This paper has been accepted for publication at the
2023 IEEE International Conference on Robotics and Automation (ICRA)

Training Efficient Controllers via Analytic Policy Gradient

Nina Wiedemann*[1], Valentin Wüest*[2], Antonio Loquercio[1], Matthias Müller[3],
Dario Floreano[2], and Davide Scaramuzza[1]

***Abstract*— Control design for robotic systems is complex and often requires solving an optimization to follow a trajectory accurately. Online optimization approaches like Model Predictive Control (MPC) have been shown to achieve great tracking performance, but require high computing power. Conversely, learning-based offline optimization approaches, such as Reinforcement Learning (RL), allow fast and efficient execution on the robot but hardly match the accuracy of MPC in trajectory tracking tasks. In systems with limited compute, such as aerial vehicles, an accurate controller that is efficient at execution time is imperative. We propose an Analytic Policy Gradient (APG) method to tackle this problem. APG exploits the availability of differentiable simulators by training a controller offline with gradient descent on the tracking error. We address training instabilities that frequently occur with APG through curriculum learning and experiment on a widely used controls benchmark, the CartPole, and two common aerial robots, a quadrotor and a fixed-wing drone. Our proposed method outperforms both model-based and model-free RL methods in terms of tracking error. Concurrently, it achieves similar performance to MPC while requiring more than an order of magnitude less computation time. Our work provides insights into the potential of APG as a promising control method for robotics.**

Supplementary Material

Video of ICRA presentation: https://youtu.be/HdfvsI126D8
Source code: https://github.com/lis-epfl/apg_trajectory_tracking

I. Introduction

Control systems for robotics are becoming increasingly complex, as they are required to cope with diverse environments and tasks. As a response to challenges in deploying robots in real-world environments, learning-based control has gained importance and has started to replace classical control such as Model Predictive Control (MPC) or PID control [1], [2]. While both (reinforcement) learning and MPC can be used for trajectory tracking by maximizing a reward or minimizing a cost with respect to the reference trajectory, they drastically differ in terms of computation of the optimization. MPC requires no prior training and computes the optimization directly at runtime, resulting in large computational demands on the robot. Subsequently, much work is invested into approximation algorithms or tailored hardware, since otherwise, the computation is not feasible to execute on the real platform [3], [4], [5]. This is especially relevant for flying vehicles where only limited computing hardware can be carried. In contrast, model-free RL is often applied by shifting the optimization to a training phase in simulation before deployment. Although RL was developed for solving long-term planning problems, it can be used for trajectory tracking by negatively rewarding the distance to a reference [6]. At runtime, inference of the policy then allows fast execution on the robot. However, RL treats dynamics as a black-box model and probes the environment to gain information about it. As a result, it is prone to require a large amount of training data, which is particularly restrictive for complex and slow-to-simulate systems. Finding a solution requiring little training data and permitting fast computation during run-time would be desirable.

Recent developments in the field of differentiable simulators have given rise to an alternative approach called Analytic Policy Gradient (APG) [7], which has the potential to fulfill these two requirements. In APG, differentiable simulators allow analytical calculation of the reward gradients, enabling direct training of a policy with backpropagation. By employing prior knowledge of the system in the form of the analytical gradients, APG was demonstrated to rely on one to four orders of magnitude less training data than RL to achieve similar performance [8], [9], [10], [11]. Since APG pre-trains a control policy, like model-free RL, it also allows fast execution at runtime and is flexibly applicable to even high-dimensional inputs such as images.

Nevertheless, APG faces several challenges when employed for robotic systems. Firstly, to train a policy with APG, it is often required to propagate gradients backwards over several time steps, to when the causing action was applied. This paradigm is termed backpropagation-through-time (BPTT) in APG and, similar to training RNNs, it is known to cause vanishing or exploding gradients [12]. Secondly, APG is prone to get trapped in local minima [7]. Both problems may prohibit training of a robust and accurate policy. Current implementations have thus focused on MuJoCo environments [7], [13] or fluid simulations [14], whereas experiments on complex robotic systems are still rare. Furthermore, APG is usually only compared to RL and not to other performant robotics control approaches. Detailed experiments are therefore still necessary to determine how its performance compares to established optimization-based control techniques, such as MPC [13].

In this work, we demonstrate the potential of APG as an efficient and accurate controller for flying vehicles. With a

*These authors contributed equally and are listed alphabetically. [1] Robotics and Perception Group, University of Zurich, Switzerland. [2] Laboratory of Intelligent Systems, École Polytechnique Fédérale de Lausanne (EPFL). [3] Embodied AI Lab, Intel. This work was supported by the Swiss National Science Foundation through the National Centre of Competence in Research (NCCR)

receding-horizon training and a curriculum learning strategy, our APG controller is able to learn trajectory-tracking tasks offline for three diverse problems: (1) balancing an inverted pendulum on a cart (standard task in RL known as CartPole), (2) navigating towards a 3D target with a fixed-wing aircraft [15], and (3) tracking a trajectory with a quadrotor. In contrast to previous work, we systematically compare APG to both learning-based approaches as well as classical control, and demonstrate its advantages in terms of runtime, tracking performance, sample efficiency for adaptation scenarios, and input flexibility, i.e. vision-based control.

