Journal of Machine Learning **ISSN**: 2790-2048(e), 2790-203X(p)

Progressive Optimal Path Sampling for Closed-Loop Optimal Control Design with Deep Neural Networks

Xuanxi Zhang * [1], Jihao Long † [2], Wei Hu‡ [2], Weinan E § [3,4,5], and Jiequn Han ¶ [6]

[1]Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, United States.
[2]Institute for Advanced Algorithms Research, Shanghai 201308, P.R. China.
[3]AI for Science Institute, Beijing 100080, P.R. China.
[4]School of Mathematical Science, Peking University, Beijing 100871, P.R. China.
[5]Center for Machine Learning Research, Peking University, Beijing 100871, P.R. China.
[6]Center for Computational Mathematics, Flatiron Institute, New York, NY 10010, United States.

Abstract. Closed-loop optimal control design for high-dimensional nonlinear systems has been a long-standing challenge. Traditional methods, such as solving the associated Hamilton-Jacobi-Bellman equation, suffer from the curse of dimensionality. Recent literature proposed a new promising approach based on supervised learning, by leveraging powerful open-loop optimal control solvers to generate training data and neural networks as efficient high-dimensional function approximators to fit the closed-loop optimal control. This approach successfully handles certain high-dimensional optimal control problems but still performs poorly on more challenging problems. One of the crucial reasons for the failure is the so-called distribution mismatch phenomenon brought by the controlled dynamics. In this paper, we investigate this phenomenon and propose the progressive optimal path sampling method to mitigate this problem. We theoretically prove that this enhanced sampling strategy outperforms both the vanilla approach and the widely used dataset aggregation method on the classical linear-quadratic regulator by a factor proportional to the total time duration. We further numerically demonstrate that the proposed sampling strategy significantly improves the performance on tested control problems, including the optimal landing problem of a quadrotor and the optimal reaching problem of a 7-DoF manipulator.

Keywords:
Closed-loop optimal control,
Distribution mismatch,
Adaptive sampling,
Supervised learning.

Article Info.:
Volume: 4
Number: 4
Pages: 223 - 263
Date: December/2025
doi.org/10.4208/jml.250213

Article History:
Received: 13/02/2025
Accepted: 08/08/2025

Communicated by:
Jianfeng Lu

1 Introduction

Optimal control aims to find a control for a dynamical system over a period of time such that a specified cost function is minimized. This cost often reflects a combination of task-specific goals such as energy usage, deviation from a tracking target, and control effort. Finding such an optimal control should be distinguished from classical stabilization control [21], which focuses only on keeping the system state bounded or driving it to an equi-

*xuanxizhang@nyu.edu
†longjh1998@gmail.com
‡weihu.math@gmail.com
§weinan@math.pku.edu.cn
¶Corresponding author. jiequnhan@gmail.com

librium, without regard to minimizing a cost. Generally speaking, there are two types of optimal controls: open-loop optimal control and closed-loop (feedback) optimal control. Open-loop optimal control, also known as trajectory optimization, deals with the problem with a given initial state, and its solution is a function of time for the specific initial data, independent of the other states of the system. In contrast, closed-loop optimal control aims to find the optimal control policy as a function of the state that gives us optimal control for general initial states.

By the nature of the problem, solving the open-loop control problem is relatively easy and various open-loop control solvers can handle nonlinear problems even when the state lives in high dimensions [6, 50]. Closed-loop control is much more powerful than open-loop control since it can cope with different initial states, and it is more robust to the disturbance of dynamics. The classical approach to obtaining a closed-loop optimal control function is by solving the associated Hamilton-Jacobi-Bellman (HJB) equation. However, traditional numerical algorithms for HJB equations such as the finite difference method or finite element method face the curse of dimensionality [5] and hence can not deal with high-dimensional problems.

There is a long history of employing neural networks (NNs) to solve the optimal control problems, see e.g. [1, 9, 19, 23, 28, 35–37, 43–46, 48, 54], and it is getting more attention recently since neural networks have demonstrate superior representation and generalization capabilities.

Generally speaking, there are two categories of methods in this promising direction. One is direct policy search approach [2, 9, 23, 35, 43, 59], which parameterizes the policy function by NNs, samples the total cost with various initial points, and directly minimizes the average total cost. When learning complex policies with hundreds of parameters or solving problems with a long time span and high nonlinearity, the corresponding optimization problems can be extremely hard and may get stuck in poor local minima [35, 58]. The other category of methods is based on supervised learning [37, 43–47]. Combining various techniques for open-loop control, one can solve complex high-dimensional open-loop optimal control problems, see [6, 30, 50] for surveys. Consequently, we can collect optimal trajectories for different initial points as training data, parameterize the control function (or value function) using NNs, and train the NN models to fit the closed-loop optimal controls (or optimal values). This work focuses on the second approach and aims to improve its performance through adaptive sampling.

As demonstrated in [46, 56, 58], NN controllers trained by the vanilla supervised-learning-based approach can perform poorly even when both the training error and test error on collected datasets are fairly small. Some existing works attribute this phenomenon to the fact that the learned controller may deteriorate badly at some difficult initial states even though the error is small in the average sense. Several adaptive sampling methods regarding the initial points are hence proposed (see Section 4 for a detailed discussion). However, these methods all focus on choosing optimal paths according to different initial points and ignore the effect of dynamics. This is an issue since the paths controlled by the NN will deviate from the optimal paths further and further over time due to the accumulation of errors. As shown in Section 6, applying adaptive sampling only on initial points is insufficient to solve challenging problems.

This work focuses on the so-called distribution mismatch or covariance shift phenomenon brought by the dynamics in the supervised-learning-based approach. This phenomenon refers to the fact that the discrepancy between the state distribution of the training data and the state distribution generated by the NN controller typically increases over time and the training data fails to represent the states encountered when the trained NN controller is used. Such phenomenon has also been identified in reinforcement learning [29, 39] and imitation learning [51, 52].

In this paper, we propose the progressive optimal path sampling (POPS) method to update the states in the training dataset to more closely match the states the controller reaches. While similar in spirit to the dataset aggregation (DAgger) method [52] in imitation learning, POPS addresses issues of applying DAgger to the optimal control problem considered, particularly the costs of solving open-loop control problems. Given the substantial computational demands of querying the expert policy, it is not feasible to employ the mixed policies strategy of DAgger. Instead, our approach involves progressively sampling new states by the current neural network, deliberately avoiding long-horizon rollouts that exceed the network's capability. Furthermore, unlike approaches that use model predictive control (MPC) as expert policy [28, 48], which utilize only the first value from a trajectory, our method incorporates entire trajectories into the training dataset, reflecting the strength of solving open-loop control problems to produce a full trajectory of optimal labels.

The resulting supervised-learning-based approach empowered by POPS can be interpreted as an instance of the exploration-labeling-training (ELT) algorithms [20, 57] for closed-loop optimal control problems. At a high level, the ELT algorithm proceeds iteratively with the following three steps: (1) exploring the state space and examining which states need to be labeled; (2) solving the control problem to label these states and adding them to the training data; (3) training the machine learning model. Through the lens of the ELT algorithm, there are at least three aspects to improve the efficiency of the supervised-learning-based approach for the closed-loop optimal control problem:

- Use the adaptive sampling method. Adaptive sampling methods aim to sequentially choose the time-state pairs based on previous results to improve the performance of the NN controller. This corresponds to the first step in the ELT algorithm and is the main focus of this work. We will discuss other adaptive sampling methods in Section 4.

- Improve the efficiency of data generation, i.e. solving the open-loop optimal control problems. Although the open-loop optimal control problem is much easier than the closed-loop optimal control problem, its time cost cannot be neglected and the efficiency varies significantly with different methods. This corresponds to the second step in the ELT algorithm and we refer to [30] for a detailed survey.

- Improve the learning of the machine learning model. This corresponds to the third step in the ELT algorithm. The recent works [25, 44, 46, 47] design a special network architecture such that the NN controller is close to the linear quadratic controller around the equilibrium point to improve the stability of the NN controller. Note that besides the popular use of neural networks, other models have been explored for approximating control/value functions [3, 4, 16, 17], and the adaptive sampling methods developed in this paper are also applicable to these models.

The main contributions of the paper can be summarized as follows:

- We investigate the distribution mismatch phenomenon brought by the controlled dynamics in the supervised-learning-based approach, which explains the failure of this approach for challenging problems. We propose POPS as an enhanced sampling method to update the training data, which significantly alleviates the distribution mismatch problem.
- We show that POPS can significantly improve the performance of the learned closed-loop controller on a uni-dimensional linear quadratic control problem (theoretically and numerically) and two high-dimensional problems (numerically), the quadrotor landing problem and the reaching problem of a 7-DoF manipulator.
- We compare POPS with other adaptive sampling methods and show that POPS gives the best performance.

2 Preliminary

2.1 Open-loop and closed-loop optimal control

We consider the following deterministic controlled dynamical system:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(t, \boldsymbol{x}(t), \boldsymbol{u}(t)\big), \quad t \in [t_0, T], \\ \boldsymbol{x}(t_0) = \boldsymbol{x}_0, \end{cases} \tag{2.1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ denotes the state, $\boldsymbol{u}(t) \in \mathcal{U} \subset \mathbb{R}^m$ denotes the control with $\mathcal{U}$ being the set of admissible controls, $\boldsymbol{f}\colon [0, T] \times \mathbb{R}^n \times \mathcal{U} \to \mathbb{R}^n$ is a smooth function describing the dynamics, $t_0 \in [0, T]$ denotes the initial time, and $\boldsymbol{x}_0 \in \mathbb{R}^n$ denotes the initial state. Given a fixed $t_0 \in [0, T]$ and $\boldsymbol{x}_0 \in \mathbb{R}^n$, solving the open-loop optimal control problem means to find a control path $\boldsymbol{u}^*\colon [t_0, T] \to \mathcal{U}$ to minimize

$$\begin{aligned} & J(\boldsymbol{u}; t_0, \boldsymbol{x}_0) = \int_{t_0}^{T} L\big(t, \boldsymbol{x}(t), \boldsymbol{u}(t)\big) \mathrm{d}t + M\big(\boldsymbol{x}(T)\big), \\ & \text{s.t.} \quad (\boldsymbol{x}, \boldsymbol{u}) \quad \text{satisfy the system (2.1)}, \end{aligned} \tag{2.2}$$

where $L\colon [0, T] \times \mathbb{R}^n \times \mathcal{U} \to \mathbb{R}$ and $M\colon \mathbb{R}^n \to \mathbb{R}$ are the running cost and terminal cost, respectively. We use $\boldsymbol{x}^*(t; t_0, \boldsymbol{x}_0)$ and $\boldsymbol{u}^*(t; t_0, \boldsymbol{x}_0)$ to denote the optimal state and control with the specified initial time t_0 and initial state $\boldsymbol{x}_0$, which emphasizes the dependence of the open-loop optimal solutions on the initial time and state. We assume the open-loop optimal control problem is well-posed, i.e. the solution always exists and is unique.

In contrast to the open-loop control being a function of time only, closed-loop control is a function of the time-state pair $(t, \boldsymbol{x})$. Given a closed-loop control $\boldsymbol{u}\colon [0, T] \times \mathbb{R}^n \to \mathcal{U}$, we can induce a family of the open-loop controls with all possible initial time-state pairs

$(t_0, \boldsymbol{x}_0)$: $\boldsymbol{u}(t; t_0, \boldsymbol{x}_0) = \boldsymbol{u}(t, \boldsymbol{x}_{\boldsymbol{u}}(t; t_0, \boldsymbol{x}_0))$, where $\boldsymbol{x}_{\boldsymbol{u}}(t; t_0, \boldsymbol{x}_0)$ is defined by the following initial value problem (IVP):

$$\mathrm{IVP}(\boldsymbol{x}_0, t_0, T, \boldsymbol{u}) : \begin{cases} \dot{\boldsymbol{x}}_{\boldsymbol{u}}(t; t_0, \boldsymbol{x}_0) = \boldsymbol{f}\big(t, \boldsymbol{x}_{\boldsymbol{u}}(t; t_0, \boldsymbol{x}_0), \boldsymbol{u}\big(t, \boldsymbol{x}_{\boldsymbol{u}}(t; t_0, \boldsymbol{x}_0)\big)\big), \quad t \in [t_0, T], \\ \boldsymbol{x}_{\boldsymbol{u}}(t_0; t_0, \boldsymbol{x}_0) = \boldsymbol{x}_0. \end{cases} \tag{2.3}$$

To ease the notation, we always use the same character to denote the closed-loop control function and the induced family of the open-loop controls. The context of closed-loop or open-loop control can be inferred from the arguments and will not be confusing. It is well known in the classical optimal control theory (see, e.g. [38]) that there exists a closed-loop optimal control function $\boldsymbol{u}^*\colon [0, T] \times \mathbb{R}^n \to \mathcal{U}$ such that for any $t_0 \in [0, T]$ and $\boldsymbol{x}_0 \in \mathbb{R}^n$, $\boldsymbol{u}^*(t; t_0, \boldsymbol{x}_0) = \boldsymbol{u}^*(t, \boldsymbol{x}^*(t; t_0, \boldsymbol{x}_0))$, which means the family of the open-loop optimal controls with all possible initial time-state pairs can be induced from the closed-loop optimal control function. Since IVPs can be easily solved, one can handle the open-loop control problems with all possible initial time-state pairs if a good closed-loop control solution is available. Moreover, the closed-loop control is more robust to dynamic disturbance and model misspecification, and hence it is much more powerful in applications. In this paper, our goal is to find a near-optimal closed-loop control $\hat{\boldsymbol{u}}$ such that for $\boldsymbol{x}_0 \in X \subset \mathbb{R}^n$ with X being the set of initial states of interest, the associated total cost is near-optimal, i.e. $|J(\hat{\boldsymbol{u}}(\,\cdot\,; 0, \boldsymbol{x}_0); 0, \boldsymbol{x}_0) - J(\boldsymbol{u}^*(\,\cdot\,; 0, \boldsymbol{x}_0); 0, \boldsymbol{x}_0)|$ is small.

2.2 Supervised-learning-based approach for closed-loop optimal control problem

Here we briefly explain the idea of the supervised-learning-based approach for the closed-loop optimal control problem. The first step is to generate training data by solving the open-loop optimal control problems with zero initial time and initial states randomly sampled in X. Then, the training data is collected by evenly choosing points in every optimal path

$$\mathcal{D} = \{(t^{i,j}, \boldsymbol{x}^{i,j}), \boldsymbol{u}^{i,j}\}_{1 \le i \le M, 1 \le j \le N},$$

where M and N are the number of sampled training trajectories and the number of points chosen in each path, respectively. Finally, a function approximator (mostly neural network, as considered in this work) with parameters θ is trained by solving the following regression problem:

$$\min_\theta \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} \|\boldsymbol{u}^{i,j} - \hat{\boldsymbol{u}}(t^{i,j}, \boldsymbol{x}^{i,j}; \theta)\|^2, \tag{2.4}$$

and gives the neural network (NN) controller $\hat{\boldsymbol{u}}$.

We emphasize that the basic supervised learning approach relies on two key assumptions. First, the open-loop optimal solutions must be unique, or at least the training data should contain only unimodal solutions. If multimodal open-loop solutions (e.g. a swing-up task that can proceed either left or right) are present in the training data, the function approximator will struggle to learn the closed-loop optimal control. Second, the function approximator must be capable of representing the closed-loop optimal control. According to the universal approximation theorem, neural networks can generally represent smooth control. However, standard neural networks may fail to represent discontinuous control, such as bang-bang control, which is beyond the scope of this work.

3 Proposed method: Progressive optimal path sampling

Although the vanilla supervised-learning-based approach can achieve a good performance in certain problems [45], it is observed that its performance on complex problems is not satisfactory (see [46, 56] and examples below). One of the crucial reasons that the vanilla method fails is the distribution mismatch phenomenon. To better illustrate this phenomenon, let μ_0 be the distribution of the initial state of interest and $\boldsymbol{u} : [0, T] \times \mathbb{R}^n \to \mathcal{U}$ be a closed-loop control function. We use $\mu_{\boldsymbol{u}}(t)$ to denote the distribution of $\boldsymbol{x}(t)$ generated by $\boldsymbol{u}$: $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t, \boldsymbol{x}(t))), \boldsymbol{x}_0 \sim \mu_0$. Note that in the training process (2.4), the distribution of the state at time t is $\mu_{\boldsymbol{u}^*}(t)$, the state distribution generated by the closed-loop optimal control. On the other hand, when we apply the learned NN controller in the dynamics, the distribution of the input state of $\hat{\boldsymbol{u}}$ at time t is $\mu_{\hat{\boldsymbol{u}}}(t)$. The error between state $\boldsymbol{x}$ driven by $\boldsymbol{u}^*$ and $\hat{\boldsymbol{u}}$ accumulates and makes the discrepancy between $\mu_{\boldsymbol{u}^*}(t)$ and $\mu_{\hat{\boldsymbol{u}}}(t)$ increases over time. Hence, the training data fails to represent the states encountered in the controlled process, and the error between $\boldsymbol{u}^*$ and $\hat{\boldsymbol{u}}$ dramatically increases when t is large. See Figs. 3.1(left) and 6.1 below for an illustration of this phenomenon.

