JOINTLY LEARNING IDENTIFICATION AND CONTROL FOR FEW-SHOT POLICY ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Complex dynamical systems are challenging to model and control. Especially when not deployed in controlled conditions, they might be subject to disturbances that cannot be predicted in advance, *e.g.* wind, a payload, or environment-specific forces. Adapting to such disturbances with a limited sample budget is difficult, especially for systems with many degrees of freedom. This paper introduces a theoretical framework to model this problem. We show that the expected error of a sensorimotor controller can be bounded by two components: the optimality of the controller and the domain gap between training and testing due to unmodelled dynamic effects. These components are usually minimized separately; the former with online or offline optimization, the latter with system identification. Motivated by this observation, we propose a differentiable programming approach to *jointly* minimize model and control errors with gradient descent. Similar to model-based methods, our algorithm learns from prior knowledge about the system, but *grounds* the model to account for observed disturbances, thereby favouring sample efficiency. Yet, it maintains the flexibility of model-free methods, which can be applied to generic systems with arbitrary inputs. We evaluate our approach on several complex systems and tasks, and experimentally analyze the advantages over model-free and model-based methods in terms of performance and sample efficiency.

1 INTRODUCTION

Consider the following scenario: a complex dynamical system, *e.g.* a drone or quadruped, is deployed outside of controlled laboratory conditions. To optimally function, this robot has to quickly adapt to the conditions it encounters in the target domain and cannot anticipate, *e.g.* a payload, wind, contact with unknown surfaces, or other environment-specific disturbances. Some of these disturbances may be hard to measure but their effect may be easy to observe from high-dimensional data such as images. In order to adapt a control policy to the target domain, expressive models that leverage measurements and environment observations are required to account for these effects. But how can this be done efficiently, with only a restricted budget of samples? In principle it is possible to completely identify a system's dynamics through extensive experiments. But, such experiments are both tedious and expensive to run in practice. This may not be feasible for systems with many degrees of freedom and too dangerous for safety critical platforms such as flying robots (Zhang et al., 2021).

Variations of this few-shot adaptation problem can be found in multiple branches of machine learning and robotics, including few shot learning (Wang et al., 2020), simulation grounding (Hanna & Stone, 2017; Chebotar et al., 2019), and simulation to reality transfer (Höfer et al., 2021; Loquercio et al., 2019). In this paper, we study few-shot domain adaptation in the context of sensorimotor control. The traditional approach for addressing this problem is to first identify a model of the system and then use it in the downstream control task. The identification step consists of either adapting an existing model from observations (Hanna & Stone, 2017; Chebotar et al., 2019; Farchy et al., 2013; Desai et al., 2020; Fu et al., 2016) or learning it end-to-end from data (Jatavallabhula et al., 2021; Deng et al., 2020; Lenz et al., 2015; Bansal et al., 2016; Jin et al., 2020). After the identification step, control commands are computed with either online optimization, *e.g.* with model predictive control (Fu et al., 2016) or linear quadratic regulators (Deng et al., 2020; Bansal et al., 2016), or offline optimization, with supervised (Desai et al., 2020), self-supervised (Jatavallabhula et al., 2021; Lenz et al., 2015; Jin et al., 2020), or reinforcement learning (RL) (Chebotar et al., 2019; Nagabandi et al., 2018). Despite achieving good performance on multiple demonstrators and on physical platforms, the main

limitation of the aforementioned methods is the separation of the identification and control step. This separation can be particularly problematic for complex systems subject to time-varying forces, *e.g.* contact or wind, where a large number of training samples might be required to learn sufficiently accurate models. In addition, it remains unclear whether an accurate model is needed in the states that are not observed during task execution. To account for this limitation, model-based RL approaches learn a model of the system and a control policy jointly (Heess et al., 2015; Fu et al., 2016; Hafner et al., 2019a;b; Nagabandi et al., 2018). Motivated by the sample efficiency of model-based RL, we extend this idea and design a specialized algorithm that can adapt a sensorimotor policy to a target domain in a sample efficient way.

We present a theoretical framework to rigorously analyze the underlying adaption problem. In light of this framework, we lay out a taxonomy of existing approaches and show that prior works either optimize the objective indirectly or minimize an upper bound. We propose a fully differentiable and sample-efficient framework that directly minimizes the adaptation error. Akin to model-based RL, we learn a model of the system and a controller jointly. However, in contrast to traditional model-based RL, we optimize the model and control error via gradient descent: we adapt prior knowledge about the system dynamics from interactions with the target domain; and we use self-supervised backpropagation-through-time (Grzeszczuk et al., 1998; Deisenroth & Rasmussen, 2011; Parmas, 2018; Degrave et al., 2019; Clavera et al., 2018; Jin et al., 2020) to tune the controller to the new domain. These two design decisions allow to both handle arbitrary changes in the dynamics and minimize the number of samples that are required in the target domain. In our framework both the controller and the simulator can learn from arbitrary inputs, for example solely from visual observations.

The proposed framework is agnostic to the system and task. We empirically validate it on several problems: (i) balancing an inverted pendulum on a cart (standard task in RL known as CartPole), (ii) navigating towards a 3D target with a fixed-wing aircraft (Oettershagen et al., 2014), and (iii) tracking a trajectory with a quadrotor (Kaufmann et al., 2020). In all of these problems, the system is subject to either time-invariant or time-variant disturbances that are only present in the target domain and should be identified and counteracted with as few samples as possible. We demonstrate that these disturbances can be identified from arbitrary inputs (low-level states, images, and state-action history) and that the resulting differential simulator can successfully adapt the controller to the target domain in a self-supervised fashion. We empirically show comparable or better results with respect to prior works in terms of performance on the downstream task and sample complexity in the target domain.

2 Theoretical Derivation and Proposed Approach

In the following, we formulate the problem of adapting a policy $\pi : S \times S \longrightarrow A$ from a source domain with dynamics $\hat{f} : S \times A \longrightarrow S$ to a target domain with dynamics f^* with a possibly different state space. We assume that the source dynamics $\hat{f}$ are differentiable and known *a priori*: this encode prior knowledge about the system. Conversely, the target dynamics f^* are unknown and possibly discontinuous. We also assume that unlimited samples can be drawn from the source domain (which is practically a simulator of the system), but only a limited sample budget is available from the target domain. We define the task as a discrete time, continuous-valued optimization problem:

$$\min_{\pi} J^*(\pi) = \mathbb{E}_{\rho(\pi)}\Big[\sum_{t=0}^{t=T} C(f^*\big(s_t, \pi(s_t, \nu_{t+1})\big), \nu_{t+1})\Big], \tag{1}$$

where C is a cost depending on a reference $\nu_{t+1} \in S$ and the robot state $s_t \in S$; $\rho(\pi)$ is the distribution of possible state-reference pairs $\{(s_0, \nu_0), \ldots, (s_T, \nu_T)\}$ induced by the policy π. The robot state is initialized to the provided reference ($s_0 = \nu_0$). As typical in the control literature (Borrelli et al., 2017), we define C as the ℓ^p-norm of the difference between the next state and the reference, *i.e.* $C(f^*\big(s_t, \pi(s_t, \nu_{t+1})\big), \nu_{t+1}) = \|f^*\big(s_t, \pi(s_t, \nu_{t+1})\big) - \nu_{t+1}\|_p$. For the sake of simplicity, we drop the norm factor p in the rest of the derivation. Similarly, we disregard the control cost $\|\pi(s_t, v_t)\|$ which is often added to C, because it does not affect the following derivations.

2.1 BOUNDING THE ADAPTATION ERROR $J^*(\pi)$

We now derive an upper bound on the performance of the controller in the target domain $J^*(\pi)$. First, we decompose $J^*(\pi)$ over time using the linearity of expectation:

$$J^*(\pi) = \mathbb{E}_{\rho(\pi)}\Big[\sum_{t=0}^{t=T} \|f^*\big(s_t, \pi(s_t, \nu_{t+1})\big) - \nu_{t+1}\|\Big] = \sum_{t=0}^{t=T} \mathbb{E}_{(s_t,\nu_{t+1})}\|f^*\big(s_t, \pi(s_t, \nu_{t+1})\big) - \nu_{t+1}\| \tag{2}$$

In the following, we derive a bound for the rightmost term of Equation equation 2 component-wise. Here and in the rest of the paper, we drop the time argument for the sake of simplicity of notation:

$$\begin{aligned}\mathbb{E}_{s,\nu}\|f^*\big(s, \pi(s,\nu)\big) - \nu\| &= \mathbb{E}_{s,\nu}\|f^*\big(s, \pi(s,\nu)\big) - \hat{f}\big(s, \pi(s,\nu)\big) + \hat{f}\big(s, \pi(s,\nu)\big) - \nu\| \\ &\leq \mathbb{E}_{s,\nu}\|f^*\big(s, \pi(s,\nu)\big) - \hat{f}\big(s, \pi(s,\nu)\big)\| + \mathbb{E}_{s,\nu}\|\hat{f}\big(s, \pi(s,\nu)\big) - \nu\| \end{aligned} \tag{3}$$

