Fast Differential Dynamic Programming for Time-Optimal Trajectory Planning

Research Portfolio

Miaomiao Dai

https://github.com/dmmsjtu-umich/time-opt-ilqr

1 Overview

Time-optimal trajectory planning—generating motions that complete a task in the minimum possible time—is a fundamental requirement for agile robotic systems. From autonomous drone racing to emergency collision avoidance in self-driving cars, the ability to jointly optimize the control sequence and the total maneuver duration T is critical for pushing physical limits.

While Differential Dynamic Programming (DDP) and its variant, the iterative Linear Quadratic Regulator (iLQR), have become standard tools for high-dimensional trajectory optimization, they typically assume a fixed planning horizon. Extending these methods to time-optimal control introduces a discrete-continuous optimization challenge: the solver must determine the optimal integer horizon T^* alongside the continuous control inputs.

2 Problem Formulation

We consider the discrete-time optimal control problem with variable horizon:

$$\begin{aligned} \min_{U_T, T} \quad & J = \phi(x_T) + \sum_{k=0}^{T-1} \ell(x_k, u_k) + wT \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k), \quad k = 0, \ldots, T-1 \\ & x_0 = \bar{x}_0, \quad T \in \{1, 2, \ldots, N\} \end{aligned} \tag{1}$$

where the decision variables include both the control sequence $U_T = \{u_0, \ldots, u_{T-1}\}$ and the finish time T. The term wT penalizes the planning horizon, encouraging time-optimal behaviour of the system.

3 The Challenge

A fundamental bottleneck lies in the structure of the standard Riccati recursion. In the LQR backward pass, the Value Function V_k is computed recursively starting from a terminal cost anchored at the final time step T (i.e., $P_T = Q_T$). Consequently, changing the horizon from T to $T + 1$ shifts the boundary condition, invalidating the entire sequence of previously computed Cost-to-Go matrices. This structural dependency prevents the reuse of historical computations across different horizons, forcing the solver to restart the backward pass from scratch for each candidate T, resulting in a prohibitive $\mathcal{O}(N^2)$ complexity.

4 Our Approach: Time-Varying Propagator

To address the loss of reusability in time-varying systems, we shift from *reusing values* to *reusing mappings*.

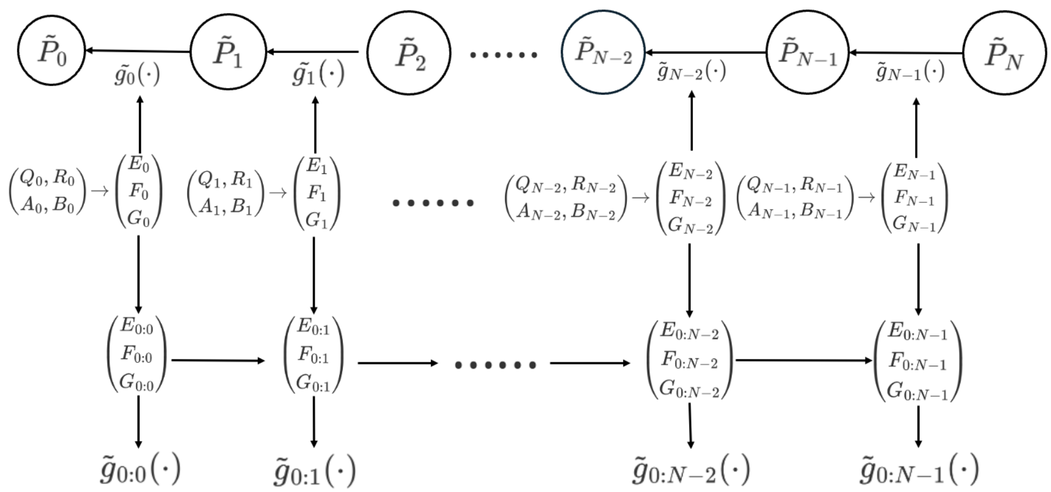

Figure 1: **Time-varying propagator.** When g_k varies with k, we switch to inverse form where each stage is an LFT $\tilde{g}_k$. The composed map $\tilde{g}_{0:k}$ remains an LFT with prefix parameters $(E_{0:k}, F_{0:k}, G_{0:k})$. This enables cheap horizon queries by reusing the composed mapping $\tilde{g}_{0:k}$ instead of reusing $\tilde{P}_k$ values.

Our key idea (Fig. 1) is to rewrite the map g_k as a new linear fractional transformation (LFT) form $\tilde{g}_{0:k}$, and some of the matrices that help compute $\tilde{g}_{0:k}$ can be reused. As a result, these matrices only need to be computed once for all possible horizons $k = 1, 2, \cdots, N$, as opposed to be repetitively computed for each possible horizon, which thus saves computational effort.

