HOP: Fast Differential Dynamic Programming for Horizon-Optimal Trajectory Planning

Author Names Omitted for Anonymous Review. Paper-ID 495

***Abstract*—This paper considers a Horizon-Optimal Control problem that seeks a dynamically feasible trajectory while minimizing the planning horizon, which is a fundamental problem in robotics with numerous applications. While many famous optimal control methods, such as LQR, iLQR/DDP, are well studied and deployed on various robots, they often have a fixed planning horizon, and their horizon-optimal counterparts are still undiscovered. The best result in the literature solves the horizon-optimal LQR problem by shifting the horizon and reusing the value functions computed by the Riccati recursion, which leads to an efficient algorithm. However, this approach is limited to LQR with time-invariant dynamics and costs only. This paper finds that the Riccati recursion can be reformulated into a form of Linear Fractional Transformation (LFT), which enjoys the structure that enables efficient computational reuse even for non-stationary dynamics and costs. Based on this insight, we develop a new efficient algorithm to solve Horizon-Optimal Time-Varying LQR problem to optimality, and further fuse it with DDP to handle general non-quadratic costs and nonlinear dynamics. Results show, our approach always finds the same optimal solution as a naive brute force baseline method, while running up to 40 times faster. For nonlinear dynamics, our method always finds better solutions than approximation using time-invariant LQR.**

I. Introduction

Optimal Control (OC) problems seek a dynamically feasible trajectory that minimizes a cost functional defined along the trajectory, which is of fundamental importance in robotics, and is the basis for numerous research topics such as agile flight of drones [16, 6], dynamic balancing of legged robots [8], trajectory planning for multiple robots [11] or information search [4]. Many famous OC approaches, such as Linear Quadratic Regulator (LQR) [1], iterative LQR (iLQR) [9], and differential dynamic programming (DDP) [7], often assume a fixed planning horizon, which limits their usage especially in applications such as drone racing [16, 6], where the horizon itself is part of the objectives to be minimized.

To bypass fixed planning horizons, horizon-optimal (or time-optimal) OC was studied and existing research can be roughly classified into two categories. The first class of methods includes the horizon itself as a decision variable and formulates the problem as a nonlinear program [16, 6, 2, 10]. While being general to handle a variety of systems and tasks, this class of methods often suffers from local minima and can be computationally expensive. The second class of methods discretizes the horizon into time steps, and seeks to extend the classic OC methods, such as LQR and DDP, to find an optimal number of time steps while optimizing the trajectories [12, 14, 5, 3]. While enjoying theoretical properties (such as solution optimality guarantees like LQR) and being computationally efficient, this class of methods is currently limited to only a few special cases that limits their usage.

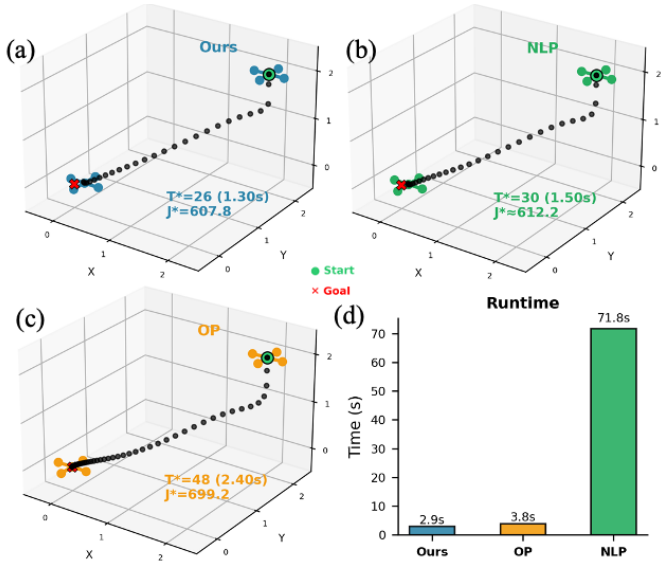


Fig. 1. For a quadrotor dynamics with 12 degrees of freedom, our HOP-DDP finds the best solution trajectory with horizon $T^* = 26$ in 2.9 seconds while the DDP based on time-invariant LQR [12] and NLP converges to local minima $T^* = 48, 30$ in 3.8 and 71.8 seconds respectively.

This paper is interested in the second class of methods, and develops new fast algorithms for horizon-optimal OC. Of close relevance to this paper, prior work [12] observes that: when solving LQR problem with Riccati recursion backwards from the end of the horizon to the starting time, as long as the dynamics and costs are stationary (i.e., time-invariant), "shifting the horizon" does not affect the value functions which allows reusing the value functions to efficiently solve the horizon-optimal LQR problem. Based on this result, prior work [12] further develops iLQR/DDP-like algorithms for nonlinear cases. However, for non-stationary dynamics or costs, the value functions are time-varying and thus cannot be reused any more, and this idea of shifting the horizon fails. This also limits the resulting iLQR/DDP, since linearizing the nonlinear dynamics along a trajectory at different times or positions can naturally lead to time-varying dynamics.

To address this challenge and bypass the assumption on stationary dynamics and costs, this paper develops a new approach HOP (Horizon-Optimal Planning). The key idea in HOP is based on an observation that, the Riccati recursion can be reformulated into a form of Linear Fractional Transformation (LFT), which enjoys the structure that enables efficient

computation by reusing a new form of value functions during the backwards pass, even for non-stationary dynamics and costs. Based on this idea, we develop HOP-LQR that can solve Horizon-Optimal Time-Varying LQR problem to optimality, and we show that its runtime complexity is same as a regular Riccati recursion for the basic LQR problem. Based on HOP-LQR, we further develop HOP-DDP by introducing an augmented state space formulation, which allows solving horizon-optimal OC problems with general nonlinear dynamics and non-quadratic costs efficiently.

We compare our HOP against several baselines on different dynamic systems. Experimental results show that, our HOP always finds the same optimal solution as a naive brute force baseline method, while running up to 40 times faster, for both linear and nonlinear systems. In comparison to the shift horizon baseline method [12], while this baseline has similar runtime as ours, this baseline gets stuck in worse local minima for almost all instances when the dynamics is nonlinear, while our HOP and the brute force baseline always find a better local minima with up to 7% cheaper costs.

II. Problem Description

A. General Problem

Let $x_{k+1} = f(x_k, u_k)$ denote the discrete-time dynamics of a system where $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$ are the state and control vectors at time step $k \in \{0, 1, \dots, T-1\}$, respectively, with $T \in \mathbb{N}$ being the planning time horizon. Let $U_T = \{u_0, \dots, u_{T-1}\}$ denote the sequence of all controls. Let $\ell : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}_+$ and $\phi : \mathbb{R}^n \to \mathbb{R}_+$ denote the stage and terminal cost functions. Let a non-negative real number $w \geq 0$ denote a weight factor.

Problem 1 (General Problem)**.**

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

Here, the decision variables include both the control sequence $U_T = \{u_0, \dots, u_{T-1}\}$ and the finish time T. The term wT penalizes the planning horizon, and thus encourages time-optimal behaviour of the system. When $w = 0$, Problem 1 becomes the regular OC problem with a fixed planning horizon. Without stage and terminal costs $\phi(x_T) = 0, \ell(x_k, u_k) = 0, k = 0, 1, \cdots, T-1$ and $w > 0$, Problem 1 becomes the regular horizon-optimal OC problem.