Our APG controller achieves similar or better tracking performance than commonly used model-based and model-free RL algorithms. It also achieves similar performance as MPC while reducing computation time by more than an order of magnitude. This demonstrates the potential of APG and addresses the problem of unstable training, enabling APG to be applied in challenging robot control tasks, where computation time for deployment and flexibility to sensor inputs is important.

II. Background and Related Work

We first introduce a framework describing the problem setting mathematically. We situate previous works within this framework and highlight the differences to our approach.

A. Problem setting

In contrast to the traditional RL setting, we target fixed-horizon problems where a reference trajectory is given as $\{\nu_t\},\ \nu_t \in S$ is to be tracked accurately. Thus, we do not maximize a reward but minimize the Mean Squared Error between states and a reference trajectory. We define the task as a discrete time, continuous-valued optimization problem:

$$\min_{\pi} J^*(\pi) = \min_{\pi} \mathbb{E}_{(s_t,\nu_t)\sim\rho(\pi)}[C(f\big(s_t, \pi(s_t,\nu)\big),\nu_t)], \tag{1}$$

where C is a cost depending on a given reference state $\nu_t \in S$ and the robot state $s_t \in S$; $\rho(\pi)$ is the distribution of possible state-reference pairs $\{(s_0,\nu_0),\dots,(s_t,\nu_t)\}$ induced by the policy π. The state at the next time step is given by the dynamics model f with $s_{t+1} = f(s_t, \pi(s_t,\nu))$. The robot state is initialized to the provided reference ($s_0 = \nu_0$).

As typical in the control literature [16], we define total cost C as the sum of the state cost (ℓ^p-norm of the difference between the next state and the reference) and the control cost (ℓ^p-norm of the actions) at time t, i.e.

$$\begin{aligned} &C(f\big(s_t, \pi(s_t,\nu)\big),\nu_t) \\ &= \|f\big(s_t, \pi(s_t,\nu)\big) - \nu_{t+1}\|_p + \|\pi(s_t,\nu)\|_p \ . \end{aligned} \tag{2}$$

B. Model-free RL

Observation 1: $J^*(\pi)$ can be optimized directly with respect to π by Monte-Carlo sampling to obtain empirical estimates of the gradients. This does not require f to be differentiable. In turn, it requires extensive sampling, especially for long-term planning. This approach is used by both model-free RL [17], [18] and some specific instances of model-based RL [19], [20], [21], [22].

C. Online optimization and model-based RL

Observation 2: Many model-based RL algorithms [23], [24], [25] optimize the nonlinear controller cost in Equation 1. They limit the horizon T, reducing the computational load and allowing online optimization with respect to actions, e.g., via iterative Linear Quadratic Regulator (iLQR) method or MPC. A drawback of such methods is that the optimization is calculated online, which is potentially both prohibitively slow and difficult to solve. This is especially true for non-convex dynamics or dynamics with a high dimensional state space.

D. Differentiable programming (DP)

Observation 3: If f and π are differentiable, $J^*(\pi)$ can be minimized directly via gradient descent.

Such end-to-end differentiable training is also known as training with Analytical Policy Gradients (APG) [7]. Neural ODEs [19] have inspired APG approaches for control, and several differentiable physics simulators were developed, such as DifftTaichi [8], DiffSim [11], or Deluca [26]. Gradu et al. [26] also demonstrate APG applications for control tasks and include a planar quadrotor model in their experiments. Other APG approaches were proposed for solving non-linear system identification and optimal control [27], [28] which outperformed state-of-the-art control methods. Jin et al. propose a framework to solve complex control tasks using (deep) learning techniques [27]. They find that this approach improves on state-of-the-art methods in solving non-linear system identification and optimal control.

When APG is used for trajectory tracking with a fixed-length horizon, it is closely related to a paradigm termed Backpropagation-Through-Time (BPTT), which has been extensively studied [27], [29], [30], [31], [32], [33]. For instance, Bakker et al. [34] use BPTT with Recurrent Neural Networks (RNN) for offline policy learning [34]. Schaefer et al. [35] and Wierstra et al. [36] apply RNN- or LSTM-based versions of BPTT on the CartPole task and already demonstrated the sample-efficiency of this method. However, BPTT has limitations. Exploding and vanishing gradients can occur when training a control policy, as discussed by Metz et al. [37] in their recent work.

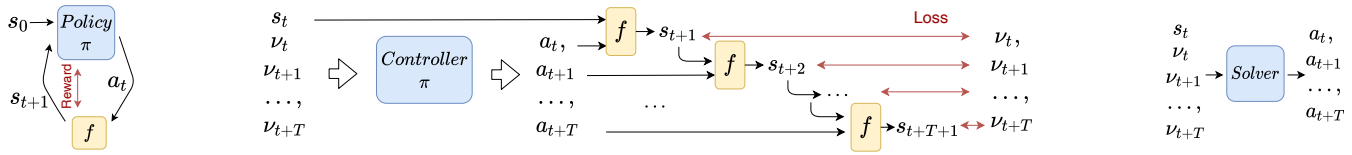
We apply a modified version of BPTT to control problems. Specifically, we tackle numerical training issues with curriculum learning and we explore different strategies to train with a fixed horizon. The method is explained in detail in the following, and is contrasted with MPC and RL in Figure 1.

