This paper has been accepted for publication at the
IEEE International Conference on Robotics and Automation (ICRA), Philadelphia, 2022. ©IEEE

A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight

Elia Kaufmann, Leonard Bauersfeld, Davide Scaramuzza

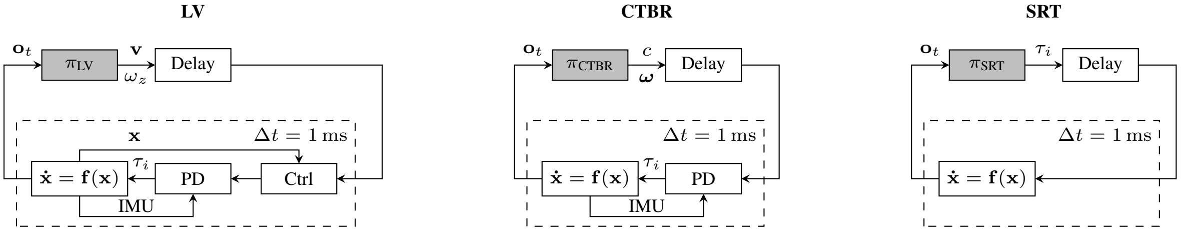

Fig. 1: In this paper, we compare three different classes of control policies for the task of agile quadrotor flight. From left to right: policies specifying desired linear velocities (LV) (they rely on a control stack that maps the output velocities to individual rotor thrusts), policies commanding collective thrust and bodyrates (CTBR) (they rely on a low-level controller that maps the output bodyrates to individual rotor thrusts), policies directly outputting single-rotor thrust (SRT).

***Abstract*— Quadrotors are highly nonlinear dynamical systems that require carefully tuned controllers to be pushed to their physical limits. Recently, learning-based control policies have been proposed for quadrotors, as they would potentially allow learning direct mappings from high-dimensional raw sensory observations to actions. Due to sample inefficiency, training such learned controllers on the real platform is impractical or even impossible. Training in simulation is attractive but requires to transfer policies between domains, which demands trained policies to be robust to such domain gap. In this work, we make two contributions: (i) we perform the first benchmark comparison of existing learned control policies for agile quadrotor flight and show that training a control policy that commands body-rates and thrust results in more robust sim-to-real transfer compared to a policy that directly specifies individual rotor thrusts, (ii) we demonstrate for the first time that such a control policy trained via deep reinforcement learning can control a quadrotor in real-world experiments at speeds over 45 km/h.**

SUPPLEMENTARY MATERIAL

A narrated video illustrating our findings is available at `https://youtu.be/zqdfVq2uWUA`

I. INTRODUCTION

Agile quadrotor flight is a challenging problem that requires fast and accurate control strategies. In recent years, numerous learning-based controllers have been proposed for quadrotors. In contrast to their traditional counterparts, learned control policies have the potential to directly map sensory information to actions, alleviating the need for explicit state estimation [1]–[4].

Prior work has proposed learned control policies that make use of various control input modalities to the underlying platform: while some directly specify motor commands [1], [5], [6], others, instead, output desired collective thrust and bodyrates [2], [7] (that are then executed by a low-level controller), or velocity commands [8], [9] (that are then executed by a control stack), or even a sequence of future waypoints [4]. Most published approaches do not justify their choice of control input. This renders performance comparisons among them and, thus, scientific progress difficult.

Due to the high sample complexity of learning-based policies, they are often trained in simulation, which then requires transferring the policy from simulation to the real world. This transfer between domains is known to be hard and is typically approached by increasing the simulation fidelity [10], [11], by randomization of dynamics [6], [12] or rendering properties [13], [14] at training time, or by abstraction of the policy inputs [2], [4]. Apart from simulation enhancements and input abstractions, also the choice of action space of the learned policy itself can facilitate transfer. Policies that generate high-level commands, such as desired linear velocity or future waypoints [4], have a reduced simulation to reality gap, as they abstract the task of flying by relying on an existing underlying control stack. However, while facilitating transfer, such abstractions also constrain the maneuverability of the platform. Approaches that do not rely on such abstractions (like those specifying collective thrust and body rates or even single-rotor-thrust commands) can potentially execute much more agile maneuvers, but have so far only been shown for near-hover trajectories [6] or require a dedicated policy for each maneuver [2].