When the dynamics and cost terms are nonlinear and twice differentiable, Problem 1 can be iteratively approximated using Taylor expansion and solved using DDP/iLQR. For the rest of the paper, all our discussion on Problem 1 relies on the following Assumption.

Assumption 1. *The cost terms ϕ, ℓ are twice differentiable everywhere and the dynamics f can be linearized everywhere.*

B. HO-LQR Problem

We first consider a simplified variant of Problem 1. Let $A_k \in \mathbb{R}^{n \times n}$ and $B_k \in \mathbb{R}^{n \times m}$ denote the system and the input matrices, which can be time-varying as indicated by the subscript k. Let $Q_k \succeq 0$ and $R_k \succ 0$ denote the positive semi-definite state cost matrix and positive definite control cost matrix, respectively. Consider the system with linear dynamics $x_{k+1} = A_k x_k + B_k u_k$. The stage cost $\ell(x_k, u_k) = x_k^\top Q_k x_k + u_k^\top R_k u_k$ and terminal cost $x_T^\top Q_T x_T$ are both quadratic. Problem 1 becomes Horizon-Optimal Linear Quadratic Regulator (HO-LQR) problem.

Problem 2 (HO-LQR Problem)**.**

$$\begin{aligned} \min_{U_T, T} \quad & J = \tfrac{1}{2} x_T^\top Q_T x_T + \sum_{k=0}^{T-1} \tfrac{1}{2} \left(x_k^\top Q_k x_k + u_k^\top R_k u_k \right) + wT \\ \textit{s.t.} \quad & x_{k+1} = A_k x_k + B_k u_k, \quad k = 0, \dots, T-1 \\ & x_0 = \bar{x}_0 \\ & T \in \{1, 2, \dots, N\} \end{aligned} \tag{2}$$

Remark 1. *Both Problem 1 and 2 can be modified to include an additional (soft) goal constraint $x_T = x_g$ where x_g is the desired goal state to be reached by the system when $t = T$. Our approach can handle these variants with some modification. As opposed to this soft constraint on the goal state, an alternative way to include goal constraint is using soft constraint as described by the terminal cost ϕ, which is common in the optimal control literature. We therefore formulate the Problem 1 and 2 without this hard goal constraint.*

III. Preliminaries

A. Fixed Horizon LQR and Riccati Recursion

The conventional discrete LQR problem (with fixed planning horizon) can be solved to optimality via dynamic programming. Let $V_k(x_k) = \frac{1}{2} x_k^\top P_k x_k$ denote the value function that describes the cost-to-go from state x_k at time step k, where P_k is computed backwards from $k = T$ to $k = 0$. When $k = T$, the cost-to-go is same as the terminal cost with $V_T(x_T) = \frac{1}{2} x_T^\top Q_T x_T$ and $P_T = Q_T$. For other $k = \{1, 2, \cdots, T-1\}$, the optimal solution is characterized by the backwards Riccati equations:

$$\begin{aligned} S_k &= R_k + B_k^\top P_{k+1} B_k, \\ K_k &= S_k^{-1} B_k^\top P_{k+1} A_k, \\ P_k &= Q_k + A_k^\top P_{k+1} A_k - (A_k^\top P_{k+1} B_k) K_k. \end{aligned} \tag{3}$$

With a backward pass from $k = T$ to $k = 1$, all K_k, P_k matrices can be computed and the optimal control for each step can be obtained by $u_k^* = -K_k x_k$.

B. Horizon-Optimal Time-Invariant LQR

As opposed to fixing the planning horizon T in LQR, T can also be optimized for systems with stationary dynamics and costs [12]. Specifically, let g denote a mapping corresponding to the computation process $P_k = g(P_{k+1})$ which obtains P_k

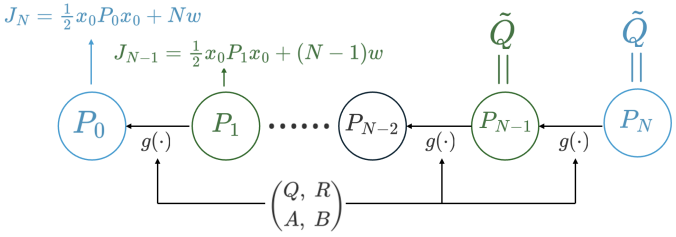


Fig. 2. Illustration of the "shift horizon" one pass approach in [12] for time-invariant HO-LQR. The map g can be computed based on (A, B, Q, R) matrices, and is identical for all steps k. One backward pass yields all $\{P_k\}$, which allows computing J_k by shifting the terminal index. For example, for horizon $k = N$ (blue), the corresponding cost J_N can be computed by using P_0. For horizon $k = N-1$ (green), the corresponding cost J_{N-1} can be computed by using P_1. The optimal horizon can be computed by finding the horizon k^* such that J_{k^*} reaches the minimum among J_k for all possible k.

from P_{k+1}. When the system dynamics satisfy $A_k = A, \forall k$, $B_k = B, \forall k$ (i.e., $x_{k+1} = Ax_k + Bu_k, \quad k = 0, \ldots, T-1$) and the cost terms satisfy $Q_k = Q, \forall k$, $R_k = R, \forall k$, the matrices A, B, Q, R in the Riccati equations are all constant for all time steps k. As a result, the mapping g becomes

$$\begin{aligned} P_k &= g(P_{k+1}) \\ &= Q + A^\top P_{k+1}\big[I - B(R + B^\top P_{k+1} B)^{-1} B^\top P_{k+1}\big] A. \end{aligned} \tag{4}$$

which is invariant across different time steps.

This enables computational reuse: a single Riccati backward pass yields $\{P_0, P_1, \ldots, P_N\}$, from which we can extract the cost for any horizon t as:

$$J_t \approx \tfrac{1}{2} x_0^\top P_{N-t} x_0 + tw. \tag{5}$$

The minimum among all these $J_t, t = 0, 1, 2, \cdots, N-1$ provides the optimal solution to the problem and the corresponding t is the optimal horizon.

This result is already shown in optimal-horizon control [12]. We illustrate this computational process in Fig. 2. The same map $g(\cdot)$ is reused at every step, so one backward pass produces all $\{P_k, k = 0, 1, 2, \cdots, T\}$. The colors emphasize horizon shifting. For example, the blue case seeks J_N, uses P_N as the terminal matrix (i.e., time index is $N{+}1$) and computes the cost via P_0, while the green case seeks J_{N-1}, uses P_{N-1} as the terminal matrix (i.e., time index is N) and computes the cost via P_1.

However, for time-varying systems, the mapping g becomes g_k that is also time-varying, and one would have to compute g_k for each time possible horizon k. As a result, the P-matrices P_k cannot be reused since for different horizon k, the set of matrices $P_t, t = 0, 1, 2, \cdots, k$ also varies and cannot be reused. A naive approach is to solve for each possible horizon $k = 1, 2, \cdots, N$ with a Riccati recursion, which leads to N Riccati recursions in total and is computationally inefficient.

IV. Optimal Horizon Time-Varying LQR

To address this challenge, our key idea (Fig. 3) is to rewrite the map g_k as a new linear fractional transformation (LFT) form $\tilde{g}_{0:k}$ (which is explained later), 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.

