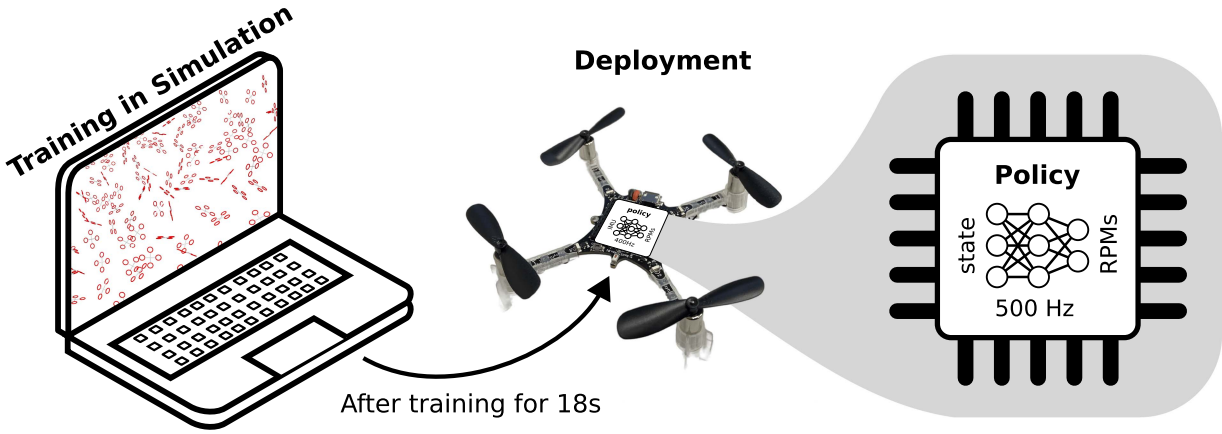
Learning to Fly in Seconds

Jonas Eschmann, Dario Albani, and Giuseppe Loianno, *Member, IEEE*



***Abstract*—Learning-based methods, particularly Reinforcement Learning (RL), hold great promise for streamlining deployment, enhancing performance, and achieving generalization in the control of autonomous multirotor aerial vehicles. Deep RL has been able to control complex systems with impressive fidelity and agility in simulation but the simulation-to-reality transfer often brings a hard-to-bridge reality gap. Moreover, RL is commonly plagued by prohibitively long training times. In this work, we propose a novel asymmetric actor-critic-based architecture coupled with a highly reliable RL-based training paradigm for end-to-end quadrotor control. We show how curriculum learning and a highly optimized simulator enhance sample complexity and lead to fast training times. To precisely discuss the challenges related to low-level/end-to-end multirotor control, we also introduce a taxonomy that classifies the existing levels of control abstractions as well as non-linearities and domain parameters. Our framework enables Simulation-to-Reality (Sim2Real) transfer for direct Revolutions Per Minute (RPM) control after only 18 seconds of training on a consumer-grade laptop as well as its deployment on microcontrollers to control a multirotor under real-time guarantees. Finally, our solution exhibits competitive performance in trajectory tracking, as demonstrated through various experimental comparisons with existing state-of-the-art control solutions using a real Crazyflie nano quadrotor. We open source the code including a very fast multirotor dynamics simulator that can simulate about 5 months of flight per second on a laptop GPU. The fast training times and deployment to a cheap, off-the-shelf quadrotor lower the barriers to entry and help democratize the research and development of these systems.**

***Index Terms*—Aerial systems: applications, machine learning for robot control, reinforcement learning.**

Manuscript received 18 November 2023; accepted 3 April 2024. Date of publication 1 May 2024; date of current version 31 May 2024. This letter was recommended for publication by Associate Editor E. Johns and Editor A. Faust upon evaluation of the reviewers' comments. This work was supported in part by the Technology Innovation Institute, the NSF CAREER under Grant 2145277, and in part by the DARPA YFA under Grant D22AP00156-00. *(Corresponding author: Jonas Eschmann.)*

Jonas Eschmann is with the New York University, Tandon School of Engineering, Brooklyn, NY 11201 USA, and also with the Autonomous Robotics Research Center, Technology Innovation Institute, Abu Dhabi, UAE (e-mail: jonas.eschmann@nyu.edu).

Dario Albani is with the Autonomous Robotics Research Center, Technology Innovation Institute, Abu Dhabi, UAE (e-mail: dario.albani@tii.ae).

Giuseppe Loianno is with the New York University, Tandon School of Engineering, Brooklyn, NY 11201 USA (e-mail: loiannog@nyu.edu).

Giuseppe Loianno serves as consultant for the Technology Innovation Institute. This arrangement has been reviewed and approved by the New York University in accordance with its policy on objectivity in research.

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2024.3396025, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3396025

Supplementary Material

Video: https://youtu.be/NRD43ZA1D-4
Code: https://github.com/arplaboratory/learning-to-fly
Parameters: {Code} → ./media/parameters.pdf

I. Introduction

With the availability of cheap Commercial Off-The-Shelf (COTS) gyroscopes and accelerometers that are implemented as Microelectromechanical Systems (MEMS), the large-scale production of cheap Unmanned Aerial Vehicles (UAVs), particularly quadrotors, became viable. Bearing Vertical Take-Off and Landing (VTOL) as well as hovering capabilities a myriad of use cases, such as search and rescue, infrastructure inspection, or package delivery emerged. Leveraging classical, cascaded control hierarchies, multirotors are able to perform a variety of tasks. However, these control approaches require domain expertise and engineering to be adapted to new platforms and use cases. At the same time, the recent developments in machine learning and particularly the success of deep learning for supervised tasks like image classification [1], [2], raise the question if these learning-based capabilities could be transferred to quadrotor control. In contrast to supervised learning,

TABLE I
TRAINING TIMES

Publication	Level	Time	Samples
2019 [27] †	4.2/5.1	30 h	84×10^6
2020 [28]	4.3	N/A	10×10^6
2021 [29]	4.3	$\sim$ 2 h	N/A
2021 [23]	2.1	27 m	1×10^6
2022 [30] †	4.2/5.1	1.2 h	16×10^6
2023 [25]	3.1	50 m	100×10^6
Ours	**5.1**	**18 s**	$\mathbf{0.3 \times 10^6}$

4.2/5.1: Policy outputs 4.2, simulation 5.1. †: Used as a baseline in Table III.

