GCIMOPT: Learning efficient goal-conditioned policies by imitating optimal trajectories

Jon Goikoetxea JON.GOIKOETXEA-MACUA@ETU.UNIV-GRENOBLE-ALPES.FR
Department of Statistics, Mathematics and Computer Science, Public University of Navarre
Ensimag, Institut National Polytechnique de Grenoble, Université Grenoble Alpes

Jesús Palacián PALACIAN@UNAVARRA.ES
Department of Statistics, Mathematics and Computer Science, Public University of Navarre
Institute for Advanced Materials and Mathematics (InaMat^2)

Editors: G. Sukhatme, L. Lindemann, S. Tu, A. Wierman, N. Atanasov

Abstract

Imitation learning is a well-established approach for machine-learning-based control. However, its applicability depends on having access to demonstrations, which are often expensive to collect and/or suboptimal for solving the task. In this work, we present GCIMOPT, an approach to learn efficient goal-conditioned policies by training on datasets generated by trajectory optimization. Our approach for dataset generation is computationally efficient, can generate thousands of optimal trajectories in minutes on a laptop computer, and produces high-quality demonstrations. Further, by means of a data augmentation scheme that treats intermediate states as goals, we are able to increase the training dataset size by an order of magnitude. Using our generated datasets, we train goal-conditioned neural network policies that can control the system towards arbitrary goals. To demonstrate the generality of our approach, we generate datasets and then train policies for various control tasks, namely cart-pole stabilization, planar and three-dimensional quadcopter stabilization, and point reaching using a 6-DoF robot arm. We show that our trained policies can achieve high success rates and near-optimal control profiles, all while being small (less than $80\,000$ neural network parameters) and fast enough (up to more than $6000\times$ faster than a trajectory optimization solver) that they could be deployed onboard resource-constrained controllers. We provide videos, code, datasets and pre-trained policies under a free software license; see our project website https://jongoiko.github.io/gcimopt/.

Keywords: Machine learning, imitation learning, goal conditioning, optimal control, trajectory optimization

1. Introduction

Optimally controlling a dynamical system enables the accomplishment of a task while minimizing a given cost measure. However, for many dynamical systems, finding an optimal closed-loop controller or policy is very difficult or impossible. For this reason, trajectory optimization (Diehl and Gros, 2011; Betts, 2010) is often used in practice, for example as part of model predictive control (MPC) (Rawlings et al., 2017): the controller computes open-loop optimal controls in a loop at high frequency, executing at each step the first segment of the optimal control. Even though MPC allows the design of near-optimal closed-loop controllers, solving optimization problems at high frequency makes it computationally expensive.

In addition to obtaining controllers that are efficient and have low computational cost, we are interested in having them be *general*: a general policy would be able to accomplish multiple control tasks on the system without having to modify the policy itself. This can be obtained by passing a representation of a *goal* to the controller so that its output is conditioned both on the current system

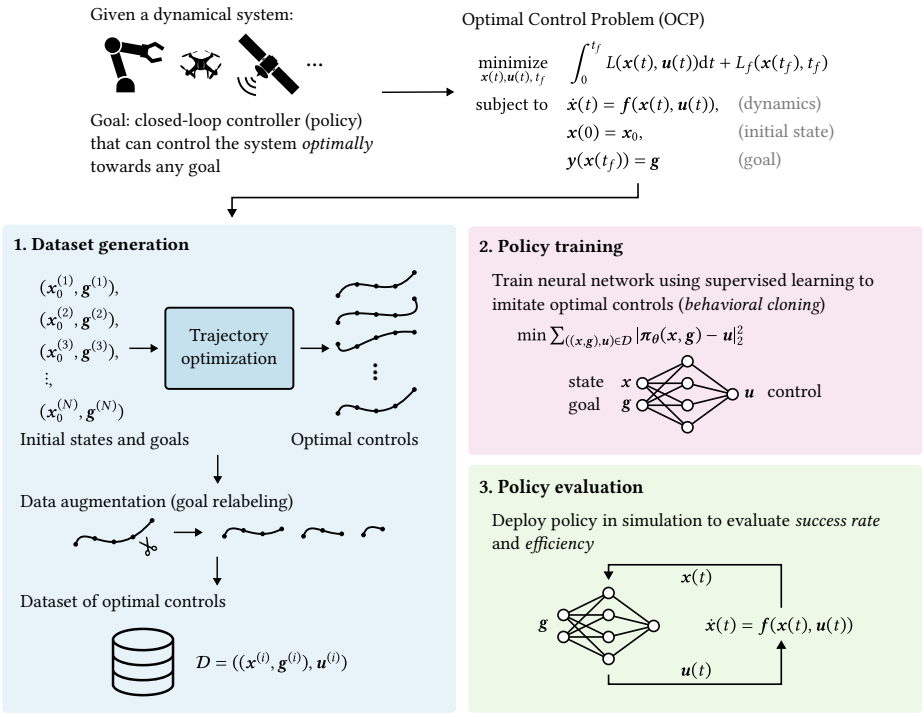

Figure 1: Schematic summary of GCIMOPT, a simple method to train and evaluate near-optimal goal-conditioned policies. A policy π_θ, parameterized by a neural network, is trained by supervised learning on a large dataset of trajectories generated by trajectory optimization. The policy is then evaluated in simulation to measure its success rate and efficiency.

state and the desired goal. Generalization is a central problem in machine learning (ML), since we aim to train models that perform well on the entire data distribution, even when faced with previously unseen data. Thus, a data-driven method can be effective to obtain near-optimal goal-conditioned policies that have low computational cost at inference: using trajectory optimization to generate a large dataset of optimal controls, we can train a small neural network to approximate the optimal controller and generalize to arbitrary goals.