In this paper, we compare and benchmark learned control policies with respect to their choice of action space. Specifically, we compare them in terms of peak performance in case of perfect model identification, as well as in terms of their transferability to a new platform with possibly different dynamics properties. We compare the learned policies with respect to their flight performance, which we characterize by the average tracking error on a set of predefined trajectories.

Our experiments, performed both in simulation and on

The authors are with the Robotics and Perception Group, Dep. of Informatics, University of Zurich, and Dep. of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland (`https://rpg.ifi.uzh.ch`). This work was supported by the National Centre of Competence in Research (NCCR) Robotics through the Swiss National Science Foundation (SNSF) and the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 871479 (AERIAL-CORE) and the European Research Council (ERC) under grant agreement No. 864042 (AGILEFLIGHT).

a real quadrotor platform, show that control policies that command collective thrust and bodyrates are more robust to changes in the dynamics of the platform without compromising agility. Additionally, compared to high-level action parameterizations, specifying collective thrust and bodyrates allows performing significantly more agile maneuvers.

Finally, we demonstrate the first learning-based controller, trained via deep reinforcement learning, that is able to perform previously unseen agile maneuvers on a real quadrotor flying at speeds over $45\,\mathrm{km\,h^{-1}}$. The policy is trained purely in simulation and transferred to the real platform without any fine-tuning.

II. Related Work

In this section, we give an overview of the related work for learning-based quadrotor control while focusing on the choice of action space. While there exists a comparison of action spaces of learned policies for 2D locomotion [15], such analysis is still lacking in the aerial robotics community. In the following, we group learned control strategies according to their action space into a) Linear Velocity Commands (LV), b) Collective Thrust and Bodyrates (CTBR), and c) Single Rotor Thrusts (SRT).

Linear Velocity. Control policies specifying high-level commands, often in the form of receding-horizon waypoints or velocity commands, have been proposed for a variety of tasks, such as forest trail navigation [8], navigation in city streets [9] and indoor environments [13], or even drone racing [16]. Recently, [17] have used model-based meta reinforcement learning to generate velocity commands that adapt to unknown payloads. While these approaches have been successfully deployed in the real world, only [16] achieved flight speeds beyond $3\,\mathrm{m\,s^{-1}}$, while the other policies result in near-hover flight. As the control policy does not take into account the dynamic constraints of the platform, it can be easily transferred, but does not exploit the platform's full dynamic capabilities. Furthermore, such approaches rely on an existing underlying control stack, which itself is dependent on high-quality state estimation.

Collective Thrust and Bodyrates. Compared to specifying linear velocity commands, controlling collective thrust and bodyrates has been shown to allow performing significantly more aggressive maneuvers. In [7], a racing policy directly maps image observations to collective thrust and bodyrate commands. Although the policy successfully races through challenging race tracks in simulation, it is not deployed on a real platform. In [18], the authors propose combining a classical controller with a learned residual command to correct for aerodynamic disturbances such as ground effect during near-hover flight. In [2], the authors use privileged learning to imitate a model predictive controller (MPC) to perform acrobatic maneuvers. While this approach successfully showed acrobatic flight on a real platform, it was constrained to a single maneuver and required a separate policy for each trajectory. In contrast to generating high-level commands, specifying collective thrust and bodyrates does not necessitate estimation of the full state of the platform, but only requires inertial measurements to perform feedback control on the bodyrates. This information is readily available at high frequency in today's flight controllers, rendering collective thrust and bodyrates the preferred control input modality for professional human pilots.