We define $\hat{\pi}$ as the optimal controller in the source domain with dynamics $\hat{f}$, *i.e.*,

$$\hat{\pi}(s, v) := \arg\min_{\pi} \mathbb{E}_{s,\nu}\|\hat{f}(s, \pi(s,\nu)) - \nu\|. \tag{4}$$

It is important to note that the reference ν might be infeasible for the dynamics $\hat{f}$. Thus, even the optimal controller can be subject to an error $\zeta_{s,\nu}$:

$$\forall s, \nu \in S \;\; \exists\, \zeta_{s,\nu} \geq 0 : \;\; \|\hat{f}(s, a) - \nu\| \leq \zeta_{s,\nu} \;\; \forall a \in A. \tag{5}$$

We further define the expected difference between the predictions of the source and target dynamics $\hat{f}$ and f^* during execution of a policy π as $\delta_{s,\nu,\pi}$. Specifically,

$$\delta_{s,\nu,\pi} := \mathbb{E}_{s,\nu}\|\hat{f}(s, \pi(s,\nu)) - f^*(s, \pi(s,\nu))\|. \tag{6}$$

We now derive a bound for the error of a policy π that was trained in the source domain and is then applied to the target domain.

Theorem 1. *Let the source dynamics $\hat{f}$ be Lipschitz-continuous with factor L in the set of observed states,* i.e.

$$\|\hat{f}(s, a_1) - \hat{f}(s, a_2)\| \leq L \cdot \|a_1 - a_2\| \tag{7}$$

Then with $\hat{\pi}$, $\zeta_{s,\nu}$, and $\delta_{s,\nu,\pi}$ as defined above, it holds that

$$\mathbb{E}_{s,\nu}\|f^*\big(s, \pi(s,\nu)\big) - \nu\| \leq \delta_{s,\nu,\pi} + L \cdot \mathbb{E}_{s,\nu}\|\pi(s,\nu) - \hat{\pi}(s,\nu)\| + \mathbb{E}_{s,\nu}[\zeta_{s,v}]. \tag{8}$$

Proof.

$$\begin{aligned}\mathbb{E}_{s,\nu}\|f^*\big(s, \pi(s,\nu)\big) - \nu\| &\underset{(3)}{\leq} \mathbb{E}_{s,\nu}\|f^*\big(s, \pi(s,\nu)\big) - \hat{f}\big(s, \pi(s,\nu)\big)\| + \mathbb{E}_{s,\nu}\|\hat{f}\big(s, \pi(s,\nu)\big) - \nu\| \\ &\underset{(6)}{=} \delta_{s,\nu,\pi} + \mathbb{E}_{s,\nu}\|\hat{f}\big(s, \pi(s,\nu)\big) - \hat{f}\big(s, \hat{\pi}(s,\nu)\big) + \hat{f}\big(s, \hat{\pi}(s,\nu)\big) - \nu\| \\ &\leq \delta_{s,\nu,\pi} + \mathbb{E}_{s,\nu}\|\hat{f}\big(s, \pi(s,\nu)\big) - \hat{f}\big(s, \hat{\pi}(s,\nu)\big)\| + \mathbb{E}_{s,\nu}\|\hat{f}\big(s, \hat{\pi}(s,\nu)\big) - \nu\| \\ &\underset{(5,7)}{\leq} \delta_{s,\nu,\pi} + L \cdot \mathbb{E}_{s,\nu}\|\pi(s,\nu) - \hat{\pi}(s,\nu)\| + \mathbb{E}_{s,\nu}[\zeta_{s,v}] \end{aligned}$$

□

Theorem 1 shows that the controller performance $J^*(\pi)$ in the target domain is upper bounded by two terms: the dynamics mismatch between source and target domain $\delta_{s,\nu,\pi}$ and the difference between the actions taken by π and the actions taken by the optimal controller $\hat{\pi}$ *in the source domain*, *i.e.* the optimality cost of π. The rightmost term of Equation equation 8, $\mathbb{E}_{s,\nu}[\zeta_{s,v}]$, is constant for fixed dynamics $\hat{f}$, and represents the unavoidable error due to the reference infeasibility. Therefore, optimizing $J^*(\pi)$, entails minimizing the first two terms of Equation equation 8. Based on the insights of Theorem 1, we lay out a taxonomy of prior works by analyzing the terms of Equation equation 8 targeted by each method.

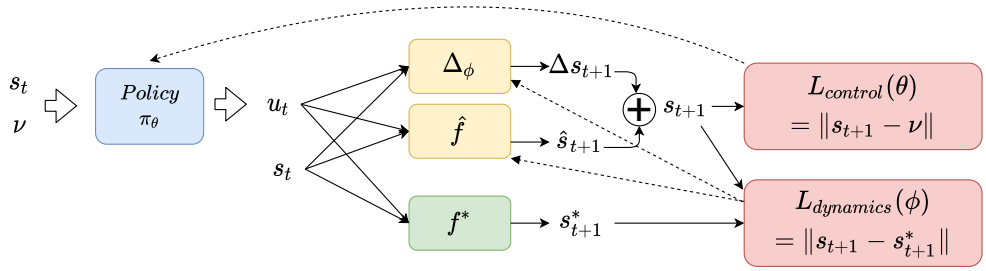


Figure 1: Framework for joint dynamics grounding and policy learning. The parameters ϕ of the residual Δ_ϕ are trained to account for the distance between $\hat{f}$ and f^*. The policy is then tuned on the updated dynamics in a self-supervised fashion.

Observation 1. *$\mathbb{E}_{s,\nu}\|f^*\big(s, \pi(s,\nu)\big) - \nu\|$ can be optimized directly with respect to π via Monte-Carlo estimates of the gradient. This does not require f^* to be differentiable and enables long-term planning, but requires extensive sampling in the target domain. This approach is used by both model-free RL (Schulman et al., 2017) and some specific instances of model-based RL (Chen et al., 2018; Heess et al., 2015; Hafner et al., 2019a;b).*

Note that it is possible to first identify f^*, *i.e.* to minimize $\delta_{s,\nu,\pi}$ from observations in the target domain, and then to train the policy by minimizing $\mathbb{E}_{s,\nu}\|\hat{f}\big(s, \pi(s,\nu)\big) - \nu\|$ via sampling. However, this separation might be sub-optimal, since it removes the coupling between the two error terms.

Observation 2. *Many model-based RL algorithms (Fu et al., 2016; Deng et al., 2020; Bansal et al., 2016; Nagabandi et al., 2018) optimize the nonlinear controller cost $\mathbb{E}_{s,\nu}\|\hat{f}\big(s, a\big) - \nu\|$ online with respect to actions a,* e.g. *via iLQR or MPC.*

In this case adaptation is also addressed by minimising the identification cost $\delta_{s,\nu,\pi}$. This can be done either separately from the control cost (Deng et al., 2020; Bansal et al., 2016) or jointly (Fu et al., 2016). One drawback for these methods is that actions are selected through online optimization, which could be both prohibitively slow and difficult to solve for non-convex dynamics depending on high-dimensional observations.

Observation 3. *Imitation learning methods solve the control task by imitating an expert π_{exp} and minimising $\mathbb{E}_{s,\nu}\|\pi(s,\nu) - \pi_{exp}(s,\nu)\|$ with gradient descent and data-set aggregation techniques (Ross et al., 2011; Kaufmann et al., 2020; Pan et al., 2018). These methods minimize an upper bound of $J^*(\pi)$, since*

$$\begin{aligned}
&\mathbb{E}_{s,\nu}\|f^*\big(s, \pi(s,\nu)\big) - \nu\| \\
&\leq \delta_{s,\nu,\pi} + L \cdot \mathbb{E}_{s,\nu}\|\pi(s,\nu) - \hat{\pi}(s,\nu)\| + \mathbb{E}_{s,\nu}\,\zeta_{s,v} \\
&\leq \delta_{s,\nu,\pi} + L \cdot \mathbb{E}_{s,\nu}\|\pi(s,\nu) - \pi_{exp}(s,\nu)\| + L \cdot \mathbb{E}_{s,\nu}\|\pi_{exp}(s,\nu) - \hat{\pi}(s,\nu)\| + \mathbb{E}_{s,\nu}\,\zeta_{s,v}
\end{aligned}$$

Thus, the adaptation problem is optimized independently from the imitation problem, leading to high sample complexity in the target domain. In summary, prior work minimizes an upper-bound or an approximation of the control cost, and generally optimizes the identification problem separately. Motivated by this observation, we propose to jointly optimize the components of equation 3.