III. Methods

In Analytic Policy Gradient learning, we use gradient descent to train the policy π with respect to the dynamics model f. Specifically, we minimize a loss function $L(\theta)$ that approximates $J^*(\pi)$ on a dataset $\mathcal{D}$:

$$L(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s,\nu)\in\mathcal{D}} \sum_{t=0}^{T} C(f\big(s_t, a_t\big),\nu_t), \tag{3}$$

where θ are the parameters of π, $\mathcal{D}$ is a set of state-reference pairs collected from interaction with f, and we refer to



(a) RL (b) APG-based training (concurrent) (c) MPC

Fig. 1: Comparison of controller training methods: Our APG controller does not require a solver, but minimizes the costs directly via gradient descent instead of maximizing a black-box reward. With the concurrent training strategy, T actions a are predicted simultaneously. Based on this loss, the parameters of π are updated with backpropagation through time.

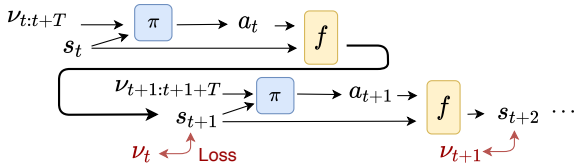


Fig. 2: Training an APG controller with an autoregressive approach: Instead of predicting the next T actions a at once, we predict and apply one action per time step and feed the resulting state back into the network. We thereby backpropagate through the policy network multiple times.

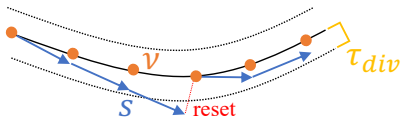


Fig. 3: Curriculum learning, where we reset the state s onto the trajectory ν if the error exceeds the threshold τ_{div}.

$a_t = \pi(s_t, \nu)$ as the action at step t. In contrast to MPC or iLQR, we can optimize π *offline* on the entire interaction data $\mathcal{D}$. This optimization is done with stochastic optimization algorithms. To favor short-term planning, we train π in a receding-horizon fashion. Analogous to MPC, the controller predicts T actions and computes the loss on the divergence of the resulting T subsequent states from the reference trajectory. The reference states in the horizon are input to the policy network at all steps, i.e. $a_t = \pi(s_t, \nu_t, \nu_{t+1}, \dots, \nu_{t+T})$.

A. Backpropagation over a horizon

At test time only one action is applied at a time. There are, however, two possibilities to train over a horizon: 1) At time t, the policy outputs T actions *concurrently*, and they are applied subsequently (Figure 1b). The action at step t is given by $a_t = [\pi(s_0, \nu_{0:T})]_t$ where $\nu_{0:T}$ denotes the T reference states within the horizon and $[\cdot]_t$ is the t-th component of an output vector. Here, BPTT passes multiple times through f, but only once through π. 2) On the other hand, an action can be predicted in *autoregressive* manner and applied in each time step, as shown Figure 2. The action at time t is $a_t = \pi(s_t, \nu_{t:t+T})$. After T steps, the loss is computed with respect to the reference trajectory and the gradients are backpropagated through the whole chain of T policy-network predictions and T times the dynamics, resulting in a longer chain of gradients.

To the best of our knowledge, only the latter approach has been analyzed in the literature. However, it is well known that BPTT with this paradigm leads to vanishing or exploding gradients [12] and was found to potentially lead to performance limitations of the policy [37]. To improve training stability, we propose the concurrent controller and provide a systematic comparison.

B. Curriculum learning

Nevertheless, large displacements of the states with respect to the reference trajectory can occur and increase the loss. Training with such a loss function is initially unstable and may end in a local optimum. We thus additionally propose a curriculum learning strategy to improve training, as depicted in Figure 3. We set a threshold τ_{div} to the divergence that is applied only when collecting the dataset $\mathcal{D}$. We run the partially-trained controller to collect new training data, but whenever $\|\hat{s}_{t+1} - \nu_{t+1}\| > \tau_{div}$ after executing action a_t, the agent is reset to ν_{t+1}. The threshold τ_{div} is then increased over time. Therefore, the robot will learn to follow the reference from gradually more distant states as training proceeds.

IV. Experiments

We compare our approach to both classical control and reinforcement learning algorithms on three diverse tasks: CartPole balancing, trajectory tracking with a quadrotor, and flying to a 3D target with a fixed-wing aircraft (see Figure 4). We design our evaluation procedure to answer the following questions: (1) How does our proposed APG method compare to commonly used RL and MPC algorithms? (2) Are the measures of limiting the backpropagation chain length and introducing the curriculum improving training stability? (3) Can the advantages of APG be leveraged to deal with high-dimensional input data (e.g. images) or to improve the sample efficiency for adaptation tasks?