(multirotor) control is a decision-making problem that can be phrased as a Markov Decision Process (MDP) where labels usually do not directly exist. To solve MDPs, RL has been employed to train policies for complex end-to-end continuous control problems in simulation [3], [4], [5], [6]. However, while the results attained in simulation are impressive, transferring end-to-end control policies to real-world systems has proven challenging. This is mainly due to model inaccuracies, partial observation of the state, observation and action noise, and other disturbances. Additionally, RL is well known to be sensitive to the choice of hyperparameters and reward function design. The requirement for hyperparameter tuning and reward function design, in combination with long training times for end-to-end control as shown in Table I, can prohibit fast iteration and create a barrier to entry. To better highlight the challenges and clarify the scope of true end-to-end control, we describe the different levels of quadrotor dynamics and control by developing a taxonomy to categorize related work. We argue that the abstractions in classic control stacks introduce information loss (e.g., actuator constraints in differential flatness-based control) and constrain the expressivity of the control policy. Due to Bellman's Principle of Optimality (BPO), the optimal policy for a decision/control problem can be represented as a function mapping from states/observations to actions. Hence, in theory, the optimal policy can be expressed as a neural network which in the limit is a general function approximator [7]. Compared to general function approximators, classical control stacks (based on multi-level abstractions) have limited expressivity and can not necessarily represent the optimal policy, especially when accounting for evermore (non-linear) factors in the simulation. Optimization-based controllers can circumvent this limitation but are usually too computationally expensive to run end-to-end control under real-time constraints on microcontrollers. Hence, we believe it can be desirable to design an end-to-end controller using RL, directly mapping the quadrotor state to RPM outputs.

We observe that the end-to-end control of quadrotors using state-of-the-art deep RL techniques (especially using off-policy RL) is not well explored and documented. It is particularly unclear which level of performance end-to-end policies can achieve compared to classic controllers when deployed directly on a real quadrotor under real-time constraints. With this work, we aim to push the boundaries of deep RL-based end-to-end quadrotor control and present the following contributions

- *RL-based end-to-end controller design:* We propose a novel asymmetric actor-critic-based architecture coupled with a highly reliable training paradigm for true end-to-end quadrotor control (level **5.1** outputs, cf. Fig. 1 and Table I). The proposed training paradigm takes advantage of the ground truth available in the simulator while explicitly accounting for the partial observability of the real system using an action history.
- *Best sample complexity:* We devise a curriculum that gradually increases the penalties in the reward function leading to better sample complexity and more reliable policies. We show the benefit of the components in our proposed training paradigm by conducting an extensive ablation study containing 300 real-world trajectories across different configurations, seeds and tasks. In contrast to existing works, our training setup uses off-policy RL and we demonstrate the training of an end-to-end quadrotor control policy using the fewest number of environment interactions reported.
- *Fastest training time:* By implementing a highly optimized simulator, we demonstrate the fastest training of an end-to-end quadrotor control policy that can be transferred to a real system.
- *Sim2Real:* We conduct extensive experiments including more than 300 flights across configurations, seeds and tasks to test the Sim2Real transfer of end-to-end policies for direct RPM control. We show that training a position controller using our setup generalizes to other tasks like (agile) trajectory tracking.
- *Open Source*[1]*:* We open-source our setup to facilitate research, enabling everyone with a consumer-grade laptop to train and deploy state-of-the-art quadrotor control policies in a matter of seconds, greatly reducing the barriers to entry in this research area and therefore contributing to democratize the use of these approaches.

Difficulty & Uncertainty

0. **Position**
1. **Velocity**
2. **Acceleration**
   - 2.1 **Attitude/orientation & thrust**: Non-linear orientation $\rightarrow$ acceleration transfer function [mass]
3. **Jerk**
   - 3.1 **Angular rate & thrust (CTBR)**: Non-linear rotational kinematics
4. **Snap**
   - 4.1 **Body torque & thrust**: Rotational dynamics [inertia]
   - 4.2 **Individual rotor thrusts (SRT)**: [vehicle geometry]
   - 4.3 **RPMs**: Non-linear torque/thrust curves [torque/thrust model parameters]
5. **Crackle**
   - 5.1 **Motor commands/RPM setpoints**: [first-order low-pass time constant (motor delay)]
   - 5.2 **Motor effort**: [battery level]

Fig. 1. Taxonomy of multirotor dynamics and control.