To overcome this problem, we propose the following progressive optimal path sampling method. The key idea is to improve the quality of the NN controller iteratively by enlarging the training dataset with the states seen by the NN controller at previous times. Given predesigned (not necessarily even-spaced) temporal grid points

$$0 = t_0 < t_1 < \cdots < t_K = T,$$

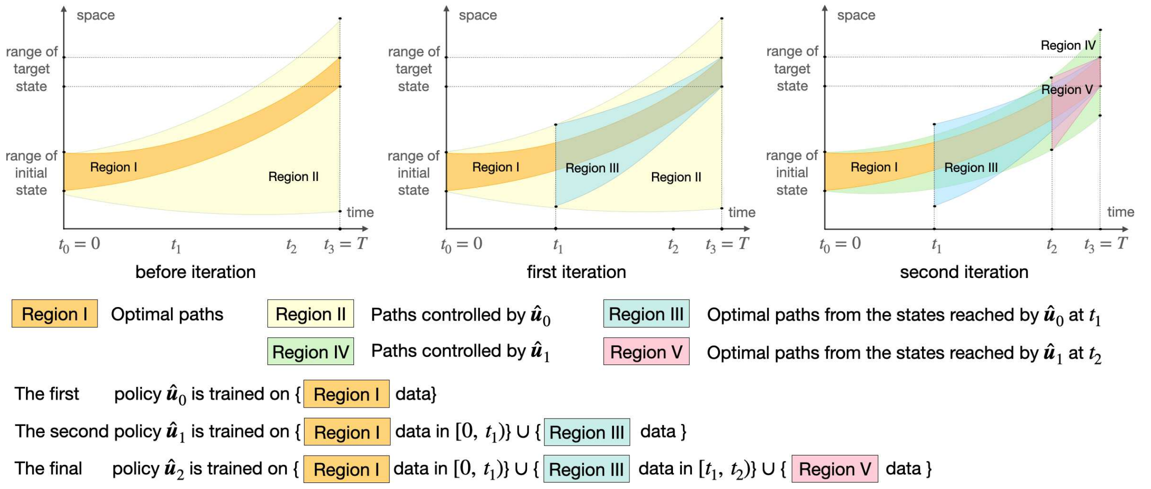
space
range of target state
range of initial state
Region I
Region II
time
$t_0 = 0$
t_1
t_2
$t_3 = T$
before iteration
space
range of target state
range of initial state
Region I
Region III
Region II
time
$t_0 = 0$
t_1
t_2
$t_3 = T$
first iteration
space
range of target state
range of initial state
Region IV
Region V
Region I
Region III
time
$t_0 = 0$
t_1
t_2
$t_3 = T$
second iteration
Region I Optimal paths
Region II Paths controlled by $\hat{\boldsymbol{u}}_0$
Region III Optimal paths from the states reached by $\hat{\boldsymbol{u}}_0$ at t_1
Region IV Paths controlled by $\hat{\boldsymbol{u}}_1$
Region V Optimal paths from the states reached by $\hat{\boldsymbol{u}}_1$ at t_2
The first policy $\hat{\boldsymbol{u}}_0$ is trained on { Region I data}
The second policy $\hat{\boldsymbol{u}}_1$ is trained on { Region I data in $[0, t_1)$} ∪ { Region III data }
The final policy $\hat{\boldsymbol{u}}_2$ is trained on { Region I data in $[0, t_1)$} ∪ { Region III data in $[t_1, t_2)$} ∪ { Region V data }

Figure 3.1: An illustration of POPS (Algorithm 1) when there are two intermediate temporal grid points t_1 and t_2.

we first generate a training dataset S_0 by solving open-loop optimal control problems on the time interval $[0, T]$ starting from points in X_0, a set of initial points sampled from μ_0, and train the initial model $\hat{\boldsymbol{u}}_0$. Under the control of $\hat{\boldsymbol{u}}_0$, the generated trajectory deviates more and more from the optimal trajectory. So we stop at time t_1, i.e. compute the IVPs (2.3) using $\hat{\boldsymbol{u}}_0$ as the closed-loop control and points in X_0 as the initial points on the time interval $[0, t_1]$, and then on the interval $[t_1, T]$ solve new optimal paths that start at the end-points of the previous IVPs. The new training dataset S_1 is then composed of new data (between t_1 and T) and the data before time t_1 in the dataset S_0, and we train a new model $\hat{\boldsymbol{u}}_1$ using S_1. We repeat this process to the predesigned temporal grid points $t_2, t_3, \dots$ until end up with T. In other words, in each iteration, the adaptively sampled data replaces the corresponding data (defined on the same time interval) in the training dataset (the size of the training data remains the same). The whole process can be formulated as Algorithm 1, and we refer to Fig. 3.1 for an illustration of the algorithm's mechanism. We refer to this method as progressive optimal path sampling because it incrementally generates new open-loop optimal paths as training data using the most recent NN controller. While this approach shares similarities with a few existing methods, as discussed in the next section, our strategy for progressively adding data is more appropriate for optimal control compared to the aggressive selection used in methods such as DAgger.

It is worthwhile mentioning that POPS is versatile enough to combine other improvements for closed-loop optimal control problems, such as efficient open-loop control problem solvers [30,56] or specialized neural network structures [44,46,47]. One design choice regarding the network structure in POPS is whether to share the same network among different time intervals. We choose to use the same network for all the time intervals in the following numerical examples, but the opposite choice is also feasible.

Algorithm 1 Progressive Optimal Path Sampling for Closed-Loop Optimal Control Design.

```
Input: Initial distribution $\mu_0$, number of time points $K$, temporal grid points
    $0 = t_0 < t_1 < \cdots < t_K = T$, time step $\delta$, number of initial points $N$.
Initialize: $S_{-1} = \emptyset$, $\hat{\boldsymbol{u}}_{-1}(t, \boldsymbol{x}) = 0$.
Independently sample $N$ initial points from $\mu_0$ to get an initial point set $X_0$.
for $i = 0, 1, \dots, K-1$ do
    For any $\boldsymbol{x}_0 \in X_0$, compute IVP $(\boldsymbol{x}_0, 0, t_i, \hat{\boldsymbol{u}}_{i-1})$ according to (2.3).    ▷ Exploration
    Set $X_i = \{\boldsymbol{x}_{\hat{\boldsymbol{u}}_{i-1}}(t_i; 0, x_0) : x_0 \in X_0\}$.
    For any $\boldsymbol{x}_i \in X_i$, call the open-loop optimal control solver to obtain
        $\boldsymbol{x}^*(t; t_i, \boldsymbol{x}_i)$ and $\boldsymbol{u}^*(t; t_i, \boldsymbol{x}_i)$ for $t \in [t_i, T]$.    ▷ Labeling
    Set $\hat{S}_i = \{(t, \boldsymbol{x}^*(t; t_i, \boldsymbol{x}_i), \boldsymbol{u}^*(t; t_i, \boldsymbol{x}_i)) : \boldsymbol{x}_i \in X_i, t \in [t_i, T], (t - t_0)/\delta \in \mathbb{N}\}$.
    Set $S_i = \hat{S}_i \bigcup \{(t, \boldsymbol{x}, \boldsymbol{u}) : t < t_i, (t, \boldsymbol{x}, \boldsymbol{u}) \in S_{i-1}\}$.
    Train $\hat{\boldsymbol{u}}_i$ with dataset $S_i$.    ▷ Training
end for
Output: $\hat{\boldsymbol{u}}_{K-1}$.
```

4 Related work

Adaptive sampling for supervised-learning-based approach. Learning the optimal policy can be framed as an imitation learning problem by treating the optimal policy as the expert's control. This leads to a similar distribution mismatch issue in both settings. However, there is a key difference regarding the mechanism of data generation between the two settings: in imitation learning, it is often assumed that one can easily access the expert's behavior at every time-state pair, e.g. through solving a sub-problem by MPC [28, 48], while in the optimal control problem, it is much more computationally expensive to access since one must solve an open-loop optimal control problem. This difference affects algorithm design fundamentally. The methods for imitation learning often assume low computational costs of obtaining the expert demonstration at each time-state pair. Take the forward training algorithm [51] as an example. To apply it to the closed-loop optimal control problem, we first need to consider a discrete-time version of the problem with a sufficiently fine time grid

$$0 = t_0 < t_1 < \cdots < t_{K'} = T.$$

At each time step t_i, we learn a policy function $\bar{\boldsymbol{u}}^i : \mathbb{R}^n \to \mathcal{U}$, where the state x in the training data are generated by sequentially applying $\bar{\boldsymbol{u}}^0, \ldots, \bar{\boldsymbol{u}}^{i-1}$ and the labels are generated by solving the open-loop optimal solutions with (t_i, x) as the initial time-state pair. Hence, the open-loop control solver is called with numbers proportionally to the discretized time steps K', and only the first value on each optimal control path is used for learning. In contrast, in Algorithm 1, we can use much more values over the optimal control paths in learning, which allows a much coarse temporal grid for adaptive sampling, and significantly reduces the total cost of solving open-loop optimal control problems.

Another popular imitation learning method is DAgger [52], which can be adapted for sampling in closed-loop optimal control. Similar to the forward training algorithm, DAgger assumes access to expert behavior at each time-state pair. Here, we retain the core idea of DAgger but modify it for closed-loop optimal control. In DAgger, in order to improve the current closed-loop controller $\hat{\boldsymbol{u}}$, one simulates the system using $\hat{\boldsymbol{u}}$ over $[0, T]$ starting from various initial states and collect the states on a time grid

$$0 = t_0 < t_1 < \cdots < t_{K-1} < T.$$

The open-loop control problems are then solved with all the collected time-state pairs as the initial time-state pair, and all the corresponding optimal solutions are used to construct a dataset for learning a new controller. The process can be repeated until a good controller is obtained. While DAgger aims to address distribution mismatch, its state selection differs from POPS. Take the data collection using the controller $\hat{\boldsymbol{u}}_1$ in the first iterative step for example. POPS focuses on the states at the time grid t_1 while DAgger collects states at all the time grids. If $\hat{\boldsymbol{u}}_1$ is still far from optimal, the data collected at later time grids may be irrelevant or even misleading due to error accumulation in states. As shown in subsequent sections, this sensitivity can reduce DAgger's effectiveness compared to POPS.

There are other adaptive sampling methods specifically developed for closed-loop optimal control rather than imitation learning. [36,37] propose iterative methods that generate sub-optimal open-loop solutions around the current policy but are tailored to stochastic policies, limiting their applicability to deterministic control problems considered in this paper. [45] propose an adaptive sampling method that chooses the initial points with large gradients of the value function as the value function tends to be steep and hard to learn around these points. [34] propose to sample the initial points on which the errors between predicted values from the NN and optimal values are large. These two adaptive sampling methods both focus on finding points that are not learned well but ignore the influence of the accumulation of the distribution mismatch over time brought by controlled dynamics.

Open-loop data for direct policy search approach. Open-loop data can also alleviate optimization challenges in direct policy search. For example, guided policy search [35] incorporates trajectory samples from differentiable dynamic programming into the optimization objective. Similarly, [43] reformulate the optimization as a constrained problem to prevent the learned policy from deviating too far from open-loop optimal trajectories.

5 Theoretical analysis on an LQR example

In this section, we analyze the superiority of POPS by considering the following unidimensional linear quadratic regulator (LQR) problem:

$$\begin{aligned}&\min_{x(t),u(t)} \frac{1}{T}\int_{t_0}^{T} |u(t)|^2 \mathrm{d}t + |x(T)|^2,\\ &\text{s.t.} \quad \dot{x}(t) = u(t), \quad t \in [t_0, T], \quad x(t_0) = x_0,\end{aligned}$$

where T is a positive integer, $t_0 \in [0, T]$ and $x_0 \in \mathbb{R}$. Classical theory on linear quadratic control (see, e.g. [53]) gives the following explicit linear form of the optimal controls:

$$\begin{cases} u^*(t;t_0,x_0) = -\dfrac{T}{T(T-t_0)+1}x_0, & \text{(open-loop optimal control)}\\ u^*(t,x) = -\dfrac{T}{T(T-t)+1}x. & \text{(closed-loop optimal control)}\end{cases}$$

We consider the following two models to approximate the closed-loop optimal control function with parameter θ:

$$\text{Model 1:} \quad u_\theta(t,x) = -\frac{T}{T(T-t)+1}x + b(t), \quad \theta = \{\theta_t\}_{0\le t\le T} = \{b(t)\}_{0\le t\le T}. \tag{5.1}$$

$$\text{Model 2:} \quad u_\theta(t,x) = a(t)x + b(t), \quad \theta = \{\theta_t\}_{0\le t\le T} = \big\{\big(a(t), b(t)\big)\big\}_{0\le t\le T}. \tag{5.2}$$

Since learning a linear model has no error with exact data, to mimic the errors encountered when learning neural networks, throughout this section, we assume the data has certain noise. To be precise, for any $t_0 \in [0, T]$ and $x_0 \in \mathbb{R}$, the open-loop optimal control solver gives the following approximated optimal path:

$$\hat{u}(t;t_0,x_0) = -\frac{T}{T(T-t_0)+1}x_0 + \epsilon Z,$$

$$\hat{x}(t;t_0,x_0) = x_0 + \int_{t_0}^{t} \hat{u}(t;t_0,x_0)\mathrm{d}t = \frac{T(T-t)+1}{T(T-t_0)+1}x_0 + (t-t_0)\epsilon Z,$$

where $\epsilon > 0$ is a small positive number to indicate the scale of the error and Z is a normal random variable whose mean is m and variance is σ^2. In other words, the obtained open-loop control is still constant in each path, just like the optimal open-loop control, but perturbed by a random constant. The random variables in different approximated optimal paths starting from different t_0 or x_0 are assumed to be independent.

Comparison results. We provide a theoretical comparison under Model 1 (5.1) and a numerical evaluation under Model 2 (5.2) for the vanilla supervised-learning-based method, DAgger, and POPS. Detailed descriptions of the setups of each method are given in Appendix A. Notably, we maintain the same total number of open-loop optimal paths, NT, across all methods, where N is an integer.

We denote four closed-loop controllers as u_o, u_v, u_d, and u_p, corresponding to the optimal controller, the controller learned by the vanilla method, the controller learned by DAgger, and the controller learned by POPS, respectively. Additionally, we define $\hat{x}^v(t)$, $\hat{x}^d(t)$, and $\hat{x}^p(t)$ as random variables whose distributions are the average distributions of state variables in the training data at time t for the vanilla method, DAgger, and POPS (in the final iteration), respectively. We define two key metrics for evaluation: (1) distribution difference, which measures the discrepancy between the variance of the training trajectory data used for the final controller and the variance of trajectory data generated by applying the final controller, and (2) performance difference, which is the difference between the total cost incurred by the final learned controller and the optimal cost. Theorem 5.1 establishes that, for both the vanilla method and DAgger, the distribution difference and performance difference increase quadratically with T, while in the case of POPS, these metrics remain independent of T. Therefore, compared to the vanilla method and DAgger, POPS mitigates the distribution mismatch phenomenon and significantly improves the performance for large T. The detailed proof is provided in Appendix A.

Theorem 5.1. *Under Model 1 (5.1), define dynamical systems:* $\dot{x}_s(t) = u_s(t) = u_s(t, x_s(t))$, $x_s(0) = x_{\text{init}}, 0 \le t \le T, s \in \{o, v, d, p\}$.