2.2 Gradient-based joint dynamics grounding and policy learning

We propose a fully differentiable framework to adapt a policy π_θ from a source to a target domain (see Figure 1). The initial policy can be trained with any algorithm, since we have an infinite sample budget with the source dynamics. According to Equation equation 3, the adaptation process entails optimizing two components: an identification and control cost. We jointly optimize these two components in an alternating process: $\hat{f}$ is trained to minimize the distance to f^* from samples collected during the execution of the current policy π_θ in the target domain, and then π_θ is tuned on the updated $\hat{f}$. We repeat this process for $\lceil N/B \rceil$ times, where N is the total sample budget in the target domain and B is the number of samples collected at each iteration. A typical training procedure is visualized in appendix A.2.

Algorithm 1: Jointly dynamics grounding and policy learning

while $|\mathcal{D}_{dynamics}| \leq N$ **do**
  Execute π_θ in f^* for B steps to collect $\mathcal{D}_{dynamics} = \{(s_t, a_t, s^*_{t+1})\}$
  Minimize $L_{dynamics}(\phi) = \sum_{(s_t,a_t,s^*_{t+1})\in\mathcal{D}_{dynamics}} \|\hat{f}(s_t, a_t) + \Delta(s_t, a_t, \phi) - s^*_{t+1}\|$
  while $\pi(\theta)$ *not converged* **do**
    Sample reference $r_0, \ldots, r_n$, set $s_0 \longleftarrow r_0$
    for *t=0...n-k* **do**
      $a_t, \ldots, a_{t+k} = \pi_\theta(s_t, \nu_{t+1}, \ldots, \nu_{t+k})$
      $\hat{s}_{t+1} = \hat{f}(s_t, a_t) + \Delta(s_t, a_t),\ \hat{s}_{t+2} = \hat{f}(\hat{s}_{t+1}, a_{t+1}) + \Delta(\hat{s}_{t+1}, a_{t+1}) \ldots$
      Add $\hat{s}_{t+1}, \ldots, \hat{s}_{t+k}$ and $\nu_{t+1}, \ldots, \nu_{t+k}$ to $\mathcal{D}_c$
      if $\|\hat{s}_{t+1} - r_{t+1}\| > \delta_{div}$ **then** $s_{t+1} \longleftarrow r_{t+1}$;
    end
    Minimize $L_{control}(\theta) = \sum_{(s_t,\nu_t)\in\mathcal{D}_c} \|s_t - \nu_t\|$
    Increase δ_{div}
  end
end

Dynamics grounding. To minimize δ, *i.e.* the difference between $\hat{f}$ and f^*, we collect tuples of state, action, and next state observed through interaction with the target dynamics. Formally :

$$\mathcal{D}_{dynamics} = \{(s_t, u_t, s^*_{t+1}) \mid t \in [1..B],\ s^*_{t+1} = f^*(s_t, u_t)\}, \tag{9}$$

This dataset is used to minimize difference of source and target domain by training a state-residual network Δ with the following loss function:

$$L_{dynamics}(\phi) = \sum_{(s_t,u_t,s^*_{t+1})\in\mathcal{D}} \|\hat{f}(s_t, u_t) + \Delta_\phi(s_t, u_t) - s^*_{t+1}\|, \tag{10}$$

where ϕ are the parameters of the network Δ_ϕ that accounts for effects that are not modeled in $\hat{f}$. If desired, $\hat{f}$ can also be parametrized in order to determine estimated properties of the system, e.g. the robot's mass or moment coefficients. The network Δ_ϕ can be conditioned on any kind of information, for example images, and can account for arbitrary unmodeled dynamics effects in the target domain.

Policy learning. We use gradient descent to tune the policy π_θ with the current dynamics $\hat{f}$. Specifically, we minimize the loss

$$L_{control}(\theta) = \frac{1}{|\mathcal{D}_c|} \sum_{(s,\nu)\in\mathcal{D}_c} \|\hat{f}\big(s, \pi_\theta(s, \nu)\big) - \nu\|, \tag{11}$$

where θ are the parameters of π_θ and $\mathcal{D}_c$ is a set of state-reference pairs collected from interaction with the current $\hat{f}$. In contrast to MPC or iLQR, we can optimize π_θ *offline* on the entire interaction data $\mathcal{D}_c$. This optimization is done with stochastic optimization algorithms. To favour short-term planning, we train π_θ in a receding-horizon fashion. Specifically, the controller predicts $k = 10$ actions at once from the current state and the next k references, where k is the horizon length. These k actions are used to forward-propagate the dynamics for the horizon length. The differences between the output of the dynamics and the reference over the horizon is then back-propagated to update θ. We additionally use a curriculum learning strategy to improve training. To do so, we set a threshold τ_{div} such that if $\|\hat{s}_{t+1} - \nu_{t+1}\| > \tau_{div}$ after executing action a_t, the agent is reset to ν_{t+1}. The threshold τ_{div} is increased over time. Therefore the robot will learn to follow the reference from gradually more distant states as training proceeds. More details of the training procedure are available in the appendix. An overview of the complete policy adaptation pipeline is provided in algorithm 1.

3 Experiments

We evaluate our approach on three diverse systems and tasks: CartPole balancing, trajectory tracking with a quadrotor, and flying to a 3D target with a fixed-wing aircraft. We design our evaluation

procedure to answer the following questions: (1) How does performance and sample efficiency compare to model-based and model-free approaches in both low-dimensional (Sections 3.2.1) and high-dimensional (Section 3.2.3) problems? (2) Does joint learning of dynamic grounding and control improve convergence (Section 3.2.2)?

3.1 EXPERIMENTAL SETUP

In the CartPole balancing task, a pole must be balanced by moving a cart. We use a model derived from first principles (Barto et al., 1983) as source dynamics $\hat{f}$, and allow for continuous actions in the range $[-1, 1]$ corresponding to ± 30N. Since the reference trajectory only consists of a single state (i.e. upright position of the pole), we linearly interpolate from the current state to the end state to generate a continuous reference $[\nu_{t+1}, \ldots, \nu_{t+k}]$. The second task consists of tracking a 3D trajectory with a quadrotor. In this problem, we use the first-principles model provided by the Flightmare simulator (Song et al., 2020) as source dynamics $\hat{f}$. For training and testing, we generate random polynomial trajectories with a maximum speed of $5\frac{m}{s}$. We evaluate performance based on 50 test trajectories not seen during training.

Finally, for reaching the 3D target with a fixed-wing aircraft, we also use a first-principles model for the source dynamics (Beard & McLain, 2012). Fixed-wing aircrafts can fly significantly faster than quadrotors, but are less maneuverable and thus more challenging to control. In our experiment, the aircraft is initialized with an initial velocity of $11.5\frac{m}{s}$ in the x-direction, and is tasked to reach a target at a distance of $x = 50m$, with y and z deviations from the initial position uniformly sampled from $[-5, 5]\mathrm{m}$. We evaluate performance with 30 target locations that have not been seen during training. Similar to the CartPole balancing task, the reference trajectory only consists of a single state, *i.e.* the target, and we generate a continuous reference $[\nu_{t+1}, \ldots, \nu_{t+k}]$ by linear interpolation from the current to the target state. Our approach works despite this approximation of the reference trajectory which might even be infeasible. This is further validated by the CartPole swing-up task in appendix A.5.

For all our experiments, we design a target domain by adding time-invariant and time-variant forces to the source domain. Such forces represent typical disturbances encountered when applying those systems in the real world (*e.g.* aereodynamic drag and contact forces). We perform all our experiments in PyTorch. Technical details are in appendix A.1 and the provided source code.

3.2 COMPARISON TO MODEL-BASED AND MODEL-FREE RL

We now compare our method to three strong baselines: (i) the model-free RL algorithm Proximal Policy Optimization (PPO) Schulman et al. (2017), (ii) the model-based model-free hybrid (MBMF) proposed by (Nagabandi et al., 2018) which applies Model Predictive Control on a learnt dynamics model, and (iii) the probabilistic-ensembles-with-trajectory-sampling (PETS) algorithm (Chua et al., 2018). The PPO baseline is available from the `stable-baselines` PyPI package (Raffin et al., 2019) under the MIT license. The second is available in our supplementary code and was adapted to use a non-linear programming solver to predict optimal actions instead of a simple random-sampling shooting method. The PETS implementation was taken from the `mbrl` toolbox (Pineda et al., 2021). All methods have access to the same inputs, produce the same type of output, and are tuned individually with grid-search on a held-out validation set. We evaluate the methods on three tasks of increasing difficulty: adaptation to a time-invariant linear disturbance, to a time-variant non-linear disturbance, and a time-variant non-linear disturbance only observable from images. We evaluate different approaches in terms of policy performance $J^*(\pi)$ and sample complexity in the target domain. For details on the baselines and further results see appendix A.3.