[1]https://github.com/arplaboratory/learning-to-fly

II. Multirotor Dynamics and Control Taxonomy

In the following, we introduce a taxonomy classifying the different abstraction levels in multirotor dynamics and control. We believe this taxonomy to be beneficial to the discussion as it allows for the precise categorization of different controllers, particularly those that we analyze in the related work (see Section III). Furthermore, it explicitly exposes at which levels non-linearities and domain parameters exert an influence on the motion of multirotors. We list the different control levels based on the order of the system when expressed in terms of a flat state [8]. The taxonomy shown in Fig. 1 is expressed from the perspective of a position controller and each subsequent level denotes a lower level of control inputs. The sub-bullets denote non-linear transformations that allow expressing the dynamical system using different inputs. These transformations are not only challenging because of the non-linearity they bear but also due to the additional system/domain parameters they might introduce (marked in square brackets). It is worth noting that every layer incorporates an additional level of indirection, manifesting in the form of integrators. The lower the input level to the system is chosen, the more detached the effect on the higher level (e.g., the position) is from the cause (e.g., RPM setpoints). For controllers on the lowest levels, the cause-effect relationship traverses through up to five orders of integration and multiple non-linear transformations that are dependent on system parameters. We would like to highlight that **3.1 Angular rate & thrust** inputs are commonly referred to as "low-level" commands but from the taxonomy, we can see that there is only one domain parameter (mass) and otherwise just the rotational kinematics of a rigid body. Alternatively, **3.1 Angular rate & thrust** inputs are also commonly referred to as Collective Thrust and Body Rates (CTBR) inputs. Compared to the CTBR level, the reality gap and complexity of the system greatly increases towards the lowest levels like **5.1 Motor commands/RPM setpoints** which is the level of control used in this work. The high order of integration between inputs and the desired output (position) poses a great challenge for RL algorithms because the high-frequency exploratory actions (from ϵ-greedy-like exploration schemes) are already suppressed in the early layers and hence lead to high-sample complexity and unreliable training behavior. In our proposed architecture, we overcome this challenge by using a combination of off-policy RL, curriculum learning, and by scheduling the exploration noise. From Fig. 1, we can observe that the complexity of the control problem and the size of the reality gap, follows a superlinear scaling where most of the non-linear dynamics and system parameters can be found in the lower levels.

III. Related Works

Simulators: In the past, a large number of general robotics and specifically quadrotor simulators have been proposed. Many of these focus on visual fidelity and photo-realistic reproduction of the environment (e.g., to enable realistic RGB camera perception) [9], [10], [11], [12]. Others, as the one presented in this work, focus on the accurate implementation of quadrotor dynamics [11], [12], [13], [14]. Among the referenced simulators, Flightmare [11] is the most related because it emphasizes the simulation speed of the quadrotor dynamics for the sake of RL. Our simulator also focuses on fast dynamics but since we are concerned with low-level control we just provide a basic User Interface (UI) instead of the photo-realism that is offered by other simulators.

Reinforcement learning for quadrotor control: As described in Section II, it is usually easier to learn controllers on the higher levels, especially for Sim2Real transfer. Hence, a lot of work that focuses on downstream tasks has been using CTBR and higher-level control inputs to train RL based agents. The use of velocity commands has been particularly common [15], [16], [17], [18], [19], [20]. Fewer works also use orientation level commands [21], [22] or angular rate commands (CTBR) [20], [23], [24], [25]. The latter is usually used when increased agility is required (e.g., for acrobatics, racing, or flying through a narrow gap). We would like to highlight that, while impressive, recent work on learned, champion-level drone racing [25] is using higher-level control outputs (level **3.1 CTBR**) and heavily relies on classic lower-level controllers to bridge the Sim2Real gap. In their work, the policy is only exposed to rotational kinematics and uncertainty about the mass, which is easy to identify (cf. Table I, Fig. 1). Our end-to-end policy (level **5.1 Motor commands**) is faced with the full stack of non-linearities and uncertainties about the dynamics parameters.

To use the quadrotor dynamic's full potential, the focus has been shifting to training agents outputting individual rotor thrusts (also referred to as Single Rotor Thrusts (SRT)) [20], [26], [27], [28], [29], [30]. The authors in [31] even go lower-level, training a controller outputting RPM. These represent the works that are closely related to our proposed solution. In the following, we discuss their similarities and differences. One of the earliest demonstrations of the successful application of deep RL for quadrotor control has been presented in [26]. The authors train a position controller that outputs individual rotor thrusts but in contrast to our approach, they are using a complex training procedure (exploration scheme) that requires a resettable simulator and drives up the sample complexity to more than 100 million environment steps. Additionally, they do not take into account rotor delays and their code is not available. In [27], the authors apply domain randomization to transfer a policy that outputs SRT to different quadrotors but they require knowledge about the particular thrust-to-weight ratio and thrust limits. In comparison, our approach demonstrates Sim2Real transfer without domain randomization and without modifications to the state estimation in the firmware as well as much faster training times (comparison in Table I). In [20] the authors perform a benchmark of training controllers outputting velocity, CTBR and SRT commands but in contrast to our work they do not manage to transfer the low-level controller (SRT) to the real world. Moreover, in all of the aforementioned works as well as in [20], [28], [29], [30], SRT control outputs are used which simplify the learning problem by not exposing the agent to the non-linear RPM $\leftrightarrow$ thrust relationship. Lastly in [31], RPM outputs are used but rather than cutting out lower-level controls to simplify the learning problem as commonly done and described in Section II, they discard the higher-level control and take CTBR as input from a high-level

