to j}(x_i, x_j) = \max_{\eta} & \frac{1}{2} x_i^\top \tilde{P}_{i,j} x_i + \tilde{p}_{i,j}^\top x_i - \frac{1}{2} \eta^\top \tilde{C}_{i,j} \eta \\ & - \eta^\top (x_j - \tilde{A}_{i,j} x_i - \tilde{b}_{i,j}), \end{aligned} \tag{33}$$

where $i < j$. Following [5], we can define the associative operator $\otimes$ for combining $V_{i\to k}$ and $V_{k\to j}$, when both are of the form in (33), to produce $V_{i\to j}$. The constituent terms of $V_{i\to j}$, as seen in (33), are obtained according to the following combination rules:

$$\tilde{P}_{i,j} = \tilde{P}_{i,k} \otimes \tilde{P}_{k,j} \coloneqq \Upsilon_{i,j}\tilde{P}_{k,j}\tilde{A}_{i,k} + \tilde{P}_{i,k}, \tag{34a}$$
$$\tilde{p}_{i,j} = \tilde{p}_{i,k} \otimes \tilde{p}_{k,j} \coloneqq \Upsilon_{i,j}\big(\tilde{p}_{k,j} - \tilde{P}_{k,j}\tilde{b}_{i,k}\big) + \tilde{p}_{i,k}, \tag{34b}$$
$$\tilde{A}_{i,j} = \tilde{A}_{i,k} \otimes \tilde{A}_{k,j} \coloneqq \Psi_{i,j}\tilde{A}_{i,k}, \tag{34c}$$
$$\tilde{C}_{i,j} = \tilde{C}_{i,k} \otimes \tilde{C}_{k,j} \coloneqq \Psi_{i,j}\tilde{C}_{i,k}\tilde{A}_{k,j}^{\top} + \tilde{C}_{k,j}, \tag{34d}$$
$$\tilde{b}_{i,j} = \tilde{b}_{i,k} \otimes \tilde{b}_{k,j} \coloneqq \Psi_{i,j}\big(\tilde{b}_{i,k} - \tilde{C}_{i,k}\tilde{p}_{k,j}\big) + \tilde{b}_{k,j}, \tag{34e}$$
$$\text{where} \quad \Upsilon_{i,j} = \tilde{A}_{i,k}^{\top}\big(I + \tilde{P}_{k,j}\tilde{C}_{i,k}\big)^{-1}, \tag{34f}$$
$$\Psi_{i,j} = \tilde{A}_{k,j}\big(I + \tilde{C}_{i,k}\tilde{P}_{k,j}\big)^{-1}. \tag{34g}$$

The initial values of the associative scan are defined as

$$\tilde{P}_{i,i+1} = \hat{Q}_i - \hat{S}_i^{\top}\hat{R}_i^{-1}\hat{S}_i, \quad \tilde{p}_{i,i+1} = \hat{q}_i - \hat{S}_i^{\top}\Omega_i \tag{35a}$$
$$\tilde{A}_{i,i+1} = A_i - B_i\hat{R}_i^{-1}\hat{S}_i, \quad \tilde{C}_{i,i+1} = B_i\hat{R}_i^{-1}B_i^{\top}, \tag{35b}$$
$$\tilde{b}_{i,i+1} = b_i - B_i\Omega_i, \quad \Omega_i = \hat{R}_i^{-1}\hat{r}_i \quad \forall i \in [N] \tag{35c}$$
$$\tilde{P}_{N,N+1} = \hat{Q}_N, \quad \tilde{p}_{N,N+1} = \hat{q}_N, \tag{35d}$$
$$\tilde{A}_{N,N+1} = 0, \quad \tilde{C}_{N,N+1} = 0, \quad \tilde{b}_{N,N+1} = 0. \tag{35e}$$

As shown by [5], using the CVF (33) and combination rules (34) with initialization (35), we can recover the Riccati terms $P_i = \tilde{P}_{i,N+1}$ $p_i = \tilde{p}_{i,N+1}$ through a reverse parallel associative scan (32). We can then recover the feedback terms

$$K_i = -\Gamma_i\big(\hat{S}_i + B_i^{\top}P_{i+1}A_i\big), \tag{36a}$$
$$k_i = -\Gamma_i\left(B_i^{\top}(p_{i+1} + P_{i+1}b_i) + \hat{r}_i\right), \tag{36b}$$
$$\text{where} \quad \Gamma_i = (\hat{R}_i + B_i^{\top}P_{i+1}B_i)^{-1}. \tag{36c}$$

To recover the nominal updates $\delta x, \delta u$, we can use a *forward* parallel associative scan to similarly parallelize the procedure. We define the conditional optimal trajectory (COT) as

$$\bar{h}_{i\to j}(\delta x_i, \delta x_j) = \bar{A}_{i,j}\delta x_i + \bar{b}_{i,j} \tag{37}$$

and using the combination rules with associated initialization,

$$\begin{aligned} &\bar{A}_{i,j} = \bar{A}_{i,k}\bar{A}_{k,j} && \bar{b}_{i,j} = \bar{A}_{k,j}\bar{b}_{i,k} + \bar{b}_{k,j}, \\ &\bar{A}_{i,i+1} = A_i + B_iK_i && \bar{b}_{i,i+1} = B_ik_i + b_i, \qquad \forall i \in [1, N] \\ &\bar{A}_{0,1} = 0 && \bar{b}_{0,1} = A_0\delta x_0 + b_0, \end{aligned} \tag{38}$$

we can recover the solution to (28) in parallel by

$$\begin{aligned} \delta x_i &= \bar{h}_{0\to 1} \otimes \bar{h}_{1\to 2} \otimes \cdots \otimes \bar{h}_{i-1\to i} \\ \delta u_i &= K_i\delta x_i + k_i. \end{aligned} \tag{39}$$

Therefore, the solution to (28) can be calculated in $\mathcal{O}(\log N \log^2 n_x + \log^2 n_u)$ time on parallel hardware. This follows because each step of the reverse parallel associative scan requires inverting an $n_x \times n_x$ matrix, which can be performed in $\mathcal{O}(\log^2 n_x)$ time in parallel [19]. Since the scan has depth $\mathcal{O}(\log N)$, the total cost of the scan is $\mathcal{O}(\log N \log^2 n_x)$. The initialization step additionally requires matrix inversions of size n_x and n_u, contributing $\mathcal{O}(\log^2 n_x + \log^2 n_u)$.