Single-Rotor Thrusts. There are several works that propose to directly learn to control individual rotor thrusts [1], [5], [6], [19]–[22]. As this control input does not require any additional control loop, it provides direct access to the actuators and as a result correctly represents the true actuation limits of the platform. It constitutes the most versatile control input investigated in this work. In [5], [6], the authors train a policy to map state observations directly to desired individual rotor thrusts. While [5] required a PID controller at data collection time to facilitate training, [6] demonstrated training of a stabilizing quadrotor control policy from scratch in simulation and deployment on multiple real platforms. [19] trains a policy for autonomous drone racing. Their approach demonstrates competitive racing performance in simulation, but is not deployed on a real quadrotor. In [1], the authors train a policy to perform obstacle avoidance using guided policy search by imitating an MPC controller that has access to privileged information about the environment. One of the few works that does not rely on simulated data for training is presented in [20], where the authors propose an approach based on deep model-based reinforcement learning to train a hovering policy for the *Crazyflie* quadrotor. The trained policy managed to control the real platform in hover for $6\,\mathrm{s}$ before crashing. A position controller is trained via reinforcement learning in [21] and extended in [22] to be robust against external disturbances such as wind. In [23], the authors train an attitude controller via deep reinforcement learning. They argue that their approach provides a better flight performance compared to a PID controller, while still being computationally lightweight. Although this method outputs individual rotor thrusts, it is still dependent on a higher-level controller that produces attitude setpoints to achieve stable flight.

While some of these works show successful deployment of their policies in the real world, none achieved agile flight, only reaching maximum speeds below $4\,\mathrm{m\,s^{-1}}$.

III. Quadrotor Dynamics

To train a control policy for agile flight, we implement the quadrotor dynamics as an environment in TensorFlow Agents[1]. The following section gives a brief overview of the dynamics implemented in the simulator.

A. Notation

Scalars are denoted in non-bold $[s, S]$, vectors in lowercase bold $\boldsymbol{v}$, and matrices in uppercase bold $\boldsymbol{M}$. World $\mathcal{W}$ and Body $\mathcal{B}$ frames are defined with orthonormal basis i.e. $\{\boldsymbol{x}_{\mathcal{W}}, \boldsymbol{y}_{\mathcal{W}}, \boldsymbol{z}_{\mathcal{W}}\}$. The frame $\mathcal{B}$ is located at the center of mass of the quadrotor. A vector from coordinate $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$

[1] `https://github.com/tensorflow/agents`

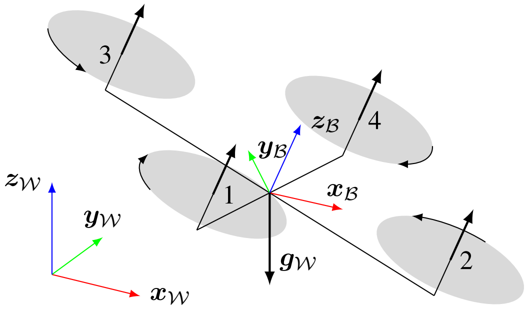

Fig. 2: Diagram of the quadrotor depicting the world and body frames and propeller numbering.

expressed in the $\mathcal{W}$ frame is written as: ${}_{\mathcal{W}}\boldsymbol{v}_{12}$. If the vector's origin coincides with the frame it is described in, the frame index is dropped, e.g. the quadrotor position is denoted as $\boldsymbol{p}_{\mathcal{WB}}$. Unit quaternions $\boldsymbol{q} = (q_w, q_x, q_y, q_z)$ with $\|\boldsymbol{q}\| = 1$ are used to represent orientations, such as the attitude state of the quadrotor body $\boldsymbol{q}_{\mathcal{WB}}$.

Finally, full SE3 transformations, such as changing the frame of reference from body to world for a point $\boldsymbol{p}_{B1}$, can be described by ${}_{\mathcal{W}}\boldsymbol{p}_{B1} = {}_{\mathcal{W}}\boldsymbol{t}_{\mathcal{WB}} + \boldsymbol{q}_{\mathcal{WB}} \odot \boldsymbol{p}_{B1}$. Note the quaternion-vector product is denoted by $\odot$ representing a rotation of the vector by the quaternion as in $\boldsymbol{q} \odot \boldsymbol{v} = \boldsymbol{q}\boldsymbol{v}\bar{\boldsymbol{q}}$, where $\bar{\boldsymbol{q}}$ is the quaternion's conjugate.