A. Linear Fractional Transformation Form

We first define necessary notations, and then rewrite the Riccati recursion into a new form of Linear Fractional Transformation (LFT) based on the inverse of cost-to-go matrices P_k. We prove that this LFT form is equivalent to the original Riccati recursion in Theorem 1. Based on this result, we further derive a LFT form for the time-varying case of LQR in Theorem 2, which then leads to an efficient algorithm HOP-LQR for the time-varying HO-LQR problem as explained in the next subsection.

Let $\tilde{P}_k := P_k^{-1}$ denote the inverse of the cost-to-go matrix P_k. Let $\tilde{g}_k$ denote the map from $\tilde{P}_{k+1}$ to $\tilde{P}_k$, i.e., $\tilde{P}_k = \tilde{g}_k(\tilde{P}_{k+1})$. Let notation $\tilde{g}_{0:k} = \tilde{g}_0 \circ \cdots \circ \tilde{g}_k$ denote a *composed map* that composes the maps $\tilde{g}_0, \tilde{g}_1, \cdots, \tilde{g}_k$ sequentially, i.e., $\tilde{P}_0 = \tilde{g}_{0:k}(\tilde{P}_{k+1})$

Theorem 1 (LFT form for Riccati Recursion)**.** *The Riccati recursion is equivalent to a Linear Fractional Transformation (LFT) on the inverse of P_k:*

$$\tilde{g}_k(\tilde{P}) \triangleq E_k - F_k(\tilde{P} + G_k)^{-1} F_k^\top, \tag{6}$$

such that $\tilde{P}_k = \tilde{g}_k(\tilde{P}_{k+1})$. where,

$$E_k = Q_k^{-1}, \; F_k = Q_k^{-1} A_k^\top, \; G_k = A_k Q_k^{-1} A_k^\top + B_k R_k^{-1} B_k^\top. \tag{7}$$

Proof: We will use the Woodbury matrix identity: for invertible matrices A and C, the following identity holds.

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}, \tag{8}$$

where A, C, U and V are matrices of sizes $n_1 \times n_1$, $n_2 \times n_2$, $n_1 \times n_2$, and $n_2 \times n_1$ respectively.

Our proof consists of two steps. First, we rewrite the Riccati equation as:

$$P_k = Q_k + A_k^\top \mathcal{K} A_k \tag{9}$$

where, $\mathcal{K} = \big[P_{k+1} - P_{k+1}B_k(R_k + B_k^\top P_{k+1} B_k)^{-1} B_k^\top P_{k+1}\big]$. We recognize the term $\mathcal{K}$ as the right-hand side of the Woodbury identity. By setting $A^{-1} = P_{k+1}$, $U = B_k$, $V = B_k^\top$, and $C^{-1} = R_k$, the identity (8) implies:

$$\mathcal{K} = (P_{k+1}^{-1} + B_k R_k^{-1} B_k^\top)^{-1}.$$

Substituting $\tilde{P}_{k+1} = P_{k+1}^{-1}$ and defining $S_k \triangleq B_k R_k^{-1} B_k^\top$, the Riccati equation becomes:

$$P_k = Q_k + A_k^\top (\tilde{P}_{k+1} + S_k)^{-1} A_k. \tag{10}$$

Next, we invert Eq. (10) to obtain the LFT form:

$$\tilde{P}_k = P_k^{-1} = \Big(Q_k + A_k^\top (\tilde{P}_{k+1} + S_k)^{-1} A_k\Big)^{-1}. \tag{11}$$

We apply the Woodbury identity (8) again by setting $A = Q_k$, $U = A_k^\top$, $V = A_k$, and $C = (\tilde{P}_{k+1} + S_k)^{-1}$, which yields:

$$\tilde{P}_k = Q_k^{-1} - Q_k^{-1} A_k^\top \Big((\tilde{P}_{k+1} + S_k) + A_k Q_k^{-1} A_k^\top\Big)^{-1} A_k Q_k^{-1}. \tag{12}$$

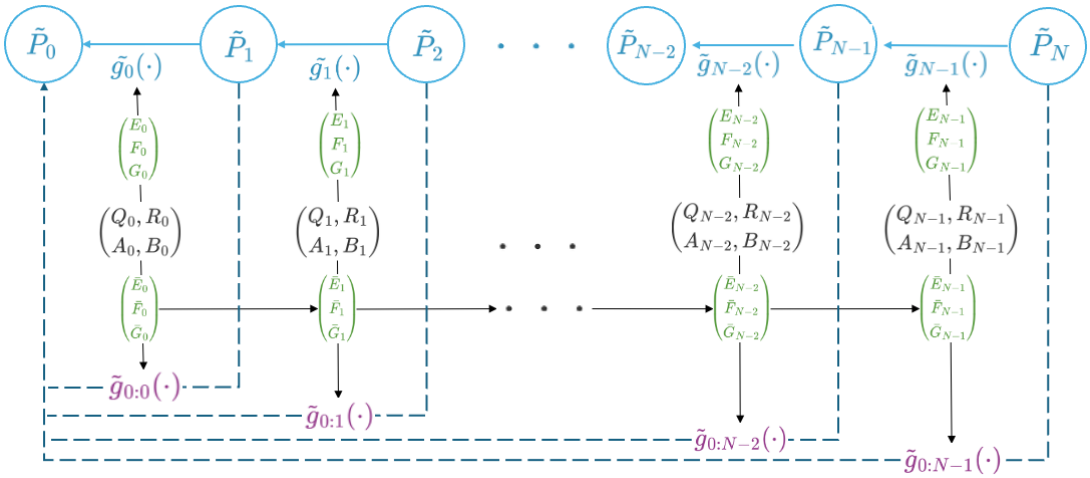


Fig. 3. Illustration of our HOP-LQR Algorithm. Green Blocks: The system matrices are used to compute matrices (E_k, F_k, G_k) and $(\overline{E}_k, \overline{F}_k, \overline{G}_k)$. Top Row (Blue): The maps $\tilde{g}_k(\cdot)$ forms the regular backward pass that maps $\tilde{P}_k$ to $\tilde{P}_{k-1}$. Bottom Row (Purple): The matrices $(\overline{E}_k, \overline{F}_k, \overline{G}_k)$ allow computing the composed maps $\tilde{g}_{0:k}(\cdot)$, which enables direct evaluation of the inverse of initial cost-to-go $(\tilde{P}_0)$ for any horizon k. For example, to evaluate the cost for horizon $N-1$, let $\tilde{P}_{N-1}$ be the terminal cost-to-go matrix $\tilde{P}_T$. Then, the initial cost-to-go matrix is $\tilde{P}_0^{(N-1)} = \tilde{g}_{0:N-2}(\tilde{P}_{N-1})$. Then, the cost can be computed as $J_{N-1} = \frac{1}{2}x_0^\top(\tilde{P}_0^{(N-1)})^{-1}x_0 + w\cdot(N-1)$. After evaluating all the time steps $J_k, k = 1, 2, \cdots, N$, we can find the minimal cost and select the corresponding time step as the optimal horizon.

Define $E_k = Q_k^{-1}$, $F_k = Q_k^{-1}A_k^\top$, $G_k = A_kQ_k^{-1}A_k^\top + B_kR_k^{-1}B_k^\top$, we obtain the desired LFT form. ∎

Theorem 1 allows us to rewrite the composite map into LFT form as well.