1. ***Distribution difference.*** *Assume* x_{init} *is a random variable following a standard normal distribution, which is independent of the initial points and noises in the training process. Then,* $\mathbb{E}\hat{x}^v(t) = \mathbb{E}x_v(t), \mathbb{E}\hat{x}^p(t) = \mathbb{E}x_p(t)$, *and*

$$\left|\mathrm{Var}\big(\hat{x}^v(t)\big) - \mathrm{Var}\big(x_v(t)\big)\right| = \epsilon^2\sigma^2\left(1 - \frac{1}{NT}\right)t^2,$$

$$\mathrm{Var}\big(\hat{x}^d(t)\big) - \mathrm{Var}\big(x_d(t)\big) \ge \epsilon^2\sigma^2\left(\frac{t^2}{3} - \frac{2t}{N}\right),$$

$$\left|\mathrm{Var}\big(\hat{x}^p(t)\big) - \mathrm{Var}\big(x_p(t)\big)\right| \le \epsilon^2\sigma^2.$$

2. ***Performance difference.*** *Assume* x_{init} *is fixed and define the total cost*

$$J_s = \frac{1}{T}\int_0^T |u_s(t)|^2\mathrm{d}t + |x_s(T)|^2, \quad s \in \{o, v, d, p\}.$$

Then,

$$\begin{aligned}
\mathbb{E}J_v - J_o &= (T^2+1)\left(m^2 + \frac{\sigma^2}{NT}\right)\epsilon^2, \\
\mathbb{E}J_d - J_o &\geq \left(\frac{T^2m^2}{4} + \frac{T\sigma^2}{3N}\right)\epsilon^2, \\
\mathbb{E}J_p - J_o &\leq 3\left(m^2 + \frac{\sigma^2}{N}\right)\epsilon^2.
\end{aligned}$$

Next, we present the numerical results when we use Model 2 (5.2) to fit the closed-loop optimal control, with the setting $\epsilon = 0.1$, $m = 0.1$ and $\sigma^2 = 1$. Fig. 5.1(top left) compares the performance of the vanilla method, DAgger, and POPS on different total times T. The performance difference is an empirical estimation of $\mathbb{E}[J_v - J_o]$, $\mathbb{E}[J_d - J_o]$ and $\mathbb{E}[J_p - J_o]$ when x_{init} follows a standard normal distribution. In this experiment, for each method, we set $N = 100$ and learn 10 different controllers with different realizations of the training data and calculate the average of the performance difference on 1000 randomly sampled initial points (from a standard normal distribution) and 10 learned controllers. Fig. 5.1(top right) shows how the time t influences the distribution differences

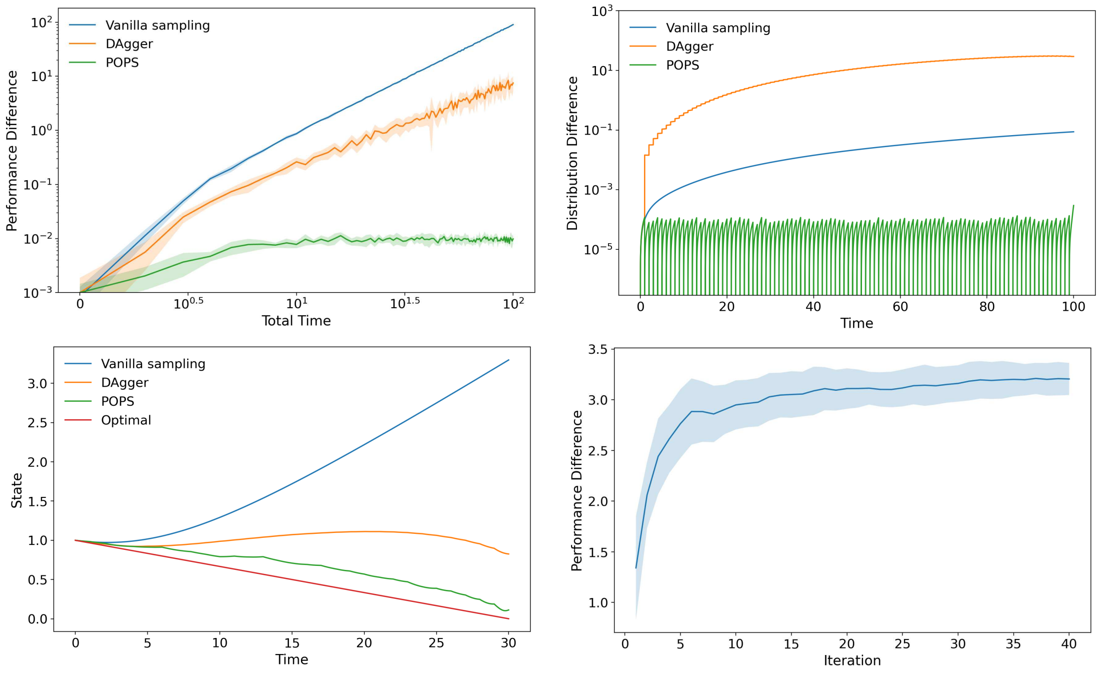

Figure 5.1: Numerical results on learning Model 2 (5.2) for the LQR model. Top left: performance differences (in the logarithm scale) of the vanilla sampling method, DAgger, and POPS for different total time span T (in the logarithm scale). Top right: differences of the second-order moments (in the logarithm scale) between the distributions of the training data and the data reached by the controllers at intermediate times. Bottom left: the optimal path and the paths generated by the vanilla sampling method, DAgger, and POPS. Bottom right: the performance difference of DAgger along multiple iterations.

when T=100 and N=100. Fig. 5.1(bottom left) compares the optimal path $x_o(t)$ with $x_v(t), x_d(t)$ and $x_p(t)$, the trajectory generated by the controllers learned by the vanilla method, DAgger, and POPS starting from $x_{\text{init}} = 1$, when $T = 30$ and $N = 100$. All three experiments consistently demonstrate the superior performance of POPS as similarly established in Theorem 5.1. Finally, Fig. 5.1(bottom right) shows the performance of DAgger with multiple iterations under $T = 30$. The results indicate that increasing the number of iterations does not improve the controller's performance in DAgger.

We conclude this section by noting that, although the analysis here is specific to linear-quadratic settings, it illustrates how POPS effectively controls the growth of distribution mismatch with respect to the time horizon. Unlike single-step imitation approaches such as DAgger, POPS leverages entire open-loop trajectories, which amortizes the cost of data generation from optimal control solvers and helps mitigate long-horizon distribution shift. This is supported both by the theoretical results in this section and by the nonlinear examples presented in the next two sections.

6 The optimal landing problem of quadrotor

In this section, we evaluate the performance of POPS on a quadrotor optimal-landing task: bringing the vehicle to a prescribed touchdown point while minimizing energy consumption. We consider the full quadrotor dynamic model with 12-dimensional state variable and 4-dimensional control variable [10,40,41]. The state variable is represented as

$$\boldsymbol{x} = \left(\boldsymbol{p}^\top, \boldsymbol{v}_b^\top, \boldsymbol{\eta}^\top, \boldsymbol{w}_b^\top\right)^\top \in \mathbb{R}^{12},$$

where $\boldsymbol{p} = (x, y, z) \in \mathbb{R}^3$ denotes the quadrotor's position in Earth-fixed coordinates, $\boldsymbol{v}_b \in \mathbb{R}^3$ is the velocity in body-fixed coordinates, $\boldsymbol{\eta} = (\phi, \theta, \psi) \in \mathbb{R}^3$ represents the attitude (roll, pitch, yaw) in Earth-fixed coordinates, and $\boldsymbol{w}_b \in \mathbb{R}^3$ is the angular velocity in body-fixed coordinates. The control variable is defined as $\boldsymbol{u} = (s, \tau_x, \tau_y, \tau_z)^\top \in \mathbb{R}^4$, where s is the total thrust, and τ_x, τ_y, τ_z are the body torques generated by the four rotors. The details of the quadrotor dynamics are provided in Appendix B.

Our goal is to compute optimal landing trajectories from an initial state $\boldsymbol{x}_0$ to a target state $\boldsymbol{x}_T = 0$ with minimum control efforts over a fixed time span $T = 16$. The initial state distribution is uniform over the set

$$\begin{aligned} X = \{&x, y \in [-40, 40],\ z \in [20, 40],\ v_x, v_y, v_z \in [-1, 1], \\ &\theta, \phi \in [-\pi/4, \pi/4],\ \psi \in [-\pi, \pi];\ \boldsymbol{w}_b = \boldsymbol{0}\}. \end{aligned} \tag{6.1}$$

The running cost and terminal cost are defined as follows:

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{u}) &= (\boldsymbol{u} - \boldsymbol{u}_d)^\top Q_u (\boldsymbol{u} - \boldsymbol{u}_d), \\ M(\boldsymbol{x}) &= \boldsymbol{p}^\top Q_{pf} \boldsymbol{p} + \boldsymbol{v}^\top Q_{vf} \boldsymbol{v} + \boldsymbol{\eta}^\top Q_{\eta f} \boldsymbol{\eta} + \boldsymbol{w}^\top Q_{wf} \boldsymbol{w} = \boldsymbol{x}^\top Q_f \boldsymbol{x}, \end{aligned}$$

where $\boldsymbol{u}_d$=$(mg, 0, 0, 0)$ is the reference control to counteract gravity, and Q_u=diag$(1, 1, 1, 1)$ is the weight matrix that penalizes deviations from the reference control. The weight matrices for the terminal cost are $Q_{pf} = 5I_3$, $Q_{vf} = 10I_3$, $Q_{\eta f} = 25I_3$, and $Q_{wf} = 50I_3$. Larger weights are used in the terminal cost to impose stricter penalties on deviations from the landing target. The open-loop optimal solutions are derived by solving the associated two-point boundary value problem using the space-marching technique [56]; additional details are provided in Appendix C. When applying POPS, the number of initial points for these open-loop optimal solutions is fixed at $N = 500$ for each iteration. To construct the training dataset, we sample time-state-action tuples at intervals of $\delta = 0.2$ time steps along the optimal trajectories, resulting in a consistent dataset size of 81×500 at every iteration.

The neural network models used in all quadrotor experiments share the same structure: a 13-dimensional input (12 for the state variables and 1 for time) and a 4-dimensional output. Each network is fully connected, with two hidden layers comprising 128 neurons per layer. We use the `tanh` activation function in both layers. The inputs are scaled to the range $[-1, 1]$, where the upper and lower bounds correspond to the maximum and minimum values of the training dataset. Since the activation function is `tanh`, we apply Xavier initialization [22] prior to training. The neural networks are trained using the Adam optimizer [32] with a learning rate of 0.001, a batch size of 1000, and 1000 epochs. At each iteration of POPS, a new neural network is trained from scratch. The quadrotor experiments were conducted on a MacBook Pro equipped with an Apple M1 Pro chip.

In the first experiment, we apply POPS and select the temporal grid points $t = 0, 10, 14, 16$. After training, we use the learned NN controllers to solve the initial value problem for the 500 training initial points, comparing the trajectories driven by the NN controllers to their corresponding training data. In Fig. 6.1, the left subfigure displays the average distance between the states reached by the NN controller and the states from the training data at various time steps. The right subfigure shows the maximum mean discrepancy (MMD) [8] between these two datasets, using a Gaussian kernel

$$k(x, y) = \exp\left(-\frac{|x-y|^2}{2}\right).$$

Both figures show noticeable jumps at $t = 10$ and 14 because the NN-controlled trajectory is continuous across time, whereas the training data is discontinuous at the points where POPS is applied. Notably, without adaptive sampling (represented by the curve labeled "policy after iteration 0" in Fig. 6.1), there is a substantial discrepancy between the

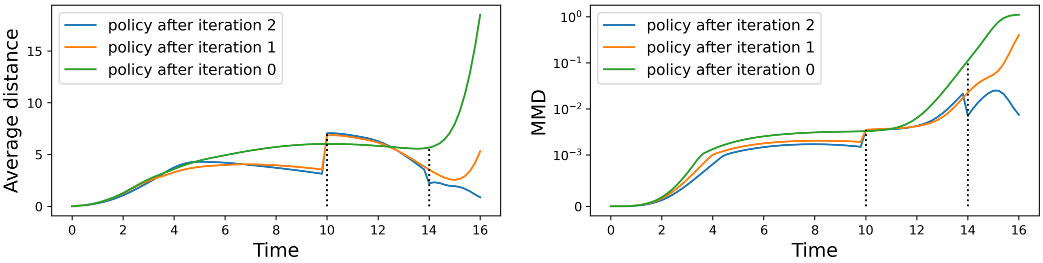

Figure 6.1: Left: the average pointwise distance between the training data and the data reached by controllers at different times. Right: the maximum mean discrepancy (in the logarithm scale) between the training data and the data reached by controllers at every time using the Gaussian kernel.

NN-controlled states and the training data while this discrepancy progressively decreases with each iteration of POPS.

Next, we report the performance of the learned NN controllers. Example trajectories driven by NN controllers are illustrated in Fig. 6.2. As the POPS iterations progress, the trajectories increasingly align with the optimal path. In the final iteration, the trajectory controlled by $\hat{\boldsymbol{u}}_3$ closely matches the entire optimal trajectory. Quantitatively, the costs of the three controlled trajectories are $3296.2, 119.9$, and 6.7, respectively, compared to the optimal cost of 6.3.

To evaluate the controller's performance, we use the cost ratio, defined as the total cost (2.2) under the given controller divided by the optimal cost. By definition, the cost ratio is always greater than or equal to 1. We perform experiments five times with different random seeds and present the results in Fig. 6.3 and Table 6.1. The performance is

Table 6.1: The columns "Mean" "90%", and "Median" represent the cost ratio statistics across 200 test initial points. The first 3 rows for $\hat{\boldsymbol{u}}_0, \hat{\boldsymbol{u}}_1, \hat{\boldsymbol{u}}_2$ denote the policy after the first, second, and third round of training, respectively and the entries are averages and standard deviations of 5 experiments. The bottom 3 rows report the results from the ensemble of five networks trained independently.

Policy	Mean	90%	Median
$\hat{\boldsymbol{u}}_0$	16.43 (± 6.16)	17.60 (± 4.80)	16.54 (± 6.92)
$\hat{\boldsymbol{u}}_1$	3.69 (± 1.62)	9.11 (± 5.54)	2.06 (± 0.72)
$\hat{\boldsymbol{u}}_2$	1.17 (± 0.09)	1.22 (± 0.09)	1.06 (± 0.02)
$\hat{\boldsymbol{u}}_0$ **ensemble**	20.37	46.85	15.05
$\hat{\boldsymbol{u}}_1$ **ensemble**	1.78	2.54	1.29
$\hat{\boldsymbol{u}}_2$ **ensemble**	1.03	1.04	1.01

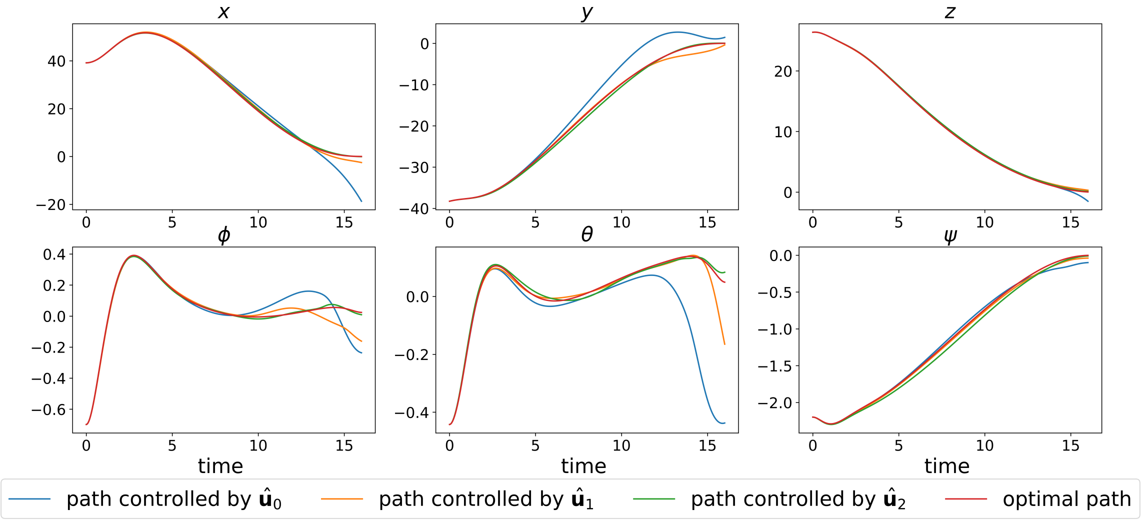

Figure 6.2: The optimal path and path controlled by learned controllers. We show the 3-dimensional position $\boldsymbol{p} = (x, y, z)$ and 3-dimensional attitude $\boldsymbol{\eta} = (\phi, \theta, \psi)$ in terms of Euler angles in Earth-fixed coordinates. The cost of $\hat{u}_0, \hat{u}_1, \hat{u}_2$ controlled paths is $3296.2, 119.9, 6.7$, respectively, and the optimal cost is 6.3.

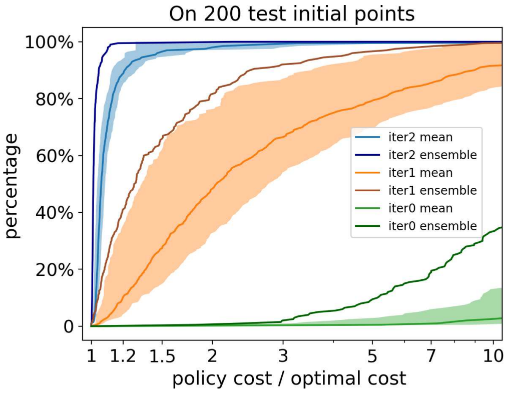

Figure 6.3: Cumulative distribution function of the cost ratio of POPS among 200 test trajectories. The shaded area represents the mean cost ratio ± the standard deviation.

evaluated on 200 test initial points which are also uniformly drawn from the initial set X defined in (6.1). Fig. 6.3 shows the cumulative distribution function (CDF) of the cost ratio under various NN controllers on test initial points. The CDF of the optimal controller's cost ratio is a horizontal line at a ratio of 1 and a percentage of 100%. Therefore, a curve closer to the top-left corner indicates better performance of the corresponding controller. It is important to note that the randomness in the first iteration arises from the initialization and batch data sampling during the training of the neural networks. In subsequent iterations, additional randomness is introduced through both the generation of distinct training data by different NN controllers and the stochastic nature of their training processes. The results indicate that the POPS algorithm improves performance with each iteration, reducing the cost from the initial model $\hat{\boldsymbol{u}}_0$, which has an average cost ratio of 16.43, to $\hat{\boldsymbol{u}}_2$, which has an average cost ratio of 1.17. Additionally, we evaluate ensembles that average the output of five independently trained networks (indexed by i) as a closed-loop controller

$$\hat{\boldsymbol{u}}^{\textbf{ensemble}}(t,x) = \frac{1}{5}\sum_{i=1}^{5}\hat{\boldsymbol{u}}_i(t,x),$$

represented by the dark curves. This ensemble demonstrates superior performance over individual networks, achieving an average cost ratio of 1.03.

We further assess the performance of NN controllers under observation noise, simulating real-world sensor errors. During simulation, we add a disturbance ϵ (including time) uniformly sampled from $[-\sigma,\sigma]^{13}$ to the network input. We test with $\sigma = 0.01, 0.05, 0.1$, and report the results in Fig. 6.4. For comparison, we also evaluate the performance of the open-loop optimal controller with a disturbance $\hat{\epsilon} \in \mathbb{R}$ added to the input time. Fig. 6.4 demonstrates that closed-loop controllers are more robust than open-loop controllers, and the controller trained with POPS continues to perform well under these noise levels.