2) Caching to Accelerate Associative Scans: A common strategy to improve ADMM convergence is to update the penalty parameter ρ only every $\sigma \in \mathbb{N}_{>1}$ iterations instead of every iteration. We adopt a similar ρ update scheme as OSQP [55] presented in Section 5.2. By exploiting this structure, we cache and reuse matrix factorizations across iterations in which ρ remains fixed, substantially reducing computational cost.

To solve (28) when ρ is fixed, only the augmented linear terms in (30) change between ADMM iterations, i.e., $\hat{Q}_{(\cdot)}$, $\hat{R}_{(\cdot)}$, and $\hat{S}_{(\cdot)}$ are unchanged. Thus, in the reverse parallel associative scan (34), only the quantities $\tilde{p}_{i,j}$ (34b) and $\tilde{b}_{i,j}$ (34e) must be recomputed. Since the dominant computational cost arises from matrix inversions, we also cache the invariant intermediate quantities $\tilde{P}_{i,j}$ (34a), $\tilde{C}_{i,j}$ (34d), $\Upsilon_{i,j}$, and $\Psi_{i,j}$ (34f)-(34g), which are needed to compute $\tilde{p}_{i,j}$ and $\tilde{b}_{i,j}$, so that they can be reused across iterations. Thus, for ADMM iterations in which ρ is unchanged, the combination rules (34) reduce to

$$\begin{aligned} \tilde{p}_{i,j} &= \Upsilon_{i,j}^{\text{cache}}\big(\tilde{p}_{k,j} - \tilde{P}_{k,j}^{\text{cache}}\tilde{b}_{i,j}\big), \\ \tilde{b}_{i,j} &= \Psi_{i,j}^{\text{cache}}\big(\tilde{b}_{i,k} - \tilde{C}_{i,k}^{\text{cache}}\tilde{p}_{i,j}\big) + \tilde{b}_{j,k}, \end{aligned} \tag{40}$$

where $\Upsilon_{i,j}^{\text{cache}}, \tilde{P}_{k,j}^{\text{cache}}, \Psi_{i,j}^{\text{cache}}, \tilde{C}_{i,k}^{\text{cache}}$ are the cached terms from the ρ-updated iteration (34). The initial values are set as

$$\begin{aligned} &\tilde{p}_{i,i+1} = \hat{q}_i - \hat{S}_i\Omega_i^{\text{cache}}, && \tilde{b}_{i,i+1} = b_i - B_i\Omega_i^{\text{cache}}, \quad \forall i \in [N] \\ &\tilde{p}_{N,N+1} = \hat{q}_N, && \tilde{b}_{N,N+1} = 0, \end{aligned} \tag{41}$$

where Ω_i^{cache} are the cached term from (35c). We can then obtain the feedback terms K_i and k_i as

$$K_i = K_i^{\text{cache}}; \quad k_i = -\Gamma_i^{\text{cache}}\left(B_i^{\top}(p_{i+1} + P_{i+1}^{\text{cache}}b_i) + \hat{r}_i\right) \tag{42}$$

where Γ_i^{cache} is the cached term from (36c). Using these feedback terms, we can follow the same procedure (37)–(39) to recover the nominal update $\delta x, \delta u$. Because procedures (40)–(42) and (37)–(39) require no inverses, we perform one iteration in $\mathcal{O}(\log N \log n_x + \log n_u)$, a reduction from $\mathcal{O}(\log N \log^2 n_x + \log^2 n_u)$ without caching.

B. Robust Nonlinear MPC via System Level Synthesis

To solve for the robust SLS controller (22), FastSLS uses N independent Riccati recursions, as shown in [38]:

$$\begin{aligned} \mathcal{G}_N^{(j)} &= \mathcal{Q}_{N,j}^{\text{x}}, \qquad \mathcal{K}_k^{(j)} = -\mathcal{G}_k^{(j)}\mathcal{B}_k^{(j)}, \\ \mathcal{P}_k^{(j)} &= \mathcal{Q}_{k,j}^{\text{x}} + \big(A_k^{(s)}\big)^{\top}\mathcal{P}_{k+1}^{(j)}A_k^{(s)} + \big(\mathcal{K}_k^{(j)}\big)^{\top}\mathcal{B}_k^{(j)}, \\ \mathcal{G}_k^{(j)} &= \big(\mathcal{Q}_{k,j}^{\text{u}} + \big(B_k^{(s)}\big)^{\top}\mathcal{P}_{k+1}^{(j)}B_k^{(s)}\big)^{-1}, \\ \mathcal{B}_k^{(j)} &= \mathcal{Q}_{k,j}^{\text{ux}} + \big(B_k^{(s)}\big)^{\top}\mathcal{P}_{k+1}^{(j)}A_k^{(s)}, \end{aligned} \tag{43}$$

and N independent forward propagations

$$\begin{aligned} &\mathbf{\Phi}_{j+1,j}^{\text{x}} = E_j, \\ &\mathbf{\Phi}_{k,j}^{u} = \mathcal{K}_k^{(j)}\mathbf{\Phi}_{k,j}^{\text{x}}, \quad \mathbf{\Phi}_{k+1,j}^{\text{x}} = (A_k^{(s)} + B_k^{(s)}\mathcal{K}_k^{(j)})\mathbf{\Phi}_{k,j}^{\text{x}}, \end{aligned} \tag{44}$$

for all $j \in [N]$ and for all $k \in [j+1, N-1]$, where j indexes the time at which the disturbance is injected, and k indexes the state and input responses at time k for the corresponding disturbance. The superscript (j) denotes the Riccati variables associated with the disturbance injected at time j.