1.1. Related work

Goal conditioning Goal-conditioned policies have been extensively studied in the context of reinforcement learning (Kaelbling, 1993; Schaul et al., 2015; Liu et al., 2022). A fundamental advantage of goal conditioning is that trajectories can be *relabeled* to have their goals be different to the originally intended goals (Andrychowicz et al., 2017), achieving greater sample efficiency than direct RL on goal-conditioned tasks. In spite of this, GCRL approaches often require careful reward shaping and online environment interaction, introducing the challenge of exploration; these difficulties can be avoided by resorting to imitation learning. In goal-conditioned imitation learning (GCIL), we require datasets where the expert solves multiple tasks, which may then be used to speed up training RL agents (Ding et al., 2019), use the expert's intentions as goals (Codevilla et al., 2018)

or imitate multimodal experts with expressive policy classes (Reuss et al., 2023). Additionally, GCIL may be scaled to use massive and heterogeneous datasets, enabling the training of general models that work on various robot morphologies (Shah et al., 2023). GCIL also relaxes the need for expert behavior in the dataset, meaning that suboptimal, uncurated data can still be leveraged for policy learning (Lynch et al., 2020; Ghosh et al., 2019).

Imitation learning on optimal trajectories Trajectories obtained by trajectory optimization can be used as expert data for imitation learning. Earlier work makes a careful consideration of the interplay between policy learning and trajectory optimization, e.g. penalizing the optimized trajectories' deviation from the policy's outputs (Mordatch and Todorov, 2014; Levine and Koltun, 2013; Zhang et al., 2016; Kahn et al., 2017) or leveraging the structure of system dynamics to ensure the policy's stability (Da and Grizzle, 2019). In recent years, there has been a renewed interest in learning from optimal trajectories computed offline, with special emphasis on aerospace applications (Sánchez-Sánchez et al., 2016; Sánchez-Sánchez and Izzo, 2018; Rubinsztejn et al., 2020; Sprague et al., 2020). A notable characteristic of these works is the simplicity of the training setup, since policies are directly trained using behavioral cloning rather than using a more complex algorithm such as DAGGER (Ross et al., 2011). Tailor and Izzo (2019) attribute this advantage to low input dimensionality and large dataset sizes, which minimize the impact of compounding errors in the policies' predictions. Policies trained using this method have been deployed in real quadcopters (Ferede et al., 2024), where they have been shown to perform comparably to differential flatness-based controllers. Although behavior cloning of policies on optimal trajectories has been extensively studied, little work has been done on extending the generality of said policies by means of goal conditioning.

1.2. Contributions

The contributions of this work are the following:

- We present GCIMOPT, a simple method to train and evaluate near-optimal goal-conditioned policies on datasets generated by trajectory optimization (Fig. 1). Unlike GCRL methods, our method does not require reward shaping or online environment interaction, since training is done on optimal trajectories computed offline. We expand on previous work by considering the effects of goal conditioning, thus training policies that can generalize to arbitrary goals while remaining near-optimal. We use a direct method for trajectory optimization, thus avoiding the need to analytically construct optimality conditions for each considered control problem; in addition, we make dataset generation fast by using the FATROP solver (Vanroye et al., 2023), which is purpose-built for optimal control applications.
- To demonstrate the generality of GCIMOPT, we evaluate it empirically on four different simulated dynamical systems. We thus also evaluate the impact of the system's dimensionality and nonlinearity on the trained policies' performance. We show that GCIMOPT policies can achieve high success rates and near-optimal control profiles. Moreover, we show that GCIMOPT policies can achieve speedups of $97\times$ to $6278\times$ versus the fast FATROP solver, suggesting that they could surpass the control frequency of MPC controllers.
- We release an implementation of the method, the code of the experiments, the generated datasets and the trained policies under a free software license. See our project website https://jongoiko.github.io/gcimopt/ for access to these resources.

In the next section, we formalize the problem of finding optimal goal-conditioned policies and recall the basic notions of numerical trajectory optimization. We detail the steps of our method in Sec. 3, and present the experimental evaluation and its results in Sec. 4. Finally, we conclude by discussing the method's advantages and limitations and describing ideas for future work. Additional experiment details and tables are provided in Appendix A.

2. Preliminaries: Closed-loop optimal control with goals

In order to generate high-quality training trajectories that leverage the system's dynamics, we use numerical trajectory optimization. This is a class of methods for solving optimal control problems (OCPs) specifying a control task to be solved while minimizing some measure of cost. We consider continuous-time dynamical systems which can be described by ODEs of the form $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t))$, where $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$ and $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$ denote the system's state and the control at time t respectively.

In this work, we solve OCPs that are constrained to a *goal* being reached at the end of the trajectory, which may be a system state or some other representation derived from a state. Thus, we assume that there is a *state-to-goal mapping* $\boldsymbol{y} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_g}$ that, given a state, returns the goal reached by the system being in that state. We aim to approximate the optimal goal-conditioned closed-loop controller (or *policy*) for the given OCP, which we define as

$$\boldsymbol{\pi}^*(\boldsymbol{x}_0, \boldsymbol{g}) = (\arg\min_{\boldsymbol{u}(t), t_f} \int_0^{t_f} L(\boldsymbol{x}(t), \boldsymbol{u}(t))\mathrm{d}t + L_f(\boldsymbol{x}(t_f), t_f))(0) \tag{1a}$$

$$\text{subject to} \quad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad t \in [0, t_f], \tag{1b}$$

$$\boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{1c}$$

$$\boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \leq \boldsymbol{0}, \quad t \in [0, t_f], \tag{1d}$$

$$\boldsymbol{y}(\boldsymbol{x}(t_f)) = \boldsymbol{g}, \tag{1e}$$

$$t_{\min} \leq t_f \leq t_{\max}. \tag{1f}$$

Thus, the optimal policy's output is the solution to a constrained OCP: the control minimizes the OCP's cost functional (Eq. 1a) with the system starting at state $\boldsymbol{x}_0$ (Eq. 1c) and demanding the goal $\boldsymbol{g}$ to have been reached after a time t_f (Eq. 1e). The cost functional is composed of a *Lagrange term* where a function L is integrated over the trajectory duration, and a *Mayer term* which may incur additional cost at the end of the trajectory, as defined by the function L_f. In Eq. 1a, we use the notation $(\arg\min_{\boldsymbol{u}(t), t_f} \cdots)(0)$ to denote that, given the optimal open-loop control function $\boldsymbol{u}(t)$ defined on $t \in [0, t_f]$, the closed-loop policy outputs the control at the current time $t = 0$, that is $\boldsymbol{u}(0)$. We use the path constraint (Eq. 1d) to incorporate state and actuator limits, and we treat the trajectory duration t_f as a decision variable constrained to a range $[t_{\min}, t_{\max}]$.