Theorem 2 (LFT form of the composed maps)**.** *There exist matrices $(\overline{E}_k, \overline{F}_k, \overline{G}_k)$, $k = 0, 1, 2, \cdots, N$ such that*

$$\tilde{g}_{0:k}(\tilde{P}) = \overline{E}_k - \overline{F}_k\,(\tilde{P} + \overline{G}_k)^{-1}\overline{F}_k^\top, \tag{13}$$

where,

$$\begin{aligned} W_k &= (E_k + \overline{G}_{k-1})^{-1}, \\ \overline{E}_k &= \overline{E}_{k-1} - \overline{F}_{k-1}W_k\overline{F}_{k-1}^\top, \\ \overline{F}_k &= \overline{F}_{k-1}W_kF_k, \\ \overline{G}_k &= G_k - F_k^\top W_kF_k, \end{aligned} \tag{14}$$

with $\overline{E}_0 = E_0,\ \overline{F}_0 = F_0,\ \overline{G}_0 = G_0$.

Note that (A_k, B_k, Q_k, R_k) matrices are known as the input of the problem, and $(E_k, F_k, G_k, \overline{E}_k, \overline{F}_k, \overline{G}_k)$ are intermediate variables computed based on (A_k, B_k, Q_k, R_k) and themselves recursively, and the composed map $\tilde{g}_k$ is computed based on $\overline{E}_k, \overline{F}_k, \overline{G}_k$ recursively. We will explain this recursive computation later in Alg. 1 with the help of Fig. 3. We now prove the correctness of this theorem.

Proof: We prove Theorem 2 by induction. First, for the base case ($k = 0$), it holds due to Theorem 1 and that $(\overline{E}_0, \overline{F}_0, \overline{G}_0) = (E_0, F_0, G_0)$. Then, for the inductive step, assume the claim holds for $k-1$:

$$\tilde{g}_{0:k-1}(\tilde{P}) = \overline{E}_{k-1} - \overline{F}_{k-1}(\tilde{P} + \overline{G}_{k-1})^{-1}\overline{F}_{k-1}^\top. \tag{15}$$

By composition, $\tilde{g}_{0:k}(\tilde{P}) = \tilde{g}_{0:k-1}\big(\tilde{g}_k(\tilde{P})\big)$. Substituting Eq. (6) into the Eq. (15) yields

$$\begin{aligned} \tilde{g}_{0:k}(\tilde{P}) = \overline{E}_{k-1} - \overline{F}_{k-1}\Big(\big[E_k - F_k(\tilde{P} + G_k)^{-1}F_k^\top\big] \\ + \overline{G}_{k-1}\Big)^{-1}\overline{F}_{k-1}^\top. \end{aligned} \tag{16}$$

We simplify the inverse term between $\overline{F}_{k-1}$ and $\overline{F}_{k-1}^\top$. Let $W_k \triangleq (E_k + \overline{G}_{k-1})^{-1}$. The term to be inverted becomes:

$$\mathcal{M} \triangleq \Big(W_k^{-1} - F_k(\tilde{P} + G_k)^{-1}F_k^\top\Big)^{-1}. \tag{17}$$

Applying the Woodbury identity (8) with $A = W_k^{-1}$:

$$\mathcal{M} = W_k + W_kF_k\Big((\tilde{P} + G_k) - F_k^\top W_kF_k\Big)^{-1}F_k^\top W_k. \tag{18}$$

Substituting (18) back into (16) yields:

$$\begin{aligned} \tilde{g}_{0:k}(\tilde{P}) = &\underbrace{\overline{E}_{k-1} - \overline{F}_{k-1}W_k\overline{F}_{k-1}^\top}_{\overline{E}_k} \\ &- \underbrace{\overline{F}_{k-1}W_kF_k}_{\overline{F}_k}(\tilde{P} + \underbrace{G_k - F_k^\top W_kF_k}_{\overline{G}_k})^{-1}\underbrace{F_k^\top W_k\overline{F}_{k-1}^\top}_{\overline{F}_k^\top}. \end{aligned}$$

The resulting terms match the desired recursive form. ∎

B. HOP-LQR Algorithm

Based on Theorem 2, the complete HOP-LQR is summarized in Alg. 1 and illustrated in Fig. 3. HOP-LQR can be divided into two phases.

1) Phase 1 Compute Composed Maps: HOP-LQR performs a single forward sweep to build the parameters of $\tilde{g}_k$. First, for each step k, we use system matrices $\{Q_i, R_i, A_i, B_i\}_{i=0}^N$ to compute the parameters (E_k, F_k, G_k) (highlighted in green). As shown in the Top Row (highlighted in blue) of Fig. 3, these parameters fully define the single-step mapping $\tilde{g}_k$, which represents the LFT form of the Riccati equation (Theorem 1). Then, instead of executing the backward recursion immediately, we use the recursive formula in Theorem 2 to accumulate these parameters forward, which helps compute the matrices $(\overline{E}_k, \overline{F}_k, \overline{G}_k)$ (highlighted in green) that will be used to compute the composed map $\tilde{g}_{0:k}$ from time 0 to k.

Algorithm 1 HOP-LQR

Input: System matrices $\{A_k, B_k, Q_k, R_k\}$, Initial state x_0, Terminal cost-to-go $\tilde{P}_T$ matrix, Time penalty w, Max horizon N
Output: Optimal costs $\{J_t\}_{t=1}^{N}$ for all arrival times

```
// Phase 1: Compute Composed Maps
```
for $k \leftarrow 0$ **to** $N-1$ **do**
  $E_k \leftarrow Q_k^{-1}$
  $F_k \leftarrow Q_k^{-1} A_k^\top$
  $G_k \leftarrow A_k Q_k^{-1} A_k^\top + B_k R_k^{-1} B_k^\top$
end
Initialize: $\overline{E}_0 \leftarrow E_0$, $\overline{F}_0 \leftarrow F_0$, $\overline{G}_0 \leftarrow G_0$
for $k \leftarrow 1$ **to** $N-1$ **do**
  $W_k \leftarrow (E_k + \overline{G}_{k-1})^{-1}$
  $\overline{E}_k \leftarrow \overline{E}_{k-1} - \overline{F}_{k-1} W_k \overline{F}_{k-1}^\top$
  $\overline{F}_k \leftarrow \overline{F}_{k-1} W_k F_k$ $\overline{G}_k \leftarrow G_k - F_k^\top W_k F_k$
end
```
// Phase 2: Fast Horizon Queries
```
for $t \leftarrow 1$ **to** N **do**
  $\tilde{P}_0 \leftarrow \overline{E}_{t-1} - \overline{F}_{t-1}(\tilde{P}_T + \overline{G}_{t-1})^{-1} \overline{F}_{t-1}^\top$
  $P_0 \leftarrow \tilde{P}_0^{-1}$
  $J_t \leftarrow \frac{1}{2} x_0^\top P_0 x_0 + t \cdot w$
end
return $\{J_t\}_{t=1}^{N}$, $\arg\min_t J_t$

2) Phase 2 Fast Horizon Query: With the composed maps $\tilde{g}_{0:k}$ stored, evaluating the cost J_k for any horizon $k = 1, 2, \cdots, N$ becomes a straightforward evaluation of J_k. We apply the stored map at index $t-1$ to the terminal condition $\tilde{P}_t = \tilde{P}_T$. Specifically, the initial inverse cost-to-go $\tilde{P}_0^{(t)}$ is:

$$\tilde{P}_0^{(t)} = \tilde{g}_{0:t-1}(\tilde{P}_T). \tag{19}$$

Once $\tilde{P}_0^{(t)}$ is obtained, the total cost is given by

$$J_t = \frac{1}{2} x_0^\top (\tilde{P}_0^{(t)})^{-1} x_0 + w \cdot t \tag{20}$$

This effectively avoids the need to solve the chain of $\tilde{g}$ functions one by one.