We also test four different temporal grid point settings in Algorithm 1 to assess their impact. The results in Table 6.2 demonstrate the algorithm's robustness, with consistent performance across different choices of temporal grid points.

Table 6.2: Average cost ratio on 200 test points of POPS applied to different temporal grid points. The outcomes were consistent after the initial iteration (16.43 ($\pm$6.16)), and thus, we have excluded them from the table. This uniformity arises from utilizing the same five random seeds for repeating the experiments. The first line indicates that we performed 2 iterations, and the corresponding temporal grid points for adaptive sampling are $0 < 14 < 16$. After iteration 0, the average ratio of policy cost over optimal cost is 16.43 and after iteration 1, it decreases to 1.36. The second line shows the same experiment in Fig. 6.3.

# of iterations	Temporal grid points ($T = 16$)				
	4	8	10	12	14
2					1.36 ($\pm$ 0.08)
3			3.69 ($\pm$ 1.62)		1.17 ($\pm$ 0.09)
4		10.86 ($\pm$ 2.27)		1.72 ($\pm$ 0.49)	1.17 ($\pm$ 0.07)
6	14.39 ($\pm$ 1.85)	12.14 ($\pm$ 5.64)	6.30 ($\pm$ 2.67)	1.73 ($\pm$ 0.46)	1.22 ($\pm$ 0.04)

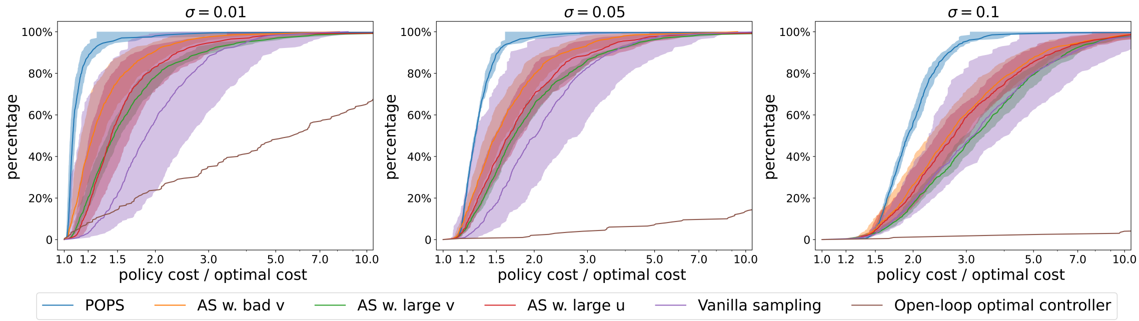

Figure 6.4: Cumulative distribution function of the cost ratio between NN controlled value under disturbance and the optimal value among 200 test trajectories.

6.1 Comparison results

In this subsection, we compared POPS with other methods in the literature. We first compare to vanilla method and three methods focusing on adaptive sampling according to the initial points:

- Vanilla sampling: Training a model on directly sampled 1500 optimal paths.
- Adaptive sampling with large control norms (AS w. large u): A method proposed by [45], selecting initial points with large gradient norms, equivalent to selecting initial points with large optimal control norms.
- Adaptive sampling with large total costs (AS w. large v): Selecting initial points whose total costs are large under the latest NN controller.
- Adaptive sampling with large value gaps (AS w. bad v): A variant of the SEAGuL algorithm (sample efficient adversarially guided learning [34]), selecting initial points with large gaps between the learned values and optimal values.

For the three adaptive sampling methods, we sample two points and select one based on the relevant criterion to add an initial point. Each method is run five times with different random seeds. All methods start with the same initial network as POPS (the policy $\hat{\boldsymbol{u}}_0$ trained on 500 paths) and progressively add 400, 300, and 300 paths, resulting in a final network trained on 1500 paths.

Data generation is the most time-intensive part of the algorithm, and all methods being compared solve 1500 open-loop problems, except for AS w. bad v, which solves 2500. In POPS, the time span of optimal paths shortens with each iteration, reducing computation time compared to other methods, where the time span remains fixed at T. The cumulative distribution functions of cost ratios for these methods, shown in Fig. 6.5, demonstrate the superior performance of POPS. Additional statistics are provided in Table 6.3.

We also compare POPS with DAgger, which augments the training dataset by sampling intermediate states rather than initial points. In DAgger, we use the same temporal grid points at $t = 10$ and 14 as in POPS. The results, summarized in Table 6.4, show that DAgger performs similarly to POPS when using 500 initial trajectories. However, when the number of trajectories is reduced from 500 to 300, DAgger shows a more significant performance drop. The main reason is that DAgger demands enough data at the beginning to have a good initial controller capable of exploring the state space over the entire time interval. However, in complicated control problems, this is often not feasible, making it necessary to employ a more gradual and adaptive sampling strategy, like POPS, to improve the controller effectively.

Table 6.3: Statistical results of different sampling methods. The columns "Mean" "90%", and "Median" represent the cost ratio statistics across 200 test initial points.

Methods	Mean	90%	Median
POPS	1.17 (± 0.09)	1.22 (± 0.09)	1.06 (± 0.02)
AS w. bad v	1.45 (± 0.18)	1.93 (± 0.33)	1.24 (± 0.12)
AS w. large v	1.94 (± 0.09)	2.97 (± 0.38)	1.46 (± 0.08)
AS w. large u	1.75 (± 0.33)	2.39 (± 0.67)	1.44 (± 0.23)
Vanilla sampling	2.15 (± 0.96)	3.18 (± 1.71)	1.89 (± 0.78)

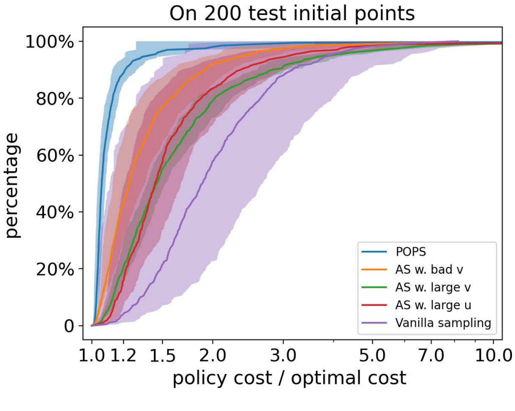

Figure 6.5: Cumulative distribution function of the cost ratio among 200 test trajectories across different sampling methods.

Table 6.4: Average cost ratio on 200 test points of controllers trained by POPS and DAgger. All models are trained with time grids $0 < 10 < 14 < 16$. Cost ratios are clipped at 10.0 for each test trajectory. For each setting, we repeat 5 times with different random seeds. DAgger performs similarly to POPS with 500 trajectories but shows a noticeable drop in performance when reduced to 300.

Methods	Number of trajectories	
	500	300
POPS	1.17 (± 0.09)	1.34 (± 0.13)
DAgger	1.19 (± 0.06)	1.50 (± 0.13)

7 The optimal reaching problem of a 7-DoF manipulator

In this section, we consider the optimal reaching problem on a 7-DoF torque-controlled manipulator, the KUKA LWR iiwa R820 14 [7, 33]. We formulate the dynamics of the manipulator as

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \big(\boldsymbol{v}, \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{u})\big),$$

where $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{v}) \in \mathbb{R}^{14}$, $\boldsymbol{q} \in \mathbb{R}^7$ is the joint angles, $\boldsymbol{v} = \dot{\boldsymbol{q}} \in \mathbb{R}^7$ is the joint velocities, $\ddot{\boldsymbol{q}} = \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}^7$ is the acceleration of joint angles, and $\boldsymbol{u} \in \mathbb{R}^7$ is the control torque. The forward dynamics $\boldsymbol{a}$ is detailed in Appendix D.

Our goal is to find the optimal torque $\boldsymbol{u} \in \mathcal{U} \subset \mathbb{R}^7$ that drives the manipulator from $\boldsymbol{x}_0$ to $\boldsymbol{x}_1$ in $T = 0.8$ seconds and minimizes a quadratic type cost. This cost reflects two competing objectives: tracking a desired terminal state and minimizing control-related effort along the way. Specifically, the running cost penalizes both dynamic acceleration and deviation from the gravity-compensating torque $\boldsymbol{u}_1$, while the terminal cost enforces accurate arrival at the target state $\boldsymbol{x}_1$. See Fig. 7.1 for an illustration of the task. In the experiments, we take

$$\begin{aligned}
&\boldsymbol{x}_0 = (\boldsymbol{q}_0, \boldsymbol{0}), \quad \boldsymbol{x}_1 = (\boldsymbol{q}_1, \boldsymbol{0}) \\
&\boldsymbol{q}_0 = [1.68, 1.25, 2.44, -1.27, -0.98, 1.12, -1.36]^\top, \\
&\boldsymbol{q}_1 = [2.77, 0.58, 1.54, -1.70, -2.17, 0.08, -2.58]^\top.
\end{aligned}$$

The initial positions $\boldsymbol{q}$ are sampled uniformly and independently in a 7-dimensional cube centered at $\boldsymbol{q}_0$ with side length 0.02. Initial velocities $\boldsymbol{v}$ are set to zero. The running cost and terminal cost are

$$\begin{aligned}
&L(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{u})^\top Q_{\boldsymbol{a}} \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{u}) + (\boldsymbol{u} - \boldsymbol{u}_1)^\top Q_{\boldsymbol{u}} (\boldsymbol{u} - \boldsymbol{u}_1), \\
&M(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_1)^\top Q_f (\boldsymbol{x} - \boldsymbol{x}_1),
\end{aligned}$$

where $\boldsymbol{u}_1$ is the torque to balance gravity at state $\boldsymbol{x}_1$, i.e. $\boldsymbol{a}(\boldsymbol{x}_1, \boldsymbol{u}_1) = \boldsymbol{0}$. Under this setting, $(\boldsymbol{x}_1, \boldsymbol{u}_1)$ is an equilibrium of the system, i.e. $\boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{u}_1) = (\boldsymbol{v}_1, a(\boldsymbol{x}_1, \boldsymbol{u}_1)) = \boldsymbol{0}$. We take $Q_{\boldsymbol{a}} = 0.005 I_7, Q_{\boldsymbol{u}} = 0.025 I_7, Q_f = 25000 I_{14}$ where we use large weights Q_f to ensure the reaching goal is approximately achieved.

To obtain training data, we solve the open-loop control problem use differential dynamic programming [26] implemented in the Crocoddy library [42]. Instead of directly

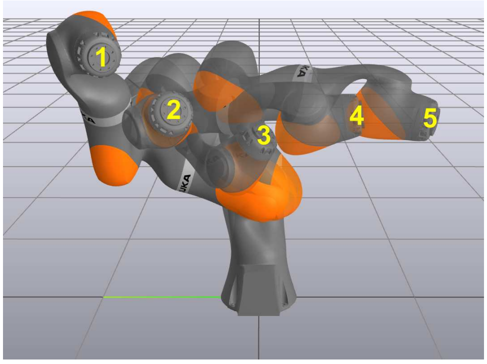

Figure 7.1: An illustration of the reaching problem of the manipulator. The solid manipulator demonstrates its initial position. We label the end effectors of the five instances of robots by "1, 2, 3, 4, 5" to indicator the position of the robot at different times $t_1 = 0.0, t_2 = 0.2, t_3 = 0.4, t_4 = 0.6, t_5 = 0.8$.

applying the open-loop solver to each collected initial state, we first sample a mini-batch of initial states around $\boldsymbol{q}_0$ from the same distribution and solve the open-loop problem for each. We then select the solution with the lowest cost and use it as an initial guess to generate all remaining trajectories. This strategy accelerates data generation, helps avoid poor local minima, and leads to a 100% success rate in solving the open-loop control problem. In the simulation and open-loop solver, we take time step $\Delta t = 0.001$ and use the semi-implicit Euler discretization. Each trajectory has $T/\Delta t = 800$ data points that are pairs of 15-dimensional input states (including time) and 7-dimensional output controls.

The backbone network for this example is the QRNet [44, 46]. QRNet exploits the solution corresponding to the linear quadratic regulator problem at equilibrium and thus improves the network performance around the equilibrium [25]. The usage of a different network structure in this example also demonstrates the genericness/versatility of POPS. We leave the details for QRNet to Appendix E.

All the QRNets $\boldsymbol{u}^{\text{QR}}$ are trained with the Adam optimizer [32] with learning rate 0.001, batch size 256 and epochs 2000. We utilize a fully-connected network with 6 hidden layers; each layer has 128 neurons. Compared to the quadrotor problem, here we used a deeper network and replaced the bounded `tanh` activation with the unbounded `ELU` [15] in the last three layers to handle the manipulator problem's increased complexity and significantly varying scale. During iterations, all networks are trained from scratch, i.e. a new network with random weights instead of loading weights from the previous iteration.

We evaluate networks trained in six different ways: four using Algorithm 1 with different temporal grid points (POPS1-POPS4), and two using the vanilla sampling method with 300 (Vanilla300) and 900 (Vanilla900) trajectories. POPS1-POPS4 are trained with an initial training data of 100 trajectories and undergo three iterations ($K = 3$), i.e. each of them requires solving the open-loop solution 300 times in total.

Each experiment repeat five times independently and evaluate them on the test dataset comprising 1200 trajectories. We again plot the cumulative distribution functions of cost ratios (clipped at 2.0) between the NN-controlled cost and optimal cost in Fig. 7.2. As reported in Table 7.1, we find that adding more data in the vanilla sampling method has very

Table 7.1: The mean ratio of policy costs / optimal costs of the optimal reaching problem of the manipulator. The ratio has been clipped at 2.0 for each test trajectory. The vanilla300/900 correspond to networks trained on 300/900 optimal trajectories, respectively. The choices of temporal grid points for adaptive sampling in the remaining rows can be inferred by the location of columns. For example, POPS1 has temporal grid points $0 < 0.16 < 0.48 < 0.8$. POPS3* has the same temporal grid points as POPS3 except that it augments the dataset directly instead of replacing them, as discussed in Section 8.

	Time grid	iter 0	iter 1	iter 2
Vanilla300		1.92 (± 0.06)		
Vanilla900		1.89 (± 0.09)		
POPS1	0.16 - 0.48 - 0.8	1.81 (± 0.16)	1.29 (± 0.21)	1.09 (± 0.08)
POPS2	0.16 - 0.56 - 0.8	1.94 (± 0.07)	1.34 (± 0.17)	1.15 (± 0.14)
POPS3	0.16 - 0.64 - 0.8	1.92 (± 0.08)	1.37 (± 0.21)	1.07 (± 0.11)
POPS3*	0.16 - 0.64 - 0.8	1.96 (± 0.05)	1.40 (± 0.17)	1.24 (± 0.28)
POPS4	0.16 - 0.72 - 0.8	1.96 (± 0.03)	1.29 (± 0.18)	1.13 (± 0.10)

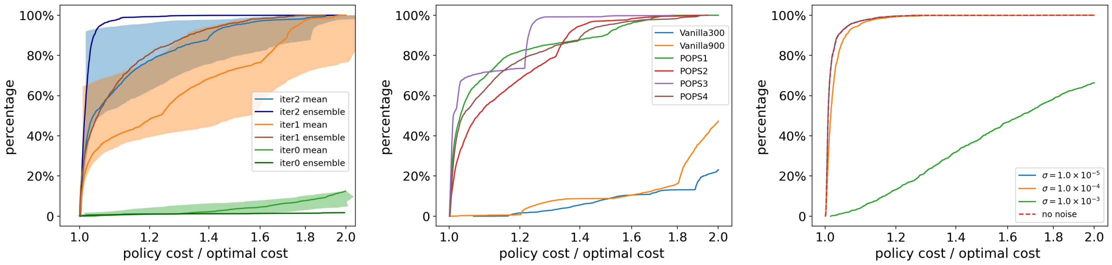

Figure 7.2: Cumulative distribution functions of cost ratios (with the ideal curve being a straight horizontal segment passing ratio = 1, percentage=100%) under different iterations when training POPS4 (left), different training schemes (middle) and different intensities of measurement noises (right). The shaded regions in the left plot represent the mean ± standard deviation. The percentages are computed based on the cost ratios of 1200 test trajectories.

limited effects on improvement while POPS greatly improves the performance. Again, such improvement is robust to different choices of temporal grid points (POPS1-POPS4). Besides, we also try augmenting the dataset with newly collected data instead of replacing them, as detailed in Section 8. Through the comparison between POPS3 and POPS3*, we find that the alternative approach does not bring further improvement.

We additionally evaluate the NN controllers in the presence of measurement errors by adding a noise uniformly sampled from $[-\sigma, \sigma]^{14}$ to the state $\boldsymbol{x}$ of the network input. See Fig. 7.2(right) for the results on the best model trained from POPS1-POPS4. The NN controller performs well at $\sigma = 10^{-4}$, and there are more than 60% of cases in which our controller achieves a ratio less than 2.0 at $\sigma = 10^{-3}$.

7.1 Comparison with DAgger

In this subsection, we compare our method with DAgger. In DAgger, we use the same temporal grid points at $t = 0.16$ and $t = 0.64$ as POPS3. The DAgger algorithm achieves a policy cost/optimal cost ratio of 1.049 on the test dataset, which is close to that achieved by POPS.