We solve OCPs of the given form numerically using a *direct* method. This is in contrast to *indirect* methods such as those based on Pontryagin's Minimum Principle (Pontryagin, 1962), which require analytically constructing the OCP's necessary and sufficient conditions for optimality. In particular, we use direct multiple shooting (Bock and Plitt, 1984) to *transcribe* the OCP to a non-linear program (NLP) in a finite number of decision variables, which is then solved numerically using an NLP solver. For a detailed discussion of numerical trajectory optimization, we refer to Diehl and Gros (2011); Betts (2010, 1998).

3. Learning efficient goal-conditioned policies by imitating optimal trajectories

We aim to train a goal-conditioned policy $\boldsymbol{\pi}_\theta$ to generalize to arbitrary system states and goals, approximate the optimal policy for the given OCP (defined in Eq. 1) and be computationally efficient. To that end, we generate optimal input-output pairs by trajectory optimization (where the inputs are state-goal pairs and the outputs are the corresponding optimal controls). This yields a large dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{g}^{(i)}), \boldsymbol{u}^{(i)}\}_{i=1}^{N_\mathcal{D}}$ of training samples that we can use to train $\boldsymbol{\pi}_\theta$ by supervised learning. For simplicity, in this work we assume deterministic environment dynamics and thus train deterministic policies.

3.1. Dataset generation

We generate a large dataset of optimal trajectories, each of which has an imposed initial state and goal. For each trajectory, we sample a pair of states from a distribution $\mathcal{T}$ over $\mathbb{R}^{n_x} \times \mathbb{R}^{n_x}$, which we call the *task distribution*: from a pair of states $(\boldsymbol{x}_0, \boldsymbol{x}_f) \sim \mathcal{T}$, the first state x_0 is the initial state in the trajectory, while the second $\boldsymbol{x}_f$ is used to generate the goal $\boldsymbol{g} = \boldsymbol{y}(\boldsymbol{x}_f)$. We then solve the corresponding trajectory optimization problem using direct multiple shooting (DMS) (Bock and Plitt, 1984). In DMS, the control and state trajectory over time is discretized into a finite number of decision variables $\boldsymbol{x}_i, i = 0, 1, \cdots, N$ and $\boldsymbol{u}_i, i = 0, 1, \cdots, N-1$. Each discretized time instant i corresponds to a multiple shooting *grid point*. The dynamic feasibility of the finite-dimensional approximation is ensured by introducing *gap-closing* constraints of the *explicit* form $\boldsymbol{x}_{i+1} = F(\boldsymbol{x}_i, \boldsymbol{u}_i)\ \forall i \in \{0, 1, \cdots, N-1\}$, where F numerically integrates system dynamics over time[1]. In this work, we use a 4th order Runge-Kutta (RK4) for the numerical integration of the system dynamics and cost functional. We choose multiple shooting as the transcription method (as opposed to e.g. collocation) because the FATROP optimizer expects the discretized dynamics to be written in the explicit form. From a successfully optimized trajectory, we recover the states and controls $(\boldsymbol{x}_i, \boldsymbol{u}_i)$ directly from the multiple shooting grid points, for $0 \leq i < N$. As for the initial guess provided to the NLP solver, we initialize all control variables $\boldsymbol{u}_i$ to 0, linearly interpolate the states $\boldsymbol{x}_i$ between $\boldsymbol{x}_0$ and $\boldsymbol{x}_f$, and set $t_f = (t_{\min} + t_{\max})/2$.

Dataset augmentation by goal relabeling To multiply the number of training samples obtained from the dataset generation process, we use a *goal relabeling* or *hindsight relabeling* technique (Andrychowicz et al., 2017). The core idea is that, if $\boldsymbol{x}_i$ is an intermediate state in a trajectory, then all previous state-control pairs $(\boldsymbol{x}_j, \boldsymbol{u}_j), j < i$ successfully control the system towards $\boldsymbol{x}_i$; thus, this *sub-trajectory* may serve as a demonstration for reaching $\boldsymbol{g}_i = \boldsymbol{y}(\boldsymbol{x}_i)$ as a goal. Hence, we augment the training dataset by taking each trajectory, and for each of its intermediate states $\boldsymbol{x}_i, 1 \leq i < N$ generating i new input-output pairs $((\boldsymbol{x}_j, \boldsymbol{y}(\boldsymbol{x}_i)), \boldsymbol{u}_j)$ for $0 \leq j < i$. This augmentation is not applied to the validation set.

Fast and parallel dataset generation Since each generated trajectory is independent of the rest, the dataset generation procedure can be easily parallelized. In addition, instead of using a general-purpose NLP solver such as IPOPT (Wächter and Biegler, 2006), we use the FATROP solver (Vanroye et al., 2023), which leverages the structure of NLPs resulting from the direct transcription of OCPs; this enables generating many more optimal trajectories in less time. We use FATROP through its CasADi interface (Andersson et al., 2019).

1. Since DMS requires system dynamics to define a well-posed initial value problem (IVP) that can be solved numerically, the dynamics function $\boldsymbol{f}$ is assumed to be locally Lipschitz in $\boldsymbol{x}(t)$ and continuously differentiable in $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$.