B. Quadrotor Dynamics

The quadrotor is assumed to be a 6 degree-of-freedom rigid body of mass m and diagonal moment of inertia matrix $\boldsymbol{J} = \operatorname{diag}(J_x, J_y, J_z)$. Furthermore, the rotational speeds of the four propellers Ω_i are modeled as first-order system with time constant k_{mot} where the commanded motor speeds $\boldsymbol{\Omega}_{\text{cmd}}$ are the input.

The state space is thus 17-dimensional and its dynamics can be written as:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{p}}_{\mathcal{WB}} \\ \dot{\boldsymbol{q}}_{\mathcal{WB}} \\ \dot{\boldsymbol{v}}_{\mathcal{WB}} \\ \dot{\boldsymbol{\omega}}_{\mathcal{B}} \\ \dot{\boldsymbol{\Omega}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{\mathcal{W}} \\ \boldsymbol{q}_{\mathcal{WB}} \cdot \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{\mathcal{B}}/2 \end{bmatrix} \\ \frac{1}{m}\left(\boldsymbol{q}_{\mathcal{WB}} \odot (\boldsymbol{f}_{\text{prop}} + \boldsymbol{f}_{\text{drag}})\right) + \boldsymbol{g}_{\mathcal{W}} \\ \boldsymbol{J}^{-1}\left(\boldsymbol{\tau}_{\text{prop}} - \boldsymbol{\omega}_{\mathcal{B}} \times \boldsymbol{J}\boldsymbol{\omega}_{\mathcal{B}}\right) \\ \frac{1}{k_{\text{mot}}}\left(\boldsymbol{\Omega}_{\text{cmd}} - \boldsymbol{\Omega}\right) \end{bmatrix}, \tag{1}$$

where $\boldsymbol{g}_{\mathcal{W}} = [0, 0, -9.81\,\mathrm{m/s^2}]^\intercal$ denotes earth's gravity, $\boldsymbol{f}_{\text{prop}}$, $\boldsymbol{\tau}_{\text{prop}}$ are the collective force and the torque produced by the propellers, and $\boldsymbol{f}_{\text{drag}}$ is a linear drag term. The quantities are calculated as follows:

$$\boldsymbol{f}_{\text{prop}} = \sum_i \boldsymbol{f}_i\,, \quad \boldsymbol{\tau}_{\text{prop}} = \sum_i \boldsymbol{\tau}_i + \boldsymbol{r}_{\text{P},i} \times \boldsymbol{f}_i\,, \tag{2}$$

$$\boldsymbol{f}_{\text{drag}} = -\begin{bmatrix} k_{vx} v_{\mathcal{B},x} & k_{vy} v_{\mathcal{B},y} & k_{vz} v_{\mathcal{B},z} \end{bmatrix}^\top, \tag{3}$$

where $\boldsymbol{r}_{\text{P},i}$ is the location of propeller i expressed in the body frame , $\boldsymbol{f}_i$, $\boldsymbol{\tau}_i$ are the forces and torques generated by the i-th propeller, and (k_{vx}, k_{vy}, k_{vz}) [24], [25] are linear drag coefficients. A commonly used [24], [26] model for the forces and torques exerted by a single propeller is presented in the following: the thrust and drag torque are assumed to be proportional to the square of the propellers' rotational speed. The corresponding thrust and drag coefficients c_l and c_d can be readily identified on a static propeller test stand. By also measuring the rotational speed of the propeller during those tests, the motor time constant k_{mot} can be estimated. Overall, the force and torque produced by a single propeller are modeled as follows:

$$\boldsymbol{f}_i(\Omega) = \begin{bmatrix} 0 & 0 & c_\text{l} \cdot \Omega^2 \end{bmatrix}^\top, \quad \boldsymbol{\tau}_i(\Omega) = \begin{bmatrix} 0 & 0 & c_\text{d} \cdot \Omega^2 \end{bmatrix}^\top \tag{4}$$

The dynamics are integrated using a symplectic Euler scheme with step size $1\,\text{ms}$. For numerical values of the identified mass, inertia, and thrust and drag constants, we refer the reader to Section IV-C.