We then increase the difficulty of the control problem by enlarging the moving distance. Specifically, we change the center of the initial position and the terminal position to

$$q_0 = [1.60, 1.30, 2.70, -0.85, -1.90, 0.95, -1.60]^\top,$$
$$q_1 = [2.75, 0.60, 2.00, -1.55, -2.15, 0.00, -2.60]^\top,$$

respectively. We also increase the size of the initial dataset from 100 trajectories to 200 trajectories. Each network is trained with 1500 epochs. The other settings remain unchanged.

For a comprehensive comparison, we choose two different temporal grids points. DAgger1 uses the same temporal grids as POPS1 while DAgger2 uses the same temporal grid points as POPS3. Each experiment has been run 5 times independently, and we report their average and best performance. The results are summarized in Table 7.2 and Fig. 7.3. As we can see, POPS is capable of finding a closed-loop controller with an average ratio between policy cost and optimal cost achieving 1.0155. However, the DAgger algorithm cannot yield such a satisfactory result.

As discussed in Section 6.1, we maintain that DAgger requires a strong initial controller for effective state exploration; otherwise, states sampled by a poor controller might even deteriorate the performance. The results in Table 7.2 support this view. First, we observe that DAgger1 performs similarly to the network trained in the second iteration of POPS, suggesting that additional data sampled at the later time $t = 0.48$ offers little benefit. Furthermore, comparing DAgger1 and DAgger2, where DAgger1 has an earlier final grid point (0.48 vs 0.64) and better performance (mean ratio 1.8528 vs 1.6327), reveals that late-time sampling actually worsens the performance. Lastly, we performed an additional iteration of DAgger (DAgger3), which requires adding 400 more trajectories. However, this led to worse performance, as the new states were sampled by a less effective controller.

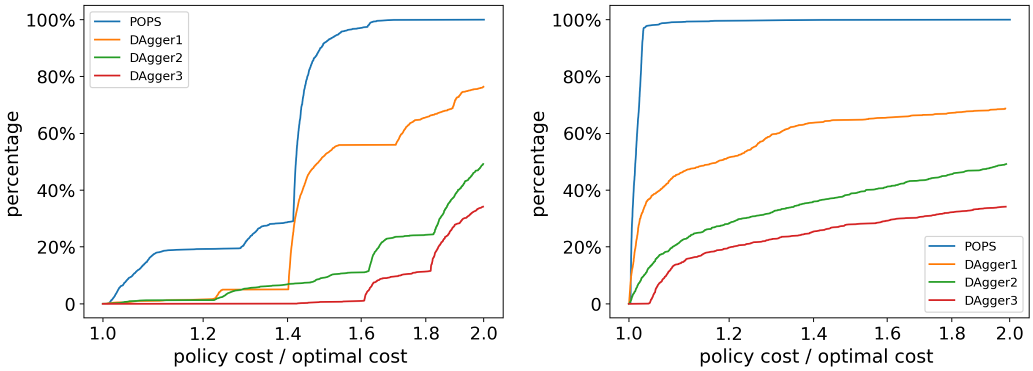

Figure 7.3: Cumulative distribution functions of average cost ratio over 5 independent experiments (left) and the cost ratio of the best controller among 5 independent experiments (right) under the proposed method and DAgger for the optimal reaching problem with a larger moving distance. The percentages are computed based on the cost ratios of 1200 test trajectories.

Table 7.2: The mean ratio between policy costs and optimal costs of the reaching problem with a larger moving distance. In each cell, the first number is averaged over 5 independent experiments and the number in parenthesis is the average ratio achieved by the best controller among them. The ratio has been clipped at 2.0 for each test trajectory. Both POPS and DAgger2 have temporal grid points $0 < 0.16 < 0.64 < 0.8$. DAgger3 repeats DAgger2 for one more DAgger iteration on the same temporal grid points.

	# of iterations	Temporal grid points ($T = 0.8$)		
		0.16	0.48	0.64
POPS	3	1.6592 (1.5417)		1.3610 (1.0155)
DAgger1	2		1.6327 (1.4004)	
DAgger2	2			1.8528 (1.6364)
DAgger3	3			1.9293 (1.7478)

8 Conclusion and future work

In this work, we propose the progressive optimal path sampling method to overcome the distribution mismatch problem in the supervised-learning-based approaches for the closed-loop optimal control problem. Both theoretical and numerical results show that the POPS method significantly improves the performance of the learned NN controller and outperforms other adaptive sampling methods.

There are several directions worth exploring in future work. One design choice in POPS is the selection of temporal grid points for adaptive sampling. In this paper, we use a heuristic that selects a common grid point for all trajectories, based on when the distance between the training data and NN-controlled trajectories begins to increase rapidly (see Fig. 6.1 for an example). We observe that POPS performs well with this strategy. A recent work [25] proposes a more systematic approach that selects grid points individually for each trajectory, based on the first significant deviation from its optimal counterpart. It would be worthwhile to further develop and generalize such strategies.

Another direction is to design more effective approaches utilizing the training data. In Algorithm 1 (lines 8-9), at each iteration, we replace parts of the training data with the newly collected data, and hence some optimal labels are thrown away, which are costly to obtain. An alternative choice is to augment data directly, i.e. setting $S_i = \hat{S}_i \bigcup S_{i-1}$ in line 9. Numerically, we observe that this choice gives similar performance to the version used in Algorithm 1, which suggests that so far the dropped data provides little value for training. However, it is still possible to find smarter ways to utilize them to improve performance. Beyond data efficiency, it is important to evaluate POPS on more challenging classes of optimal control problems, such as those involving state or control constraints. Another important challenge is the presence of multi-modal optimal policies. One strategy to address this is to pre-train a neural network and use it to warm-start an open-loop optimal-control solver, thereby restricting the training set to a single mode that the network can accurately approximate [45]. A complementary direction is to model the policy stochastically with a generative approach, such as a diffusion model, so that multiple modes are captured implicitly [14, 18, 24, 27]. It is of interest to explore how POPS can be integrated with neural-network warm starts and stochastic policy representations.

Finally, POPS can be straightforwardly applied to learning general dynamics from multiple trajectories as the controlled system under the optimal policy can be viewed as a special dynamical system. It is of interest to investigate its performance in such general settings. Theoretical analysis beyond the LQR setting is also an interesting and important problem.

Appendix A Detailed setting and proof in Section 5

We first present the settings for the vanilla supervised-learning-based method, DAgger, and POPS when applied to the LQR problem in Section 5.

Vanilla method. For the vanilla method, we first randomly sample NT initial states $\{\tilde{x}^v_j\}_{j=1}^{NT}$[1] from a standard normal distribution where N is a positive integer (recalling T is a positive integer). Then NT approximated optimal paths are collected starting at $t_0 = 0$

$$\begin{aligned}\hat{u}^v_j(t) &= -\frac{T}{T^2+1}\tilde{x}^v_j + \epsilon Z^v_j,\\ \hat{x}^v_j(t) &= \frac{T(T-t)+1}{T^2+1}\tilde{x}^v_j + \epsilon t Z^v_j,\end{aligned} \tag{A.1}$$

where $\{Z^v_j\}_{j=1}^{NT}$ are i.i.d. normal random variables with mean m and variance σ^2, and independent of initial states.

Finally, the parameters θ are learned by solving the following least square problems:

$$\min_\theta \int_0^T \sum_{j=1}^{NT} \left|\hat{u}^v_j(t) - u_\theta\big(t, \hat{x}^v_j(t)\big)\right|^2 \mathrm{d}t. \tag{A.2}$$

For the first and second models, we optimize θ_t independently for each t:

$$\begin{aligned}\theta_t &= \arg\min_b \sum_{j=1}^{NT} \left|\hat{u}^v_j(t) + \frac{T}{T(T-t)+1}\hat{x}^v_j(t) - b\right|^2\\ \text{or}\quad \theta_t &= \arg\min_{(a,b)} \sum_{j=1}^{NT} \left|\hat{u}^v_j(t) - a\hat{x}^v_j(t) - b\right|^2.\end{aligned}$$

We will use u_v to denote the closed-loop controller determined in this way. Notice that $\{\hat{x}^v_j(t)\}_{j=1}^N$ share the same distribution, we have that $\hat{x}^v(t)$ has the same distribution of $\hat{x}^v_1(t)$.

[1] Through the LQR analysis, all symbols having a hat are open-loop optimal paths sampled for training, e.g. $\hat{u}^v_j, \hat{x}^v_j, \hat{u}^p_{i,j}, \hat{x}^p_{i,j}$. Let $\tilde{x}$ denote a single state instead of a state trajectory. The clean symbol x without hat or tilde is the IVP solution generated by specific controllers which are specified in the subscript; e.g. x_o, x_v, x_d, x_p are trajectories generated by u_o, u_v, u_d, u_p which are optimal, vanilla, DAgger and POPS controllers, respectively. The positive integer j in the subscript always denotes the index of the optimal path. Symbols with superscript i are related to the i-th temporal grid points in DAgger or POPS.

DAgger. In DAgger, we again choose $K = T$ and the temporal grid points $t_i = i$ for $0 \leq i \leq K$. We first sample N initial points $\{\tilde{x}^d_{0,j}\}_{j=1}^N$ from the normal standard distribution and then generated N approximated optimal paths starting at $t_0 = 0$

$$\begin{aligned} \hat{u}^d_{0,j}(t) &= -\frac{T}{T^2+1}\tilde{x}^d_{0,j} + \epsilon Z^d_{0,j}, \\ \hat{x}^d_{0,j}(t) &= \frac{T(T-t)+1}{T^2+1}\tilde{x}^d_{0,j} + \epsilon t Z^d_{0,j}, \end{aligned}$$

where $\{Z^d_{0,j}\}_{j=1}^N$ are i.i.d. normal random variables and independent with initial states whose mean is m and variance is σ^2. We then train the closed-loop controller u_0 by solving the following least square problems:

$$\min_\theta \int_0^T \sum_{j=1}^N \left|\hat{u}^d_{0,j}(t) - u_0\big(t, \hat{x}^d_{0,j}(t)\big)\right|^2 \mathrm{d}t.$$

Then, we use u^0_d to solve the IVPs on the whole time horizon $[0, T]$ with initial states $\{\tilde{x}^d_{0,j}\}$

$$\dot{x}^d_{0,j}(t) = u_0\big(t, x^d_{0,j}(t)\big), \quad x^d_{0,j}(0) = \tilde{x}^d_{0,j}, \quad 1 \leq j \leq N, \tag{A.3}$$

and collect $\{\tilde{x}^d_{i,j}\}_{j=1}^N$ as $\tilde{x}^d_{i,j} := x^d_{0,j}(i)$ for $i = 1, 2, \dots, T-1$. At each time step $t_i = i$, we then compute N approximated optimal paths starting from $\{\tilde{x}^d_{i,j}\}_{j=1}^N$

$$\begin{aligned} \hat{u}^d_{i,j}(t) &= -\frac{T}{T(T-i)+1}\tilde{x}^d_{i,j} + \epsilon Z^d_{i,j}, \\ \hat{x}^d_{i,j}(t) &= \frac{T(T-t)+1}{T(T-i)+1}\tilde{x}^d_{i,j} + (t-i)\epsilon Z^d_{i,j}, \quad t \in [i, T], \end{aligned} \tag{A.4}$$

where $\{Z^d_{i,j}\}_{1 \leq i \leq T-1}$ are i.i.d. normal random variables and independent with $\{\tilde{x}^d_{0,j}\}_{j=1}^N$ and $\{Z^d_{0,j}\}_{j=1}^N$ whose mean is m and variance is σ^2. Finally, we collect the optimal paths $\{(\hat{u}^d_{i,j}, \hat{x}^d_{i,j})\}_{0 \leq i \leq T-1, 1 \leq j \leq N}$ to train the closed-loop controller u_d by solving the following least square problems:

$$\min_\theta \int_i^{i+1} \sum_{k=0}^i \sum_{j=1}^N \left|\hat{u}^d_{k,j}(t) - u_\theta\big(t, \hat{x}^d_{k,j}(t)\big)\right|^2 \mathrm{d}t \tag{A.5}$$

for $i = 0, 1, \dots, T-1$. In the theoretical part, we will only do a single iteration and use u_d to denote the policy learned by (A.5). In the numerical part, we can replace u_0 by the policy learned by (A.5) and repeat this process multiple times.

POPS. In POPS, we choose $K = T$ and the temporal grid points $t_i = i$ for $0 \leq i \leq K$. We first sample N initial points $\{\tilde{x}^p_{0,j}\}_{j=1}^N$ from the normal standard distribution, denote the parameters optimized at i-th iteration as θ^i and initialize $\theta^{-1} = 0$. At the i-th iteration ($0 \le i \le T-1$), we use $u_{\theta^{i-1}}$ to solve the IVPs on the time horizon $[0, i]$

$$\dot{x}^p_{i,j}(t) = u_{\theta^{i-1}}\big(t, x^p_{i,j}(t)\big), \quad x^p_{i,j}(0) = \tilde{x}^p_{0,j}, \quad 1 \le j \le N, \tag{A.6}$$

and collect $\{\tilde{x}^p_{i,j}\}_{j=1}^N$ as $\tilde{x}^p_{i,j} := x^p_{i,j}(i)$. We then compute N approximated optimal paths starting from $\{\tilde{x}^p_{i,j}\}_{j=1}^N$ at $t_i = i$

$$\begin{aligned} \hat{u}^p_{i,j}(t) &= -\frac{T}{T(T-i)+1}\tilde{x}^p_{i,j} + \epsilon Z^p_{i,j}, \\ \hat{x}^p_{i,j}(t) &= \frac{T(T-t)+1}{T(T-i)+1}\tilde{x}^p_{i,j} + (t-i)\epsilon Z^p_{i,j}, \quad t \in [i, T], \end{aligned} \tag{A.7}$$

where $\{Z^p_{i,j}\}_{0 \le i \le T-1, 1 \le j \le N}$ are i.i.d. normal random variables with mean m and variance σ^2, and independent of $\{\tilde{x}^p_{0,j}\}_{j=1}^N$. Note that $\hat{u}^p_{i,j}$ and $\hat{x}^p_{i,j}$ are only defined in $t \in [i, T]$ (for $i \ge 1$), we then fill their values in interval $[0, i)$ with values from previous iteration,

$$\hat{u}^p_{i,j}(t) = \hat{u}^p_{i-1,j}(t), \quad \hat{x}^p_{i,j}(t) = \hat{x}^p_{i-1,j}(t), \quad t \in [0, i). \tag{A.8}$$

Finally, we solve the least squares problems to determine θ^i:

$$\min_\theta \int_0^T \sum_{j=1}^N \big|\hat{u}^p_{i,j}(t) - u_\theta\big(t, \hat{x}^p_{i,j}(t)\big)\big|^2 \mathrm{d}t. \tag{A.9}$$

We will then use u_p to denote $u_{\theta^{T-1}}$, the policy learned in the last iteration. Again, notice that $\hat{x}^p_{T-1,j}(t)$ share the same distribution, we know that $\hat{x}^p(t)$ has the same distribution of $\hat{x}^p_{T-1,1}(t)$.

Below, we give the proof of Theorem 5.1.