3.2.1 ADAPTATION TO TIME-INVARIANT LINEAR FORCES

We consider a benchmark problem for flying vehicles, which is to estimate and counteract a linear translational drag acting on the platform (Faessler et al., 2017; Franchi et al., 2018). This force is modeled as as $\dot{v} \longleftarrow \dot{v} - r \cdot v$ where v is the three dimensional velocity and $r = 0.3$ the drag factor for the quadrotor and the fixed-wing systems. Performance is evaluated with the average tracking error. For the CartPole balancing task we assume a constant force along the x-axis of the system, and evaluate performance based on stability of the cart, *i.e.* how fast the system moves along the x-axis.

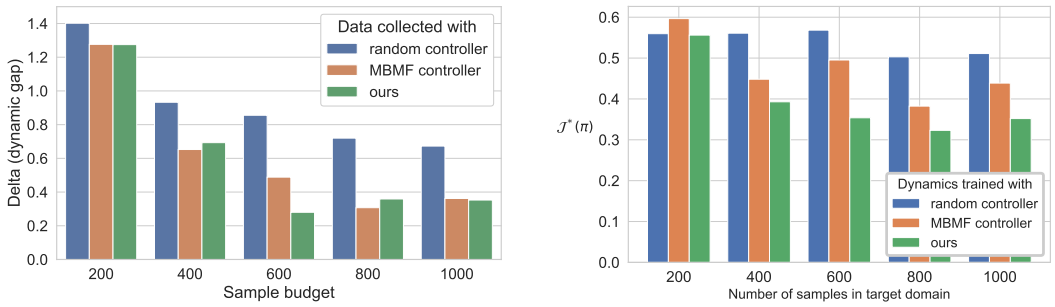
	Domain	Policy pretrained in $\hat{f}$ PPO	MBMF	PETS	Ours	Finetuned (samples in f^*) PPO	MBMF	PETS	Ours
CartPole	Source	0.04	0.01	0.24	0.05	**(60K)**	**(50)**	**(400)**	**(50)**
$L \approx 56.7$	Target	0.63	0.35	unstable	1.02	0.08	0.02	0.26	0.09
Quadrotor	Source	0.36	0.03	0.18 (*)	0.05	**(150K)**	**(1K)**	**(2.5K)**	**(1K)**
$L \approx 8.3$	Target	0.42	0.28	0.38 (*)	0.56	0.4	0.03	0.1 (*)	0.07
Fixed-wing	Source	0.09	0.0	0.24	0.01	**(150K)**	**(2K)**	**(5K)**	**(2K)**
$L \approx 41.2$	Target	0.43	0.08	8.39	0.07	0.09	0.01	0.27	0.02

Table 1: Performance comparison when the source and target domain differ of a time-invariant force, i.e. linear velocity drag for quadrotor and fixed-wing aircraft, and a lateral force on the pendulum. We measure performance in terms of average distance to the reference (in m) and the velocity of the cart (in m/s) respectively. We additionally report the Lipschitz constant L of the systems to give an idea of their dynamics. In this simple scenario, all methods nearly regain their performance in the target domain after finetuning, but the the model-based RL baselines and our method require significantly less interaction with the target domain. For the standard deviations see appendix A.3.1.

In these experiments, we assume that the exact functional of the disturbance in f^* is unknown. Both for the MBMF and ours, the dynamics $\hat{f}$ are grounded by training the network Δ_ϕ (cf. Equation equation 10). We represent Δ_ϕ by a two-layer MLP with tanh activation and 64 hidden nodes.

The results of these experiments are available in Table 1. All approaches perform similarly in the adaptation task, but PPO requires a significantly larger sample budget than MBMF and our method (e.g. 150 vs 1K samples). For an example of the convergence of PPO see appendix A.3.2. Online optimization via MBMF obtains slightly better performance than our approach after fine-tuning. This is in line with previous findings (Faessler et al., 2017), which show that velocity drag can be accurately counteracted with online optimization. The PETS algorithm (Chua et al., 2018) instead achieves very good sample efficiency, but no comparable performance could be achieved for the two more complex tasks of controlling flying vehicles. The quadrotor was simplified significantly to achieve good performance (marked (*) in Table 1): Instead of tracking an arbitrary polynomial at $\sim 3\frac{m}{s}$, we track only a single reference trajectory at lower speed ($1\frac{m}{s}$) with PETS. In contrast to that, we show in appendix A.6 that our controller for the quadrotor generalizes to acrobatic maneuvers at up to $5\frac{m}{s}$. Finally, our approach also achieves lower runtime than other model-based RL approaches (appendix A.4).

3.2.2 The advantages of joint optimization by means of time-variant force scenarios



(a) Dynamics gap

(b) Policy performance

Figure 2: Evolution of the dynamics mismatch cost $\delta_{s,\nu,\pi}$ and the policy performance $J^*(\pi_\theta)$ in function of samples collected with the target dynamics. Three policies are used for data collection: a random controller, and an MBMF controller, and the policy π_θ, which is jointly trained. Coupling the control and dynamic learning process favours sample efficiency.

Our approach tightly couples the identification and policy learning problems during training. In this section, we show that joint optimization improves sampling efficiency in the target domain. We perform this analysis on the scenario of time-variant non-linear forces. Specifically, in the target domain we apply a force of $2\cos t$ along the x-axis during the quadrotor flight, and a force of $2(\sin t + 1)$ along the x-axis of the CartPole system. Since the force is time-variant, it cannot be identified from a single state s_t. Therefore, we provide the history of the last 5 states and actions to both π_θ and the state-residual network Δ_ϕ. Both the policy and state residual networks are parametrized by a 2-layer MLP with 64 hidden units and tanh activations.

First, we analyse the effect of the data collection policy on the performance of the quadrotor. We use the dynamics mismatch cost δ and the policy performance $J^*(\pi_\theta)$ as metrics for a quantitative analysis. Data in the target domain is then collected with three policies: a random policy, a MBMF controller, and our iteratively trained policy π_θ. The results in Figure 2 show that using data collected with the policy π_θ results in quicker adaptation to the target dynamics. Doing joint identification and control leads to up to 16% and 33% better policy performance than collecting data with an MBMF or a random policy, respectively. Complementary results for the fixed-wing aircraft can be found in A.2.1. These findings confirm the intuition that for complex systems the model only needs to be accurate in the states observed during task execution, but not necessarily accurate in the unobserved regions of the state space.

Second, Figure 3 visualizes the sample efficiency of our joint approach compared to MBMF and PPO. For a fair comparison, both the MBMF and our method use exactly the same Δ_ϕ network. Here, our approach clearly outperforms the online optimization baseline in terms of both control performance and sample efficiency. Indeed, with a 1K sample budget, our approach reduces the velocity error by 24% with respect to the MBMF. This indicates that for higher-dimensional problems online optimization can struggle to find optimal solutions.

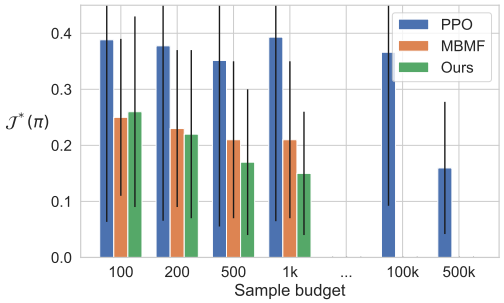


Figure 3: CartPole subject to a time-variant force.

3.2.3 VISION-BASED CONTROL

Finally, we repeat the same experiment as in the previous paragraph, but condition the network Δ_ϕ and the policy π_θ only on a history of the last 5 images and actions. We represent both Δ_ϕ and π_θ as a 4-layer convolutional network with kernel size 3 and ReLU activations. We additionally pretrain a differentiable renderer network Ψ that transforms the system state s_t (not accessible to π_θ and Δ_ϕ) into a 3rd-person-view image of the CartPole. This network is used for training the controller in a receding-horizon fashion. We parametrize Ψ via a 4-layer MLP with $[32, 128, 256]$ hidden nodes and tanh activation; it outputs an image of size 120×100. The results of this experiment are presented in Table 2. In this case, online optimization in the image space either does not converge or returns poor solutions, both in the source and in the target domain. In contrast, our method and the model-free baseline (PPO) both obtain very good performance.However, our approach can adapt the policy with a fraction of the sample budget (1K vs 230K samples) while obtaining a 29% better performance in the target domain. This experiment shows that our approach combines the best of both worlds. It can leverage prior knowledge about the system to increase sample efficiency and at the same time retain the flexibility of model-free methods to cope with high-dimensional inputs.

Domain	PPO (0)	MBMF (0)	Ours (0)	PPO (230K)	MBMF (1K)	Ours (1K)
Source	0.28 ± 0.23	unstable	0.15 ± 0.11	-	-	-
Target	unstable	unstable	unstable	0.39 ± 0.29	unstable	0.29 ± 0.17

Table 2: Image-based control and identification for the task of CartPole balancing. We report the sample budget in parentheses. The source and target domains differ by a time-variant non-linear force. Our approach outperforms the baselines in terms of performance and sample efficiency.