With all J_k computed, HOP-LQR can find the k^* such that J_{k^*} reaches the minimum, and this k^* is the optimal horizon to the problem. The controls can be obtained in the same way as in regular LQR:

$$u_k = -\underbrace{R_k^{-1} B_k^\top (\tilde{P}_{k+1} + B_k R_k^{-1} B_k^\top)^{-1} A_k}_{K_k} x_k, \tag{21}$$

C. Complexity Analysis

Recall that N is the number of all possible horizons to be chosen from, and n is the dimensionality of the system's state x. As aforementioned, a naive approach for the Horizon-Optimal Time-Varying LQR problem requires solving the Riccati equation repeatedly for every horizon $k = 1, \ldots, N$, which has a runtime complexity of $\mathcal{O}(N^2 n^3)$.

In contrast, our HOP-LQR only runs the forward recursion once to compute the composed maps, and then performs simple queries based on these maps. The runtime complexity for computing these composed map is $\mathcal{O}(Nn^3)$ and the runtime complexity for the queries is $O(Nn^3)$ as well. The total runtime complexity is $\mathcal{O}(Nn^3)$. We summarize this result with the following theorem.

Theorem 3. *With N being the number of all possible horizons to be chosen from, and n being the dimensionality of the system's state x, the proposed HOP-LQR algorithm has a runtime complexity of $\mathcal{O}(Nn^3)$.*

In other words, compared to the naive approach, HOP-LQR reduces the complexity from quadratic to linear with respect to the number of horizons N, which is of the same runtime complexity of the Riccati recursion for the fixed-horizon LQR.

V. Optimal Horizon iLQR and DDP

A. Augmented Dynamics and Cost

Standard DDP/iLQR [9] solves nonlinear OC by iteratively approximating them as linear-quadratic sub-problems. Linearization introduces affine terms, which break the standard pure-quadratic LQR form ($x^\top P x$) required by our LFT-based method as aforementioned. To address this, we perform a transformation including (1) approximation of the dynamics and cost function, (2) quadratization via completing the square, and (3) state augmentation.

1) Approximation: Given a nominal trajectory $(\bar{x}_k, \bar{u}_k)$, we define the deviations $\delta x_k := x_k - \bar{x}_k$ and $\delta u_k := u_k - \bar{u}_k$. The linearized dynamics are:

$$\delta x_{k+1} = A_k \delta x_k + B_k \delta u_k + a_k, \tag{22}$$

where $A_k = \nabla_x f(\bar{x}_k, \bar{u}_k)$, $B_k = \nabla_u f(\bar{x}_k, \bar{u}_k)$, and $a_k = f(\bar{x}_k, \bar{u}_k) - \bar{x}_{k+1}$.

The stage cost is Taylor expanded to the second order:

$$\begin{aligned} \ell_k \approx\ & \ell(\bar{x}_k, \bar{u}_k) + w + \ell_{x,k}^\top \delta x_k + \ell_{u,k}^\top \delta u_k \\ & + \tfrac{1}{2} \delta x_k^\top \ell_{xx,k} \delta x_k + \delta x_k^\top \ell_{xu,k} \delta u_k \\ & + \tfrac{1}{2} \delta u_k^\top \ell_{uu,k} \delta u_k, \end{aligned} \tag{23}$$

which are partial derivatives evaluated at $(\bar{x}_k, \bar{u}_k)$, and the time penalty w is included as a constant term.

2) Quadratization: The cross-term $\delta x_k^\top \ell_{xu,k} \delta u_k$ prevents direct application of standard LQR methods. To eliminate it, we complete the square for the control terms. We first group all terms dependent on δu_k:

$$\tfrac{1}{2} \delta u_k^\top \ell_{uu,k} \delta u_k + \delta u_k^\top (\ell_{ux,k} \delta x_k + \ell_{u,k}).$$

Recalling the identity $\frac{1}{2} v^\top R v = \frac{1}{2}(u + R^{-1} b)^\top R (u + R^{-1} b) = \frac{1}{2} u^\top R u + u^\top b + \text{const.}$, we can match terms to define the new control variable v_k:

$$v_k := \delta u_k + \ell_{uu,k}^{-1} (\ell_{ux,k} \delta x_k + \ell_{u,k}). \tag{24}$$

Substituting v_k back into the cost function allows us to express the stage cost in a decoupled form. The expansion is:

$$\begin{aligned} \ell_k \approx\ & \tfrac{1}{2} \delta x_k^\top (\ell_{xx,k} - \ell_{xu,k} \ell_{uu,k}^{-1} \ell_{ux,k}) \delta x_k \\ & + (\ell_{x,k} - \ell_{xu,k} \ell_{uu,k}^{-1} \ell_{u,k})^\top \delta x_k \\ & + \tfrac{1}{2} v_k^\top \ell_{uu,k} v_k \\ & + \left(\ell(\bar{x}_k, \bar{u}_k) + w - \tfrac{1}{2} \ell_{u,k}^\top \ell_{uu,k}^{-1} \ell_{u,k} \right). \end{aligned} \tag{25}$$

3) Augmentation: Even after using the new control variable, the dynamics and cost still contain state-dependent affine terms. We eliminate these by lifting the system into a homogeneous coordinate space via the augmented state $z_k = [\delta x_k,\ 1]^T$. This allows us to absorb all linear and constant terms into the quadratic form.

a) Augmented Dynamics: Substituting the new control variable v_k (Eq. 24) into the dynamics yields the explicit update equation:

$$\begin{aligned}\delta x_{k+1} = &\left(A_k - B_k \ell_{uu,k}^{-1} \ell_{ux,k}\right) \delta x_k + B_k v_k \\ &+ \left(a_k - B_k \ell_{uu,k}^{-1} \ell_{u,k}\right).\end{aligned} \tag{26}$$

We then rewrite it based on z_k and get the new augmented system dynamics:

$$z_{k+1} = A_k^{\text{aug}} z_k + B_k^{\text{aug}} v_k, \tag{27}$$

$$A_k^{\text{aug}} = \begin{bmatrix} A_k - B_k \ell_{uu,k}^{-1} \ell_{ux,k} & a_k - B_k \ell_{uu,k}^{-1} \ell_{u,k} \\ 0 & 1 \end{bmatrix}, \tag{28}$$

$$B_k^{\text{aug}} = \begin{bmatrix} B_k \\ 0 \end{bmatrix}. \tag{29}$$

b) Augmented Cost: Similarly, we define notations for the modified cost terms:

$$\tilde{Q}_k = \ell_{xx,k} - \ell_{xu,k} \ell_{uu,k}^{-1} \ell_{ux,k}, \quad \tilde{q}_k = \ell_{x,k} - \ell_{xu,k} \ell_{uu,k}^{-1} \ell_{u,k}.$$