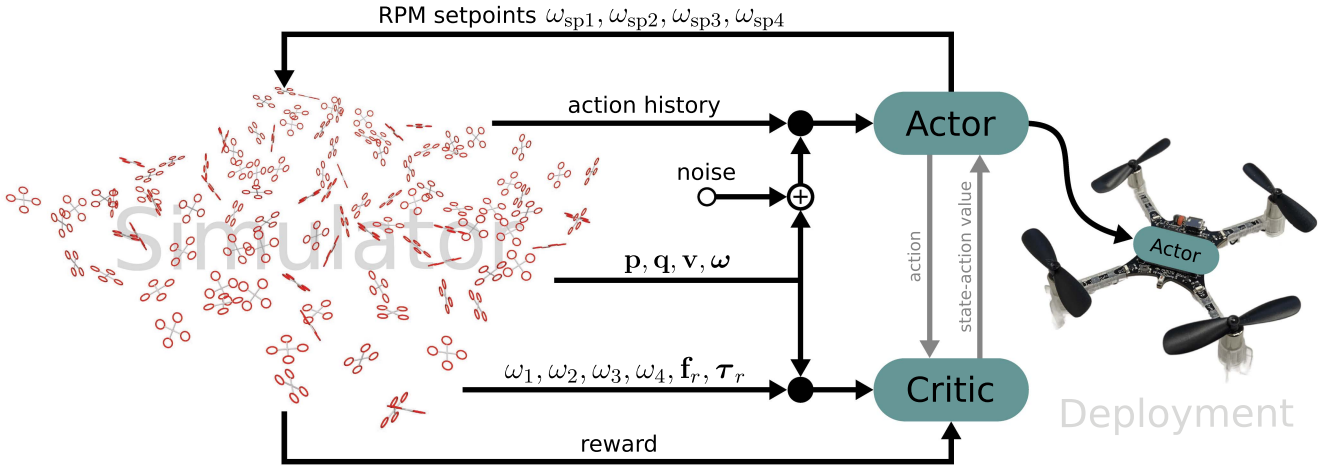


Fig. 2. Overview of the training and inference setup with a view of the simulator UI (left).

controller. In addition, in contrast to our method, [31] requires a state history and domain randomization as well as training for 2 h and 100 M steps. Furthermore, [31] only works under near-hover conditions and fails for agile trajectories (unlike ours, cf. Table III, Fig. 5(c) and the video). Lastly, the complexity of the method and the lack of an open-source implementation inhibit replication. In contrast to all the most related work which rely on on-policy, policy gradient RL algorithms (particularly Proximal Policy Optimization (PPO) [6]), we use Twin Delayed Deep Deterministic policy gradient (TD3) [32], an off-policy RL algorithm which offers better sample complexity and achieves very fast wall-clock training times.

IV. Methodology

To be able to take full advantage of the capabilities of the robot, we phrase our control problem as an MDP where the policy directly maps states to motor commands in the form of RPMs. We select quaternions to represent orientation as they are a compact and global representation. However, for the observations fed into the actor and critic, we convert them to rotation matrices to remove the ambiguity stemming from the quaternion's double coverage of the space of rotations. Since we model the motors as a first-order low-pass filter, RPMs are also part of the state. This makes the state 17 dimensional with the following structure $\mathbf{s} = \{\mathbf{p}, \mathbf{q}, \mathbf{v}, \boldsymbol{\omega}, \boldsymbol{\omega}_m\}$, consisting of position, orientation, linear and angular velocity and motor speeds respectively. We implement the standard dynamics of a quadrotor subject to motor delays (please refer to e.g. [11], [20], [25] or our parameter reference link). On real platforms, motor speeds are usually not observable and RPM setpoints are fed to the Electronic Speed Controller (ESC) in a feed-forward, one-way fashion through Pulse-Width Modulation (PWM). Hence, we implement an asymmetric actor-critic scheme [33], where the critic, being only required during training, has access to privileged information from the simulator as shown in Fig. 2. In particular, we let the critic access the RPMs and a random force $\mathbf{f}_r$ and torque $\boldsymbol{\tau}_r$ disturbance that are sampled at the beginning of each episode to increase the robustness of the trained policies. Hence, the privileged observations of the critic are represented by a 28 dimensional vector $\mathbf{o_c} = \{\mathbf{p}, \mathbf{R}, \mathbf{v}, \boldsymbol{\omega}, \boldsymbol{\omega}_m, \mathbf{f}_r, \boldsymbol{\tau}_r\}$ consisting of position, orientation, linear velocity, angular velocity, rotor speeds, force disturbance, and torque disturbance respectively.

We also notice that the delay in the step-response is significant and caused by the motors' low-pass behavior. This behavior also strongly impacts the dynamics of the 27 g nano quadrotor (Crazyflie) used in this work. Common values for the RC equivalent time constant ($1 - e^{-1} \approx 63\%$ step response) are between 0.05 s and 0.25 s. We empirically find 0.15 s to be suitable for Sim2Real transfer which is also supported by the manufacturers measurements.[2] These delays are significantly larger than the usual control interval of low-level controllers which usually run at around hundreds of Hz. Therefore, these delays lead to actions only impacting the state after 5 to 25 control steps. To mitigate this large level of partial observability, we add a history of control actions to the actor's observation. The action history is a proprioceptive measurement that can be trivially implemented in software on any real-world platform without requiring additional hardware (in contrast to direct RPM feedback measurements). Therefore, the actor's observations are $18 + N_H \cdot 4$ dimensional, where N_H is the length of the action history, and are defined as follows: $\mathbf{o_a} = \{\mathbf{p}, \mathbf{R}, \mathbf{v}, \boldsymbol{\omega}, \mathbf{H}\}$ with $\mathbf{H}$ being the action history.

While the critic observes the ground truth state, the actor's observations are additionally perturbed using observation noise to account for imperfections in the sensors (the scale of the noise components is described with all other parameters in the supplementary material). The actions consist of the RPM setpoints as $\mathbf{a} = \{\omega_{sp1}, \omega_{sp2}, \omega_{sp3}, \omega_{sp4}\}$ which places our policy/controller on the lowest order (level 5.1) of the taxonomy described in Section II.

For the initial state distribution, we sample from a diverse set of positions, orientations, linear and angular velocities as well as rotor speeds. We use a negative squared cost with an additive constant incentivizing survival to mitigate the "learning

[2]Crazyflie motor step response: https://web.archive.org/web/20220309092320/https://www.bitcraze.io/wp-content/uploads/2015/02/M1-step-response.png

to terminate" problem [34]

$$r(\mathbf{s},\mathbf{a},\mathbf{s}') = -C_{rp}\|\mathbf{p}\|_2^2 - C_{rq}\left(1-{q_w}^2\right) - C_{rv}\|\mathbf{v}\|_2^2 - C_{r\omega}\|\boldsymbol{\omega}\|_2^2 - C_{ra}\|\mathbf{a}-C_{rab}\|_2^2 + C_{rs}.$$

The values of the constants C_* are supplied in the supplementary material. Additionally, we find that a simple curriculum that is transitioning from an initial set of constants $C_{\text{init},*}$ to a more restrictive $C_{\text{target},*}$ (punishing position errors and particularly control actions more harshly) benefits the sample complexity and Sim2Real transfer (as described in Section V). Every 100 000 steps the weights are adjusted by multiplying them by the C_{p*} factors described in the supplementary material until they reach the $C_{\text{target},*}$ values, where they remain constant. We also decay the exploration noise using the same exponential scheme as in the curriculum of the reward function.