We use parallel associative scans to calculate both (43) and (44) to get the solution for (22). For a given Riccati recursion for the j-th disturbance, we define the CVF

$$\mathcal{V}^{(j)}_{i\to l}(\mathbf{\Phi}_{i,j}, \mathbf{\Phi}_{l,j}) = \max_{\theta} \frac{1}{2}\mathbf{\Phi}^{\top}_{i,j}\tilde{\mathcal{P}}^{(j)}_{i,l}\mathbf{\Phi}_{i,j} - \frac{1}{2}\theta^{\top}\tilde{\mathcal{D}}^{(j)}_{i,l}\theta - \theta^{\top}\left(\mathbf{\Phi}_{l,j} - \tilde{\mathcal{A}}^{(j)}_{i,l}\mathbf{\Phi}_{i,j}\right), \tag{45}$$

and the associated combination rules as

$$\begin{aligned}
\tilde{\mathcal{P}}^{(j)}_{i,l} &= \left(\tilde{\mathcal{A}}^{(j)}_{i,k}\right)^{\top}\left(I + \tilde{\mathcal{P}}^{(j)}_{k,l}\tilde{\mathcal{D}}^{(j)}_{i,k}\right)^{-1}\tilde{\mathcal{P}}^{(j)}_{k,l}\tilde{\mathcal{A}}^{(j)}_{i,k} + \tilde{\mathcal{P}}^{(j)}_{i,k},\\
\tilde{\mathcal{A}}^{(j)}_{i,l} &= \tilde{\mathcal{A}}^{(j)}_{k,l}\left(I + \tilde{\mathcal{D}}^{(j)}_{i,k}\tilde{P}^{(j)}_{k,l}\right)^{-1}\tilde{A}^{(j)}_{i,k},\\
\tilde{\mathcal{D}}^{(j)}_{i,l} &= \tilde{\mathcal{A}}^{(j)}_{k,l}\left(I + \tilde{\mathcal{D}}^{(j)}_{i,k}\tilde{P}^{(j)}_{k,l}\right)^{-1}\tilde{\mathcal{D}}^{(j)}_{i,k}\left(\tilde{\mathcal{A}}^{(j)}_{k,l}\right)^{\top} + \tilde{\mathcal{D}}^{(j)}_{k,l}.
\end{aligned} \tag{46}$$

We define the initial values as

$$\begin{aligned}
\tilde{\mathcal{A}}^{(j)}_{i,i+1} &= A^{(s)}_i - B^{(s)}_i\left(\mathcal{Q}^{\mathrm{u}}_{i,j}\right)^{-1}\mathcal{Q}^{\mathrm{xu}}_{i,j},\\
\tilde{\mathcal{D}}^{(j)}_{i,i+1} &= B^{(s)}_i\left(\mathcal{Q}^{\mathrm{u}}_{i,j}\right)^{-1}\left(B^{(s)}_i\right)^{\top},\\
\tilde{\mathcal{P}}^{(j)}_{i,i+1} &= \mathcal{Q}^{\mathrm{x}}_{i,j} - \mathcal{Q}^{\mathrm{ux}}_{i,j}\left(\mathcal{Q}^{\mathrm{u}}_{i,j}\right)^{-1}\mathcal{Q}^{\mathrm{xu}}_{i,j},\\
\tilde{\mathcal{P}}^{(j)}_{N,N+1} &= \mathcal{Q}^{\mathrm{x}}_{N+1} \quad \tilde{\mathcal{A}}^{(j)}_{N,N+1} = 0, \quad \tilde{\mathcal{D}}^{(j)}_{N,N+1} = 0.
\end{aligned} \tag{47}$$

We can perform a reverse parallel associative scan (32) using combination rules and initialization (46) for each disturbance E_j independently in parallel to recover the Riccati terms by $\mathcal{P}^{(j)}_k = \tilde{\mathcal{P}}^{(j)}_{k,N+1}$. Using the backward pass terms (43), we can recover all Riccati feedback gains $\mathcal{K}^{(j)}_k$. To retrieve the forward pass terms (44), we follow a similar procedure as the forward propagation of the nominal trajectory (37)–(39). We define the Conditional Optimal Propagation (COP) $\bar{\mathcal{N}}_{i\to l}(\mathbf{\Phi}^{x}_{i,j}, \mathbf{\Phi}^{x}_{l,j}) = \bar{\mathcal{A}}^{(j)}_{i,l}\mathbf{\Phi}^{\mathrm{x}}_{i,j}$ with combination rule and initialization

$$\bar{\mathcal{A}}^{(j)}_{i,l} = \bar{\mathcal{A}}^{(j)}_{i,k}\bar{\mathcal{A}}^{(j)}_{k,l}, \qquad \bar{\mathcal{A}}^{(j)}_{i,i+1} = A^{(s)}_i + B^{(s)}_i\mathcal{K}_{i,j}, \tag{48}$$

for $i = 1, \ldots, N$, while for $i = 0$ we set $\bar{\mathcal{A}}_{0,1} = 0$. From the associative scan, we obtain the solution to (22) by

$$\mathbf{\Phi}^{\mathrm{x}}_{i,j} = \bar{\mathcal{N}}_{0\to 1} \otimes \ldots \otimes \bar{\mathcal{N}}_{i-1\to i}, \qquad \mathbf{\Phi}^{\mathrm{u}}_{i,j} = \mathcal{K}_{i,j}\mathbf{\Phi}^{\mathrm{x}}_{i,j}. \tag{49}$$

Therefore, we similarly can calculate the robust control policy in $\mathcal{O}(\log N \log^2 n_x + \log^2 n_u)$ time on parallel hardware.