A.1 Vanilla sampling method

We first give the closed-form expressions of u_v using Model 1 (5.1). Recalling $\hat{u}^v_j(t)$ and $\hat{x}^v_j(t)$ given in Eq. (A.1), we have

$$\hat{u}^v_j(t) = -\frac{T}{T(T-t)+1}\hat{x}^v_j(t) + \frac{T^2+1}{T(T-t)+1}\epsilon Z^v_j, \quad 1 \le j \le NT.$$

Therefore, recalling u_v is learned through the least square problem (A.2), we have

$$u_v(t,x) = -\frac{T}{T(T-t)+1}x + \frac{T^2+1}{T(T-t)+1}\epsilon \bar{Z}^v, \tag{A.10}$$

where

$$\bar{Z}^v = \frac{1}{NT}\sum_{j=1}^{NT} Z^v_j.$$

Distribution difference. Using the control (A.10), we have

$$\dot{x}_v(t) = -\frac{T}{T(T-t)+1}x_v(t) + \frac{T^2+1}{T(T-t)+1}\epsilon\bar{Z}^v, \quad x_v(0) = \tilde{x}_{\text{init}}.$$

Solving this ODE gives

$$x_v(t) = \frac{T(T-t)+1}{T^2+1}\tilde{x}_{\text{init}} + \epsilon t\bar{Z}^v. \tag{A.11}$$

Combining the last equation with the fact that

$$\hat{x}_j^v(t) = \frac{T(T-t)+1}{T^2+1}\tilde{x}_j^v + \epsilon t Z_j^v,$$

$\tilde{x}_j^v$, $\tilde{x}_{\text{init}}$ and $\{Z_j^v\}_{j=1}^{NT}$ are independent normal random variables and

$$\tilde{x}_j^v, \tilde{x}_{\text{init}} \sim \mathcal{N}(0,1), \quad Z_j^v \sim \mathcal{N}(m,\sigma^2),$$

we know that $\hat{x}_j^v(t)$ and $x_v(t)$ are normal random variables with

$$\begin{aligned}&\mathbb{E}\hat{x}_1^v(t) = \mathbb{E}x_v(t),\\ &\left|\text{Var}\left|\hat{x}_1^v(t)\right|^2 - \text{Var}|x_v(t)|^2\right| = \sigma^2\left(1-\frac{1}{NT}\right)\epsilon^2 t^2.\end{aligned}$$

Performance difference. First, with the optimal solution

$$u_o(t) = -\frac{T}{T^2+1}\tilde{x}_{\text{init}}, \quad x_o(T) = \frac{1}{T^2+1}\tilde{x}_{\text{init}},$$

we have

$$J_o = \frac{1}{T}\int_0^T \left|\frac{T}{T^2+1}\tilde{x}_{\text{init}}\right|^2 \mathrm{d}t + \left|\frac{1}{T^2+1}\tilde{x}_{\text{init}}\right|^2 = \frac{1}{T^2+1}|\tilde{x}_{\text{init}}|^2.$$

Recalling Eq. (A.11) and plugging (A.11) into (A.10), we know that

$$\begin{aligned}x_v(T) &= \frac{1}{T^2+1}\tilde{x}_{\text{init}} + \epsilon T\bar{Z}^v,\\ u_v(t) &= -\frac{T}{T^2+1}\tilde{x}_{\text{init}} + \epsilon\bar{Z}^v.\end{aligned}$$

Hence,

$$J_v = \epsilon^2|\bar{Z}^v|^2 - \frac{2T}{T^2+1}\tilde{x}_{\text{init}}\epsilon\bar{Z}^v + \epsilon^2T^2|\bar{Z}^v|^2 + \frac{2T}{T^2+1}\tilde{x}_{\text{init}}\epsilon\bar{Z}^v + \frac{1}{T^2+1}|\tilde{x}_{\text{init}}|^2,$$

which gives

$$\mathbb{E}J_v - J_o = (T^2+1)\left(m^2 + \frac{\sigma^2}{NT}\right)\epsilon^2.$$

A.2 POPS

Similarly, we first compute u_p. Recalling Eqs. (A.7) and (A.8), when $0 \le i \le T-1$, $1 \le j \le N$ and $t \in [i, i+1)$, we have

$$\begin{aligned} \hat{u}^p_{T-1,j}(t) &= \hat{u}^p_{i,j}(t) = -\frac{T}{T(T-i)+1}\tilde{x}^p_{i,j} + \epsilon Z^p_{i,j}, \\ \hat{x}^p_{T-1,j}(t) &= \hat{x}^p_{i,j}(t) = \frac{T(T-t)+1}{T(T-i)+1}\tilde{x}^p_{i,j} + (t-i)\epsilon Z^p_{i,j}. \end{aligned} \tag{A.12}$$

Therefore,

$$\hat{u}^p_{T-1,j}(t) = -\frac{T}{T(T-t)+1}\hat{x}^p_{T-1,j}(t) + \frac{T(T-i)+1}{T(T-t)+1}\epsilon Z^p_{i,j}.$$

Hence, recalling u_p is learned through the least square problem (A.9), we have, when $t \in [i, i+1)$

$$u_p(t,x) = -\frac{T}{T(T-t)+1}x + \frac{T(T-i)+1}{T(T-t)+1}\epsilon \bar{Z}^p_i, \tag{A.13}$$

where

$$\bar{Z}^p_i = \frac{1}{N}\sum_{j=1}^{N} Z^p_{i,j}, \quad 0 \le i \le T-1.$$

Eq. (A.13) also holds when $i = T-1$ and $t = T$.

We then compute the starting points $\{\tilde{x}^p_{i,j}\}_{0\le i\le T-1, 1\le j\le N}$ in the POPS method. By Eq. (A.8), we know that when $1 \le i \le i' \le T-1$ and $0 \le t < t_i$, $\theta^i_t = \theta^{i'}_t$. Together with Eq. (A.6), we know that when $0 \le i \le T-2$,

$$\begin{aligned} &u_{\theta^i}(t,x) = u_{\theta^{T-1}}(t,x) = u_p(t,x), && t \in [i,i+1), \\ &x^p_{i+1,j}(t) \equiv x^p_{i,j}(t), && t \in [0,i], \end{aligned}$$

which implies

$$x^p_{i+1,j}(i) = x^p_{i,j}(i) = \tilde{x}^p_{i,j}.$$

Therefore, for $1 \le j \le N$, when $0 \le i \le T-2$, we have

$$\begin{cases} \dot{x}^p_{i+1,j}(t) = u_{\theta^i}\big(t, x^p_{i+1,j}(t)\big) = u_p\big(t, x^p_{i+1,j}(t)\big) \\ \qquad\qquad = -\dfrac{T}{T(T-t)+1}x^p_{i+1,j}(t) + \dfrac{T(T-i)+1}{T(T-t)+1}\epsilon\bar{Z}^p_i, \quad t \in [i,i+1], \\ x^p_{i+1,j}(i) = \tilde{x}^p_{i,j}. \end{cases}$$

Solving the above ODE, we get the solution

$$x^p_{i+1,j}(t) = \frac{T(T-t)+1}{T(T-i)+1}\tilde{x}^p_{i,j} + (t-i)\epsilon\bar{Z}^i_p, \quad t \in [i,i+1].$$

Hence, by definition, for $0 \le i \le T-2$,

$$\tilde{x}^p_{i+1,j} = x^p_{i+1,j}(i+1) = \frac{T(T-i-1)+1}{T(T-i)+1}\tilde{x}^p_{i,j} + \epsilon \bar{Z}^i_p.$$

Utilizing the above recursive relationship, we obtain, for $0 \le i \le T-1$[2],

$$\tilde{x}^p_{i,j} = \frac{T(T-i)+1}{T^2+1}\tilde{x}^p_{0,j} + \sum_{k=0}^{i-1}\frac{T(T-i)+1}{T(T-k-1)+1}\epsilon\bar{Z}^k_p. \tag{A.14}$$

Distribution difference. For $0 \le i \le T-1$ and $t \in [i, i+1)$, using the control (A.13), we have

$$\dot{x}_p(t) = -\frac{T}{T(T-t)+1}x_p(t) + \frac{T(T-i)+1}{T(T-t)+1}\epsilon\bar{Z}^p_i.$$

Solving the above ODE with the initial condition $x_p(0) = \tilde{x}_{\text{init}}$, we can get the solution

$$x_p(t) = \frac{T(T-t)+1}{T^2+1}\tilde{x}_{\text{init}} + \sum_{k=0}^{i-1}\frac{T(T-t)+1}{T(T-k-1)+1}\epsilon\bar{Z}^k_p + (t-i)\epsilon\bar{Z}^p_i, \tag{A.15}$$

when $0 \le i \le T-1$ and $t \in [i, i+1)$. The above equation also holds when $i = T-1$ and $t = T$.

On the other hand, combining Eqs. (A.12) and (A.14), we know that when $0 \le i \le T-1$ and $t \in [i, i+1)$ or $i = T-1$ and $t = T$,

$$\begin{aligned}\hat{x}^p_{T-1,j}(t) &= \frac{T(T-t)+1}{T(T-i)+1}\tilde{x}^p_{i,j} + (t-i)\epsilon Z^p_{i,j}\\ &= \frac{T(T-t)+1}{T^2+1}\tilde{x}^p_{0,j} + \sum_{k=0}^{i-1}\frac{T(T-t)+1}{T(T-k-1)+1}\epsilon\bar{Z}^k_p + (t-i)\epsilon Z^p_{i,j}.\end{aligned}$$

The above equation also holds when $i = T-1$ and $t = T$.

Combining the last equation with Eq. (A.15) and the fact that $\{Z^p_{i,j}\}_{0\le i\le T-1, 1\le j\le N}$, $\tilde{x}^p_{0,j}$ and $\tilde{x}_{\text{init}}$ are independent normal random variables and

$$\tilde{x}^p_{0,j}, \tilde{x}_{\text{init}} \sim \mathcal{N}(0,1), \quad Z^p_{i,j} \sim \mathcal{N}(m, \sigma^2),$$

we know that $\hat{x}^p_{T-1,1}(t)$ and $x_p(t)$ are normal random variables with

$$\begin{aligned}&\mathbb{E}\hat{x}^p_{T-1,1}(t) = \mathbb{E}x_p(t),\\ &\left|\mathrm{Var}|\hat{x}^p_{T-1,1}(t)|^2 - \mathrm{Var}|x_p(t)|^2\right| = \sigma^2\epsilon^2(t-i)^2\left(1-\frac{1}{N}\right) \le \sigma^2\epsilon^2.\end{aligned}$$

[2] In this section, we take by convention that summation $\sum_{k=m}^{n} c_k = 0$ if $m > n$.

Performance difference. Recalling Eq. (A.15) and plugging (A.15) into (A.13), we know that

$$
\begin{aligned}
x_p(T) &= \frac{1}{T^2+1}\tilde{x}_{\text{init}} + \sum_{k=0}^{T-1} \frac{1}{T(T-k-1)+1}\epsilon \bar{Z}_p^k,\\
u_p(t) &= -\frac{T}{T^2+1}\tilde{x}_{\text{init}} - \sum_{k=0}^{i-1} \frac{T}{T(T-k-1)+1}\epsilon \bar{Z}_p^k + \epsilon \bar{Z}_i^p, \quad 0 \le i \le T-1, \quad t \in [i, i+1).
\end{aligned}
$$

To compute the difference between J_p and J_o, we first notice that

$$
\begin{aligned}
\mathbb{E}J_p - J_o = &\left[\frac{1}{T}\int_0^T \mathrm{Var}\big(u_p(t)\big)\mathrm{d}t + \mathrm{Var}\big(x_p(T)\big)\right]\\
&+ \left[\frac{1}{T}\int_0^T |\mathbb{E}u_p(t)|^2 \mathrm{d}t + |\mathbb{E}x_p(T)|^2 - J_o\right] =: \mathrm{I}_1 + \mathrm{I}_2.
\end{aligned}
$$

By the independence of $\{\bar{Z}_i^p\}_{i=0}^{T-1}$, we know that

$$
\begin{aligned}
\mathrm{I}_1 &= \frac{\sigma^2\epsilon^2}{NT}\sum_{i=0}^{T-1}\sum_{k=0}^{i-1}\frac{T^2}{[T(T-k-1)+1]^2} + \frac{\sigma^2\epsilon^2}{N} + \sum_{k=0}^{T-1}\frac{1}{[T(T-k-1)+1]^2}\frac{\sigma^2\epsilon^2}{N}\\
&= \frac{\sigma^2\epsilon^2}{N}\left[1 + \sum_{i=0}^{T-1}\sum_{k=0}^{i-1}\frac{T}{[T(T-k-1)+1]^2} + \sum_{k=0}^{T-1}\frac{1}{[T(T-k-1)+1]^2}\right]\\
&= \frac{\sigma^2\epsilon^2}{N}\left[1 + \sum_{k=0}^{T-2}\sum_{i=k+1}^{T-1}\frac{T}{[T(T-k-1)+1]^2} + \sum_{k=0}^{T-1}\frac{1}{[T(T-k-1)+1]^2}\right]\\
&= \frac{\sigma^2\epsilon^2}{N}\left[1 + \sum_{k=0}^{T-1}\frac{T(T-k-1)+1}{[T(T-k-1)+1]^2}\right]\\
&= \frac{\sigma^2\epsilon^2}{N}\left[1 + \sum_{k=0}^{T-1}\frac{1}{Tk+1}\right] \le \frac{3\sigma^2\epsilon^2}{N}.
\end{aligned}
$$

Meanwhile, noticing that

$$
\begin{aligned}
\mathbb{E}x_p(T) &= \frac{1}{T^2+1}\tilde{x}_{\text{init}} + \sum_{k=0}^{T-1}\frac{1}{T(T-k-1)+1}\epsilon m,\\
\mathbb{E}u_p(t) &= -\frac{T}{T^2+1}\tilde{x}_{\text{init}} - \sum_{k=0}^{i-1}\frac{T}{T(T-k-1)+1}\epsilon m + \epsilon m,
\end{aligned}
$$

it is straightforward to compute that

$$
\mathrm{I}_2 = \frac{2\epsilon m \tilde{x}_{\text{init}}}{T^2+1}\mathrm{I}_3 + \epsilon^2 m^2(\mathrm{I}_4 + 1),
$$

where

$$\begin{aligned}
\mathrm{I}_3 &= \sum_{k=0}^{T-1} \frac{1}{T(T-k-1)+1} + \sum_{i=0}^{T-1}\sum_{k=0}^{i-1} \frac{T}{T(T-k-1)+1} - T \\
&= \sum_{k=0}^{T-1} \frac{1}{T(T-k-1)+1} + \sum_{k=0}^{T-1}\sum_{i=k+1}^{T-1} \frac{T}{T(T-k-1)+1} - T \\
&= \sum_{k=0}^{T-1} \frac{T(T-k-1)+1}{T(T-k-1)+1} - T = 0, \\
\mathrm{I}_4 &= \left(\sum_{k=0}^{T-1} \frac{1}{T(T-k-1)+1}\right)^2 + T\sum_{i=0}^{T-1}\left(\sum_{k=0}^{i-1} \frac{1}{T(T-k-1)+1}\right)^2 \\
&\quad - \sum_{i=0}^{T-1}\sum_{k=0}^{i-1} \frac{2}{T(T-k-1)+1} \\
&= \sum_{k=0}^{T-1} \frac{1+T(T-k-1)}{[1+T(T-k-1)]^2} + 2\sum_{i=0}^{T-1}\sum_{k=0}^{i-1} \frac{T(T-i-1)+1}{[T(T-k-1)+1][T(T-i-1)+1]} \\
&\quad - 2\sum_{i=0}^{T-1}\sum_{k=0}^{i-1} \frac{1}{T(T-k-1)+1} \\
&= \sum_{k=0}^{T-1} \frac{1}{1+T(T-k-1)} = \sum_{k=0}^{T-1} \frac{1}{Tk+1} \le 2.
\end{aligned}$$

Therefore,

$$\mathbb{E}J_p - J_o = \mathrm{I}_1 + \frac{2\epsilon m \tilde{x}_{\mathrm{init}}}{T^2+1}\mathrm{I}_3 + \epsilon^2 m^2(\mathrm{I}_4+1) \le 3\left(m^2 + \frac{\sigma^2}{N}\right)\epsilon^2.$$

A.3 Dagger

With the same approach of computing u_v in (A.10), we have

$$u_0(t,x) = -\frac{T}{T(T-t)+1}x + \frac{T^2+1}{T(T-t)+1}\epsilon \bar{Z}_0^d,$$

where

$$\bar{Z}_0^d = \frac{1}{N}\sum_{j=1}^{N} Z_{0,j}^d.$$

Recalling the definition of $x_{0,j}^d$ in Eq. (A.3), we have

$$x_{0,j}^d(t) = \frac{T(T-t)+1}{T^2+1}\tilde{x}_{0,j}^d + \epsilon t \bar{Z}_0^d.$$

Hence, for $0 \le i \le T-1$, we have

$$\tilde{x}^d_{i,j} = x^d_{0,j}(i) = \frac{T(T-i)+1}{T^2+1}\tilde{x}^d_{0,j} + \epsilon i \bar{Z}^d_0.$$

Plugging the last equation into Eq. (A.4), we have that for $t \in [i, T]$,

$$\begin{aligned}
\hat{u}^d_{i,j}(t) &= -\frac{T}{T^2+1}\tilde{x}^d_{0,j} - \epsilon\frac{Ti}{F(i)}\bar{Z}^d_0 + \epsilon Z^d_{i,j},\\
\hat{x}^d_{i,j}(t) &= \frac{T(T-t)+1}{T^2+1}\tilde{x}^d_{0,j} + \epsilon\frac{iF(t)}{F(i)}\bar{Z}^d_0 + (t-i)\epsilon Z^d_{i,j},
\end{aligned}$$

where $F(t) = T(T-t)+1$. Therefore,

$$\hat{u}^d_{i,j}(t) = -\frac{T}{F(t)}\hat{x}^d_{i,j}(t) + \frac{F(i)}{F(t)}\epsilon Z^d_{i,j}.$$

We can then compute the least squares problem (A.5) to obtain that for $0 \le i \le T-1$ and $t \in [i, i+1)$, we have

$$u_d(t,x) = -\frac{T}{F(t)}x + \frac{1}{i+1}\sum_{k=0}^{i}\frac{F(k)}{F(t)}\epsilon\bar{Z}^d_k,$$

where

$$\bar{Z}^d_i = \frac{1}{N}\sum_{j=1}^{N} Z^d_{i,j}$$

for $0 \le i \le T-1$. Hence, we have that when $0 \le i \le T-1$ and $t \in [i, i+1)$,

$$\begin{aligned}
x_d(t) = {}& \frac{T(T-t)+1}{T^2+1}\tilde{x}_{\text{init}} + \sum_{k=0}^{i-1}\frac{F(t)}{F(k+1)F(k)}\frac{1}{k+1}\sum_{l=0}^{k}\epsilon F(l)\bar{Z}^d_l\\
&+ \frac{t-i}{(i+1)F(i)}\sum_{k=0}^{i}F(k)\epsilon\bar{Z}^d_k.
\end{aligned}$$

Therefore, when $0 \le i \le T-1$ and $t \in [i, i+1)$,

$$\begin{aligned}
u_d(t) = {}& -\frac{T}{T^2+1}\tilde{x}_{\text{init}} - \sum_{k=0}^{i-1}\frac{T}{(k+1)F(k)F(k+1)}\sum_{l=0}^{k}\epsilon F(l)\bar{Z}^d_l\\
&+ \frac{1}{(i+1)F(i)}\sum_{k=0}^{i}F(k)\epsilon\bar{Z}^d_k.
\end{aligned}$$