3.2. Policy training

After generating a large dataset of input-output pairs, we can proceed to train the policy $\boldsymbol{\pi}_\theta$ using supervised learning, thus doing behavior cloning (BC) on the optimal demonstrations[2]. Given that the inputs to the policies are low-dimensional state and goal representations, we define the policies as simple multi-layer perceptrons (MLPs). We favor MLP architectures that are smaller and thus faster for training and inference.

Having $\boldsymbol{\pi}_\theta(\boldsymbol{x}, \boldsymbol{g}) = \text{MLP}_\theta(\boldsymbol{w})$, where $\boldsymbol{w}$ is a vector representing the state-goal pair $(\boldsymbol{x}, \boldsymbol{g})$, the policy's output should be an n_u-dimensional vector (i.e. of the dimension of the controls). We thus simply train $\boldsymbol{\pi}_\theta$ to minimize the mean squared error $\text{MSE}(\boldsymbol{\pi}_{\boldsymbol{\theta}}, \mathcal{D}_{\text{train}}) = \mathbb{E}_{((\boldsymbol{x},\boldsymbol{g}),\boldsymbol{u})\sim\mathcal{D}_{\text{train}}}\left[|\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{g}) - \boldsymbol{u}|_2^2\right]$, that is, to minimize the deviation between the policy's outputs and the optimal controls obtained by solving the OCP 1. Our implementation of the policies and their training is based on the Equinox library (Kidger and Garcia, 2021) which is based on JAX (Bradbury et al., 2018).

3.3. Policy evaluation

Once a policy $\boldsymbol{\pi}_\theta$ has been trained by imitation learning on $\mathcal{D}_{\text{train}}$, we evaluate its success rate and efficiency when controlling the dynamical system. Tailor and Izzo (2019) found that a lower mean absolute error (MAE) in imitating the expert controller leads to more performant policies according to the OCP's cost functional; however, there comes a point where small gains in validation error no longer improve the efficiency of the policy. Thus, in this section we present our method to empirically evaluate a trained policy, which is heavily inspired by that of Sánchez-Sánchez and Izzo (2018).

Checking for successful goal reaching For a given goal $\boldsymbol{g}$, we simulate the dynamical system where the policy acts as a controller, i.e. $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x}(t), \boldsymbol{g}))$. In all experiments, we simulate the environment dynamics with $\boldsymbol{\pi}_\theta$ as a controller using Dormand-Prince's 7/8 method for numerical integration (Dormand and Prince, 1980), clipping the policy's output to enforce actuator limits. We define a function $\text{GOALREACHED} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_g} \to \{0, 1\}$, which indicates whether the input state reaches the input goal up to a sufficiently small tolerance. We can then use this function to decide whether a rollout was successful or not in reaching $\boldsymbol{g}$: given the state trajectory $\boldsymbol{x}(t)$ obtained in simulation, we take the *completion time* t^* as the earliest time in $[t_{\min}, t_{\max}]$ such that $\text{GOALREACHED}(\boldsymbol{x}(t^*), \boldsymbol{g}) = 1$; if no such time exists, then we take the trajectory as unsuccessful, that is, the policy was not able to reach the commanded goal.

Evaluating efficiency Assume that the policy successfully reached its goal at completion time t^*. To assess how efficiently it reached the goal with respect to the OCP's cost functional, we compare its achieved cost J (as defined by the cost functional of Eq. 1a) to the optimal cost J^* obtained through trajectory optimization for the same initial state-goal pair. Since the learned policy is generally suboptimal ($J^* < J$), we quantify its performance using the relative error $\delta_r = 100 \times \frac{J - J^*}{J^*}$, which expresses (as a percentage) how much higher the policy's cost is than the optimal one. A lower δ_r indicates a more near-optimal policy.

The evaluation procedure estimates the policy's success rate and efficiency over the full task distribution $\mathcal{T}$, by sampling initial state-goal pairs independently and computing the average success rate and relative cost error across tasks.

2. Any other imitation learning algorithm may be used, such as DAGGER. Also, the policy could be parameterized by a more expressive model such as a diffusion model (Chi et al., 2023; Reuss et al., 2023).

4. Experimental evaluation

To evaluate GCIMOPT, we perform experiments on four different continuous control tasks of varying complexity: a *cart-pole* system, a two-dimensional (planar) quadrotor, a three-dimensional quadrotor and a 6 degree-of-freedom (DoF) robot arm. To model and simulate these systems, for the first three we use the `safe-control-gym` library (Yuan et al., 2022), while for the robot arm we use the `urdf2casadi` library (Johannessen et al., 2019) to parse a dynamic model of the robot from its URDF file.

4.1. Dynamical systems and associated OCPs

Cart-pole The cart-pole consists of a cart moving along a frictionless axis with a freely hinged pole that must be balanced by applying horizontal forces to the cart. Its 4D state vector is $\boldsymbol{x} = [x, \dot{x}, \theta, \dot{\theta}]^\intercal$, describing the cart position in meters, velocity in meters per second, and the angle and angular velocity of the pole in radians and radians per second respectively. The system's nonlinear dynamics depend on the cart and pole masses $m_c = m_p = 1\,\mathrm{kg}$ and pole length $l = 0.5\,\mathrm{m}$. Initial states are sampled uniformly within the bounds $[[-3, -1, 0, -1]^\intercal, [3, 1, 2\pi, 1]^\intercal]$, while goals correspond to equilibrium points with $\dot{x} = \dot{\theta} = 0$ and $\theta \in \{0, \pi\}$. As a success indicator function for the evaluation, we require that all state variables approximate their goal values with corresponding tolerances, that is

$$\textsc{GoalReached}(\boldsymbol{x}, \boldsymbol{g}) = \bigwedge_{i=1}^{4} \Big(|\boldsymbol{x}_i - \boldsymbol{g}_i| \leq \varepsilon_i \Big), \tag{2}$$

where $\wedge$ denotes logical conjunction. We use tolerances of $\varepsilon_x = 0.05\,\mathrm{m}$, $\varepsilon_{\dot{x}} = 0.1\,\mathrm{m\,s^{-1}}$, $\varepsilon_\theta = 0.1\,\mathrm{rad}$ and $\varepsilon_{\dot{\theta}} = 0.1\,\mathrm{rad\,s^{-1}}$.