A. Experimental setup

As a low-dimensional state space task, we test APG algorithm on the CartPole balancing task where a pole must be balanced by moving a cart. We use a model derived from first principles [38] as dynamic model f, and allow for continuous actions in the range $[-1, 1]$ corresponding to ± 30 N force on the cart. Since the reference trajectory only consists of a single state (i.e. upright position of the pole), we linearly

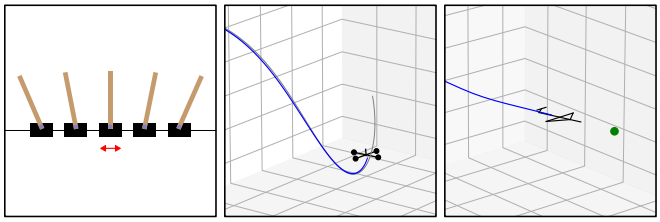


Fig. 4: Experiments are conducted on three diverse dynamic systems and tasks: Balancing a pole on a cart (left), tracking a trajectory with a quadrotor (center) and passing through a target point (green) with a fixed wing drone (right).

interpolate from the current state to the desired end state (upright and zero velocity) to generate a continuous reference $[\nu_{t+1}, \ldots, \nu_{t+k}]$.

The second task consists of tracking a 3D trajectory with a quadrotor. We use the first-principles model provided by the Flightmare simulator [39] as the dynamic model f. For training and testing, we generate random polynomial trajectories with an average speed of 3 $\frac{\text{m}}{\text{s}}$. We evaluate performance based on 50 test trajectories that were not seen during training.

Finally, for reaching a 3D target with a fixed-wing aircraft, we implement a first-order model as described by Beard et al. [40]. 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 a forward velocity of 11.5 $\frac{\text{m}}{\text{s}}$ in the x-direction, and is tasked to reach a target at a distance of $x = 50$ m, with y and z deviations from the initial position uniformly sampled from $[-5, 5]$ 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, however maintaining constant forward speed. Our approach works despite this approximation of the reference trajectory which may be infeasible. We perform all experiments using PyTorch [41]; see the Appendix for technical details on model definition and training. We publish the source code on the project website to reproduce our experiments, and provide example videos of the APG trained policies on all three tasks in the supplementary material.

B. Comparison to model-based RL, model-free RL, and MPC

We compare our best-performing APG controller to three strong baselines (1) the model-free RL algorithm Proximal Policy Optimization (PPO) [17], (2) the model-based RL algorithm termed probabilistic-ensembles-with-trajectory-sampling (PETS) [42], and (3) model predictive control (MPC). The PPO baseline is available from the `stable-baselines` PyPI package [43]. The PETS implementation was taken from the `mbrl` toolbox [44]. The MPC is implemented as a non-linear programming solver with `Casadi` [45] and is provided in the supplementary code.

	MPC	Learning-based control PPO	PETS	Ours
CartPole				
Cart velocity (in m/s)	0.01±0.03	**0.04 ±0.04**	0.24 ±0.10	0.05 ±0.05
Runtime (in ms)	4.9	0.28	2200	**0.16**
Quadrotor				
Tracking error (in m)	0.03 ±0.01	0.36 ±0.10	unstable	**0.05 ±0.01**
Runtime (in ms)	5.2	0.31	4919	**0.17**
Fixed-wing				
Tracking error (in m)	0.0 ±0.0	0.09 ±0.05	0.24 ±0.15	**0.01 ±0.01**
Runtime (in ms)	11.4	0.25	2775	**0.23**

TABLE I: Tracking performance and runtimes for classical and learning-based control methods. The best learning-based results are marked bold. While MPC has very low tracking error, differentiable control is the only learning-based method that achieves comparable performance. At the same time, it has significant advantages in terms of computational efficiency at inference time.

All methods have access to the same inputs, produce the same type of output, are tuned individually with grid-search on a held-out validation set, and the policies are trained with the same curriculum.

We train and test the methods described above on all three systems, and evaluate the performance in tracking randomly-sampled polynomial reference trajectory. For the CartPole system, we evaluate the performance based on the stability of the cart, i.e. how fast the system moves along the x-axis. For the quadrotor and fixed-wing tasks, performance is evaluated with the average tracking error (in m). The results of these experiments are shown in Table I.

From the table, we can clearly see that classical control with non-linear optimization achieves the best performance in terms of tracking error. This is to be expected, given the availability of the model dynamics and an (approximate) reference trajectory. The major drawback of MPC is its computational load at runtime, with a calculation time of several milliseconds for each time step. In flying vehicles, the runtime is a major limitation and has therefore received much attention in the literature [4], [15], [46], [47]. Similarly, model-based RL methods, such as PETS, also employ online-optimization to compute the next action and thus suffer from the same problem. The runtime (and the performance) for PETS - and presumably other model-based RL methods - are worse due to the assumption of a black box model, which demands an approximation of the system dynamics with a complex learnt function. For the quadrotor tracking task, the PETS model only converged in a simplified scenario, namely tracking with lower speed (≈ 1 $\frac{\text{m}}{\text{s}}$) and only on a single reference trajectory, which still results in a tracking error of 0.18 m.