C. Real-Time Iteration

As standard practice for achieving real-time performance in MPC, we employ a Real-Time Iteration (RTI) scheme in Fig. 1. In RTI, only a single SQP iteration is performed per control step, sacrificing linearization accuracy for speed. This approximation is typically acceptable, as subsequent MPC updates relinearize the dynamics around the newly executed state, thus correcting the linearization error. By only performing one iteration, this drastically improves the computation time for one control, making these MPC loops suitable for real-time.

We adopt an RTI framework for our solver, leading to the RTI GPU-SLS solver. In this setting, we modify our algorithm to instead perform a single linearization of the dynamics and constraints. We warm-start the controller update using the previous iteration's τ variables, from which we can calculate the new constraint tightenings. These are then used to solve the tightened nominal trajectory optimization, which is applied for the current control step. Thus, the total solve time for one RTI update consists of the required time for one linearization, one controller update, and one tightened nominal trajectory update.

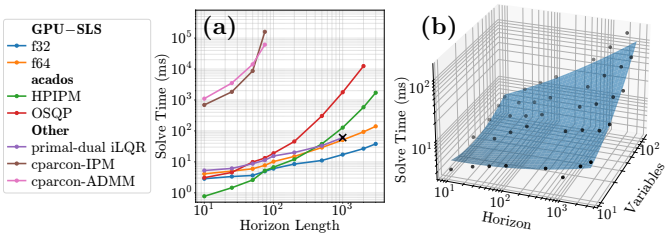

Fig. 3. **(a)**: Comparison of average solve-time scaling with horizon length for various NMPC solvers on a torque-constrained 10-link pendulum stabilization task under random external disturbances. Our method consistently outperforms all baselines for long-horizon problems. **(b)**: Solve-time scaling of GPU-SLS with a family of n-link torque-constrained pendulums.

V. Results

In this section, we evaluate the computational efficiency of our nonlinear MPC framework against state-of-the-art CPU and GPU-based solvers (Sec. V-A), assess the robustness and speedups of our proposed SLS solver against existing methods (Sec. V-B), and demonstrate the practical applicability of our work through both simulation and hardware experiments on legged locomotion systems (Sec. V-C). All CPU benchmarks were run on a desktop computer equipped with an *AMD Ryzen 9900X* processor (12 cores, 24 threads). GPU-based hardware experiments were conducted on a system with an *NVIDIA RTX 4070 Ti Super*, while all other remaining GPU simulation experiments were conducted using a *NVIDIA RTX 4090*.

A. Constrained Nonlinear MPC Benchmarks

To show the scalability of our formulation to long horizons and large problem sizes, we compare our method against various constrained NMPC solvers on a torque-constrained 10-link inverted pendulum. At each control step, the system is randomly perturbed at each joint, requiring the NMPC to stabilize the system. All benchmarks were solved to convergence to a maximum residual tolerance of 10^{-2}. Reported runtimes correspond to the average solve time over 1000 MPC iterations.

As shown in Fig. 3, the proposed method substantially outperforms CPU-based solvers OSQP and HPIPM, reducing solve times by **99.8%** and **98.9%**, respectively, in long-horizon scenarios. Our method also improves upon the GPU-accelerated augmented Lagrangian approach (primal-dual iLQR AL) by **71.8%** and significantly outperforms both *cparcon-IPM* and *cparcon-ADMM*. Furthermore, we evaluate the impact of using high precision arithmetic (FP64) to handle occasionally ill-conditioned problems and observe that our approach continues to outperform all baselines.

These scaling trends are due to the computational structure of each solver. CPU-based methods such as HPIPM and OSQP rely on largely sequential linear-algebra operations, which becomes a bottleneck as the horizon grows. Primal-dual iLQR AL, while GPU-parallelized, incurs higher per-iteration cost due to increased sensitivity of the penalty parameter under reduced GPU precision, and in certain ill-conditioned problems,

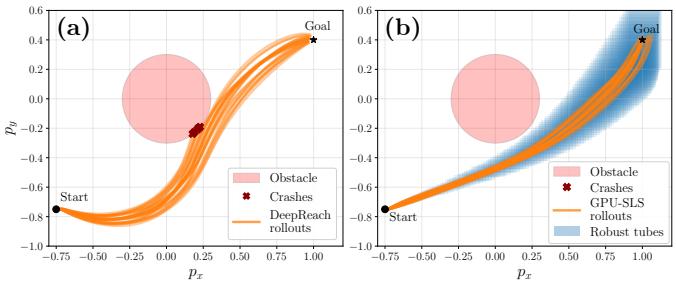

Fig. 4. DeepReach rollouts **(a)** and GPU-SLS rollouts **(b)** for an obstacle avoidance Dubins car system subject to both adversarial and random disturbance. Deepreach is shown to fail to consistently certify safety, while our method certifies safety 100% of all rollouts.

Horizon N	10	25	50	100	200	500	1000
FastSLS (ms)	10	65	180	630	2,538	35,057	189,052
GPU-SLS (ms)	9	11	15	27	50	198	796
Speedup	1.11	5.91	12	23.3	50.8	193.5	237.5

Fig. 5. Runtime comparison of FastSLS vs GPU-SLS for solving the system in 4. GPU-SLS consistently outperforms FastSLS due to its logarithmic scaling vs FastSLS's cubic scaling.

will lead to divergence, as reflected in our experiments. To improve convergence, *cparcon* uses full second-order information rather than a Gauss-Newton approximation, leading to multiple expensive associative scan operations that increases wall-clock time. ADMM on the other hand, can tolerate inexact solves on the GPU more reliably than AL methods. This attribute, paired with our caching parallel associative scan routine, enables substantial performance improvements.