Then the augmented cost matrix becomes:

$$Q_k^{\text{aug}} = \begin{bmatrix} \tilde{Q}_k & \tilde{q}_k \\ \tilde{q}_k^\top & 2\left(\ell(\bar{x}_k, \bar{u}_k) + w - \frac{1}{2} \ell_{u,k}^\top \ell_{uu,k}^{-1} \ell_{u,k}\right) \end{bmatrix}. \tag{30}$$

$$R_k = \ell_{uu,k}.$$

The stage cost in augmented form is quadratic:

$$\ell_k \approx \tfrac{1}{2} z_k^\top Q_k^{\text{aug}} z_k + \tfrac{1}{2} v_k^\top R_k v_k \tag{31}$$

Similarly, the terminal cost matrix is:

$$Q_N^{\text{aug}} = \begin{bmatrix} \phi_{xx,N} & \phi_{x,N} \\ \phi_{x,N}^\top & 2\phi(\bar{x}_N) \end{bmatrix}. \tag{32}$$

B. HOP-DDP Algorithm

We now present the complete HOP-DDP Algorithm. With the augmented state space derived in Section V-A, the iterative optimization can be described in four steps (Alg. 2).

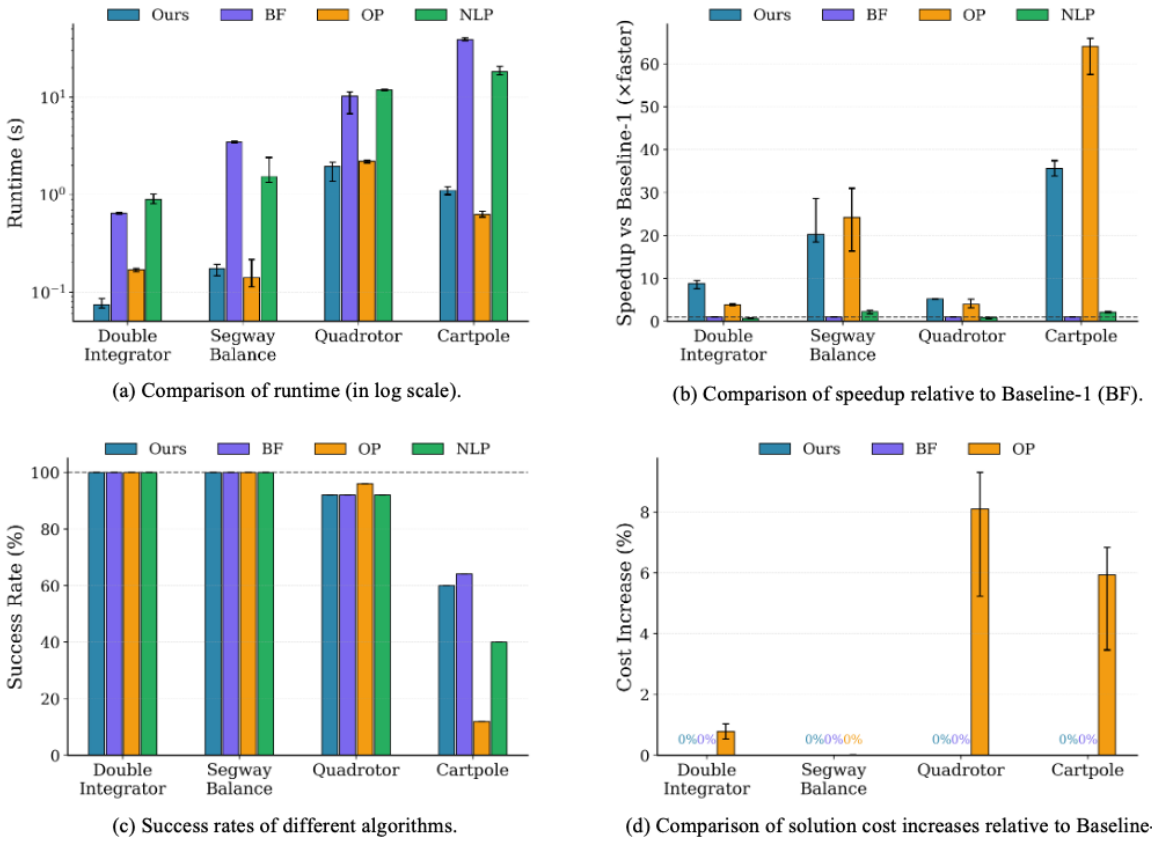
1) Step 1: Linearization and Augmentation: The iteration begins by linearizing the nonlinear dynamics and quadratizing the cost function around the current nominal trajectory $(\bar{x}, \bar{u})$ as mentioned in Sec. V-A, which yields the augmented system matrices $A_k^{\text{aug}}, B_k^{\text{aug}}$ (Eq. 28) and the augmented cost matrices Q_k^{aug}, R_k (Eq. 31).

2) Step 2: Horizon Selection: With the system $\{(A_k^{\text{aug}}, B_k^{\text{aug}}, Q_k^{\text{aug}}, R_k)\}_{k=0}^{N-1}$ in a standard linear-quadratic form, we can now find an optimal horizon T^* by calling the aforementioned HOP-LQR. This step identifies the optimal stopping time T^* without requiring a full backward pass for each candidate horizon.