4.1 Linear Fractional Transformation Form

Let $\tilde{P}_k := P_k^{-1}$ denote the inverse matrix of P_k. Let notation $\tilde{g}_{0:k} = \tilde{g}_0 \circ \cdots \circ \tilde{g}_k$ denote a *composed map* that composes the maps $g_0, g_1, \cdots, g_k$ sequentially.

Theorem 1 (LFT Form). *There exist matrices $(E_{0:k}, F_{0:k}, G_{0:k})$, $k = 0, 1, 2, \cdots, N$ such that*

$$\tilde{g}_{0:k}(\tilde{P}) = E_{0:k} - F_{0:k}(\tilde{P} + G_{0:k})^{-1} F_{0:k}^\top, \tag{2}$$

where the prefix parameters obey the recursion (for $k \geq 1$):

$$\begin{aligned} W_k &= (E_k + G_{0:k-1})^{-1}, \\ E_{0:k} &= E_{0:k-1} - F_{0:k-1} W_k F_{0:k-1}^\top, \\ F_{0:k} &= F_{0:k-1} W_k F_k, \\ G_{0:k} &= G_k - F_k^\top W_k F_k, \end{aligned} \tag{3}$$

with base case $E_{0:0} = E_0$, $F_{0:0} = F_0$, $G_{0:0} = G_0$, and stage parameters:

$$E_k = Q_k^{-1}, \quad F_k = Q_k^{-1} A_k^\top, \quad G_k = A_k Q_k^{-1} A_k^\top + B_k R_k^{-1} B_k^\top. \tag{4}$$

4.2 Efficient Query for All Arrival Times

Given the prefix triple at $t-1$, the initial inverse for any candidate arrival time t is:

$$\tilde{P}_0^{(t)} = E_{0:t-1} - F_{0:t-1}(\tilde{P}_t + G_{0:t-1})^{-1} F_{0:t-1}^\top. \tag{5}$$

Then $P_0^{(t)} = (\tilde{P}_0^{(t)})^{-1}$ and the cost for horizon t is:

$$J_t = \tfrac{1}{2} x_0^\top P_0^{(t)} x_0 + wt. \tag{6}$$

Thus all $\{J_t\}_{t=1}^N$ are obtained from a single forward prefix build plus N terminal updates.

Complexity. The propagator has the same order as a single LQR backward sweep, $\mathcal{O}(Nn^3)$. Brute forcing all horizons by re-solving Riccati is $\mathcal{O}(N^2 n^3)$.

4.3 Augmented State Formulation for iLQR

To extend the propagator to nonlinear iLQR, we introduce an **Augmented State Formulation** that absorbs the time-varying affine linearization terms into a homogeneous coordinate system:

$$z_k = \begin{bmatrix} \delta x_k \\ 1 \end{bmatrix}. \tag{7}$$

The augmented system matrices become:

$$A_k^{\text{aug}} = \begin{bmatrix} A_k - B_k \ell_{uu,k}^{-1} \ell_{ux,k} & -B_k \ell_{uu,k}^{-1} \ell_{u,k} \\ 0 & 1 \end{bmatrix}, \quad B_k^{\text{aug}} = \begin{bmatrix} B_k \\ 0 \end{bmatrix}. \tag{8}$$

This unifies the treatment of linear and nonlinear problems, allowing the propagator to compute the *exact* LQR cost for all horizons in a single $\mathcal{O}(N)$ pass.

5 Main Contributions

1. **Propagator-based Horizon Selection:** We develop an LFT-based solver that enables the reuse of backward pass computations, reducing the complexity of horizon selection from $\mathcal{O}(N^2 n^3)$ to $\mathcal{O}(Nn^3)$.

2. **Augmented State Formulation:** We propose a state augmentation technique that embeds affine linearization terms into a homogeneous coordinate system, extending the efficient propagator method to general nonlinear iLQR problems.

3. **Performance and Robustness:** We validate our algorithm on four benchmark systems, including a 12-DOF Quadrotor. Experimental results show that our method achieves speedups of up to **43**× compared to brute-force search while guaranteeing global optimality with respect to the linearized model.

6 Results

We validate our proposed Propagator-based iLQR on four benchmark systems: Double Integrator, Segway Balance, Cartpole Swing-Up, and a 12-DOF Quadrotor. We compare three horizon-selection strategies:

- **Ours (Propagator):** The proposed method using the Augmented State Propagator.
- **Baseline-1 (Bruteforce):** Evaluating all horizons via standard backward Riccati sweeps. This serves as the ground truth but has $\mathcal{O}(N^2)$ complexity.
- **Baseline-2 (OnePass):** The current state-of-the-art approximate method, which estimates costs for neighboring horizons by reusing the value function from a single nominal backward pass. While efficient, it is only an *approximation* that can introduce significant errors for time-varying systems.