We would also like to highlight that we train a position controller, not merely a stabilizing controller that only works around a particular state. The goal of our position controller is to return to the origin point with zero linear velocity from any initial conditions (within reasonable bounds described in the parameters in the supplementary material). Hence, we train a policy that can go to any position or velocity setpoint by shifting the current position and velocity (so there is no need to apply goal-conditioned RL). For stable behavior, the position and velocity errors induced by the shifted setpoints should not exceed the errors seen during training which can easily be accommodated for by clipping.

To facilitate fast training times we implement a highly optimized simulator for multirotor dynamics. A sample view of the interface is shown in Fig. 2 (left). By leveraging C++ template metaprogramming the simulation code can be highly optimized by the compiler and can be tightly integrated into the RLtools [35] deep RL framework.

V. Experimental Setup

Simulation: We run our multirotor dynamics simulator on a Nvidia T2000 laptop GPU and attain 1284 million steps/s. To reach this level of performance, 64 blocks of 128 threads are each executing the forward dynamics in parallel. This amounts to 8192 environments in total which are run for 1 000 000 steps each. The required execution time of the GPU kernel is 6380 ms amounting to 1284 million steps/s. At a simulation frequency of 100 Hz this is about 5 months of simulated flight per second. Compared to Flightmare [11] which is the state of the art in terms of dynamics simulation speed with a reported frequency of 200 000 steps/s on a laptop, our simulation is about 6420× faster.

Training: Leveraging the training setup described in Section III and using the RLtools RL framework [35], we train a low-level quadrotor control policy. Fig. 3 shows the learning curve in terms of the returns (sum of rewards per episode). Purely looking at the reward/returns, it is hard to judge the level of flying capabilities of a particular policy because it is highly dependent on the reward function formulation. We believe the episode length is a more understandable measure than the return because if the agent crashes or flies away from a tight box around the origin, the episode is terminated. Hence, from Fig. 4 we can see that after about 300 000 steps (total number of interactions with the environment) or 18 s of training on a 2020 MacBook Pro, the policy has learned to fly relatively reliably. In comparison to related work (listed in Table I) our approach is substantially faster and requires an order of magnitude less samples when compared to learned policies on a similar level of control outputs.

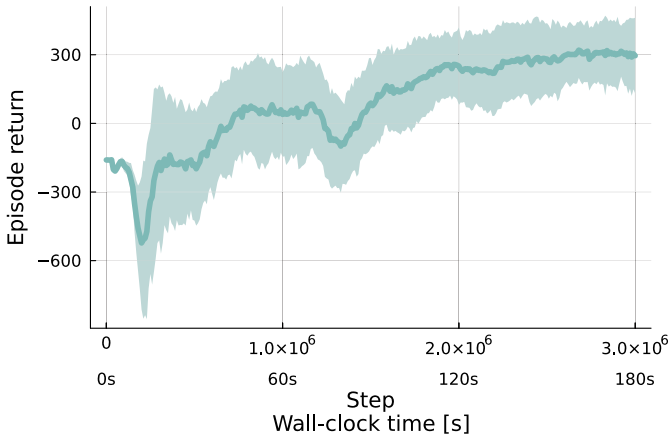


Fig. 3. Episode return (μ and σ over 50 runs with different initial seeds).

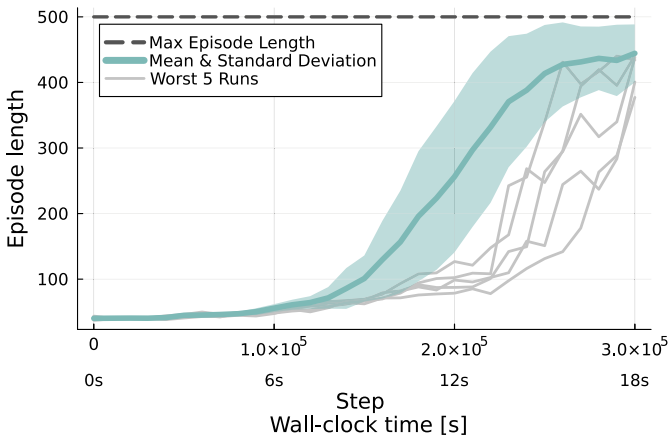


Fig. 4. Episode lengths (μ and σ over 50 runs with different initial seeds). Note, compared to Fig. 3 the horizontal axis is zoomed in to highlight the initial phase.

We investigate the 50 differently seeded training runs by selecting the 5 seeds with the worst cumulative number of steps over the whole training run (area under the curve) shown in Fig. 4. We notice that even the worst 5 runs learn to fly rapidly. This shows the remarkable reliability of our training approach which is not commonly expected from RL training pipelines (e.g., in [27] the authors state that cherry-picking across many seeds is required to find a policy that can fly). This remarkable level of reliability is also confirmed by real-world experiments that are described in the following.