Distribution difference. We first give a lower bound of $\mathrm{Var}(\hat{x}^d(t))$. For any $t \in [i, i+1)$, $\hat{x}^d(t)$ can be viewed as the random variable which uniformly samples an element from $\{\hat{x}_{k,j}(t)\}_{0\le k\le i, 1\le j\le N}$. Noticing that

$$\mathrm{Var}(X) = \mathrm{Var}\big(\mathbb{E}(X|Y)\big) + \mathbb{E}\mathrm{Var}(X|Y),$$

we know that for any $t \in [i, i+1)$,

$$\mathrm{Var}\big(\hat{x}^d(t)\big) \geq \frac{1}{N(i+1)} \sum_{k=0}^{i} \sum_{j=1}^{N} \mathrm{Var}\big(\hat{x}^d_{k,j}(t)\big).$$

Then, through the independence of $\tilde{x}^d_{0,j}$, $Z^d_{0,j}$ and $Z^d_{i,j}$, we have that

$$\begin{aligned}\mathrm{Var}\big(\hat{x}^d(t)\big) &\geq \left[\frac{T(T-t)+1}{T^2+1}\right]^2 + \frac{1}{i+1} \sum_{k=0}^{i} (t-k)^2 \epsilon^2 \sigma^2 \\ &= \left[\frac{T(T-t)+1}{T^2+1}\right]^2 + \epsilon^2\sigma^2 \left[t^2 - ti + \frac{i(2i+1)}{6}\right].\end{aligned}$$

Note that

$$t^2 - ti + \frac{i(2i+1)}{6} - \frac{t^2}{3} = t(t-i) + \frac{i}{6} - \frac{1}{3}(t+i)(t-i) \geq \left(\frac{2}{3}t - \frac{1}{3}i\right)(t-i) \geq 0.$$

Hence,

$$\mathrm{Var}\big(\hat{x}^d(t)\big) \geq \left[\frac{T(T-t)+1}{T^2+1}\right]^2 + \frac{t^2\epsilon^2\sigma^2}{3}. \tag{A.16}$$

Next, we give an upper bound of $\mathrm{Var}(x_d(t))$. In this part, without loss of generality, we assume $m = 0$, as the value of m does not influence of the value of $\mathrm{Var}(x_d(t))$. Let

$$\begin{aligned}x'_d(t) &= x_d(t) - \frac{T(T-t)+1}{T^2+1}\tilde{x}_{\text{init}} \\ &= \sum_{k=0}^{i-1} \frac{F(t)}{F(k+1)F(k)} \frac{1}{k+1} \sum_{l=0}^{k} \epsilon F(l) \bar{Z}^d_l \\ &\quad + \frac{t-i}{(i+1)F(i)} \sum_{k=0}^{i} F(k)\epsilon \bar{Z}^d_k.\end{aligned}$$

Then,

$$\mathrm{Var}\big(x_d(t)\big) = \left[\frac{T(T-t)+1}{T^2+1}\right]^2 + \mathrm{Var}\big(x'_d(t)\big), \tag{A.17}$$

and for any $t \in [i, i+1)$, $\dot{x}'_d(t) = u_d(t, x'_d(t))$. Define

$$V(t) = \mathrm{Var}\big(x'_d(t)\big) = \mathbb{E}|x'_d(t)|^2.$$

Then, we have that

$$\dot{V}(t) = 2\mathbb{E}x'_d(t)u_d\big(t, x'_d(t)\big) = -\frac{2T}{F(t)}V(t) + 2\frac{\epsilon^2\sigma^2}{N}\mathrm{I}_5,$$

where

$$\mathrm{I}_5 = \sum_{k=0}^{i-1} \frac{1}{F(k+1)F(k)(k+1)(i+1)} \sum_{l=0}^{k} F^2(l) + \frac{t-i}{(i+1)^2 F(i)F(t)} \sum_{k=0}^{i} F^2(k).$$

Noticing that $F(l) \leq F(0)$, we have

$$\begin{aligned} I_5 &\leq \frac{F^2(0)}{i+1} \left[\sum_{k=0}^{i-1} \frac{1}{F(k+1)F(k)} + \frac{(t-i)}{F(i)F(t)} \right] \\ &= \frac{F^2(0)}{T(i+1)} \left[\sum_{k=0}^{i-1} \left(\frac{1}{F(k+1)} - \frac{1}{F(k)} \right) + \frac{1}{F(t)} - \frac{1}{F(0)} \right] \\ &= \frac{F^2(0)}{T(i+1)} \left[\frac{1}{F(t)} - \frac{1}{F(0)} \right] = \frac{F^2(0)}{T(i+1)} \frac{Tt}{F(0)F(t)} \leq \frac{F(0)}{F(t)}. \end{aligned}$$

Therefore,

$$\dot{V}(t) \leq -\frac{2T}{F(t)} V(t) + \frac{2\epsilon^2\sigma^2}{N} \frac{F(0)}{F(t)}.$$

Let

$$W(t) = 2\frac{\epsilon^2\sigma^2}{N} t - V(t),$$

then $W(0) = 0$, and

$$\begin{aligned} \dot{W}(t) + \frac{2T}{F(t)} W(t) &\geq \frac{\epsilon^2\sigma^2}{N} \left[2 + \frac{4Tt}{F(t)} \right] - \dot{V}(t) - \frac{2T}{F(t)} \\ &\geq \frac{\epsilon^2\sigma^2}{N} \left[2 + \frac{4Tt}{F(t)} - 2\frac{F(0)}{F(t)} \right] \\ &= \frac{\epsilon^2\sigma^2}{N} \left[2 + \frac{4Tt}{F(t)} - 2\frac{2F(0)}{F(t)} \right] = \frac{\epsilon^2\sigma^2}{N} \frac{2Tt}{F(t)} \geq 0. \end{aligned}$$

Hence, $W(t) \geq 0$, which means that

$$\mathrm{Var}\big(x_d'(t)\big) = V(t) \leq 2\frac{\epsilon^2\sigma^2}{N} t$$

Combining the last inequality with (A.16) and (A.17), we have

$$\mathrm{Var}\big(\hat{x}^d(t)\big) - \mathrm{Var}\big(x_d(t)\big) \geq \epsilon^2\sigma^2 \left(\frac{t^2}{3} - \frac{2t}{N} \right).$$

Performance difference. Define

$$\begin{aligned} e_i = &- \sum_{k=0}^{i-1} \frac{T}{(k+1)F(k)F(k+1)} \sum_{l=0}^{k} \epsilon F(l) \bar{Z}_l^d \\ &+ \frac{1}{(i+1)F(i)} \sum_{k=0}^{i} F(k) \epsilon \bar{Z}_k^d, \quad 0 \leq i \leq T-1. \end{aligned}$$

We have

$$\begin{aligned}
J_d &= \frac{1}{T}\sum_{i=0}^{T-1}\left|-\frac{T}{T^2+1}\tilde{x}_{\text{init}}+e_i\right|^2+\left|\tilde{x}_{\text{init}}-\frac{T^2}{T^2+1}\tilde{x}_{\text{init}}+\sum_{i=0}^{T-1}e_i\right|^2 \\
&= \frac{T^2|x_{\text{init}}|^2}{(T^2+1)^2}-\sum_{i=0}^{T-1}\frac{2\tilde{x}_{\text{init}}e_i}{T^2+1}+\frac{1}{T}\sum_{i=0}^{T-1}|e_i|^2+\frac{|\tilde{x}_{\text{init}}|^2}{(T^2+1)^2}+\sum_{i=0}^{T-1}\frac{2e_i\tilde{x}_{\text{init}}}{T^2+1}+\left|\sum_{i=0}^{T-1}e_i\right|^2 \\
&= \frac{|\tilde{x}_{\text{init}}|^2}{T^2+1}+\frac{1}{T}\sum_{i=0}^{T-1}|e_i|^2+\left|\sum_{i=0}^{T-1}e_i\right|^2.
\end{aligned}$$

Therefore,

$$\mathbb{E}J_d - J_o \geq \mathbb{E}\left|\sum_{i=0}^{T-1}e_i\right|^2 = \left(\mathbb{E}\sum_{i=0}^{T-1}e_i\right)^2+\text{Var}\left(\sum_{i=0}^{T-1}e_i\right). \tag{A.18}$$

We can then compute that

$$\begin{aligned}
\sum_{i=0}^{T-1}e_i &= -\sum_{i=0}^{T-1}\sum_{k=0}^{i-1}\sum_{l=0}^{k}\frac{\epsilon TF(l)\bar{Z}_l^d}{(k+1)F(k)F(k+1)}+\sum_{i=0}^{T-1}\sum_{k=0}^{i}\frac{F(k)\epsilon\bar{Z}_k^d}{(i+1)F(i)} \\
&= \sum_{i=0}^{T-1}\sum_{k=0}^{i}\frac{F(k)\epsilon\bar{Z}_k^d}{(i+1)F(i)}-\sum_{k=0}^{T-1}\sum_{l=0}^{k}\sum_{i=k+1}^{T-1}\frac{\epsilon TF(l)\bar{Z}_l^d}{(k+1)F(k)F(k+1)} \\
&= \sum_{i=0}^{T-1}\sum_{k=0}^{i}\frac{F(k)\epsilon\bar{Z}_k^d}{(i+1)F(i)}-\sum_{i=0}^{T-1}\sum_{k=0}^{i}\frac{\epsilon T(T-i-1)F(k)\bar{Z}_k^d}{(i+1)F(i)F(i+1)} \\
&= \sum_{i=0}^{T-1}\sum_{k=0}^{i}\frac{F(k)\epsilon\bar{Z}_k^d}{(i+1)F(i)F(i+1)}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbb{E}\sum_{i=0}^{T-1}e_i &= \epsilon m\sum_{i=0}^{T-1}\sum_{k=0}^{i}\frac{F(k)}{(i+1)F(i)F(i+1)} \\
&= \epsilon m\sum_{i=0}^{T-1}\frac{(T^2+1)(i+1)-Ti(i+1)/2}{(i+1)F(i)F(i+1)} \\
&= \epsilon m\sum_{i=0}^{T-1}\frac{T^2+1-Ti/2}{[T(T-i)+1][T(T-i-1)+1]} \\
&\geq \epsilon m\frac{T^2+T+2}{2T}\sum_{i=0}^{T-1}\left[\frac{1}{T(T-i-1)+1}-\frac{1}{T(T-i)+1}\right] \\
&= \epsilon m\frac{T^2+T+2}{2T}\left(1-\frac{1}{T^2+1}\right)\geq\frac{\epsilon mT}{2}.
\end{aligned} \tag{A.19}$$

On the other hand, noticing that

$$\sum_{i=0}^{T-1} e_i = \sum_{k=0}^{T-1}\sum_{i=k}^{T-1} \frac{F(k)\epsilon \bar{Z}_k^d}{(i+1)F(i)F(i+1)},$$

we have

$$\begin{aligned}
\mathrm{Var}\left(\sum_{i=0}^{T-1} e_i\right) &= \frac{\epsilon^2\sigma^2}{N}\sum_{k=0}^{T-1} F^2(k)\left(\sum_{i=k}^{T-1}\frac{1}{(i+1)F(i)F(i+1)}\right)^2 \\
&\geq \frac{\epsilon^2\sigma^2}{NT^4}\sum_{k=0}^{T-1}[T(T-k)+1]^2\left(\sum_{i=k}^{T-1}\frac{1}{T(T-i-1)+1} - \frac{1}{T(T-i)+1}\right)^2 \\
&= \frac{\epsilon^2\sigma^2}{NT^4}\sum_{k=0}^{T-1}[T(T-k)+1]^2\left[1-\frac{1}{T(T-k)+1}\right]^2 \\
&= \frac{\epsilon^2\sigma^2}{NT^2}\sum_{k=0}^{T-1}(T-K)^2 = \frac{\epsilon^2\sigma^2}{NT^2}\frac{T(T+1)(2T+1)}{6} \geq \frac{\epsilon^2\sigma^2 T}{3N}. \qquad \text{(A.20)}
\end{aligned}$$

Combining Eqs. (A.18)-(A.20), we conclude our result.

Appendix B Full dynamics of quadrotor

In this section, we introduce the full dynamics of quadrotor [10,40,41] that are considered in Section 6. The state variable of a quadrotor is $\boldsymbol{x} = (\boldsymbol{p}^\top, \boldsymbol{v}_b^\top, \boldsymbol{\eta}^\top, \boldsymbol{w}_b^\top)^\top \in \mathbb{R}^{12}$ where $\boldsymbol{p} = (x,y,z) \in \mathbb{R}^3$ is the position of quadrotor in Earth-fixed coordinates, $\boldsymbol{v}_b \in \mathbb{R}^3$ is the velocity in body-fixed coordinates, $\boldsymbol{\eta} = (\phi,\theta,\psi) \in \mathbb{R}^3$ (roll, pitch, yaw) is the attitude in terms of Euler angles in Earth-fixed coordinates, and $\boldsymbol{w}_b \in \mathbb{R}^3$ is the angular velocity in body-fixed coordinates. Control $\boldsymbol{u} = (s,\tau_x,\tau_y,\tau_z)^\top \in \mathbb{R}^4$ is composed of total thrust s and body torques (τ_x,τ_y,τ_z) from the four rotors. Then we can model the quadrotor's dynamics as

$$\begin{cases}
\dot{\boldsymbol{p}} = \boldsymbol{R}^\top(\boldsymbol{\eta})\boldsymbol{v}_b, \\
\dot{\boldsymbol{v}}_b = -\boldsymbol{w}_b \times \boldsymbol{v}_b - \boldsymbol{R}(\boldsymbol{\eta})\boldsymbol{g} + \dfrac{1}{m}A\boldsymbol{u}, \\
\dot{\boldsymbol{\eta}} = \boldsymbol{K}(\boldsymbol{\eta})\boldsymbol{w}_b, \\
\dot{\boldsymbol{w}}_b = -\boldsymbol{J}^{-1}\boldsymbol{w}_b \times \boldsymbol{J}\boldsymbol{w}_b + \boldsymbol{J}^{-1}B\boldsymbol{u}
\end{cases} \qquad \text{(B.1)}$$

with matrix A and B defined as

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The constant mass m and inertia matrix $\boldsymbol{J} = \mathrm{diag}(J_x, J_y, J_z)$ are the parameters of the quadrotor, where J_x, J_y, and J_z are the moments of inertia of the quadrotor in the x-axis,

y-axis, and z-axis, respectively. We set $m = 2$ kg and

$$J_x = J_y = \frac{1}{2} J_z = 1.2416 \text{ kg} \cdot \text{m}^2,$$

which are the same system parameters as in [40]. The constants $\boldsymbol{g} = (0, 0, g)^\top$ denote the gravity vector where $g = 9.81 \text{ m/s}^2$ is the acceleration of gravity on Earth. The direction cosine matrix $\boldsymbol{R}(\boldsymbol{\eta}) \in SO(3)$ represents the transformation from the Earth-fixed coordinates to the body-fixed coordinates

$$\boldsymbol{R}(\boldsymbol{\eta}) = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\theta\cos\psi\sin\phi - \sin\psi\cos\phi & \sin\theta\sin\psi\sin\phi + \cos\psi\cos\phi & \cos\theta\sin\phi \\ \sin\theta\cos\psi\cos\phi + \sin\psi\sin\phi & \sin\theta\sin\psi\cos\phi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{bmatrix},$$

and the attitude kinematic matrix $\boldsymbol{K}(\boldsymbol{\eta})$ relates the time derivative of the attitude representation with the associated angular rate

$$\boldsymbol{K}(\boldsymbol{\eta}) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix}.$$

Note that in practice the quadrotor is directly controlled by the individual rotor thrusts $\boldsymbol{F} = (F_1, F_2, F_3, F_4)^\top$, and we have the relation $\boldsymbol{u} = E\boldsymbol{F}$ with

$$E = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & l & 0 & -l \\ -l & 0 & l & 0 \\ c & -c & c & -c \end{bmatrix},$$

where l is the distance from the rotor to the UAV's center of gravity and c is a constant that relates the rotor angular momentum to the rotor thrust (normal force). So once we obtain the optimal control $\boldsymbol{u}^*$, we are able to get the optimal $\boldsymbol{F}^*$ immediately by the relation $\boldsymbol{F}^* = E^{-1}\boldsymbol{u}^*$.

Appendix C PMP and space marching method

In this section, we introduce the open-loop optimal problem solver used for solving the optimal landing problem of a quadrotor. The solver is based on Pontryagin's minimum principle (PMP) [49] and space-marching method [56]. The optimal landing problem is defined as

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{u}} \int_0^T L\big(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau)\big) d\tau + M\big(\boldsymbol{x}(T)\big), \\ &\text{s.t.} \quad \begin{cases} \dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big), & t \in [0, T], \\ \boldsymbol{x}(0) = \boldsymbol{x}_0, \end{cases} \end{aligned} \tag{C.1}$$

where $\boldsymbol{x}(t) : [0, T] \to \mathbb{R}^{12}$ and $\boldsymbol{u}(t) : [0, T] \to \mathbb{R}^4$ denote the state trajectory and control trajectory, respectively, and f is the full dynamics of quadrotor introduced in Appendix B.