On the other hand, model-free RL such as PPO is fast at inference time, but leads to much larger tracking error, with the exception of the CartPole system. Overall, APG offers a way to achieve very low tracking error while maintaining the low inference runtime of learning-based methods like PPO. All times were computed on a single CPU. While

better implementations could increase the performance of optimization-based methods, our approach stands out as a method that achieves both low tracking error, low sample complexity, and a low runtime.

C. Validation of receding-horizon training and curriculum

1) Training with receding-horizon: Furthermore, we provide a systematic comparison of the training strategies discussed in Section III, namely training with a receding horizon by predicting all actions at once (*concurrent*) or sequentially (*autoregressive*), see Figure 1b and Figure 2 respectively. Furthermore, we test an autoregressive model with memory, using an LSTM[1]. We evaluate these methods with varying horizon length T in the task of trajectory tracking with a quadrotor, and evaluate performance in terms of tracking success and tracking error. Tracking success is defined as the ratio of trajectories that was followed successfully without diverging more than 5 m from ν, whereas the average tracking error is only computed on the successful tracking runs.

The experiments reported in Table II show that if the horizon is too short, i.e. below 5 steps, APG is not able to look far enough into the future to achieve a desirable trajectory tracking performance. However, with a long horizon, i.e. above 12 steps, the performance degrades again. We conjecture that this is caused by the increasing length of the gradient chains, which lead to poorly tractable gradients. Interestingly, the autoregressive approach works well with longer horizon, but generally converges to worse tracking error. Adding memory in from of an LSTM does not improve performance for long horizon lengths (8-15) and fails when the horizon length is short (1-5). We hypothesize that this is due to the Markov property of the task. Given the better performance of our new approach with a horizon of $T = 10$, we use this model for all other experiments.

2) Curriculum learning: We verify the importance of our curriculum learning for the training stability of APG by training the model with and without the curriculum explained in Section III-B. During training we evaluate the control performance after each epoch by tracking 100 trajectories until the quadrotor is either too far away from the trajectory (5 m) or has reached the end of the trajectory.

In Figure 5 we report the tracking error throughout the training. We note three things: (1) Training with curriculum leads to a significant reduction of the tracking error, while without it does not. (2) There are jumps in the tracking error of the training without curriculum, which may be caused by unstable training. (3) There is minimal change between episodes 95 and 15 in the training without curriculum, which may have been caused by a local minimum. Based on these observations, we argue that the introduction of curriculum learning stabilizes APG training and reduces the chance of ending in a local minimum.

[1]All training details for the three methods are provided in the code repository at https://github.com/lis-epfl/apg_trajectory_tracking/blob/main/training_details.pdf

	Tracking error (in m)			Tracking success		
T	Con-current	Auto-regressive	Recurrent (LSTM)	Con-current	Auto-regressive	Recurrent (LSTM)
1	–	–	–	0.00	0.00	0.00
3	2.43 ± 0.00	**1.57** ± 0.27	–	0.01	0.01	0.00
5	**0.10** ± 0.05	0.15 ± 0.11	–	0.91	**0.98**	0.00
8	**0.07** ± 0.05	0.09 ± 0.04	0.30 ± 0.09	**0.99**	0.97	0.92
10	**0.05** ± 0.01	0.12 ± 0.04	0.18 ± 0.07	**0.99**	0.98	0.97
12	**0.11** ± 0.06	0.17 ± 0.06	0.48 ± 0.13	**0.99**	**0.99**	0.91
15	–	**0.28** ± 0.09	0.61 ± 0.18	0.00	**0.99**	0.76

TABLE II: Table showing the influence of the control horizon length (T) on the control performance. If the horizon is too short, the control lacks the ability to plan ahead; if it is too long, the errors become difficult to backpropagate.

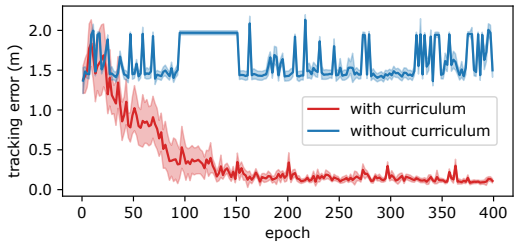


Fig. 5: Tracking performance with and without curriculum learning, where the line reports the average and shaded area the standard deviation. The curriculum leads to a stable learning process.

D. Vision-based control

In Table I, MPC achieved very low tracking errors in all tasks. However, besides its inferiority in terms of runtime, a shortcoming of classical control algorithms is their inability to deal with high-dimensional inputs such as images. In contrast, here, we can demonstrate the ability of APG to learn a control policy for the CartPole task with only images as inputs.

We use a 4-layer convolutional network with kernel size 3 and ReLU activations as policy. Instead of the 4-dimensional CartPole state, we input a sequence of four images of the CartPole in 3rd-person view (see Figure 4). This sequence allows the network to infer the current angular pole direction and velocity.