4 RELATED WORK

Adaptive control for real-world dynamics Many works regard the transfer to a target domain in the context of robust adaptive control (Ioannou & Sun, 2012). Next to optimization methods for adaptive model predictive control (Dean et al., 2018; Dougherty & Cooper, 2003; Song et al., 2019), also learning approaches for robust and safe control were proposed (Berkenkamp et al., 2016; Dean et al., 2019; Zhang et al., 2020) and tested in real-world settings. Kaufmann et al. (2020) learn a controller in simulation with environment randomization and abstract representations in order to transfer it to the real world. In contrast to our work and optimization based methods for adaptive control, robust control deals with uncertainties and noise in the environment (Saltık et al., 2018). Analyzing the robustness of our method with respect to noisy state observations, or leveraging uncertainty methods for sim-to-real scenarios may be an interesting opportunity for future work.

Grounded simulation learning In the context of reinforcement learning, Farchy et al. (2013) introduced "grounded simulation learning" (i.e. altering the simulator to better match the real world). Hanna & Stone (2017) extend this work and learn an action transformation (GAT) from data for simulator grounding. Recently, Desai et al. (2020) showed that grounded action transformation can be formulated as an imitation from observation problem. These works are similar to ours in the sense that the simulator is altered to match the target dynamics. However, action transformations can not account for arbitrary changes in the dynamics. Similarly, for simple identification tasks it is sufficient to estimate parameters of the target system from data (Chebotar et al., 2019), whereas in this work also external disturbances were considered.

Differentiable programming (DP) Neural ODEs (Chen et al., 2018) have inspired DP approaches for control, and several differential physics simulators were developed (Hu et al., 2019; Qiao et al., 2020; Gradu et al., 2021). To adapt to changes in the dynamics system, Heiden et al. (2019; 2020) extend a differentiable physics simulator with neural networks, however not in the context of control tasks. Other DP approaches where proposed for solving non-linear system identification and optimal control (Jatavallabhula et al., 2021; Jin et al., 2019) and show that it outperforms state-of-the-art control methods. We arrive at a similar conclusion, but our work jointly optimizes for both identification and control.

5 DISCUSSION

While there is a large body of work on domain adaptation for image recognition tasks, the same problem in the context of sensorimotor control has received little attention. However, efficient adaptation of sensorimotor policies could have fundamental implications for several real-world applications. While a lot of industrial robots have operated in controlled and constrained environments, modern robots require more flexibility and are deployed in a variety of more natural and unconstrained environments. Hence, they need to quickly adapt to novel conditions (*e.g.* environment effects, contact dynamics, human-robot interaction) that cannot always be anticipated. In this work, we analyze this domain adaptation problem. We use our framework to review existing methods and motivate a novel approach which jointly minimizes the control and identification cost with gradient descent. Through controlled experiments, we show that our approach maintains the sample efficiency of online optimization methods while keeping the flexibility of offline model-free algorithms.

We see a number of opportunities for future work. First, the presented methodology is only demonstrated on control tasks with a relatively short horizon (up to $5\,\mathrm{s}$ in the future). We hypothesize that combining differentiable programming with stochastic gradient descent could be beneficial for sample efficient domain adaptation in long-term planning problems, *e.g.* maze navigation (Wijmans et al., 2020; Hafner et al., 2019b). Second, we believe that our methodology can be used to enable autonomous operation of systems that are very difficult to model and control, *e.g.* bio-inspired robots (Ajanic et al., 2020; Karásek et al., 2018). Overall, we hope that this work will inspire future research at the intersection of machine learning and robotics on the task of few-shot domain adaptation.

STATEMENT OF REPRODUCIBILITY

The source code to reproduce all experiments is provided in the supplementary material and will be published upon acceptance. All dependencies are available under the MIT license as discussed in subsection 3.2. Our code further includes the checkpoints for the best-performing model for each CartPole, quadrotor and fixed-wing aircraft. The data is collected in simulation and mostly generated on the fly. Hence, we consider our work fully reproducible with the information in the README as well as the training details given in appendix A.1.

REFERENCES

Enrico Ajanic, Mir Feroskhan, Stefano Mintchev, Flavio Noca, and Dario Floreano. Bioinspired wing and tail morphing extends drone flight capabilities. *Science robotics*, 5(47):eabc2897, 2020.

Somil Bansal, Anayo K Akametalu, Frank J Jiang, Forrest Laine, and Claire J Tomlin. Learning quadrotor dynamics using neural network for flight control. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 4653–4660. IEEE, 2016.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5): 834–846, 1983.

Randal W Beard and Timothy W McLain. *Small unmanned aircraft: Theory and practice*, pp. 16, 44ff. Princeton university press, 2012.

Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. Safe controller optimization for quadrotors with gaussian processes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 491–496. IEEE, 2016.

Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979. IEEE, 2019.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *NeurIPS*, 2018.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, pp. 617–629, 2018.

Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. Regret bounds for robust adaptive control of the linear quadratic regulator. *arXiv preprint arXiv:1805.09388*, 2018.

Sarah Dean, Stephen Tu, Nikolai Matni, and Benjamin Recht. Safely learning to control the constrained linear quadratic regulator. In *2019 American Control Conference (ACC)*, pp. 5582–5588. IEEE, 2019.

Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13:6, 2019. doi: 10.3389/fnbot.2019.00006. URL https://www.frontiersin.org/article/10.3389/fnbot.2019.00006.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

Yitong Deng, Yaorui Zhang, Xingzhe He, Shuqi Yang, Yunjin Tong, Michael Zhang, Daniel DiPietro, and Bo Zhu. Soft multicopter control using neural dynamics identification. *arXiv preprint arXiv:2008.07689*, 2020.

Siddarth Desai, Ishan Durugkar, Haresh Karnan, Garrett Warnell, Josiah Hanna, and Peter Stone. An imitation from observation approach to sim-to-real transfer. *arXiv preprint arXiv:2008.01594*, 2020.

Danielle Dougherty and Doug Cooper. A practical multiple model adaptive strategy for single-loop mpc. *Control engineering practice*, 11(2):141–159, 2003.

Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories. *IEEE Robotics and Automation Letters*, 3(2):620–626, 2017.

Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone. Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 39–46, 2013.

Antonio Franchi, Ruggero Carli, Davide Bicego, and Markus Ryll. Full-pose tracking control for aerial robotic systems with laterally bounded input force. *IEEE Transactions on Robotics*, 34(2): 534–541, 2018.

Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4019–4026. IEEE, 2016.

Paula Gradu, John Hallman, Daniel Suo, Alex Yu, Naman Agarwal, Udaya Ghai, Karan Singh, Cyril Zhang, Anirudha Majumdar, and Elad Hazan. Deluca–a differentiable control library: Environments, methods, and benchmarking. *arXiv preprint arXiv:2102.09968*, 2021.

Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 9–20, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919998. doi: 10.1145/280814.280816. URL https://doi.org/10.1145/280814.280816.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019b.

Josiah Hanna and Peter Stone. Grounded action transformation for robot learning in simulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, volume 28, 2015. URL https://proceedings.neurips.cc/paper/2015/file/148510031349642de5ca0c544f31b2ef-Paper.pdf.

Eric Heiden, David Millard, Hejia Zhang, and Gaurav S Sukhatme. Interactive differentiable simulation. *arXiv preprint arXiv:1905.10706*, 2019.

Eric Heiden, David Millard, Erwin Coumans, and Gaurav S Sukhatme. Augmenting differentiable simulators with neural networks to close the sim2real gap. *arXiv preprint arXiv:2007.06045*, 2020.

Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Camilo Gamboa, Melissa Mozifian, Florian Golemo, Chris Atkeson, Dieter Fox, Ken Goldberg, John Leonard, et al. Sim2real in robotics and automation: Applications and challenges. *IEEE Transactions on Automation Science and Engineering*, 18(2):398–400, 2021.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

Petros A Ioannou and Jing Sun. *Robust adaptive control*. Courier Corporation, 2012.

Krishna Murthy Jatavallabhula, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jerome Parent-Levesque, Kevin Xie, Kenny Erleben, et al. gradsim: Differentiable simulation for system identification and visuomotor control. *arXiv preprint arXiv:2104.02646*, 2021.

Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. *arXiv preprint arXiv:1912.12970*, 2019.

Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. 2020.

Matěj Karásek, Florian T Muijres, Christophe De Wagter, Bart DW Remes, and Guido CHE de Croon. A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns. *Science*, 361(6407):1089–1094, 2018.

Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020. doi: 10.15607/RSS.2020.XVI.040.

Ian Lenz, Ross A Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*. Rome, Italy, 2015.

Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 2019. doi: 10.1109/TRO.2019.2942989.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018.

Philipp Oettershagen, Amir Melzer, Stefan Leutenegger, Kostas Alexis, and Roland Siegwart. Explicit model predictive control and l 1-navigation strategies for fixed-wing uav path tracking. In *22nd Mediterranean Conference on Control and Automation*, pp. 1159–1165. IEEE, 2014.

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntak Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile autonomous driving using end-to-end deep imitation learning. In *Robotics: science and systems*, 2018.

Paavo Parmas. Total stochastic gradient algorithms and applications in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper/2018/file/0d59701b3474225fca5563e015965886-Paper.pdf.

Luis Pineda, Brandon Amos, Amy Zhang, Nathan O. Lambert, and Roberto Calandra. Mbrl-lib: A modular library for model-based reinforcement learning. *Arxiv*, 2021. URL https://arxiv.org/abs/2104.10159.

Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C Lin. Scalable differentiable physics for learning and control. *arXiv preprint arXiv:2007.02168*, 2020.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík (eds.), *International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, 2011.

M Bahadır Saltık, Leyla Özkan, Jobert HA Ludlage, Siep Weiland, and Paul MJ Van den Hof. An outlook on robust model predictive control algorithms: Reflections on performance and computational aspects. *Journal of Process Control*, 61:77–102, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Yan Song, Kaiqun Zhu, Guoliang Wei, and Jianhua Wang. Distributed mpc-based adaptive control for linear systems with unknown parameters. *Journal of the Franklin Institute*, 356(5):2606–2624, 2019.

Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. *arXiv preprint arXiv:2009.00563*, 2020.

Antony Waldock, Colin Greatwood, Francis Salama, and Thomas Richardson. Learning to perform a perched landing on the ground using deep reinforcement learning. *Journal of Intelligent & Robotic Systems*, 92(3):685–704, 2018.

Yaqing Wang, Quanming Yao, James T. Kwok, and Lionel M. Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Comput. Surv.*, 53(3), 2020. ISSN 0360-0300. doi: 10.1145/3386252. URL https://doi.org/10.1145/3386252.

Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *International Conference on Learning Representations*, 2020.

Kaiqing Zhang, Bin Hu, and Tamer Basar. On the stability and convergence of robust adversarial reinforcement learning: A case study on linear quadratic systems. *Advances in Neural Information Processing Systems*, 33, 2020.

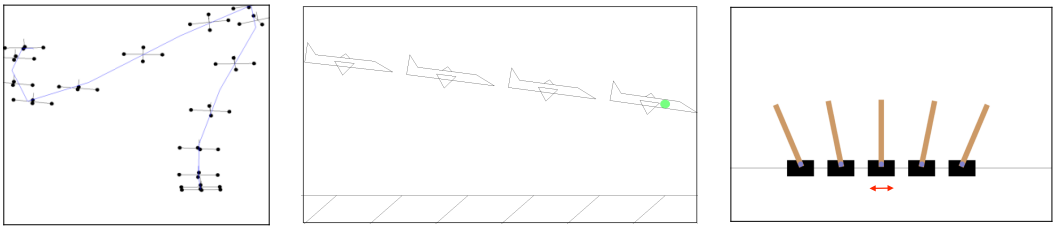
Weixuan Zhang, Marco Tognon, Lionel Ott, Roland Siegwart, and Juan Nieto. Active model learning using informative trajectories for improved closed-loop control on real robots. In *Int. Conf. on Robotics and Automation*. IEEE, 2021.

Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5745–5753, 2019.

A APPENDIX

A.1 HYPERPARAMETERS AND TRAINING DETAILS

We tested our method on three dynamic systems that are visualized in Figure 4: a quadrotor, a fixed-wing aircraft, and the CartPole task. Our framework is implemented in Pytorch, enabling the use of autograd for backpropagation through time. All source code is attached, and the parameters can be found as part of the source code in the folder `configs`. Example videos for all three applications are also attached in the supplementary material. In the following we provide details on training and evaluation.



(a) Quadrotor (b) Fixed-wing drone (c) CartPole

Figure 4: Experiments are conducted on three diverse dynamic systems and tasks: Tracking a trajectory with a quadrotor, passing through a target with a fixed wing drone, and balancing a pole on a cart. The `OpenAI Gym` renderer was used for simple 2D visualizations.

A.1.1 QUADROTOR

Our implementation of a quadrotor environment is loosely based on the implementation provided at `https://github.com/ngc92/quadgym` (MIT license). However, our model of the quadrotor is a Pytorch implementation of the equations in the Flightmare simulator Song et al. (2020). We also provide an interface to test our models in Flightmare, and a video of our model controlling a quadrotor in Flightmare is attached in the supplementary material. In all our experiments, we set the time between the discrete time steps to $0.1s$.
Furthermore, for training and testing we generate 10000 random polynomials of $10s$ length, where all trajectories are guaranteed to be feasible to track with the used platform. From this dataset, 1000 trajectories are left out as a test set to ensure that the policy generalizes to any given reference. The maximum desired velocity on such a reference trajectory is usually around $3 - 5m/s$, whereas the average velocity is $1 - 2m/s$.

Policy network At each time step, the current state and the next reference states are given as input to the network. As the current state, we input the velocity in the world frame and in the body frame, the first two columns of the rotation matrix to describe the attitude (as recommended in Zhou et al. (2019)), and the angular velocity. For the reference, we input the next 10 desired positions *relative* to the current drone position, as well as the next 10 desired velocities (in the world frame).
The state is first passed through a linear layer with 64 neurons (tanh activation), while the reference is processed with a 1D convolutional layer (20 filters, kernel size 3) to extract time-series features of the reference. The outputs are concatenated and fed through three layers of 64 neurons each with tanh activation. The output layer has 40 neurons to output 10 four-dimensional actions. Here, an action corresponds to the total thrust T and the desired body rates ω_{des} to be applied to the system. The network outputs are first normalized with a sigmoid activation and then rescaled to output a thrust between $2.21N$ and $17.31N$ (such that an output of 0.5 corresponds to $9.81N$) and body rates between -0.5 and 0.5.

Loss function The loss (equation 11) is a weighted MSE between the reached states and the reference states. The weights are aligned to the ones used for the optimization-based MBMF, namely a weight of 10 for the position loss, 1 for the velocity, 5 to regularize the predicted thrust command and 0.1 for the predicted body rates as well as the actualy angular velocity. Formally, these weights

yields the following loss:

$$L = \sum_{k=1}^{10} 10 \cdot (x_{t+k,\pi} - x_{t+k,\nu})^2 + (\dot{x}_{t+k,\pi} - \dot{x}_{t+k,\nu})^2 + 5 \cdot (T_k - 0.5)^2 + 0.1 \cdot (\omega_{k,des} - 0.5)^2 + 0.1 \cdot \omega_{t+k}$$

where x_t is the position at time step t, $\dot{x}_t$ is the velocity, and the subscript π indicates the states reached with the policy while the subscript ν denotes the positions and velocities of the reference. T and ω_{des} correspond to the action after sigmoid activation but before rescaling (such that values lie between 0 and 1), and ω_t is the actual angular velocity of the system at each state. The loss is minimized with the Pytorch SGD optimizer with learning rate 10^{-5} and momentum 0.9.

Curriculum learning As explained in subsection 2.2, we use a curriculum learning strategy with a threshold τ_{div} on the allowed divergence from the reference trajectory. We set $\tau_{div} = 0.1m$ initially and increase it by $0.05m$ every 5 epochs, until reaching $2m$. Additionally, we start by training on slower reference trajectories (half the speed). Once the quadrotor is stable and tracks the full reference without hitting τ_{div}, the speed is increased to 75% of the desired speed and τ_{div} is reset to $0.1m$. This is repeated to train at the full speed in the third iteration.

A.1.2 FIXED-WING DRONE

We implemented a realistic model of the dynamics based on the equations and parameters described in Beard & McLain (2012) and Waldock et al. (2018). In our discrete-time formulation, we set $\delta t = 0.05s$.

Reference trajectory In contrast to the reference trajectories for the quadrotor, the reference for the fixed-wing aircraft is only given implicitly with the target position. As an approximate reference, we train the policy to follow the linear trajectory towards the target point. In the following, the term "linear reference" will be used to refer to the straight line from the current position to the target point. At each time step, we compute the next 10 desired states as the positions on the linear reference while assuming constant velocity.

Policy network Similar as for the quadrotor, the state and reference are pre-processed before being input to the network policy. The state is normalized by subtracting the mean and dividing by the standard deviation per variable in the state (mean and standard deviation are computed over a dataset of states encountered with a random policy). This normalized state together with the relative position of the 10-th desired state on the reference are passed to the policy network as inputs. Using all 10 reference states as input is redundant since they are only equally-distant positions on a line.
The state and the reference are each separately fed through a linear layer with 64 neurons and then concatenated. The feed-forward network then corresponds to the one used for the quadrotor training (three further layers with 64 neurons each and an output layer with 40 neurons). The output actions are also normalized with sigmoid activations and then scaled to represent thrust $T \in [0, 7]N$, elevator angle $a_1 \in [-20, 20]^\circ$, aileron angle $a_2 \in [-2.5, 2.5]^\circ$ and rudder angle $a_3 \in [-20, 20]^\circ$.