VI. Results

Ablation study: To show the impact of the different components of the training setup (as shown in Fig. 2) on the

TABLE II
ABLATION STUDY

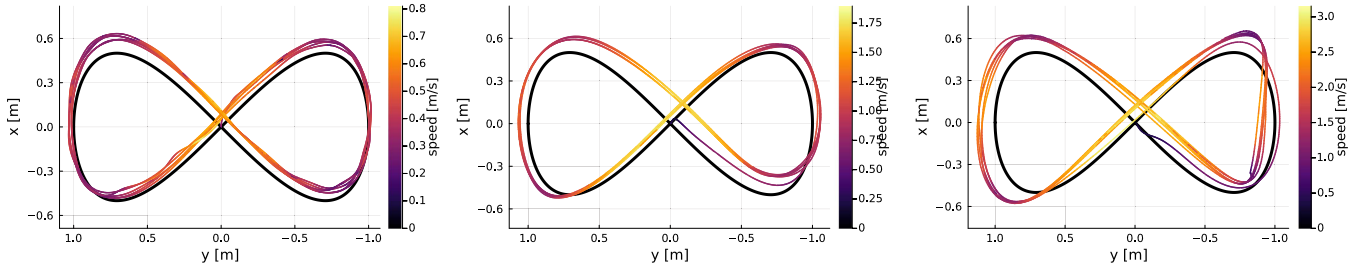
	Training (Simulation)				Inference (Real World)											
Task	Position Control				Position Control				Trajectory Tracking							
Checkpoint [steps]	300 000		3 000 000		300 000								3 000 000			
Ablation	N	R	N	R	#	$\bar{e}$	e_{med}	e_{min}	#	$\bar{e}$	e_{med}	e_{min}	#	$\bar{e}$	e_{med}	e_{min}
All Components	444	-171	366	296	**10/10**	0.26	0.24	0.1	**10/10**	0.34	0.38	**0.18**	**10/10**	0.27	0.26	0.21
Observation Noise	446	-195	364	294	**10/10**	0.26	0.27	0.1	8/10	0.33	0.31	0.23	**10/10**	**0.21**	0.21	0.16
Reward Recalculation	442	-311	397	**325**	9/10	**0.24**	**0.15**	**0.07**	9/10	0.34	**0.3**	0.25	7/10	0.22	0.22	0.2
Exploration Noise Decay	444	-171	351	285	9/10	0.25	0.22	0.08	8/10	**0.31**	**0.3**	0.19	**10/10**	**0.21**	**0.18**	**0.15**
Disturbances	**446**	**-146**	379	316	9/10	0.28	0.27	0.08	8/10	0.33	0.32	0.21	**10/10**	**0.21**	0.21	0.17
Asymmetric Actor-Critic	290	-313	431	268	6/10	0.28	0.29	0.08	4/10	0.4	0.34	0.32	9/10	0.25	0.26	0.19
Action History	173	-355	300	38	5/10	1.18	1.41	0.25	5/10	0.63	0.6	0.25	0/10	∞	∞	∞
Curriculum	444	-340	**450**	-285	1/10	0.18	0.18	0.18	1/10	0.23	0.23	0.23	9/10	0.2	0.2	0.15
Rotor Delay	46	-207	44	-195	0/10	∞	∞	∞	0/10	∞	∞	∞	0/10	∞	∞	∞
AAC & Curriculum	296	-341	428	-540	0/10	∞	∞	∞	0/10	∞	∞	∞	2/10	0.26	0.26	0.23

The *Ablation* column describes the component that is removed, except for the first row which contains all components. Returns R and number of steps per episode (N) are mean over 50 training runs with different seeds each. The mean($\bar{e}$)/median(e_{med})/minimum(e_{min}) of the position error (in the xy-plane) are statistics over 10 flights of the real quadrotor with policies from different seeds each (no cherry-picking, seeds are the first 10 of the 50 trained in simulation). The # column shows the number of successful flights (without crashing). For the trajectory tracking task we use the figure-eight trajectory shown in Fig. 5b with an interval of $T = 5.5$ s. For each metric, the best value is marked in bold. The best values of the real-world tests are awarded provided that at least 5/10 of the runs/seeds of a particular configuration are successful for each of the three tests.

performance during training and real-world deployment, we conduct a large-scale ablation study and present the results in Table II. To account for the inherent stochasticity of the observation noise and training process, we take a moving average over the previous 10 evaluations for the episode return and length metrics. The evaluation takes place every 1000 steps, hence the moving window covers 10 000 steps. We train each configuration for up to 3 000 000 steps using 50 different initial seeds each. Figs. 3 and 4 are generated from the 50 runs of the baseline configuration (containing all components). Furthermore, we execute the resulting policy after 300 000 and 3 000 000 steps of training using the first 10 different seeds of each configuration on a real Crazyflie quadrotor. We manually terminate the position control episodes after 20 s of flying because in our experience, at this level, policies can fly without crashing until the battery runs out. In the case of the trajectory tracking task, we complete 4 subsequent cycles or report a failed attempt if the quadrotor crashes beforehand. We report the mean and median as well as the minimum of the position error across the 10 runs/seeds. The minimum positional error is a particularly interesting metric because it corresponds to cherry-picking which is common in related works (e.g., [27]). Since the curriculum entails changes in the reward function and we apply reward recalculation of all rewards in the replay buffer after each modification of the reward function, we also ablate the setup without reward recalculation. Note that the configurations differ in the added/removed complexity. The components observation noise, reward recalculation, exploration noise decay, and disturbances are minor changes, while the asymmetric actor-critic structure, action history as well as rotor delay, and the curriculum are larger modifications.