Algorithm 2 HOP-DDP Algorithm

Input: Dynamics f, costs ℓ, ϕ, initial state x_0, initial controls U, horizon bounds $[T_{\min}, T_{\max}]$, time penalty w, $N = T_{\max}$
Output: Optimal trajectory and controls
repeat
  `// Step 1: Linearization and augmentation`
  Rollout X using f and U **for** $k = 0$ **to** $N-1$ **do**
    Compute A_k, B_k at $(\bar{x}_k, \bar{u}_k)$
    Compute derivatives: $\ell_{x,k}, \ell_{u,k}, \ell_{xx,k}, \ell_{ux,k}, \ell_{uu,k}$
    Compute $A_k^{\text{aug}}, B_k^{\text{aug}}, Q_k^{\text{aug}}, R_k$ via Eqs. (28)-(30)
  end
  Build Q_T^{aug} from ϕ at $\bar{x}_T$
  `// Step 2: Horizon selection`
  $J \leftarrow$ HOP-LQR $(A^{\text{aug}}, B^{\text{aug}}, Q^{\text{aug}}, R, z_0, Q_T^{\text{aug}}, N)$
  $T^* \leftarrow \arg\min_{t \in [T_{\min}, T_{\max}]} J[t]$
  `// Step 3: Backward pass on` $[0, T^*-1]$
  $V_{xx}[T^*] \leftarrow \phi_{xx}$, $V_x[T^*] \leftarrow \phi_x$, $V_0[T^*] \leftarrow \phi(\bar{x}_{T^*})$
  for $k = T^*-1, T^*-2, \cdots, 0$ **do**
    `// Compute derivatives (DDP terms in parentheses, iLQR ignores them)`
    $Q_x \leftarrow \ell_{x,k} + A_k^\top V_{x,k+1}$ $\left(+ A_k^\top V_{xx,k+1}\, a_k\right)$
    $Q_u \leftarrow \ell_{u,k} + B_k^\top V_{x,k+1}$ $\left(+ B_k^\top V_{xx,k+1}\, a_k\right)$
    $Q_{xx} \leftarrow \ell_{xx,k} + A_k^\top V_{xx,k+1} A_k$ $\left(+ \sum_{i=1}^{n_x} V_{x,k+1}^{(i)} f_{xx,k}^{(i)}\right)$
    $Q_{ux} \leftarrow \ell_{ux,k} + B_k^\top V_{xx,k+1} A_k$ $\left(+ \sum_{i=1}^{n_x} V_{x,k+1}^{(i)} f_{ux,k}^{(i)}\right)$
    $Q_{uu} \leftarrow \ell_{uu,k} + B_k^\top V_{xx,k+1} B_k$ $\left(+ \sum_{i=1}^{n_x} V_{x,k+1}^{(i)} f_{uu,k}^{(i)}\right)$
    $Q_{xu} \leftarrow Q_{ux}^\top$
    $Q_{uu} \leftarrow Q_{uu} + \lambda I$ `// LM regularization`
    `// Compute gains`
    $\kappa_k \leftarrow -Q_{uu}^{-1} Q_u$, $K_k \leftarrow -Q_{uu}^{-1} Q_{ux}$
    `// Update value function`
    $V_{xx,k} \leftarrow Q_{xx} - Q_{ux}^\top Q_{uu}^{-1} Q_{ux}$
    $V_{x,k} \leftarrow Q_x - Q_{ux}^\top Q_{uu}^{-1} Q_u$
    $V_{0,k} \leftarrow V_{0,k+1} + \ell(\bar{x}_k, \bar{u}_k) + w + \kappa_k^\top Q_u + \frac{1}{2} \kappa_k^\top Q_{uu} \kappa_k$
  end
  `// Step 4: Forward rollout with line search`
  for $\alpha \in I_\alpha$ **do**
    $x_{\text{new}}[0] \leftarrow x_0$
    for $k = 0$ **to** T^*-1 **do**
      $\delta x \leftarrow x_{\text{new}}[k] - \bar{x}_k$
      $\delta u \leftarrow K_k \cdot \delta x + \alpha \cdot \kappa_k$
      $u_{\text{new}}[k] \leftarrow \bar{u}_k + \delta u$
      $x_{\text{new}}[k+1] \leftarrow f(x_{\text{new}}[k], u_{\text{new}}[k])$
    end
    if *TrueCost*$(x_{new}, u_{new}, T^*) <$ *TrueCost*$(\bar{x}, \bar{u}, T^*)$ **then**
      Accept trajectory; **break**
    end
  end
  if *not accepted* **then**
    $\lambda* = 10$ `// Increase LM regularization`
  end
  else
    Update nominal (X, U)
  end
until *convergence*

3) Step 3: Truncated Backward Pass: Once T^* is known, we perform the *truncated* backward pass, which is executed only on the interval $[0, T^*-1]$, rather than the maximum possible horizon N. Specifically, we initialize the value function at T^* using the terminal cost $\phi(\bar{x}_{T^*})$. Then, proceeding backward from $k = T^*-1$ to 0, we compute the derivatives of



(a) Comparison of runtime (in log scale).

(b) Comparison of speedup relative to Baseline-1 (BF).

(c) Success rates of different algorithms.

(d) Comparison of solution cost increases relative to Baseline-1 (BF).

Fig. 4. Experimental results. (a) Comparison of runtime in log scale. Error bars indicate variability across trials. (b) Comparison of speedup relative to Baseline-1 (BF) over the four systems. This figure shows the same results in (a) in a different way. (c) Success rates of different algorithms. (d) Comparison of solution cost increases relative to Baseline-1 (BF). Our method finds solutions of the better quality than OP and NLP, while enjoying fast running speed.

Q-matrices and extract the feedback gains as in regular DDP:

$$K_k = -Q_{uu,k}^{-1} Q_{ux,k}, \quad \kappa_k = -Q_{uu,k}^{-1} Q_{u,k}. \tag{33}$$

This yields the modification on the current control $\delta u_k = K_k \delta x_k + \kappa_k$ for the selected horizon T^*.

4) Step 4: Forward Rollout and Update: Finally, we apply the computed gains to the original nonlinear system $x_{k+1} = f(x_k, u_k)$ to generate a new trajectory. We perform a line search on the step size $\alpha \in I_\alpha$ to ensure convergence, where I_α is a user-specified set of possible step sizes. To generate the new trajectory, we compute the new controls as follows.

$$u_k^{\text{new}} = \bar{u}_k + K_k(x_k^{\text{new}} - \bar{x}_k) + \alpha \kappa_k. \tag{34}$$

Based on the computed new controls, a new state trajectory can be rolled out. This rollout is terminated at time step T^*. If the total nonlinear cost of the trajectory decreases, the new trajectory is accepted as the nominal trajectory for the next iteration; otherwise, the regularization parameter is increased similarly to [13].

VI. Experimental Results

We compare our HOP-DDP against several baselines on four systems including linear and nonlinear dynamics: Double Integrator, Segway Balance, Cartpole Swing-Up, and a 12-DOF Quadrotor with 25 cases each. Here, Double Integrator and Segway (linearized about the equilibrium) have two linear dynamics, while Cartpole and Quadrotor have nonlinear dynamics. We use three baseline methods for comparison.

1) Baseline-1 (BruteForce, BF) evaluates all horizons via backward Riccati recursions to select the horizon.
2) Baseline-2 (OnePass, OP) [12] uses time-invariant LQR to approximate nonlinear systems.
3) Baseline-3 (NLP) includes the time horizon as a decision variable, uses time-scaled transcription, and solves the resulting NLP using IPOPT [15].

A. Experiment 1: Overall Performance

a) Runtime and Speedup: Fig. 4(a) reports the runtime on log scale, and Fig. 4(b) shows the corresponding speedup relative to Baseline-1 (BF). For all systems, our method consistently runs faster than Baseline-1 (BF): $\sim 9\times$ faster on Double Integrator, $\sim 20\times$ faster on Segway Balance, $\sim 5\times$ faster on Quadrotor, and $\sim 40\times$ faster on Cartpole. Baseline-2 (OP) can be even faster on some tasks (e.g. Cartpole and Segway), but at the cost of worse solution quality as explained later. Baseline-3 (NLP) usually has a similar runtime to Baseline-1 (BF), yet runs slower on some systems, which indicates the expensive computation of solving a large nonlinear program.

b) Success Rates: Fig. 4(c) reports success rates, which is the percentage of trials where the cost function converges

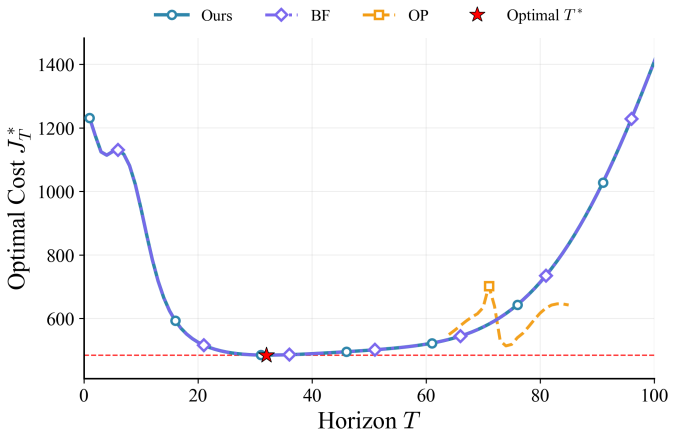


Fig. 5. Cost values for various horizons for the hovering task of Quadrotor. Our method and Baseline-1 (BF) both finds an optimal horizon $T^* = 32$ whose corresponding optimal costs are $J_{32}^{\text{ours}} \approx 484.79$, $J_{32}^{\text{BF}} \approx 484.80$, while Baseline-2 (OP) converges to a different local minimum with a longer horizon ($T \approx 74$) and 5.97% higher cost ($J_{74}^{\text{OP}} \approx 513.75$).