We also show in Fig. 3 that we can solve a family of torque-constrained n-link pendulum problems with horizons as large as $N = 3000$, corresponding to more than 1.35×10^5 decision variables, within 73 ms. These results reflect our method's logarithmic scaling with respect to the state, control, constraint, and horizon dimensions, allowing our method to solve much larger problems than existing MPC methods.

B. SLS Benchmarks

We now evaluate GPU-SLS on a suite of Dubins car benchmarks to assess safety, scalability, and computational performance. We also compare the benefits of robust MPC against classical MPC on a planar quadrotor. The associated state, control, and dynamics definitions are given in App. C–D.

To show the superior safety of our method compared to data-driven methods such as DeepReach, we perform closed-loop rollouts (14) of a Dubins car navigating around an obstacle. In Fig. 4, we plot the rollouts of both controllers subject to random and adversarial disturbance. DeepReach fails to consistently certify safety, crashing into the obstacle in 6 out of 31 rollouts. This is because DeepReach relies on offline value-function approximations, which requires large amounts of training data and provides limited robustness guarantees. In contrast, GPU-SLS explicitly enforces robustness via disturbance feedback and is safe in all tested rollouts, highlighting the limitations of purely data-driven safety certificates.

We further demonstrate the advantages of robust MPC over classical MPC on a 6D planar quadrotor. Under disturbances, standard MPC (Fig. 7**a**) fails to consistently avoid all obstacles, colliding in 6/31 executions, whereas our method (Fig. 7**b**)

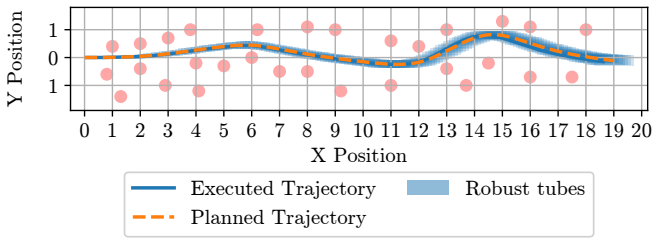

Fig. 6. Dubins car experiment using GPU-SLS to calculate a robust trajectory for 20 meters through a dense constraint field of 30 obstacles. This demonstrates our solver's ability to handle large-scale trajectory optimization and robust constraint satisfaction at scale.

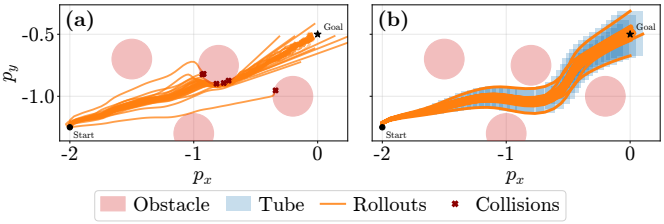

Fig. 7. Rollouts for uncertain nonlinear dynamics on a 6D planar quadrotor using **(a)**: Classical Closed-Loop MPC and **(b)**: robust MPC via GPUSLS. Classical MPC fails to reliably avoid the obstacle, whereas our method consistently achieves safe navigation.

successfully avoids all obstacles across all rollouts. Notably, the robustness procedure adds minimal overhead, with the controller solve accounting for ∼3% of total computation time.

A core bottleneck with traditional FastSLS is its computational speed. In Fig. 5, we benchmark the average solve times of GPU-SLS and FastSLS on a similar system as in Fig. 4. As the prediction horizon increases, our method achieves increasingly large speedups compared to FastSLS, with the largest horizon offering a 237.5× speedup. This contrast reflects FastSLS' cubic scaling in problem dimension, while our GPU-parallel routine scales logarithmically in the same dimensions, enabling GPU-SLS to solve robust OCPs at scale.

We further evaluate the scalability of our formulation to large environments with multiple constraints. Fig. 6 illustrates a long horizon Dubins car trajectory (20 m) navigating through a dense constraint field of 30 obstacles subject to external disturbances. The system is able to retain safety while navigating difficult terrain, consistently remaining within the calculated reachable tubes. These results demonstrate the ability of GPU-SLS to handle large-scale, highly constrained problems without compromising robustness.

C. Legged Experiments

We evaluate our solver's ability to plan robust whole-body trajectories for a Unitree H1 humanoid robot with 75 states and 19 control inputs in a MuJoCo simulation environment adapted from [5] using the humanoid's full, rigid-body dynamics model. As demonstrated in Fig. 8, we task the humanoid with robustly navigating around an obstacle subject to external disturbances. The humanoid can be seen successfully navigating around the obstacle using the synthesized robust controller (14), which ensures that the system state remains within its robust tubes as shown in Fig. 9. Due to the parallelization of the GPU and the formal guarantees of SLS, we are able to solve constrained, high dimensional nonlinear systems, while

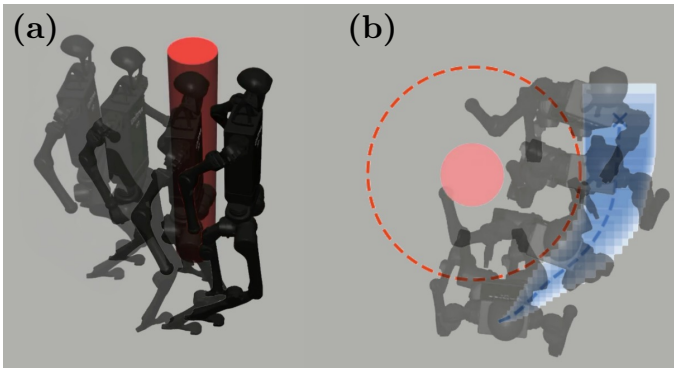

Fig. 8. **(a)**: Whole-body humanoid (75D, 19C) navigating around an obstacle using (14) calculated from GPU-SLS. **(b)**: A visualization of the robust forward tubes (blue squares) generated by (18) and the humanoid's rolled out trajectory remaining within the tubes.