Planar quadrotor The second task considers controlling a planar (2D) quadrotor, modeled as a flat plate with one vertical thruster at each edge, so there are only two control inputs $\boldsymbol{u} = [T_1, T_2]^\intercal$. The system state is 6-dimensional, $\boldsymbol{x} = [x, \dot{x}, z, \dot{z}, \theta, \dot{\theta}]^\intercal$, consisting of the center of mass position and velocity, pitch angle θ, and pitch rate $\dot{\theta}$. The physical parameters model the Bitcraze Crazyflie 2.0. Initial positions are sampled uniformly from a $10\,\mathrm{m}$ side box, with lower and upper state bounds $[[-5, -5, -5, -5, -\pi, -1]^\intercal, [5, 5, 5, 5, \pi, 1]^\intercal]$, while goal states always correspond to a hover at a sampled position (x_g, z_g) with zero velocity and pitch. Goal achievement is defined by tolerances on position, velocity, pitch, and pitch rate: $\varepsilon_{\text{dist}} = 5\,\mathrm{cm}, \varepsilon_{\text{vel}} = 5\,\mathrm{cm\,s^{-1}}, \varepsilon_\theta = 0.1\,\mathrm{rad}, \varepsilon_{\dot{\theta}} = 0.1\,\mathrm{rad\,s^{-1}}$.

Three-dimensional quadrotor The third task extends the Crazyflie 2.0 quadrotor to its full three-dimensional model, with a 12-dimensional state $\boldsymbol{x} = [x, \dot{x}, y, \dot{y}, z, \dot{z}, \phi, \theta, \psi, p, q, r]^\intercal$, including 3D position, velocity, roll ϕ, pitch θ, yaw ψ and their body frame rates. Controls are 4-dimensional, corresponding to the quadrotor's motors. Optimal trajectories are generated with initial and goal positions sampled from a $4\,\mathrm{m}$ box and all non-position variables initialized to zero for hover, which improves solver convergence. Goal achievement is defined with the tolerances $\varepsilon_{\text{dist}} = 5\,\mathrm{cm}, \varepsilon_{\text{vel}} = 5\,\mathrm{cm\,s^{-1}}, \varepsilon_\theta = \varepsilon_\phi = \varepsilon_\psi = 0.1\,\mathrm{rad}, \varepsilon_p = \varepsilon_q = \varepsilon_r = 0.1\,\mathrm{rad\,s^{-1}}$.

Franka Emika Panda robot arm The final task considers controlling the Franka Emika Panda robot arm, which has 7 revolute joints. Only 6 of the joints (A1-A6) are controlled, since the last joint (A7) does not affect the end-effector (EE) position, making the state space 6-dimensional. The

control objective is to move the EE to a specified point in space, ignoring orientation. The EE position is obtained via forward kinematics: if $FK(\boldsymbol{x})$ yields the 3D position of the EE in configuration $\boldsymbol{x}$, the EE position (and thus the goal for the OCPs) is $\boldsymbol{y}(\boldsymbol{x}) = FK(\boldsymbol{x})$. Controls correspond to joint angular velocities, so the dynamics simplify to $\dot{\boldsymbol{x}}(t) = \boldsymbol{u}(t)$. Initial and goal joint angles are sampled uniformly within joint limits. We define goal reaching success by the end-effector being within $\varepsilon = 2\,\mathrm{cm}$ of the goal, i.e. $\textsc{GoalReached}(\boldsymbol{x}, \boldsymbol{g}) = |\boldsymbol{y}(\boldsymbol{x}) - \boldsymbol{g}|_2 \leq \varepsilon$.

For all systems, angles are encoded as sine-cosine pairs, and when the dynamics are invariant to absolute positions, policies receive the difference between goal and current positions rather than absolute positions. We use cost functionals that balance control effort and trajectory duration, by defining $L(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \alpha|\boldsymbol{u}(t)|_2^2$ and $L_f(\boldsymbol{x}(t_f), t_f) = (1-\alpha)t_f$ in Eq. 1a. Thus, the cost functionals that we minimize in the generated datasets take the form

$$\alpha \int_0^{t_f} |\boldsymbol{u}(t)|_2^2 \mathrm{d}t + (1-\alpha)t_f, \tag{3}$$

where $\alpha \in [0, 1]$. Setting $\alpha = 0$ minimizes the time to reach the goal, $\alpha = 1$ minimizes control effort, and intermediate values interpolate between these objectives. See Appendix A.1 for details on the dataset generation parameters.

4.2. Policy training

We train MLP policies of varying sizes for each task: 3 hidden layers for cart-pole, planar quadrotor, and robot arm, and 5 layers for the 3D quadrotor due to its higher state dimensionality. We test hidden layer sizes of 32, 64, and 128, using the Swish activation (Ramachandran et al., 2017) for smooth, continuous outputs. Policies are trained using the Adam optimizer (Kingma and Ba, 2014) with task-specific learning rates. 90% of the trajectories are used for training and the other 10% for validation; we do not use a separate testing or evaluation set, since our final performance metric of interest is not regression error but closed-loop success rate and optimality gap. We only apply goal relabeling to the training split, such that for each trajectory of N grid points, we obtain $N(N+1)/2$ initial/goal state pairs. For the 3D quadrotor, early stopping based on success rate determined the number of epochs. We provide details on policy training in Appendix A.2.

4.3. Results

We provide the results of the experimental evaluation on closed-loop control tasks in Table 1.