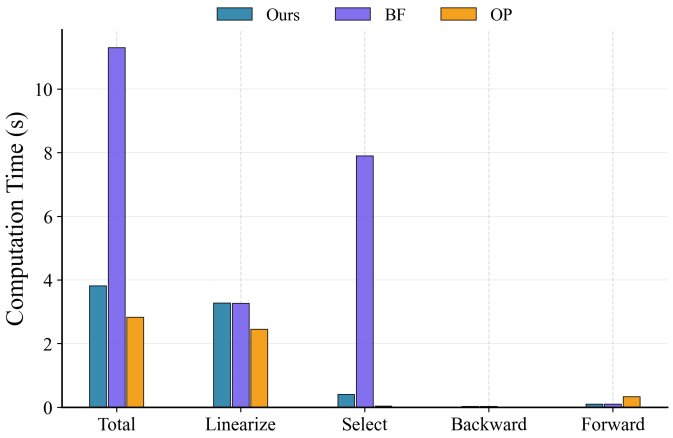


Fig. 6. Runtime breakdown for the hovering task of Quadrotor. Ours and Baseline-2 (OP) run faster than Baseline-1 (BF) as they bypass the expensive computation for horizon selection.

and the system's terminal state is within a small error threshold from the goal state, i.e., $\|x_{T^*} - x_g\| \leq 0.5$. All methods achieve high success rates on Double Integrator and Segway, which are linear systems. For Quadrotor, all methods remain above 90% success rates, suggesting that the task is feasible for a wide range of horizons and that local optimization is relatively stable once a reasonable rollout is found. Cartpole here is the most challenging case, and our method and the Baseline-1 (BF) succeed for most trials, while Baseline-2 (OP) exhibits much lower success rates, due to the mis-selection of horizon caused by the time-invariant LQR approximation of nonlinear systems. Baseline-3 (NLP) has success rates higher than OP yet lower than Ours and Baseline-1 (BF), as the underlying NLP can get trapped in local minima.

c) Solution Quality: Fig. 4(d) reports the normalized cost increase ratio relative to the solution costs found by Baseline-1 (BF). Here, Baseline-1 (BF), Baseline-2 (OP) and our methods all share the same objective function, while Baseline-3 (NLP) has slightly different objective functions due to the different formulation, and we thus remove it from the figure for clarity. As shown in Fig. 4(d), our method always finds almost the same solution costs as Baseline-1 (BF) for all systems, since the cost increases are 0%. It indicates that the fast horizon querying in our method does not compromise the final solution quality. In contrast, Baseline-2 (OP) often finds more expensive solutions for those nonlinear systems, due to the approximation using time-invariant LQR.

B. Experiment 2: Case Study on Quadrotor

We then look into a hovering task for Quadrotor. We compare the solution costs and breakdown the runtime to explain why ours is as accurate as BF and as fast as OP.

a) Horizon optimality: During the iterations of Alg. 2, both ours and Baseline-1 (BF) need to iteratively select a horizon T^* and solve. Fig. 5 shows the cost J for all possible horizons during the iteration. Baseline-2 (OP) has a different computational process and Fig. 5 only shows the costs J of horizons in its last iteration.

We observe that, our HOP-LQR and Baseline-1 (BF) have almost the same cost functions values J for the entire horizon range, and select the same optimal horizon $T^* = 32$ whose corresponding optimal costs are $J_{32}^{\text{ours}} \approx 484.79, J_{32}^{\text{BF}} \approx 484.80$. In contrast, Baseline-2 (OP) converges to a local minimum and selects a longer horizon ($T \approx 74$) with higher cost ($J_{74}^{\text{OP}} \approx 513.75$), which is about 5.97% higher than the cost returned by Baseline-1 (BF).

b) Runtime breakdown: We then look at the runtime breakdown of the algorithms. Fig. 6 explains why our algorithm runs faster. Baseline-1 (BF) spends most of the runtime in selecting the horizon, since it must evaluate many candidate horizons via repeated backward Riccati recursions. Baseline-2 (OP) runs fast because it avoids the exhaustive horizon selection. Similarly, our method can also bypass this expensive selection by using HOP-LQR for horizon selection.

VII. Conclusion and Future Work

This paper develops a new approach HOP for horizon-optimal OC problems with linear and nonlinear dynamics. The key insight is to rewrite the Riccati recursion into a LFT form which allows computational reuse. For time-varying HO-LQR problems, HOP-LQR reduces the runtime complexity from quadratic to linear with respect to the maximum possible horizon N. For nonlinear system, the proposed HOP-DDP runs fast while finding high-quality solutions. We plan to include more sophisticated constraints such as collision avoidance into our HOP in our future work.

References

[1] Brian D. O. Anderson and John B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice Hall, 1990.

[2] John T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, March 1998. doi: 10.2514/2.4231.

[3] Alberto De Marchi and Matthias Gerdts. Free finite horizon lqr: A bilevel perspective and its application to model predictive control. *Automatica*, 100:299–311, 2019.

[4] Dayi Ethan Dong, Henry Berger, and Ian Abraham. Time-optimal ergodic search: Multiscale coverage in minimum time. *The International Journal of Robotics Research*, 44(10-11):1664–1683, 2025.

[5] N. El Alami, A. Ouansafi, and N. Znaidi. On the discrete linear quadratic minimum-time problem. *Journal of the Franklin Institute*, 335(3):525–532, 1998.

[6] Philipp Foehn, Angel Romero, and Davide Scaramuzza. Time-optimal planning for quadrotor waypoint flight. *Science robotics*, 6(56):eabh1221, 2021.

[7] David Jacobson and David Mayne. *Differential Dynamic Programming*. American Elsevier Publishing Company, 1970.

[8] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016. doi: 10.1007/s10514-015-9479-3.

[9] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 222–229, 2004. doi: 10.5220/0001143902220229.

[10] Anil V. Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1): 497–528, 2009.

[11] Zhongqiang Ren, Allen George Philip, Shizhe Zhao, Sivakumar Rathinam, and Howie Choset. Cp-milp: Mixed integer linear programming for multi-agent motion planning with linear dynamics. *IEEE Robotics and Automation Letters*, 10(12):12573–12579, 2025. doi: 10.1109/LRA.2025.3623912.

[12] Kyle Stachowicz and Evangelos A. Theodorou. Optimal-horizon model-predictive control with differential dynamic programming. 2021.

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

[14] E. L. Verriest and F. L. Lewis. On the linear quadratic minimum-time problem. *IEEE Transactions on Automatic Control*, 36(7):859–863, July 1991.

[15] Andreas Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, 2002.

[16] Zhepei Wang, Xin Zhou, Chao Xu, and Fei Gao. Geometrically constrained trajectory optimization for multicopters. *IEEE Transactions on Robotics*, 38(5):3259–3278, 2022.