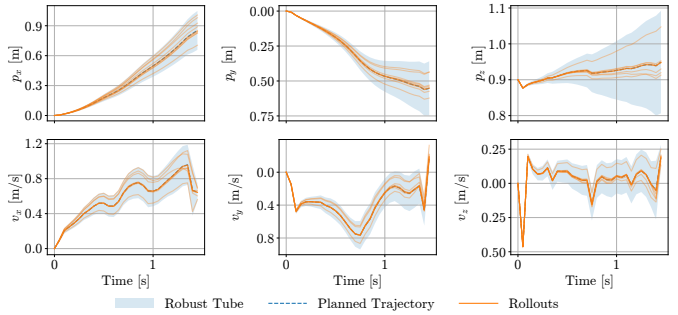

Fig. 9. Simulated rollouts of 8 under adversarial disturbances. Using the controller (14) the humanoid is able to safely remain within the tubes, demonstrating our method's ability to develop robust control policies for complex, high-dimensional nonlinear systems.

also calculating robust controllers that guarantee system safety.

The high computation times of traditional constrained NMPC solvers preclude their applicability to legged hardware experiments requiring high control update rates. To address this, we first benchmark the computation times between CPU-based solver OSQP and our solver for controlling a whole rigid-body quadruped (61D) in navigating around an obstacle. Both methods are implemented in an RTI scheme and solved to the same convergence tolerance of 10^{-2}, with solve times averaged across 700 MPC iterations. In Fig. 10 we show the benefit of our GPU accelerated method, outperforming OSQP in all horizon lengths, achieving a 95% speedup at the longest horizon, while requiring fewer ADMM iterations (27/25 mean/median vs. 107/50 for OSQP). The difference in computation time stems from the parallel structure of our method, which is particularly advantageous for high-dimensional systems and longer horizons, enabling real-time performance where CPU-based solvers become computationally prohibitive.

To validate the real-world practicality of our formulation for legged robots, we perform whole-body constrained NMPC and SLS on a physical Unitree Go2 EDU quadruped (61D). In our hardware experiments, we use a Vicon Motion Capture System to track the quadruped's center of mass. In the first experiment, shown in Fig. 1, the quadruped is tasked with safely navigating through two obstacles using the nominal trajectory generator. The controller runs at 50 Hz with an average solve time of 16ms, prediction horizon of 25, and a discretization timestep of

Horizon N	25	50	75	100	150	200
GPUSLS (ms)	10	13	18	21	27	36
CPU SQP + OSQP (ms)	127	202	299	392	602	760

Fig. 10. Runtime comparisons between our GPU solver and the CPU-based solver OSQP for varying horizons on a whole rigid-body quadruped (61D) navigating around an obstacle. Across all horizons, our method outperforms OSQP, achieving up to a 95% reduction in solve time at the longest horizon.

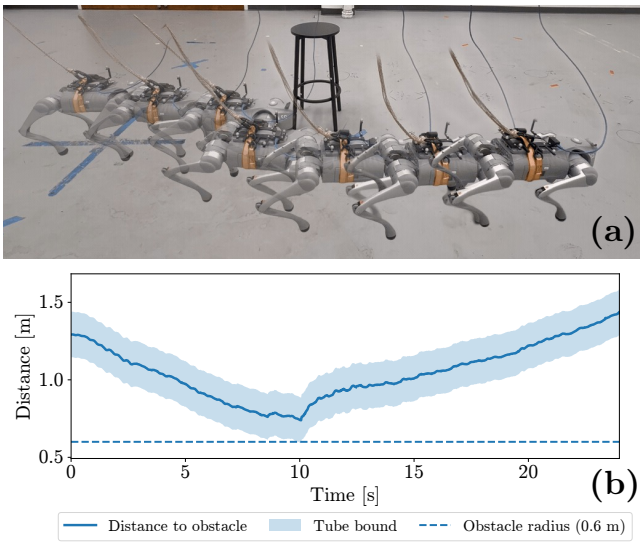

Fig. 11. **(a)**: GPU-accelerated robust whole-body control (61 states, 12 controls) using RTI GPU-SLS executed on hardware with a Unitree Go2 EDU quadruped. The robot successfully navigates robustly around an obstacle in real-time at 50 Hz. **(b)**: Graph illustrating the robot's distance to the obstacle and the maximum tube size for each time step along the trajectory. The tubes are used to tighten the constraints in the nominal trajectory generator, ensuring that the nominal policy remains safe despite disturbances.

0.02s. From Fig. 1 we can see that the quadruped successfully navigates through the obstacles in a collision-free manner, highlighting the practical viability of the proposed constrained optimization framework on hardware.

We also evaluate the practicality of the RTI GPU-SLS pipeline for real-time reachability analysis on hardware. We task the robot with navigating around an obstacle with a disturbance magnitude of 0.02. In Fig. 11, we show that the quadruped successfully navigates around the obstacle while remaining within its tubes throughout the entire trajectory, demonstrating the practicality on hardware of our RTI GPU-SLS algorithm for high-dimensional nonlinear systems.