Loss function. As for the quadrotor, we align the loss function to the cost function of the online optimization model predictive control in the MBMF baseline. The MSE between the reached positions and the target positions on the linear reference is minimized while regularizing the action, formally

$$L = \sum_{k=1}^{10} 10 \cdot (x_{t+k,\pi} - x_{t+k,\nu})^2 + 0.1 \cdot ((a_{k,1} - 0.5)^2 +, (a_{k,2} - 0.5)^2 + (a_{k,3} - 0.5)^2) \,,$$

where t is the current time step and $x_{t+k,\pi}$ is the position of the aircraft after executing the k-th action $(T_k, a_{k,1}, a_{k,2}, a_{k,3})$, and $x_{t+k,\nu}$ is the corresponding reference state. The loss is minimized with an SGD optimizer with learning rate of 10^{-4} and momentum of 0.9.

Finally, the curriculum is initialized to allow a divergence of $4m$ from the linear reference and increased by $0.5m$ every epoch until reaching $20m$. Note that in contrast to the quadrotor, the model converges in few epochs.

A.1.3 CARTPOLE

In the CartPole problem, a pole should be balanced on a cart by pushing the cart left and right, while maintaining low cart velocity. The time step is again set to $0.05s$. The state of the system can be described by position x and velocity $\dot{x}$ of the cart, as well as angle α and angular velocity $\dot{\alpha}$ of the pole. The "reference trajectory" is defined only by the target state, which is the upright position of the pole, $\alpha = 0$, $\dot{\alpha} = 0$ and $\dot{x} = 0$.

Policy network Since the reference state is constant, it is sufficient for the policy to observe the state at each time step. Here, we input the raw state without normalization. It is passed through a five-layer MLP with 32, 64, 64, 32 and 10 neurons respectively. All layers including the output layer use tanh activation, in order to scale the actions to values between -1 (corresponding to $30N$ force to the left) and 1 (force of $30N$ pushing the cart to the right). As for the quadrotor and fixed-wing drone, the next 10 (1-dimensional) actions are predicted.

Loss function To compute the loss with respect to a reference trajectory, we interpolate between the current state and the target state. Note that the intermediate states are often infeasible to reach; for example an increase of velocity might be required to reduce the pole angle. The interpolation where both velocity and angle decrease is thus not realistic. As before, a weighted MSE between the reference states and the actual states is computed, where the angle difference is weighted with a factor of 10, the cart velocity with factor 3, and the angular velocity with factor 1. The loss is minimized with SGD optimizer with learning rate 10^{-7}.

A.2 ITERATIVE TRAINING

As described in subsection 2.2, we propose a training procedure that iterates between grounding the dynamics and fine-tuning the controller. An example is shown Figure 5, where the quadrotor model is fine-tuned to account for velocity drag (see subsubsection 3.2.1). The iterative algorithm starts from a pre-trained policy that achieves low tracking error in $\hat{f}$ but higher tracking error ($0.42m$, see Table 1) in the target environment f^* due to velocity drag. When no improvement was found for five consecutive epochs in the dynamics or control test data performance, we switch to the other one respectively. Before the dynamics model is trained, new data in f^* is collected with the most recent policy (always before red sections in Figure 5). Specifically, we start with 200 samples at epoch 0, adding 200 additional samples at epoch 15, again at epoch 38, etc. Thus, overall only 1800 samples in f^* are used in this example.
It can be observed in Figure 5 that 200 samples are insufficient to fine-tune the dynamics model, but with 400 samples already both δ as well as the tracking error decrease significantly. With a sample budget of 1000, the tracking error is already below 0.1, almost matching the performance of our best model in $\hat{f}$ (tracking error of $0.05m$). Observe also that fine-tuning the policy (blue parts) is extremely fast thanks to differentiable programming.

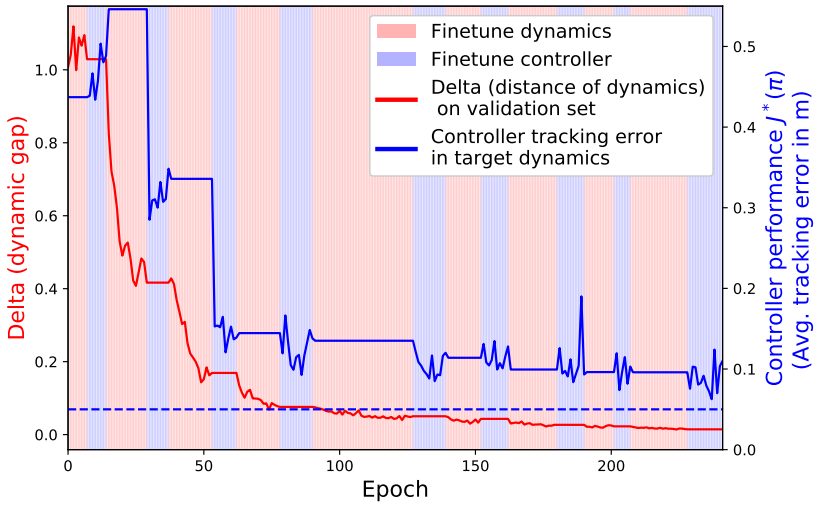
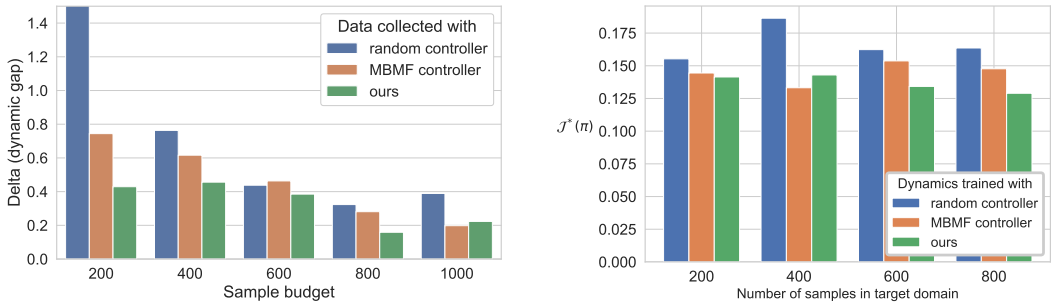


Figure 5: Iterative fine-tuning of model and policy. Here, a residual network Δ is trained to account for velocity drag in the target dynamics f^* with low sample budget. The policy π is fine-tuned (blue parts) by backpropagation through the updated dynamics model. When converged, the new policy π is used to collect new samples in f^* to further decrease the dynamic gap Δ (red parts).

A.2.1 Advantages of coupled training of dynamics and control

In accordance to the experiment on the quadrotor shown in subsubsection 3.2.2, we test the performance of coupled training also on the fixed-wing aircraft. A time-variant force of $2 \cos t$ is applied orthogonal to the initial flight direction. In Figure 6a we compare the dynamics error δ yielded when collecting data with a random policy, the MBMF controller or our pre-trained policy. The mismatch cost δ is reduced by up to 55% compared to the MBMF and up to 65% in contrast to a random policy. The policy performance improves accordingly (Figure 6b). This result further supports the advantages of joint dynamics and control optimization.



(a) Dynamics gap

(b) Policy performance

Figure 6: Evolution of the dynamics mismatch cost $\delta_{s,\nu}$ and the policy performance for tracking a target point with a fixed-wing aircraft. Data in the target domain f^* is collected with either a random policy, a MBMF or our pre-trained policy. The error decreases faster when our joint learning approach is used, confirming the result in subsubsection 3.2.2

A.3 Model-free and model-based baselines

The online-optimization model predictive control in the MBMF baseline is implemented with the toolbox `CasADi` for numerical optimization, using its nonlinear programming solver. The same reference trajectories and loss functions as for our neural controller are used.

As a baseline for model-free RL, the PPO algorithm of the `stable-baselines3` PyPI package Raffin et al. (2019) is used with the default parameters. The maximal episode length corresponds to the length of the reference trajectory for the quadrotor, the time to pass the x-position of the target point for the fixed-wing, and is infinite for the CartPole. In our experiments on PPO, we first pre-train a policy in the environment corresponding to $\hat{f}$ until convergence. We then start with the model performing best in $\hat{f}$ and fine-tune in the target domain, i.e. an environment corresponding to f^*. For example, for the experiment in Table 1 the PPO baseline is fine-tuned in a modified environment with velocity drag.