Table III shows the results. In this case, online optimization in the image space either does not converge or returns poor solutions. In contrast, our method and the model-free baseline (PPO) both obtain very good performance. However, our approach can train 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.

E. Exploring the sample efficiency in adaptation tasks

In contrast to other learning-based methods, APG is highly sample-efficient. This is particularly relevant for adaptation scenarios, where we need to learn from real-world examples. For example, consider a scenario where the dynamics of a

	MPC	PPO	Ours
Vision-based CartPole Cart velocity (in $\frac{\text{m}}{\text{s}}$)	unstable	0.28 $\pm$0.23	0.15 $\pm$0.11

TABLE III: Image-based control and identification for the task of CartPole balancing. Our approach outperforms the baselines in terms of tracking performance.

quadrotor in simulation (source domain f) *differ* from the dynamics in reality (target domain f^*). The sample size to adapt to the new dynamics should be kept as low as possible. We evaluate the adaptation performance of APG in a benchmark problem for flying vehicles, which is to estimate and counteract a linear translational drag acting on the platform [48], [49]. This force is modeled as $\dot{v} \longleftarrow \dot{v} - r \cdot v$ where v is the three dimensional velocity and $r = 0.3\ \frac{1}{\text{s}}$ the drag factor.

Assuming that the reason of the change in dynamics (i.e. the drag) is unknown, we propose to train a neural network Δ_ϕ that acts as a residual on top of the original dynamics model f, essentially learning the difference between f and target f^*. The residual is trained on a small dataset $\mathcal{D}_{\text{dyn}}$ of state-action-state triples:

$$\mathcal{D}_{\text{dyn}} = \{(s_t, a_t, s^*_{t+1}) \mid t \in [1..B],\ s^*_{t+1} = f^*(s_t, a_t)\}, \quad (4)$$

$\mathcal{D}_{\text{dyn}}$ is used to minimize the difference of source and target domain by training a state-residual network Δ_ϕ with the following loss function:

$$L_{\text{dyn}}(\phi) = \sum_{\mathcal{D}_{\text{dyn}}} \| f(s_t, a_t) + \Delta_\phi(s_t, a_t) - s^*_{t+1} \|, \quad (5)$$

where ϕ are the parameters of the network Δ_ϕ that accounts for effects that are not modeled in f. 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.

An example is given in Figure 6. After the dynamics Δ_ϕ are trained with samples from f^* (red), the pre-trained APG controller is fine-tuned and converges with few samples. Table IV provides the results upon fine-tuning pre-trained policies on dynamics with velocity drag. Our method, i.e. training a residual on a differentiable dynamics model and fine-tuning a differentiable learnt policy, requires orders of magnitudes fewer samples than RL methods that assume a black-box dynamics model.

V. Conclusion

APG algorithms have the potential to rely on little training data and allow fast computation during runtime, bridging the gap between RL and MPC. We proposed a receding-horizon implementation of APG with a curriculum learning scheme that improves training stability and enables the application of APG on more complex and longer-horizon problems. Through controlled experiments on a CartPole, a quadrotor, and a fixed-wing task, we showed that our approach outperforms both commonly used model-free and model-based RL algorithms

Task (#samples)	PPO	PETS	Ours
Quadrotor			
Source domain f	0.36 $\pm$ 0.10	unstable	**0.05** $\pm$ **0.01**
Zero-shot to f^*	**0.42** $\pm$ **0.09**	unstable	0.56 $\pm$ 0.07
Few-shot to f^* (1K)	0.42 $\pm$ 0.09	unstable	**0.07** $\pm$ **0.01**
Many-shot to f^* (150K)	0.40 $\pm$ 0.12	unstable	–
Fixed-wing			
Source domain f	0.09 $\pm$ 0.05	0.24 $\pm$ 0.15	**0.01** $\pm$ **0.01**
Zero-shot to f^*	0.43 $\pm$ 1.45	8.39 $\pm$ 10.15	**0.07** $\pm$ **0.10**
Few-shot to f^* (2K)	0.34 $\pm$ 0.11	5.86 $\pm$ 5.86	**0.02** $\pm$ **0.03**
Few-shot to f^* (5K)	0.17 $\pm$ 0.07	0.27 $\pm$ 0.14	–
Many-shot to f^* (150K)	0.09 $\pm$ 0.06	–	–

TABLE IV: Adaptation to velocity drag. Our APG method converges to the initial performance with few samples, whereas other RL approaches require significantly more training.

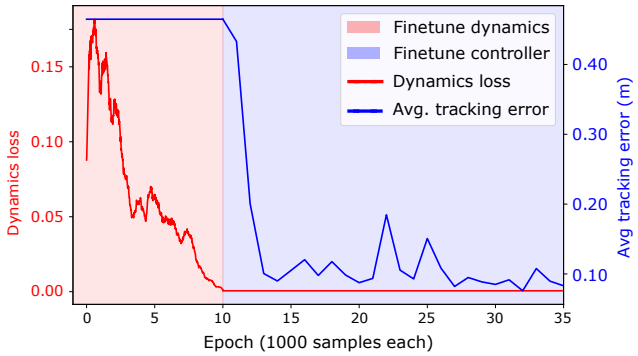


Fig. 6: Fine-tuning an APG controller for adaptation. A residual dynamics model is trained (red) and the controller is fine-tuned (blue). It converges with few samples.

in terms of tracking error. We showed that the tracking performance is comparable to MPC, while reducing the computation time on deployment by more than an order of magnitude. Finally, we highlight the advantages of learning-based control by applying our APG controller on a vision-based control task, and demonstrate the sample efficiency of APG when adaptating to new environments.