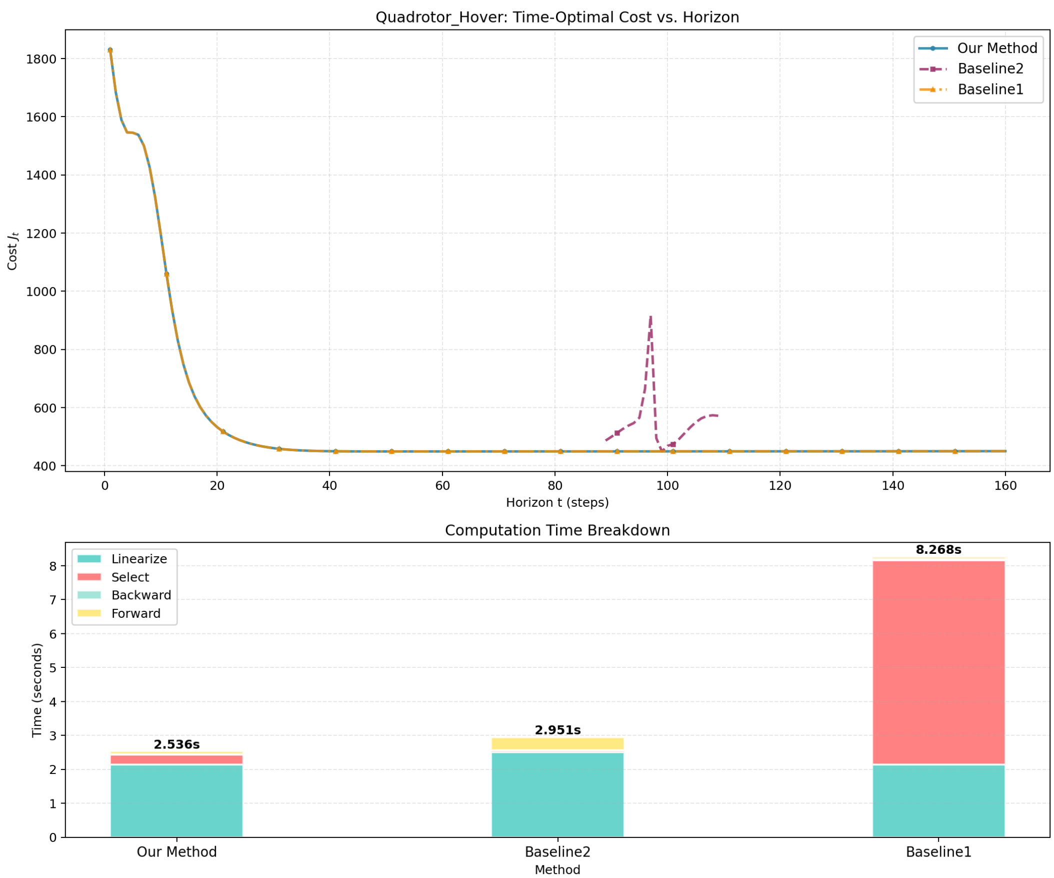

Figure 2: **Case Study on Quadrotor Hover.** (Top) Comparison of cost landscapes (J_t) computed by different methods. (Bottom) Breakdown of total runtime into linearization, selection, backward, and forward phases.

Cost Landscape (Top Panel). The top panel of Fig. 2 compares the cost curves (J_t vs. horizon t). The OnePass method (Purple), as the current SOTA, approximates the cost landscape by projecting the value function from a single nominal horizon. However, because it is merely an *estimation* method, it suffers from severe distortions when the system dynamics vary significantly over time (e.g., the artifact spike near $t = 82$). Consequently, OnePass converges to a wrong local minimum, failing to find the true optimal horizon.

In contrast, our Propagator curve (Blue) overlaps *perfectly* with the Bruteforce ground truth (Yellow). This confirms that our augmented formulation correctly captures the *exact* time-varying LQR cost—not an approximation—allowing the solver to locate the true global optimum.

Runtime Breakdown (Bottom Panel). The bottom panel decomposes the runtime to reveal the source of efficiency. The Bruteforce method (Baseline-1) computes the exact cost but at prohibitive expense—the massive "Select" phase (Red) represents the $\mathcal{O}(N^2)$ cost of repeated Riccati sweeps, taking **8.21s** total.

The OnePass method (Baseline-2) is faster (**2.88s**) but, as shown above, produces incorrect results due to its approximate nature.

Our Propagator method achieves the best of both worlds: it computes the *exact* cost like Bruteforce while running even faster than the approximate OnePass (**2.46s**). By compressing the horizon evaluation into an $\mathcal{O}(N)$ LFT propagation, we effectively eliminate the selection bottleneck while maintaining rigorous optimality.