The results indicate that relatively small MLP policies can achieve high success rates and competitive efficiencies without extensive hyperparameter tuning. However, there is not always a direct correlation between validation error and policy performance: in some cases, networks with higher regression errors perform better in terms of success rate or optimality gap. This can be explained by the mismatch between the states covered by the training dataset and those visited by the policy, which appears when doing simple behavioral cloning like in our experiments. For the cart-pole, the 128-unit MLP achieves the lowest validation error, but the 64-unit and 128-unit MLPs attain the highest average success rates, while the 32-unit MLP performs best in terms of optimality gap. In the planar quadrotor task, all architectures achieve similar success rates, and lower validation errors correspond to smaller relative costs.

For the robot arm, success rates are consistently high, but the cost relative error decreases with larger MLPs until they become negative. Even though $\delta_r < 0$ would imply that the trained policy is

Table 1: Final validation error J_{val}, success rate s and average cost relative error δ_r for all tasks. Results are shown as mean $\pm$ standard deviation over 5 random seeds (see detailed results in Appendix A.4). Lower J_{val} and δ_r are better ($\downarrow$); higher s is better ($\uparrow$).

Task	MLP hidden layer size	J_{val} ($\downarrow$)	s ($\uparrow$)	δ_r ($\downarrow$)
Cart-pole	32	0.07±0.005	89.75±5.51%	21.082±3.429%
	64	0.0446±0.0037	94.83±2.17%	27.252±3.444%
	128	0.0321±0.0016	94.29±2.21%	29.896±12.475%
Planar quadrotor	32	0.0343±0.0015	99.77±0.19%	16.888±0.932%
	64	0.0146±0.0012	99.86±0.08%	7.973±0.581%
	128	0.0078±0.0009	99.77±0.08%	5.282±0.838%
3D quadrotor	32	0.1293±0.0688	28.04±18.54%	207.432±22.521%
	64	0.0765±0.0699	69.83±25.55%	121.453±15.648%
	128	0.0336±0.0094	97.8±2.13%	60.145±16.259%
Robot arm reaching	32	2.6586±0.0416	94.76±0.64%	19.698±1.458%
	64	2.2203±0.0398	98.48±0.13%	-0.297±1.39%
	128	2.3748±0.0331	98.71±0.21%	-0.198±0.768%

more near-optimal than the optimal policy, we observed that it is due to the non-zero tolerance of GOALREACHED: when the trained policy very closely approximates the optimal control, it drives the end effector near-optimally to $\leq 2\,\text{cm}$ of the goal, whereas the optimal control reaches the goal point exactly. This goal-reaching tolerance results in the trajectory completion time (as defined in Section 3.3) being slightly earlier than that of the optimal trajectory, leading to a lower trajectory cost. In effect, this phenomenon indicates that these robot arm policies very closely approximate the optimal controller.

The 3D quadrotor stabilization task is notably more challenging: smaller networks failed to achieve acceptable performance, necessitating 5-layer MLPs. Wider networks improve success rates, yet relative cost errors remain high (over 60% on average), indicating limited policy efficiency. Furthermore, training exhibits a counterintuitive phenomenon: as regression error decreases during the training process, success initially rises but then sharply drops. This is likely due to the sensitivity of quadrotor dynamics, where a closer approximation of optimal controls results in aggressive maneuvers that can destabilize the system; these destabilizations could be mitigated by covering more "unsafe" states in the training dataset to improve the policy's ability to correct deviations. Overall, these findings suggest that careful task-specific tuning, more varied datasets that better cover the distribution of states visited by the policy, and incorporating domain knowledge (particularly for challenging tasks like 3D quadrotor control) would likely improve both success rates and efficiency, consistent with successful real-world deployments of similar neural network policies (Ferede et al., 2024).

In order to assess the gains in computational efficiency achieved by GCIMOPT policies with respect to a fast trajectory optimization solver, we compare the policies' inference time and the solve time of FATROP. We show the measured times and the speedups of GCIMOPT with respect to FATROP in Table 2. See Appendix A.3 for details.

Table 2: Comparison of average running time between the FATROP optimizer (T_{FATROP}) and GCIMOPT policies of u units per hidden layer (T_u), all in milliseconds. We also report the speedup $S_u = T_{\text{FATROP}}/T_u$.

Task	T_{FATROP}	T_{32}	T_{64}	T_{128}	S_{32}	S_{64}	S_{128}
Cart-pole	7.530	0.012	0.011	0.013	619.857	669.991	588.896
Planar quadrotor	32.810	0.013	0.012	0.014	2621.197	2724.668	2393.192
3D quadrotor	84.018	0.013	0.014	0.015	6278.703	6188.510	5578.079
Robot arm reaching	4.671	0.042	0.043	0.048	109.951	107.418	97.844

As shown by the comparison in inference time, GCIMOPT policies can obtain very large speedups with respect to the fast FATROP solver. The speedups are largest in the case of the 3D quadrotor, since the high dimensionality and nonlinearity of the dynamical system makes trajectory optimization more computationally expensive. Given that our policies are parameterized by MLPs, they have a constant computation latency; in contrast, the execution time of an iterative trajectory optimization solver can vary depending on the initial state and goal. The obtained speedups, together with the small size of the trained MLPs, suggest that GCIMOPT policies could be deployed on resource-constrained controllers and achieve control frequencies rivaling or surpassing those obtained by MPC controllers.

5. Conclusion

In this work, we demonstrate that small neural networks trained via behavioral cloning on datasets of optimal trajectories can achieve high success rates and near-optimal goal-conditioned control across multiple tasks. While performance generally improves with model capacity, success rates and optimality gaps do not always correlate directly with regression error. The three-dimensional quadrotor task was the most challenging, requiring larger networks and careful tuning due to its high dimensionality and sensitivity to control errors. In contrast, lower-dimensional tasks such as cart-pole and planar quadrotor control were solved effectively even with small MLPs. Besides achieving good closed-loop performance, GCIMOPT policies yield considerable reductions in control latency (up to more than $6000\times$) with respect to a fast trajectory optimization solver such as FATROP.