We note that our problem formulation as a fixed-horizon tracking task implies two limitations. Firstly, it is only demonstrated on control tasks with a relatively short horizon (up to 5 s in the future). A long horizon is in principle possible but may lead to training instability (e.g. exploding gradients). Secondly, we tested our framework on settings with (a) known dynamics and (b) given reference trajectory only. Future research on real robots will therefore be necessary and may lead to additional insights. However, this paper's evaluation of common robotics tasks in simulation, provides initial evidence that the presented approach works with highly non-realistic references (e.g. linear trajectory in fixed-wing flight), and with learned dynamics. We thus consider gradient-based policies, in particular, combined with curriculum learning schemes, promising for various control tasks. It offers a good trade-off between performance, efficiency, and flexibility, and should receive more attention in the control field.

ACKNOWLEDGMENTS

We thank Elia Kaufmann, René Ranftl, and Yunlong Song for the fruitful discussion throughout the project.

REFERENCES

[1] Y. Li, H. Li, Z. Li, H. Fang, A. K. Sanyal, Y. Wang, and Q. Qiu, "Fast and accurate trajectory tracking for unmanned aerial vehicles based on deep reinforcement learning," pp. 1–9, 2019.

[2] R. Yu, Z. Shi, C. Huang, T. Li, and Q. Ma, "Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle," pp. 4958–4965, 2017.

[3] T. Salzmann, E. Kaufmann, M. Pavone, D. Scaramuzza, and M. Ryll, "Neural-mpc: Deep learning model predictive control for quadrotors and agile robotic platforms," *arXiv preprint arXiv:2203.07747*, 2022.

[4] D. Wang, Q. Pan, Y. Shi, J. Hu *et al.*, "Efficient nonlinear model predictive control for quadrotor trajectory tracking: Algorithms and experiment," *IEEE Transactions on Cybernetics*, vol. 51, no. 10, pp. 5057–5068, 2021.

[5] M. Abdolhosseini, Y. M. Zhang, and C. A. Rabbath, "An efficient model predictive control scheme for an unmanned quadrotor helicopter," *Journal of intelligent & robotic systems*, vol. 70, no. 1, pp. 27–38, 2013.

[6] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1205–1212.

[7] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax-a differentiable physics engine for large scale rigid body simulation," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[8] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "Difftaichi: Differentiable programming for physical simulation," in *International Conference on Learning Representations*, 2020.

[9] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," *Advances in neural information processing systems*, vol. 31, 2018.

[10] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt, "A differentiable programming system to bridge machine learning and scientific computing," *arXiv preprint arXiv:1907.07587*, 2019.

[11] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin, "Scalable differentiable physics for learning and control," *International Conference on Machine Learning*, 2020.

[12] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, 1994.

[13] S. Gillen and K. Byl, "Leveraging reward gradients for reinforcement learning in differentiable physics simulations," *arXiv preprint arXiv:2203.02857*, 2022.

[14] P. Holl, N. Thuerey, and V. Koltun, "Learning to control pdes with differentiable physics," in *International Conference on Learning Representations*, 2020.

[15] P. Oettershagen, A. Melzer, S. Leutenegger, K. Alexis, and R. Siegwart, "Explicit model predictive control and l 1-navigation strategies for fixed-wing uav path tracking," *IEEE 22nd Mediterranean Conference on Control and Automation*, pp. 1159–1165, 2014.

[16] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[19] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *NeurIPS*, 2018.

[20] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[21] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *International Conference on Learning Representations*, 2019.

[22] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2555–2565.

[23] J. Fu, S. Levine, and P. 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)*. IEEE, 2016, pp. 4019–4026.

[24] Y. Deng, Y. Zhang, X. He, S. Yang, Y. Tong, M. Zhang, D. DiPietro, and B. Zhu, "Soft multicopter control using neural dynamics identification," in *Conference on Robot Learning*. PMLR, 2021, pp. 1773–1782.

[25] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 4653–4660.

[26] P. Gradu, J. Hallman, D. Suo, A. Yu, N. Agarwal, U. Ghai, K. Singh, C. Zhang, A. Majumdar, and E. Hazan, "Deluca–a differentiable control library: Environments, methods, and benchmarking," *arXiv preprint arXiv:2102.09968*, 2021.

[27] W. Jin, Z. Wang, Z. Yang, and S. Mou, "Pontryagin differentiable programming: An end-to-end learning and control framework," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7979–7992, 2020.

[28] J. K. Murthy, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Lévesque, K. Xie, K. Erleben *et al.*, "gradsim: Differentiable simulation for system identification and visuomotor control," in *International Conference on Learning Representations*, 2021.