VI. Conclusion

We presented GPU-SLS, a GPU-parallelized framework for safe, large-scale robust NMPC that unifies constrained trajectory optimization and disturbance-feedback controller synthesis within a real-time pipeline. By exploiting the structure of ADMM and FastSLS, our method leverages parallel associative scans and factorization caching to achieve logarithmic-depth scaling in horizon, control, and state dimensions, enabling millisecond-scale solutions to robustly constrained optimization problems. Our method removes a major computational bottleneck, enabling significant speedups over existing CPU and GPU solvers, along with 100% empirical safety across high-dimensional systems, including both simulation and hardware validation on humanoids and quadrupeds.

Acknowledgments

We thank Saman Zonouz and the Indoor Flight Laboratory at the Georgia Institute of Technology for support with hardware and experimental facilities.

References

[1] Arshiya Taj Abdul, Augustinos D. Saravanos, and Evangelos A. Theodorou. Nonlinear robust optimization for planning and control. In *2025 IEEE 64th Conference on Decision and Control (CDC)*, pages 3383–3390, 2025. doi: 10.1109/CDC57313.2025.11312909.

[2] Emre Adabag, Miloni Atal, William Gerard, and Brian 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)*, pages 9787–9794. IEEE, 2024.

[3] Emre Adabag, Marcus Greiff, John Subosits, and Thomas Lew. Differentiable model predictive control on the gpu. *arXiv preprint arXiv:2510.06179*, 2025.

[4] Matthias Althoff, Goran Frehse, and Antoine Girard. Set propagation techniques for reachability analysis. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):369–395, 2021.

[5] Lorenzo Amatucci, João Sousa-Pinto, Giulio Turrisi, Dominique Orban, Victor Barasuol, and Claudio Semini. Primal-dual ilqr for gpu-accelerated learning and control in legged robots. *arXiv preprint arXiv:2506.07823*, 2025.

[6] Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European control conference (ECC)*, pages 3420–3431. Ieee, 2019.

[7] James Anderson, John C. Doyle, Steven H. Low, and Nikolai Matni. System level synthesis. *Annu. Rev. Control.*, 47:364–393, 2019.

[8] James Anderson, John C Doyle, Steven H Low, and Nikolai Matni. System level synthesis. *Annual Reviews in Control*, 47:364–393, 2019.

[9] Somil Bansal and Claire J Tomlin. Deepreach: A deep learning approach to high-dimensional reachability. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1817–1824. IEEE, 2021.

[10] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2242–2253. IEEE, 2017.

[11] Marcell Bartos, Alexandre Didier, Jerome Sieber, Johannes Köhler, and Melanie N Zeilinger. Stochastic mpc with online-optimized policies and closed-loop guarantees. *arXiv preprint arXiv:2502.06469*, 2025.

[12] Arun L Bishop, John Z Zhang, Swaminathan Gurumurthy, Kevin Tracy, and Zachary Manchester. Reluqp: A gpu-accelerated quadratic programming solver for model-predictive control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13285–13292. IEEE, 2024.

[13] Guy E Blelloch. Scans as primitive parallel operations. *IEEE Transactions on computers*, 38(11):1526–1538, 2002.

[14] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[15] Shaoru Chen, Victor M Preciado, Manfred Morari, and Nikolai Matni. Robust model predictive control with polytopic model uncertainty through system level synthesis. *Automatica*, 162:111431, 2024.

[16] Glen Chou, Necmiye Ozay, and Dmitry Berenson. Model error propagation via learned contraction metrics for safe feedback motion planning of unknown systems. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 3576–3583. IEEE, 2021.

[17] Glen Chou, Necmiye Ozay, and Dmitry Berenson. Safe output feedback motion planning from images via learned perception modules and contraction theory. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 349–367. Springer, 2022.

[18] Long Kiu Chung, Wonsuhk Jung, Chuizheng Kong, and Shreyas Kousik. Goal-reaching trajectory design near danger with piecewise affine reach-avoid computation. *arXiv preprint arXiv:2402.15604*, 2024.

[19] Laszlo Csanky. Fast parallel matrix inversion algorithms. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 11–12. IEEE, 1975.

[20] Hongkai Dai and Frank Permenter. Convex synthesis and verification of control-lyapunov and barrier functions with input constraints. *arXiv preprint arXiv:2210.00629*, 2022.

[21] Charles Dawson, Zengyi Qin, Sicun Gao, and Chuchu Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pages 1724–1735. PMLR, 2022.

[22] Jonathan Eckstein and Wang Yao. Augmented lagrangian and alternating direction methods for convex optimization: A tutorial and some illustrative computational results. *RUTCOR Research Reports*, 32(3):44, 2012.

[23] Jonathan Eckstein and Chang Yu. Two innovations in inexact augmented lagrangian methods for convex optimization. *arXiv preprint arXiv:2503.11809*, 2025.

[24] Gianluca Frison and Moritz Diehl. Hpipm: a high-performance quadratic programming framework for model predictive control. *IFAC-PapersOnLine*, 53(2): 6563–6569, 2020.

[25] Paul J Goulart, Eric C Kerrigan, and Jan M Maciejowski. Optimization over state feedback policies for robust control with constraints. *Automatica*, 42(4):523–533, 2006.

[26] Sylvia L Herbert, Mo Chen, SooJean Han, Somil Bansal, Jaime F Fisac, and Claire J Tomlin. Fastrack: A modular framework for fast and guaranteed safe motion planning.

In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1517–1522. IEEE, 2017.

[27] Casian Iacob, Hany Abdulsamad, and Simo Särkkä. A parallel-in-time newton's method for nonlinear model predictive control. *IEEE Transactions on Control Systems Technology*, 2025.

[28] Wilson Jallet, Ewen Dantec, Etienne Arlaud, Justin Carpentier, and Nicolas Mansard. Parallel and proximal constrained linear-quadratic methods for real-time nonlinear mpc. *arXiv preprint arXiv:2405.09197*, 2024.