By PMP, problem (C.1) can be solved through solving a two-point boundary value problem (TPBVP). Introduce costate variable $\boldsymbol{\lambda} \in \mathbb{R}^{12}$ and Hamiltonian

$$H(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{u}) = L(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\lambda} \cdot f(\boldsymbol{x}, \boldsymbol{u}).$$

The TPBVP is defined as

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \partial_{\boldsymbol{\lambda}}^T H\big(\boldsymbol{x}(t), \boldsymbol{\lambda}(t), \boldsymbol{u}^*(t)\big), \\ \dot{\boldsymbol{\lambda}}(t) = -\partial_{\boldsymbol{x}}^T H\big(\boldsymbol{x}(t), \boldsymbol{\lambda}(t), \boldsymbol{u}^*(t)\big). \end{cases} \tag{C.2}$$

We have the boundary conditions

$$\begin{cases} \boldsymbol{x}(0) = \boldsymbol{x}_0, \\ \boldsymbol{\lambda}(T) = \nabla M\big(\boldsymbol{x}(T)\big), \end{cases} \tag{C.3}$$

and the optimal control $\boldsymbol{u}^*(t)$ should minimize Hamiltonian at each t

$$\boldsymbol{u}^*(t) = \arg\min_{\boldsymbol{u}} H\big(\boldsymbol{x}(t), \boldsymbol{\lambda}(t), \boldsymbol{u}\big). \tag{C.4}$$

We use *solve_bvp* function of *scipy* [31] to solve TPBVP (C.2)-(C.4) and set tolerance to 10^{-5}, *max_nodes* to 5000. We note that when the initial state $\boldsymbol{x}_0$ is far from the target state $\boldsymbol{x}_T$, solving the TPBVP directly often fails. Thus we use the space-marching method proposed in [56]. We uniformly select K points in the line segment from $\boldsymbol{x}_T$ to $\boldsymbol{x}_0$, and denote them as $\{\boldsymbol{x}_0^1, \boldsymbol{x}_0^2, \cdots, \boldsymbol{x}_0^K\}$ according to their increasing distances to $\boldsymbol{x}_T$ ($\boldsymbol{x}_0^K = \boldsymbol{x}_0$). These K TPBVPs will be solved in order and at every step we use the previous solution as the initial guess to the current problem. With this strategy, every open-loop trajectory in our experiments converged. In later runs, we warm-start the solver with the IVP trajectory generated by our trained neural controller, which almost eliminates the need for space-marching.

Appendix D Details on the 7-DoF manipulator

In this section, we introduce the dynamics for the 7-DoF torque-controlled manipulator. Recall that the dynamics of the manipulator is,

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \big(\boldsymbol{v}, \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{u})\big),$$

where $\boldsymbol{u} \in \mathbb{R}^7$ is the control torque, $\boldsymbol{x} = (\boldsymbol{q}, \boldsymbol{v}) \in \mathbb{R}^{14}$, $\boldsymbol{q} \in \mathbb{R}^7$ is the joint angles, $\boldsymbol{v} = \dot{\boldsymbol{q}} \in \mathbb{R}^7$ is the joint velocities, $\ddot{\boldsymbol{q}} = \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}^7$ is the acceleration of joint angles. To close the equation, we write down the inverse dynamics of the manipulator,

$$M(\boldsymbol{q})\boldsymbol{a} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{u},$$

where one can compute the acceleration in terms of $\boldsymbol{x}$ and $\boldsymbol{u}$. Here $M(\boldsymbol{q})$ is the generalized inertia matrix, $C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ represents the centrifugal forces and Coriolis forces, and $\boldsymbol{g}(\boldsymbol{q})$ is the generalized gravity.

Appendix E The QRNet

In this section, we introduce the structure of QRNet. QRNet utilize the linear quadratic regulator controller at an equilibrium and thus improve stability around the equilibrium. Suppose we have the LQR controller, $\boldsymbol{u}^{\mathrm{LQR}}$, for the problem with linearized dynamics and quadratized costs at $(\boldsymbol{x}_1, \boldsymbol{u}_1)$, the QRNet can be formulated as

$$\boldsymbol{u}^{\mathrm{QR}}(t, \boldsymbol{x}) = \sigma\big(\boldsymbol{u}^{\mathrm{LQR}}(t, \boldsymbol{x}) + \hat{\boldsymbol{u}}(t, \boldsymbol{x}; \theta) - \hat{\boldsymbol{u}}(T, \boldsymbol{x}_1; \theta)\big),$$

where $\hat{\boldsymbol{u}}(t, \boldsymbol{x}; \theta)$ is any neural network with trainable parameters θ, and σ is a saturating function that satisfies $\sigma(\boldsymbol{u}_1) = \boldsymbol{u}_1, \sigma_{\boldsymbol{u}}(\boldsymbol{u}_1) = I$, where I is the identity matrix. The σ used in this example is defined coordinate-wisely as

$$\sigma(u) = u_{\min} + \frac{u_{\max} - u_{\min}}{1 + c_1 \exp[-c_2(u - u_1)]},$$

where

$$c_1 = \frac{u_{\max} - u_1}{u_1 - u_{\min}}, \quad c_2 = \frac{u_{\max} - u_{\min}}{(u_{\max} - u_1)(u_1 - u_{\min})}$$

with $u_{\min}, u_{\max}$ being minimum and maximum values for u. Here $u, u_{\min}$ and $u_{\max}$ are the corresponding values at each coordinate of $\boldsymbol{u}, \boldsymbol{u}_{\min}, \boldsymbol{u}_{\max}$, respectively. In the first experiment evaluated in Section 7, we set $u_{\min} = -150$ and $u_{\max} = 15$. In the more challenging scenario in Section 7.1, we adjust the bounds to $u_{\min} = -2000$ and $u_{\max} = 2000$ to prevent saturation, as DAgger explores a wider range of states, leading to larger control torques.

To get the $\boldsymbol{u}^{\mathrm{LQR}}$, we expand the dynamics linearly as

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \approx \boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{x}_1, \boldsymbol{u}_1)(\boldsymbol{x} - \boldsymbol{x}_1) + \boldsymbol{f}_{\boldsymbol{u}}(\boldsymbol{x}_1, \boldsymbol{u}_1)(\boldsymbol{u} - \boldsymbol{u}_1),$$

and the term related to acceleration in the running cost quadratically as

$$\begin{aligned} &\boldsymbol{a}(\boldsymbol{x}, \boldsymbol{u})^\top Q_{\boldsymbol{a}} \boldsymbol{a}(\boldsymbol{x}, \boldsymbol{u}) \\ \approx\ &\mathcal{L}_{\boldsymbol{a}}(\boldsymbol{x}, \boldsymbol{u})^\top Q_{\boldsymbol{a}} \mathcal{L}_{\boldsymbol{a}}(\boldsymbol{x}, \boldsymbol{u}) \\ =\ &(\boldsymbol{x} - \boldsymbol{x}_1)^\top \boldsymbol{a}_{\boldsymbol{x}}^\top Q_{\boldsymbol{a}} \boldsymbol{a}_{\boldsymbol{x}} (\boldsymbol{x} - \boldsymbol{x}_1) + (\boldsymbol{u} - \boldsymbol{u}_1)^\top \boldsymbol{a}_{\boldsymbol{u}}^\top Q_{\boldsymbol{a}} \boldsymbol{a}_{\boldsymbol{u}} (\boldsymbol{u} - \boldsymbol{u}_1) \\ &\quad + 2(\boldsymbol{x} - \boldsymbol{x}_1)^\top \boldsymbol{a}_{\boldsymbol{x}}^\top Q_{\boldsymbol{a}} \boldsymbol{a}_{\boldsymbol{u}} (\boldsymbol{u} - \boldsymbol{u}_1), \end{aligned}$$

where

$$\mathcal{L}_{\boldsymbol{a}} = \boldsymbol{a}_{\boldsymbol{x}}(\boldsymbol{x}_1, \boldsymbol{u}_1)(\boldsymbol{x} - \boldsymbol{x}_1) + \boldsymbol{a}_{\boldsymbol{u}}(\boldsymbol{x}_1, \boldsymbol{u}_1)(\boldsymbol{u} - \boldsymbol{u}_1),$$

and we exploit $\boldsymbol{a}(\boldsymbol{x}_1, \boldsymbol{u}_1) = \boldsymbol{0}$ and $\boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{u}_1) = \boldsymbol{0}$. The derivatives boil down to $\boldsymbol{a}_{\boldsymbol{x}}$ and $\boldsymbol{a}_{\boldsymbol{u}}$ which can be analytically computed in the Pinocchio library [11–13]. In the experiment, we solve the LQR by the implementation in the Drake library [55].

References

[1] M. Abu-Khalaf and F. L. Lewis, Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach, *Automatica*, 41(5):779–791, 2005.

[2] S. Ainsworth, K. Lowrey, J. Thickstun, Z. Harchaoui, and S. Srinivasa, Faster policy learning with continuous-time gradients, in: *Proceedings of the 3rd Conference on Learning for Dynamics and Control*, PMLR, 144:1054-1067, 2021.

[3] C. Atkeson and J. Morimoto, Nonparametric representation of policies and value functions: A trajectory-based approach, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 15:1611–1618, 2002.

[4] C. Atkeson and B. Stephens, Random sampling of states in dynamic programming, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 20:369–376, 2007.

[5] R. E. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[6] J. T. Betts, Survey of numerical methods for trajectory optimization, *J. Guid. Control Dyn.*, 21(2):193–207, 1998.

[7] R. Bischoff et al., The KUKA-DLR Lightweight Robot arm – a new reference platform for robotics research and manufacturing, in: *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, VDE, 1–8, 2010.

[8] K. M. Borgwardt, A. Gretton, M. J. Rasch, H.-P. Kriegel, B. Schölkopf, and A. J. Smola, Integrating structured biological data by kernel maximum mean discrepancy, *Bioinformatics*, 22(14):e49–e57, 2006.

[9] L. Böttcher, N. Antulov-Fantulin, and T. Asikis, AI Pontryagin or how artificial neural networks learn to control dynamical systems, *Nat. Commun.*, 13(1):333, 2022.

[10] S. Bouabdallah, P. Murrieri, and R. Siegwart, Design and control of an indoor micro quadrotor, in: *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 5, IEEE, 4393–4398, 2004.

[11] J. Carpentier and N. Mansard, Analytical derivatives of rigid body dynamics algorithms, in: *Robotics: Science and Systems*, 2018. `https://api.semanticscholar.org/CorpusID:44070783`

[12] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives, in: *2019 IEEE/SICE International Symposium on System Integration (SII)*, IEEE, 614–619, 2019.

[13] J. Carpentier et al., Pinocchio: Fast forward and inverse dynamics for poly-articulated systems. https://stack-of-tasks.github.io/pinocchio, 2015–2021.

[14] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, Diffusion policy: Visuomotor policy learning via action diffusion, *Int. J. Robot. Res.*, 44(10-11):1684–1704, 2025.

[15] D. Clevert, T. Unterthiner, and S. Hochreiter, Fast and accurate deep network learning by exponential linear units (ELUs), in: *4th International Conference on Learning Representations*, ICLR, 1–14, 2016.

[16] A. Coates, P. Abbeel, and A. Y. Ng, Learning for control from multiple demonstrations, in: *Proceedings of the 25th International Conference on Machine Learning*, ACM, 144–151, 2008.

[17] M. P. Deisenroth, C. E. Rasmussen, and J. Peters, Gaussian process dynamic programming, *Neurocomputing*, 72(7-9):1508–1524, 2009.

[18] C. Domingo-Enrich, J. Han, B. Amos, J. Bruna, and R. T. Q. Chen, Stochastic optimal control matching, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 37:112459–112504, 2024.

[19] W. E, J. Han, and J. Long, Empowering optimal control with machine learning: A perspective from model predictive control, *IFAC-PapersOnLine*, 55(30):121–126, 2022.

[20] W. E, J. Han, and L. Zhang, Machine-learning-assisted modeling, *Phys. Today*, 74:36–41, 2021.

[21] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Upper Saddle River, 2002.

[22] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, PMLR, 9:249-256, 2010.

[23] J. Han and W. E, Deep learning approximation for stochastic control problems, *arXiv:1611.07422*, 2016.

[24] S. Hegde, S. Batra, K. Zentner, and G. Sukhatme, Generating behaviorally diverse policies with latent diffusion models, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 36:7541–7554, 2023.

[25] W. Hu, Y. Zhao, W. E, J. Han, and J. Long, Learning free terminal time optimal closed-loop control of manipulators, in: *Proceedings of the 2025 American Control Conference (ACC)*, IEEE, 127–133, 2025.

[26] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*, in: *Modern Analytic and Computational Methods in Science and Mathematics*, Vol. 24, Elsevier Publishing Company, 1970.

[27] M. Janner, Y. Du, J. Tenenbaum, and S. Levine, Planning with diffusion for flexible behavior synthesis,

in: *Proceedings of the 39th International Conference on Machine Learning*, PMLR, 162:9902-9915, 2022.

[28] G. Kahn, T. Zhang, S. Levine, and P. Abbeel, PLATO: Policy learning using adaptive trajectory optimization, in: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 3342–3349, 2016.

[29] S. Kakade and J. Langford, Approximately optimal approximate reinforcement learning, in: *Proceedings of the Nineteenth International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc., 267–274, 2002.

[30] W. Kang, Q. Gong, T. Nakamura-Zimmerer, and F. Fahroo, Algorithms of data generation for deep learning and feedback design: A survey, *Physica D*, 425:132955, 2021.

[31] J. Kierzenka and L. F. Shampine, A BVP solver based on residual control and the Maltab PSE, *ACM Trans. Math. Software*, 27(3):299–316, 2001.

[32] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, in: *Proceedings of the 3rd International Conference on Learning Representations*, Ithaca, 2015.

[33] Kuka AG, KUKA Robotics Official Website, 2022. `https://www.kuka.com/`

[34] B. Landry, H. Dai, and M. Pavone, SEAGul: Sample efficient adversarially guided learning of value functions, in: *Proceedings of the 3rd Conference on Learning for Dynamics and Control*, PMLR, 144:1105–1117, 2021.

[35] S. Levine and V. Koltun, Guided policy search, in: *Proceedings of the 30th International Conference on Machine Learning*, PMLR, 28(3):1–9, 2013.

[36] S. Levine and V. Koltun, Variational policy search via trajectory optimization, in: *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 26:207–215, 2013.

[37] S. Levine and V. Koltun, Learning complex neural network policies with trajectory optimization, in: *International Conference on Machine Learning*, PMLR, 829–837, 2014.

[38] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*, Princeton University Press, 2011.

[39] J. Long and J. Han, Perturbational complexity by distribution mismatch: A systematic analysis of reinforcement learning in reproducing kernel hilbert space, *J. Mach. Learn.*, 1:1–34, 2022.

[40] T. Madani and A. Benallegue, Control of a quadrotor mini-helicopter via full state backstepping technique, in: *Proceedings of the 45th IEEE Conference on Decision and Control*, IEEE, 1515–1520, 2006.

[41] R. Mahony, V. Kumar, and P. Corke, Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor, *IEEE Rob. Autom. Mag.*, 19(3):20–32, 2012.

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

[43] I. Mordatch and E. Todorov, Combining the benefits of function approximation and trajectory optimization, in: *Robotics: Science and Systems*, 2014. `https://api.semanticscholar.org/CorpusID:5564734`

[44] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, QRnet: Optimal regulator design with LQR-augmented neural networks, *IEEE Control Syst. Lett.*, 5(4):1303–1308, 2020.

[45] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, Adaptive deep learning for high-dimensional Hamilton-Jacobi-Bellman equations, *SIAM J. Sci. Comput.*, 43(2):A1221–A1247, 2021.

[46] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, Neural network optimal feedback control with enhanced closed loop stability, in: *2022 American Control Conference (ACC)*, IEEE, 2373–2378, 2022.

[47] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, Neural network optimal feedback control with guaranteed local stability, *IEEE Open J. Control Syst.*, 1:210–222, 2022.

[48] C. Pinneri, S. Sawant, S. Blaes, and G. Martius, Extracting strong policies for robotics tasks from zero-order trajectory optimizers, in: *9th International Conference on Learning Representations*, ICLR, 2021.

[49] L. S. Pontryagin, *Mathematical Theory of Optimal Processes*, CRC Press, 1987.

[50] A. V. Rao, A survey of numerical methods for optimal control, *Adv. Astronaut. Sci.*, 135(1):497–528, 2009.

[51] S. Ross and D. Bagnell, Efficient reductions for imitation learning, in: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, PMLR, 9:661-668, 2010.

[52] S. Ross, G. Gordon, and D. Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning, in: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and*

Statistics, PMLR, 15:627-635, 2011.

[53] E. D. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, in: *Texts in Applied Mathematics*, Vol. 6, Springer, 2013.

[54] Y. Tassa and T. Erez, Least squares solutions of the HJB equation with neural network value-function approximators, *IEEE Trans. Neural Networks*, 18(4):1031–1041, 2007.

[55] R. Tedrake et al., Drake: Model-based design and verification for robotics, 2019. `https://drake.mit.edu`

[56] Y. Zang, J. Long, X. Zhang, W. Hu, W. E, and J. Han, A machine learning enhanced algorithm for the optimal landing problem, in: *3rd Annual Conference on Mathematical and Scientific Machine Learning*, PMLR, 1–20, 2022.

[57] L. Zhang, H. Wang, and W. E, Reinforced dynamics for enhanced sampling in large atomic and molecular systems, *J. Chem. Phys.*, 148(12):124113, 2018.

[58] Y. Zhao and J. Han, Offline supervised learning vs online direct policy optimization: A comparative study and a unified training paradigm for neural network-based optimal feedback control, *Physica D*, 462:134130, 2024.

[59] Z. Zhao, S. Zuo, T. Zhao, and Y. Zhao, Adversarially regularized policy learning guided by trajectory optimization, *Proc. Mach. Learn. Res.*, 168:844–857, 2022.