From the results in Table II, we can see that overall (and particularly when executed on the real system) the baseline is the most reliable with no crashes in any of the tasks/seed combinations. In general, we can observe a trend that removing the smaller modifications still yields reliable policies (early during the training as well as after convergence). In contrast, when removing the more complex components, we can see a more pronounced drop in reliability as well as tracking performance. Here, we can observe that the curriculum indeed strongly impacts the training speed. Without the curriculum, the policies are not able to fly early on while after convergence they reach a comparable performance to the baseline. We can observe a similar behavior when removing the Asymmetric Actor-Critic (AAC) and hence also ablate removing both, the asymmetric actor-critic and the curriculum and find that the training takes considerably longer and even after 3 000 000 steps most of the policies are crashing. As deduced from first principles, we can also confirm that training without simulating the rotor delay leads to no usable policy. When simulating the rotor delays but not including the action history we can still see a considerable drop in reliability and positional accuracy, particularly after 3 000 000 steps where the policy seems to be overfitting the system dynamics. This validates the need for an action history to account for the partial observability in our training setup. When taking into account the average episode lengths and returns achieved by the different configurations during training in simulation we can only observe a loose correlation between simulation and real-world performance. As described earlier, the episode lengths appear to be a better gauge for the real-world performance as well.

We conclude that overall the baseline configuration with all components and the configurations with minor ablations yield the best performance when taking into account the speed of training (sample complexity) and position error. We observe that the baseline configuration yields the most reliable policies at all stages and anecdotally is also most robust with respect to e.g. turbulent wind (cf. the supplementary video). For trajectory tracking, after 3 000 000 steps in particular, we find that the configuration without exploration noise gives the lowest tracking



(a) Slow ($T = 15\,\mathrm{s}$) (b) Normal ($T = 5.5\,\mathrm{s}$) (c) Fast ($T = 3.5\,\mathrm{s}$)

Fig. 5. Real-world tracking of a Lissajous trajectory with different cycle times (reference trajectory in black).

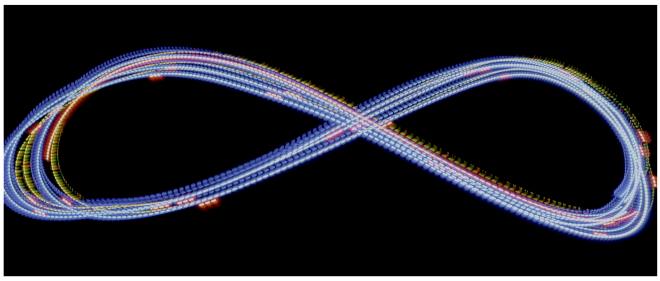


Fig. 6. Long exposure photo of the real-world tracking of a Lissajous trajectory with a $5.5\,\mathrm{s}$ cycle time.

TABLE III
REAL-WORLD TRAJECTORY TRACKING

Interval	Slow (15 s)		Normal (5.5 s)		Fast (3.5 s)	
Controller	$\bar{e}$	$\bar{e}_{xy}$	$\bar{e}$	$\bar{e}_{xy}$	$\bar{e}$	$\bar{e}_{xy}$
PID	0.23	0.22	0.72	0.72	0.88	0.87
Geometric [8]	**0.06**	**0.04**	**0.16**	0.16	0.36	0.36
Nonlinear [36]	0.29	0.11	0.38	0.32	∞	∞
INDI [37]	0.21	0.21	1.13	1.13	1.04	1.04
Baseline [27]	0.15	0.13	0.25	0.24	0.52	0.50
Baseline [30]	**0.06**	0.05	0.23	0.21	∞	∞
Ours	0.08	0.08	0.17	**0.15**	**0.24**	**0.22**

Error $\bar{e}$ (RMSE including z in meter) and $\bar{e}_{xy}$ (RMSE excluding z in meter) when tracking the Lissajous trajectory in Fig. 5 using different controllers.

error among the tested seeds and hence chose it for the trajectory tracking experiments in the next section.

Trajectory tracking: In addition, even though we train a position controller, we find that the resulting policies can track trajectories like the Lissajous in Fig. 5 when deployed on the real quadrotor. This is obtained by feeding into the policy an offset. We test the tracking performance using a figure-eight Lissajous trajectory $\mathbf{p}(t) = \begin{bmatrix}\cos(2\pi t/T) & \sin(4\pi t/T)/2 & \text{const}\end{bmatrix}^\top$ with varying cycle times T and show the tracking performance in Fig. 5. We conduct the experiments in a flying space of $10 \times 6 \times 4\,\mathrm{m}^3$ at the Agile Robotics and Perception Lab (ARPL) at New York University (equipped with a Vicon motion capturing system). We can see that our trained policy is able to track even agile trajectories like in Fig. 5(c) where it reaches up to $3\,\mathrm{ms}^{-1}$ and accelerations of up to 0.9 g. In Fig. 6, we show a long-exposure photo of the trajectory in Fig. 5(b) being tracked by one of our policies.

We execute the same trajectory using different types of classical controllers: Proportional-Integral-Derivative (PID), geometric [8], nonlinear [36] and Incremental Non-Linear Dynamic Inversion (INDI) [37]. We report the results in Table III. Additionally, we also compare the proposed solution to additional RL baselines. We replicate the related works [27] and [30]. In accordance with the original work (cf. Table I), we train them for 84 000 000 and 16 000 000 steps respectively. In the case of [27], we find that none out of 10 seeds leads to a policy that can fly the real drone. We extracted a trained checkpoint from the firmware provided by the authors. We found this checkpoint to be able to fly the trajectories but exhibit major oscillations and tracking error for the normal and fast trajectories. Based on the authors' instructions we believe this checkpoint was cherry-picked across even more seeds. In comparison, we found the method by [30] to train more reliably and to result in low position error for the slow trajectory. On the other hand, the policies trained using [30] exhibited poor yaw control (spinning around z while tracking the trajectory) and crashed for the normal-speed trajectory for 7/10 seeds and for 10/10 seeds in the fast case. We also test training [27] and [30] for only 300 000 and 3 000 000 steps for 10 seeds each (same as our method) but none of the resulting policies were able to fly. Across the classic controllers, we find that for slow trajectories, the geometric controller [8] achieves lower tracking error than our method. While for normal-speed trajectories our method is on par with the best classic controllers, it outperforms other approaches in the fast case.