A.3.1 Baseline comparison with standard deviations

Table 3 complements the results from in Table 1 by including standard deviations.

		Policy pretrained in $\hat{f}$				Finetuned (samples in f^*)			
	Domain	**PPO**	**MBMF**	**PETS**	**Ours**	**PPO**	**MBMF**	**PETS**	**Ours**
CartPole	Source	0.04 ± 0.04	0.01 ± 0.03	0.24 ± 0.10	0.05 ± 0.05	**PPO (60K)**	**MBMF (0.3K)**	**PETS (400)**	**Ours (50)**
$L \approx 56.7$	Target	0.63 ± 0.54	0.35 ± 0.06	unstable	1.02 ± 0.35	0.08 ± 0.06	0.02 ± 0.02	0.26 ± 0.08	0.09 ± 0.04
Quadrotor	Source	0.36 ± 0.10	0.03 ± 0.01	0.18 ± 0.08 (*)	0.05 ± 0.01	**PPO (150K)**	**MBMF (1K)**	**PETS (2.5K)**	**Ours (1K)**
$L \approx 8.3$	Target	0.42 ± 0.09	0.28 ± 0.04	0.38 ± 0.09 (*)	0.56 ± 0.07	0.40 ± 0.12	0.03 ± 0.02	0.1 ± 0.03 (*)	0.07 ± 0.01
Fixed-wing	Source	0.09 ± 0.05	0.0 ± 0.0	0.24 ± 0.15	0.01 ± 0.01	**PPO (150K)**	**MBMF (2K)**	**PETS (5K)**	**Ours (2K)**
$L \approx 41.2$	Target	0.43 ± 1.45	0.08 ± 0.08	8.39 ± 10.15	0.07 ± 0.10	0.09 ± 0.06	0.01 ± 0.02	0.27 ± 0.14	0.02 ± 0.03

Table 3: Performance comparison of model-free and model-based RL baselines on the CartPole, quadrotor, and fixed-wing control tasks. Here, in addition to the results from in Table 1, the standard deviation is shown.

A.3.2 Convergence of model-freel RL fine-tuned in f^*

To give a better intuition on the sample budget of fine-tuning our model-free RL baseline, the convergence of the PPO baseline for the fixed-wing drone is shown in Figure 7. Here, the training is started from a pre-trained model that achieves strong performance in the source domain $\hat{f}$, but has not seen any samples from the target domain f^* yet. Thus, the distance of the drone from the target point is initially $0.53m$ on average. During fine-tuning, the performance in the modified environment f^* is measured every 3000 samples on 40 runs with random target points. Only with more than $150k$ steps in f^*, the model has regained the performance that could be achieved in $\hat{f}$, which is an average error of $0.09m$.

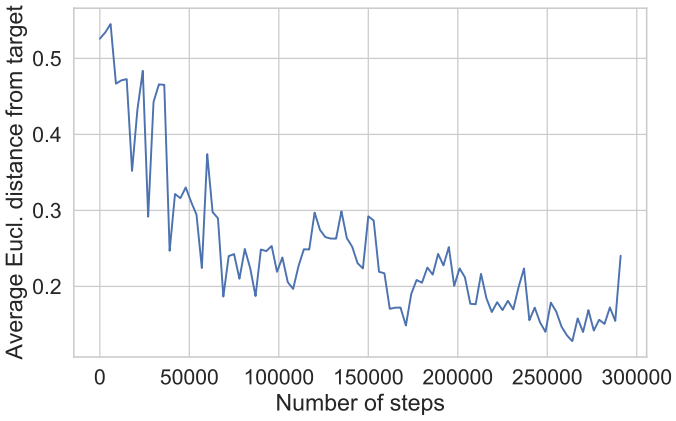


Figure 7: Convergence of model-free PPO baseline, when fine-tuned in the target domain f^* on a fixed-wing aircraft. In contrast to our approach, model-free RL requires a much higher sample budget to achieve similar performance in f^* as in $\hat{f}$.

A.3.3 TRAINING CONVERGENCE BY SAMPLE BUDGET

In order to further analyse how quickly the residual network learns to account for a dynamic mismatch, we varied the number of samples collected in f^* and observe the convergence of the dynamic loss. For this experiment we again use the velocity drag scenario for the quadrotor (Table 1). Note that one could also set translational drag itself as a trainable parameter, such that the drag is adjusted with gradient descent. This leads to almost immediate closure of the dynamics gap. Here, we are instead interested to estimate the ability of the residual network to account for arbitrary (black box) dynamic changes.

Figure 8 shows the convergence when training the dynamics model with different dataset sizes. The validation loss is computed on 200 validation samples that neither the controller nor the dynamics model has seen during training. Although convergence is slower for smaller sample budgets, and more overfitting can be observed, even with just 200 samples (20 seconds of flight) the loss converges in fifty epochs.

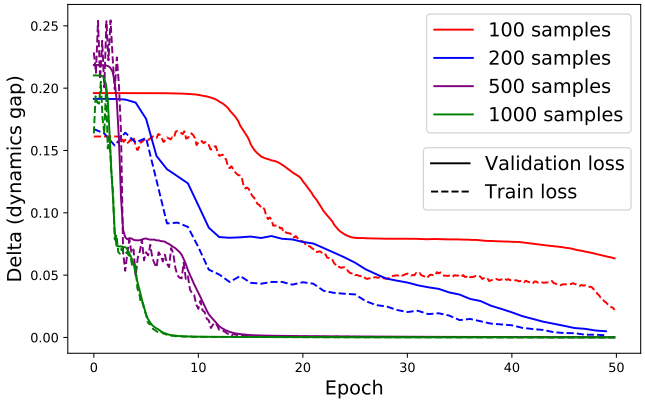


Figure 8: Learning to account for velocity drag of a quadrotor. The convergence of the mean squared error between learnt dynamics and target dynamics during training is shown. With only 200 samples, the error converges close to zero in 50 epochs. With 1000 samples, 10 epochs suffice to ground the simulator.

A.4 RUNTIME COMPARISON TO MODEL-BASED RL METHODS

Model-based RL approaches oftentimes use online-optimization such as model predictive control (MPC) to predict the next action. Our approach is instead trained in differentiable programming manner and one action prediction is only a forward pass in the network. For example, our approach takes $0.00027s$ on average to output an action for the CartPole balancing task. Instead, our implementation of the MBMF with a non-linear programming solver requires $0.005s$, and the PETS algorithm (as implemented in the MBRL-toolbox) even needs $2.2s$ on average. All times were computed on a single CPU. While better implementations could increase performance, our approach stands out as a method that achieves low sample complexity at very low runtime.

A.5 CARTPOLE SWING-UP

Our backpropagation-through-time approach depends on the existence of a reference trajectory. However, the reference trajectory does not need to be feasible. This is already shown in the experiments with a fixed-wing aircraft, where we use a linear interpolation as the reference. To further support this point, we conduct an experiment on the CartPole swing-up task. For a swing-up, the pendulum must swing back and forth and can not follow a direct sequence of reference states towards the upright goal state. Nevertheless, we train our controller with the same parameters and network as for the simple balancing task (appendix A.1.3), and also with the same reference, namely an interpolation from the current state to the goal state. We count a swing-up trial as a success if the pendulum angle is

	Max. speed	Avg. tracking error (in m)	Max. tracking error (in m)
Polynomials (train/test)	$\sim$ 3 m/s	0.05	0.09
Power Loop	2.6 m/s	0.05	0.13
Power Loop	3.4 m/s	0.09	0.23
Power Loop	4.1 m/s	0.12	0.33
Power Loop	4.6 m/s	0.21	0.79
Power Loop	5.0 m/s	0.31	1.29

Table 4: High-speed flight with a quadrotor. Our trained controller can track the Power Loop trajectory at a speed of up to $5ms^{-1}$.

less than 10° from the vertical position for at least 200 time steps, after an initial burn-in (swing-up) period of 100 steps. Our method achieves a 100% success rate from randomized initial states after training on 20000 samples. The x-position and velocity of the cart are also stable as desired (0.01m from center and $0.29\frac{m}{s}$ on average). For comparison, the PPO baseline needs 300000 samples to achieve a 100% success rate, and requires more motion of the cart ($0.74\frac{m}{s}$ on average) although the velocity is penalized.

A.6 High-speed quadrotor flight

Controlling a quadrotor to follow an arbitrary trajectory in high speed is a challenging task. Our model that was trained on 10000 random polynomial trajectories showed excellent ability to generalize to unseen trajectories and to operate with a different desired speed. Here, we push the controller to its limits by applying it at higher speed on an acrobatic maneuver proposed in Kaufmann et al. (2020), namely the "Power loop". Table 4 demonstrates that our controller can successfully track the Power loop at up to $5m/s$. The increase in tracking error is expected since the observed states become more and more different to the ones observed at training time as the speed increases.