Our approach is computationally feasible, with dataset generation and policy training achievable in under an hour on standard hardware, and readily parallelizable to scale to larger datasets. GCIMOPT relies on having access to an accurate system model and full observability of system states; although these enable generating high-quality expert data for policy training, they can also result limiting in more complex applications. Future work could address these limitations by exploring alternative policy architectures, applying the method to tasks with partial observability and sensor/actuator noise, deploying GCIMOPT policies on real hardware, and evaluating the method on more control tasks. In addition, although our OCP formulation can include expressive path constraints by Eq. 1d, in this work we only incorporate actuator limits and simple state bounds (such as the joint limits of the robot arm); training constrained goal-conditioned policies is an important direction for future work. Overall, our results show that imitating optimal trajectories is an effective method for training goal-conditioned policies that are near-optimal and computationally efficient.

Acknowledgments

This research was partially funded by the European Research Council under grant agreement No 101042702 Intevol-ERC2021-STG. We acknowledge support by the UpnaLab research group of the Public University of Navarre (UPNA). This work is based on Jon Goikoetxea's Bachelor's thesis (Goikoetxea, 2025) defended in UPNA: we thank Patricia Yanguas, José Antonio Sanz, Daniel Aláez and Mikel Martínez-Goikoetxea for their valuable feedback during the development of the thesis. We also thank the anonymous L4DC 2026 reviewers for their insightful feedback and suggestions.

References

Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11:1–36, 2019.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

John T. Betts. Survey of Numerical Methods for Trajectory Optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, March 1998. ISSN 0731-5090, 1533-3884. doi: 10.2514/2.4231. URL https://arc.aiaa.org/doi/10.2514/2.4231.

John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984. Publisher: Elsevier.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4693–4700. IEEE, 2018.

Xingye Da and Jessy Grizzle. Combining trajectory optimization, supervised machine learning, and model structure for mitigating the curse of dimensionality in the control of bipedal robots. *The International Journal of Robotics Research*, 38(9):1063–1097, 2019. Publisher: SAGE Publications Sage UK: London, England.

Moritz Diehl and Sébastien Gros. Numerical optimal control. *Optimization in Engineering Center (OPTEC)*, 258, 2011.

Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. *Advances in neural information processing systems*, 32, 2019.

John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

Robin Ferede, Guido de Croon, Christophe De Wagter, and Dario Izzo. End-to-end neural network based optimal quadcopter control. *Robotics and Autonomous Systems*, 172:104588, 2024.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*, 2019.

Jon Goikoetxea. Learning efficient goal-conditioned policies by imitating optimal trajectories. 2025. URL https://academica-e.unavarra.es/handle/2454/55206. Publisher: Public University of Navarre: Bachelor's theses repository.

Lill Maria Gjerde Johannessen, Mathias Hauan Arbo, and Jan Tommy Gravdahl. Robot Dynamics with URDF & CasADi. In *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)*. IEEE, 2019.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, volume 2, pages 1094–8. Citeseer, 1993.

Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3342–3349. IEEE, 2017.

Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *arXiv preprint arXiv:2111.00254*, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-conditioned reinforcement learning: Problems and solutions. *arXiv preprint arXiv:2201.08299*, 2022.

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR, 2020.

Igor Mordatch and Emo Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems*, volume 4, page 23, 2014.

Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. New York: John Wiley and Sons, 1962.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 7(1):5, 2017.

James Blake Rawlings, David Q Mayne, Moritz Diehl, and others. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI, 2017.

Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. *arXiv preprint arXiv:2304.02532*, 2023.

Stéphane Ross, Geoffrey Gordon, and Drew 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*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

Ari Rubinsztejn, Rohan Sood, and Frank E Laipert. Neural network optimal control in astrodynamics: Application to the missed thrust problem. *Acta astronautica*, 176:192–203, 2020.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

Dhruv Shah, Ajay Sridhar, Arjun Bhorkar, Noriaki Hirose, and Sergey Levine. Gnm: A general navigation model to drive any robot. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7226–7233. IEEE, 2023.

Christopher Iliffe Sprague, Dario Izzo, and Petter Ögren. Learning dynamic-objective policies from a class of optimal trajectories. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 597–602. IEEE, 2020.

Carlos Sánchez-Sánchez and Dario Izzo. Real-Time Optimal Control via Deep Neural Networks: Study on Landing Problems. *Journal of Guidance, Control, and Dynamics*, 41(5):1122–1135, May 2018. ISSN 0731-5090, 1533-3884. doi: 10.2514/1.G002357. URL https://arc.aiaa.org/doi/10.2514/1.G002357.

Carlos Sánchez-Sánchez, Dario Izzo, and Daniel Hennes. Optimal real-time landing using deep networks. In *Proceedings of the sixth international conference on astrodynamics tools and techniques, ICATT*, volume 12, pages 2493–2537, 2016.

Dharmesh Tailor and Dario Izzo. Learning the optimal state-feedback via supervised imitation learning. *Astrodynamics*, 3(4):361–374, 2019.

Lander Vanroye, Ajay Sathya, Joris De Schutter, and Wilm Decré. FATROP: A fast constrained optimal control problem solver for robot trajectory optimization and control. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10036–10043. IEEE, 2023.

Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.

Zhaocong Yuan, Adam W. Hall, Siqi Zhou, Lukas Brunke, Melissa Greeff, Jacopo Panerati, and Angela P. Schoellig. Safe-Control-Gym: A Unified Benchmark Suite for Safe Learning-Based Control and Reinforcement Learning in Robotics. *IEEE Robotics and Automation Letters*, 7(4): 11142–11149, 2022. doi: 10.1109/LRA.2022.3196132.

Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 528–535. IEEE, 2016.

Appendix A. Experiment details

All experiments, from dataset generation to policy training and evaluation, are performed on a laptop with a 13th Gen Intel i9 CPU (24 cores, 0.8-5.6 GHz) and an RTX 4090 Laptop GPU with 16 GB VRAM.