VII. CONCLUSION

In this letter, we presented an unprecedentedly fast end-to-end RL architecture for quadrotor control that directly outputs RPMs and can be trained to fly a real quadrotor in $18\,\mathrm{s}$ on consumer-grade laptops. The approach directly transfers to real-world platforms even without domain randomization. Compared to prior work, our approach leads to very reliable training behavior and does not require the cherry-picking of trained policies. Furthermore, we conducted a large ablation study validating our design decisions and found that our method is competitive with other classic and learned controllers. We open-source our approach and simulator setup to the community to democratize learning-based quadrotor control. Due to the curse of dimensionality, the design space (in terms of hyperparameters and other design decisions) of RL-based end-to-end quadrotor control is still sparsely explored and we believe that our experimental results constitute a foundation that future research can build

upon. Our proposed training paradigm and the resulting highly optimized implementation allow for greatly reduced training times and hence more rapid iteration. Furthermore, the Crazyflie quadrotor uses open-source firmware, is widely available and inexpensive.

Future works will push the training speed and robustness as well as the tracking performance of learned low-level controllers through automatic hyperparameter optimization. Furthermore, we are interested in extending the policy to be adaptive to changing system or environment parameters like battery levels or wind, possibly using integral compensation [38] or meta-RL [39].

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[3] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2016.

[5] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. Int. Conf. Learn. Representations*, 2016.

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

[7] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[8] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 2520–2525.

[9] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, and O. von Stryk, "Hector open source modules for autonomous mapping and navigation with rescue robots," in *Proc. RoboCup 2013: Robot World Cup XVII*, 2014, vol. 8371, pp. 624–631.

[10] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Proc. Field Serv. Robot.: Result 11th Int. Conf.*, 2018, vol. 5, pp. 621–635.

[11] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Proc. Conf. Robot Learn.*, 2020, pp. 1147–1157.

[12] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly—A gym environment with PyBullet physics for reinforcement learning of multi-agent quadcopter control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 7512–7519.

[13] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS—A modular gazebo MAV simulator framework," in *Proc. Robot Operating Syst.: Complete Reference*, 2016, pp. 595–625.

[14] G. Li, X. Liu, and G. Loianno, "RotorTM: A flexible simulator for aerial transportation and manipulation," *IEEE Trans. Robot.*, vol. 40, pp. 831–850, 2024.

[15] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," in *Proc. Robotics: Sci. Syst.*, 2017.

[16] R. Polvara et al., "Toward end-to-end control for UAV autonomous landing via deep reinforcement learning," in *Proc. In. Conf. Unmanned Aircr. Syst.*, 2018, pp. 115–123.

[17] C. Sampedro, A. Rodriguez-Ramos, I. Gil, L. Mejias, and P. Campoy, "Image-based visual servoing controller for multirotor aerial robots using deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 979–986.

[18] S. Belkhale, R. Li, G. Kahn, R. McAllister, R. Calandra, and S. Levine, "Model-based meta-reinforcement learning for flight with suspended payloads," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 1471–1478, Apr. 2021.

[19] B. Rubí, B. Morcego, and R. Pérez, "Deep reinforcement learning for quadrotor path following with adaptive velocity," *Auton. Robots*, vol. 45, no. 1, pp. 119–134, 2021.

[20] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 10504–10510.

[21] P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt, "Learning to Fly via deep model-based reinforcement learning," Aug. 2020.

[22] J. Lin, L. Wang, F. Gao, S. Shen, and F. Zhang, "Flying through a narrow gap using neural network: An end-to-end planning and control approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 3526–3533.

[23] J. E. Kooi and R. Babuska, "Inclined quadrotor landing using deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 2361–2368.

[24] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," in *Proc. Robotics: Sci. Syst.*, 2020.

[25] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.

[26] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 2, no. 4, pp. 2096–2103, Oct. 2017.

[27] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(Multi)-Real: Transfer of low-level robust control policies to multiple quadrotors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 59–66.

[28] C.-H. Pi, K.-C. Hu, S. Cheng, and I.-C. Wu, "Low-level autonomous control and tracking of quadrotor using reinforcement learning," *Control Eng. Pract.*, vol. 95, 2020, Art. no. 104222.

[29] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 1205–1212.

[30] S. Gronauer, M. Kissel, L. Sacchetto, M. Korte, and K. Diepold, "Using simulation optimization to improve zero-shot policy transfer of quadrotors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 10170–10176.

[31] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller, "Learning a single near-hover position controller for vastly different quadcopters," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 1263–1269.

[32] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.

[33] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," in *Proc. Robotics: Sci. Syst. XIV*, 2018.

[34] J. Eschmann, "Reward function design in reinforcement learning," in *Proc. Reinforcement Learn. Algorithms: Anal. Appl.*, 2021, pp. 25–33 .

[35] J. Eschmann, D. Albani, and G. Loianno, "RLtools: A fast, portable deep reinforcement learning library for continuous control," 2023, *arXiv:2306.03530*.

[36] D. Brescianini, M. Hehn, and R. D'Andrea, "Nonlinear quadrocopter attitude control: Technical report," ETH Zurich, Tech. Rep., 2013. [Online]. Available: https://doi.org/10.3929/ethz-a-009970340

[37] E. J. J. Smeur, Q. Chu, and G. C. H. E. de Croon, "Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles," *J. Guid., Control, Dyn.*, vol. 39, no. 3, pp. 450–461, 2016.

[38] J. Xu et al., "Learning to fly: Computational controller design for hybrid UAVs with reinforcement learning," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 1–12, 2019, doi: 10.1145/3306346.3322940.

[39] J. Eschmann, "Partially unsupervised deep meta-reinforcement learning," Master's thesis, Technical University of Darmstadt, Darmstadt, Germany, 2021. [Online]. Available: https://thesis.jonas.es