[29] Wilson Jallet, Antoine Bambade, Etienne Arlaud, Sarah El-Kazdadi, Nicolas Mansard, and Justin Carpentier. Proxddp: Proximal constrained trajectory optimization. *IEEE Transactions on Robotics*, 2025.

[30] Se Hwan Jeon, Seungwoo Hong, Ho Jae Lee, Charles Khazoom, and Sangbae Kim. Cusadi: A gpu parallelization framework for symbolic expressions and optimal control. *IEEE Robotics and Automation Letters*, 2024.

[31] Frank Jiang, Glen Chou, Mo Chen, and Claire J Tomlin. Using neural networks to compute approximate and guaranteed feasible hamilton-jacobi-bellman pde solutions. *arXiv preprint arXiv:1611.03158*, 2016.

[32] Craig Knuth, Glen Chou, Necmiye Ozay, and Dmitry Berenson. Planning with learned dynamics: Probabilistic guarantees on safety and reachability via lipschitz constants. *IEEE Robotics and Automation Letters*, 6(3): 5129–5136, 2021.

[33] Craig Knuth, Glen Chou, Jamie Reese, and Joe Moore. Statistical safety and robustness guarantees for feedback motion planning of unknown underactuated stochastic systems. *arXiv preprint arXiv:2212.06874*, 2022.

[34] Johannes Köhler, Raffaele Soloperto, Matthias A Müller, and Frank Allgöwer. A computationally efficient robust model predictive control framework for uncertain nonlinear systems. *IEEE Transactions on Automatic Control*, 66(2):794–801, 2020.

[35] Shreyas Kousik, Sean Vaskov, Fan Bu, Matthew Johnson-Roberson, and Ram Vasudevan. Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. *The International Journal of Robotics Research*, 39(12):1419–1469, 2020.

[36] Benoit Landry, Mo Chen, Scott Hemley, and Marco Pavone. Reach-avoid problems via sum-or-squares optimization and dynamic programming. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4325–4332. IEEE, 2018.

[37] Wilbur Langson, Ioannis Chryssochoos, SV Raković, and David Q Mayne. Robust model predictive control using tubes. *Automatica*, 40(1):125–133, 2004.

[38] Antoine P Leeman, Johannes Kohler, Florian Messerer, Amon Lahr, Moritz Diehl, and Melanie N Zeilinger. Fast system level synthesis: Robust model predictive control using riccati recursions. *IFAC-PapersOnLine*, 58(18): 173–180, 2024.

[39] Antoine P Leeman, Johannes Köhler, Andrea Zanelli, Samir Bennani, and Melanie N Zeilinger. Robust nonlinear optimal control via system level synthesis. *IEEE Transactions on Automatic Control*, 2025.

[40] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[41] David Q. Mayne, Maria M. Seron, and Saša V. Raković. Robust output feedback model predictive control of constrained linear systems. *Automatica*, 41(2):219–224, 2005.

[42] David Q Mayne, Erric C Kerrigan, EJ Van Wyk, and Paola Falugi. Tube-based robust nonlinear model predictive control. *International journal of robust and nonlinear control*, 21(11):1341–1353, 2011.

[43] Devesh Nath, Haoran Yin, and Glen Chou. Formal safety verification and refinement for generative motion planners via certified local stabilization. *arXiv preprint arXiv:2509.19688*, 2025.

[44] Khai Nguyen, Sam Schoedel, Anoushka Alavilli, Brian Plancher, and Zachary Manchester. Tinympc: Model-predictive control on resource-constrained microcontrollers. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE, 2024.

[45] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, 2nd edition, 2006.

[46] Pieter Pas and Panagiotis Patrinos. Cyqlone: A parallel, high-performance linear solver for optimal control. *arXiv preprint arXiv:2512.09058*, 2025.

[47] James B. Rawlings, David Q. Mayne, and Moritz Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, Madison, WI, 2nd edition, 2017.

[48] Alexander Robey, Haimin Hu, Lars Lindemann, Hanwen Zhang, Dimos V Dimarogonas, Stephen Tu, and Nikolai Matni. Learning control barrier functions from expert demonstrations. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 3717–3724. Ieee, 2020.

[49] Simo Särkkä and Ángel F García-Fernández. Temporal parallelization of dynamic programming and linear quadratic control. *IEEE Transactions on Automatic Control*, 68(2):851–866, 2022.

[50] Matteo Saveriano and Dongheui Lee. Learning barrier functions for constrained motion planning with dynamical systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 112–119. IEEE, 2019.

[51] Sumeet Singh, Mo Chen, Sylvia L Herbert, Claire J Tomlin, and Marco Pavone. Robust tracking with model mismatch for fast and safe planning: an sos optimization approach. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 545–564. Springer, 2018.

[52] Sumeet Singh, Benoit Landry, Anirudha Majumdar, Jean-Jacques Slotine, and Marco Pavone. Robust feedback

motion planning via contraction theory. *The International Journal of Robotics Research*, 42(9):655–688, 2023.

[53] Oswin So, Zachary Serlin, Makai Mann, Jake Gonzales, Kwesi Rutledge, Nicholas Roy, and Chuchu Fan. How to train your neural control barrier function: Learning safety filters for complex input-constrained systems. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11532–11539. IEEE, 2024.

[54] Anutam Srinivasan, Antoine Leeman, and Glen Chou. Safety beyond the training data: Robust out-of-distribution mpc via conformalized system level synthesis. *arXiv preprint arXiv:2602.12047*, 2026.

[55] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[56] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.

[57] Shuyu Zhan, Chih-Yuan Chiu, Antoine P Leeman, and Glen Chou. Robustly constrained dynamic games for uncertain nonlinear dynamics. *arXiv preprint arXiv:2509.16826*, 2025.