A.1. Dataset generation

Dataset generation is fast even with an unoptimized implementation, taking less than 12 minutes on all tasks; generation could be accelerated further with JIT compilation, which was applied only for the 3D quadrotor. We provide in Table 3 the dataset generation hyperparameters per task: the number of optimized trajectories and multiple shooting grid points, the parameter α of the cost functional defined in Eq. 3 and the number of processes used for parallel generation. We also report the wall-clock duration of dataset generation per task.

Table 3: Parameters and duration of the dataset generation process for each task: number of trajectories N_T, number of multiple shooting grid points N, parameter α of cost functional, number of parallel processes P. Wall clock duration is shown as minutes:seconds.

Task	N_T	N	α	P	Wall-clock duration
Cart-pole	20 000	35	0.05	5	03:08
Planar quadrotor	20 000	40	1	20	05:27
Three-dimensional quadrotor	20 000	40	1	20	11:23
Robot arm reaching	20 000	35	0.1	20	00:19

A.2. Policy training and closed-loop evaluation

Training each MLP policy, including periodic evaluation on 2000 fixed initial state-goal pairs, took about 30 minutes. The 2000 optimal trajectories used for evaluation were generated using $N = 50$ direct multiple shooting grid points to minimize discretization error. Table 4 details the hyperparameters related to policy training, namely the minibatch size and learning rate, the number of epochs and the size of the training and validation datasets.

A.3. Inference time comparison

In order to obtain the results of Table 2, we take T_u as the average CPU inference time over 10^6 policy forward passes. We measure trajectory optimization solve time T_{FATROP} as the average over the 2000 initial-state/goal times sampled from the task distribution $\mathcal{T}$, with the same number of multiple shooting grid points as used for dataset generation (Table 3). We always enable JIT compilation when measuring T_{FATROP} so that the solver is compiled to an efficient C language routine with the GCC flags `-O7 -march=native`, maximizing performance as would be done in an MPC application.

Table 4: Summary of training hyperparameters for each task: minibatch size m, Adam learning rate η, number of training epochs N_{epochs}, size of training and validation splits $|\mathcal{D}_{\text{train}}|$ and $|\mathcal{D}_{\text{val}}|$. Note that goal relabeling is applied to the training split trajectories, yielding a large number of samples $|\mathcal{D}_{\text{train}}|$.

Task	m	η	N_{epochs}	$\lvert\mathcal{D}_{\text{train}}\rvert$	$\lvert\mathcal{D}_{\text{val}}\rvert$
Cart-pole	8192	3×10^{-4}	300	11 340 000	70 000
Planar quadrotor	8192	1×10^{-4}	200	14 760 000	80 000
Three-dimensional quadrotor	65536	5×10^{-4}	Early stopping	14 760 000	80 000
Robot arm reaching	8192	5×10^{-4}	100	11 340 000	70 000

A.4. Detailed closed-loop evaluation results

Table 5 shows the results of closed-loop evaluation for 5 random seeds for each task/MLP size combination. The random seeds are used to initialize MLP parameters and for stochastic minibatch training. For the 3D quadrotor experiment, we select for each seed the checkpoint with the highest success rate and report its performance: we thus effectively implement early stopping since performance degrades when training beyond this point, as discussed in Section 4.3.

Table 5: Final validation error J_{val}, success rate s and average cost relative error δ_r for all tasks, for 5 random seeds for each task/MLP hidden layer size (d_{MLP}) combination.

Task	d_{MLP}	J_{val} (↓)	s (↑)	δ_r (↓)
Cart-pole	32	0.06534	94.4%	19.175%
		0.06638	95.85%	22.024%
		0.07293	84.8%	16.491%
		0.06809	90.1%	22.132%
		0.0774	83.6%	25.59%
	64	0.04246	96.2%	29.945%
		0.04002	92.6%	29.642%
		0.04448	96.2%	25.795%
		0.04667	96.8%	21.849%
		0.04961	92.35%	29.029%
	128	0.0322	90.55%	34.814%
		0.03043	95.5%	12.976%
		0.03172	96.1%	46.777%
		0.03165	94.1%	30.205%
		0.03475	95.2%	24.706%
Planar quadrotor	32	0.03556	99.9%	16.378%
		0.03197	99.9%	16.259%
		0.03519	99.45%	16.011%
		0.03362	99.85%	17.99%
		0.03494	99.75%	17.8%
	64	0.01581	99.9%	8.1%
		0.01541	99.9%	8.078%
		0.01505	99.75%	7.55%
		0.01292	99.95%	7.319%
		0.01364	99.8%	8.82%
	128	0.00881	99.7%	5.206%
		0.0085	99.7%	5.223%
		0.00801	99.75%	6.669%
		0.00697	99.9%	4.869%
		0.00693	99.8%	4.444%

Task	d_{MLP}	J_{val} (↓)	s (↑)	δ_r (↓)
3D quadrotor	32	0.1361	54.1%	213.464%
		0.13126	8.85%	198.342%
		0.22958	23.65%	237.89%
		0.03699	14.45%	211.151%
		0.11245	39.15%	176.314%
	64	0.1937	93.35%	116.171%
		0.06484	79.45%	102.798%
		0.01682	26.3%	144.431%
		0.03046	72.1%	128.06%
		0.07687	77.95%	115.803%
	128	0.04012	99.35%	50.075%
		0.01858	94.25%	86.329%
		0.03586	99.0%	50.372%
		0.03128	97.35%	65.773%
		0.04233	99.05%	48.176%
Robot arm reaching	32	2.665	95.15%	18.062%
		2.63001	93.65%	19.214%
		2.72684	95.2%	20.604%
		2.62262	94.8%	18.885%
		2.64873	95.0%	21.725%
	64	2.236	98.4%	-0.84%
		2.16017	98.4%	-1.834%
		2.23723	98.5%	-0.043%
		2.20416	98.7%	-0.675%
		2.26407	98.4%	1.908%
	128	2.364	98.7%	-0.336%
		2.35559	99.05%	0.699%
		2.40519	98.7%	-0.636%
		2.33619	98.6%	0.446%
		2.41328	98.5%	-1.162%