[29] R. Grzeszczuk, D. Terzopoulos, and G. 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*, 1998, pp. 9–20.

[30] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the International Conference on Machine Learning*, 2011, pp. 465–472.

[31] P. Parmas, "Total stochastic gradient algorithms and applications in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[32] J. Degrave, M. Hermans, J. Dambre *et al.*, "A differentiable physics engine for deep learning in robotics," *Frontiers in neurorobotics*, p. 6, 2019.

[33] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," in *Conference on Robot Learning*, 2018, pp. 617–629.

[34] B. Bakker, V. Zhumatiy, G. Gruener, and J. Schmidhuber, "A robot that reinforcement-learns to identify and memorize important previous observations," vol. 1, pp. 430–435, 2003.

[35] A. M. Schaefer, S. Udluft, and H.-G. Zimmermann, "A recurrent control neural network for data efficient reinforcement learning," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. IEEE, 2007, pp. 151–157.

[36] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber, "Recurrent policy gradients," *Logic Journal of the IGPL*, vol. 18, no. 5, pp. 620–634, 2010.

[37] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman, "Gradients are not all you need," *arXiv preprint arXiv:2111.05803*, 2021.

[38] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[39] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Conference on Robot Learning*. PMLR, 2021, pp. 1147–1157.

[40] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012, pp. 16, 44ff.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, pp. 8026–8037, 2019.

[42] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[43] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto,

and N. Dormann, "Stable baselines3," https://github.com/DLR-RM/stable-baselines3, 2019.
[44] L. Pineda, B. Amos, A. Zhang, N. O. Lambert, and R. Calandra, "Mbrl-lib: A modular library for model-based reinforcement learning," *arXiv preprint arXiv:2104.10159*, 2021.
[45] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.
[46] T. J. Stastny, A. Dash, and R. Siegwart, "Nonlinear mpc for fixed-wing uav trajectory tracking: Implementation and flight experiments," in *AIAA guidance, navigation, and control conference*, 2017, p. 1512.
[47] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 690–697, 2021.
[48] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2017.
[49] A. Franchi, R. Carli, D. Bicego, and M. Ryll, "Full-pose tracking control for aerial robotic systems with laterally bounded input force," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 534–541, 2018.
[50] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5745–5753.
[51] A. Waldock, C. Greatwood, F. Salama, and T. Richardson, "Learning to perform a perched landing on the ground using deep reinforcement learning," *Journal of Intelligent & Robotic Systems*, vol. 92, no. 3, pp. 685–704, 2018.

APPENDIX

HYPERPARAMETERS AND TRAINING DETAILS

We tested our method on three dynamic systems that are visualized in Figure 4: the CartPole balancing task, trajectory tracking with a quadrotor and point-goal navigation with a fixed-wing aircraft. 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. 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 0.05 s. 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 30 N force to the left) and 1 (force of 30 N 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}.

B. 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 [39]. 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 0.1 s.
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 $3 - 5\ \frac{m}{s}$, whereas the average velocity is $1 - 2\ \frac{m}{s}$.

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 [50]), 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 2.21 N and 17.31 N (such that an output of 0.5 corresponds to 9.81 N) and body rates between -0.5 and 0.5.

Loss function. The loss (equation 3) is a weighted MSE between the reached states and the reference states. The weights are aligned to the ones used for the optimization-based MPC, 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 actual

angular velocity. Formally, these weights yield the following loss:

$$L = \sum_{k=1}^{10} 10 \cdot (x_{t+k,\pi} - x_{t+k,\nu})^2 + (\dot{x}_{t+k,\pi} - \dot{x}_{t+k,\nu})^2 + 5 \cdot (T_k - 0.5)^2 + 0.1 \cdot (\omega_{k,des} - 0.5)^2 + 0.1 \cdot \omega_{t+k} \quad (6)$$

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 a learning rate of 10^{-5} and momentum of 0.9.

Curriculum learning. As explained in Section III, we use a curriculum learning strategy with a threshold τ_{div} on the allowed divergence from the reference trajectory. We set $\tau_{div} = 0.1$ m initially and increase it by 0.05 m every 5 epochs, until reaching 2 m. 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 0.1 m. This is repeated to train at the full speed in the third iteration.

C. Fixed-wing drone

We implemented a realistic model of the dynamics based on the equations and parameters described in [40] and [51]. In our discrete-time formulation, we set the time step to 0.05 s.

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 to 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 $a_1 \in [-20, 20]$ °, aileron angle $a_2 \in [-2.5, 2.5]$ ° and rudder angle $a_3 \in [-20, 20]$ °.

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

$$L = \sum_{k=1}^{10} 10 \cdot (x_{t+k,\pi} - x_{t+k,\nu})^2 + 0.1 \cdot ((a_{k,1} - 0.5)^2 + (a_{k,2} - 0.5)^2 + (a_{k,3} - 0.5)^2) \ , \quad (7)$$

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 a learning rate of 10^{-4} and momentum of 0.9.

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