IV. Methodology

We address the challenge of robust and agile quadrotor flight using learned control policies by identifying the best choice of action space for the task. We train deep neural control policies that directly map observations $\boldsymbol{o}_t$ in the form of platform state and a reference trajectory to control actions $\boldsymbol{u}_t$. The control policies are trained using model-free reinforcement learning (PPO [27]) purely in simulation on a set of over 600 reference trajectories that cover the full performance envelope of the quadrotor. We train policies of three different types that only differ in their choice of action space $\boldsymbol{u}_t$, as illustrated in Figure 1:

1) **Linear Velocity & Yaw Rate (LV):** Each action specifies a desired linear velocity and yaw rate, which are then tracked by a full control stack with access to accurate state estimation. $\pi_{\text{LV}}(\boldsymbol{o}_t) \Rightarrow \boldsymbol{u}_t = \{v_x, v_y, v_z, \omega_z\}$
2) **Collective Thrust & Bodyrate (CTBR):** Each action represents desired collective thrust and bodyrates, which are tracked by a low-level controller using measurements from an inertial sensor. $\pi_{\text{CTBR}}(\boldsymbol{o}_t) \Rightarrow \boldsymbol{u}_t = \{c, \omega_x, \omega_y, \omega_z\}$
3) **Single-Rotor Thrust (SRT):** Each action directly specifies desired individual rotor thrusts, which are then applied for the duration of a control step. $\pi_{\text{SRT}}(\boldsymbol{o}_t) \Rightarrow \boldsymbol{u}_t = \{f_1, f_2, f_3, f_4\}$

All policy types feature a 4-dimensional action space, are fed the same observations $\boldsymbol{o}_t$, and are represented by the same network architecture.

TABLE I: Input features to the policy and value networks. The state is represented by a sliding window of length H of current and previous states, the reference is represented by a receding-horizon window of length R of current and future reference states. Both networks observe the same state and reference, but only the value network observes privileged information, such as biases in mass, inertia, drag and gravity applied during training with domain randomization.

Input	Components	Dimensions	Policy NW	Value NW
State	z-Position	$H \times 1$	✓	✓
	Velocity	$H \times 3$	✓	✓
	Attitude	$H \times 9$	✓	✓
	Bodyrates	$H \times 3$	✓	✓
	Privileged Info.	$H \times 7$	✗	✓
Reference	Position	$R \times 3$	✓	✓
	Velocity	$R \times 3$	✓	✓
	Attitude	$R \times 9$	✓	✓
	Bodyrates	$R \times 3$	✓	✓

TABLE II: Physical parameters of the simulation. At the start of each rollout, the parameters are sampled from a uniform distribution around the nominal values with the randomization specified above.

Parameter	Nominal Value	Randomization
Mass [kg]	0.768	±30%
Inertia [$\mathrm{kg\,m^2}$]	[2.5e-3, 2.1e-3, 4.3e-3]	±30%
Gravity [$\mathrm{m\,s^{-2}}$]	[0.0, 0.0, -9.81]	±0.4
k_{vx} [$\mathrm{N\,s\,m^{-1}}$]	0.3	±0.3
k_{vy} [$\mathrm{N\,s\,m^{-1}}$]	0.3	±0.3
k_{vz} [$\mathrm{N\,s\,m^{-1}}$]	0.15	±0.15
c_l [$\mathrm{N\,rad^{-1}\,s^{-1}}$]	1.563e-6	±0.0
c_d [$\mathrm{N\,m\,rad^{-1}\,s^{-1}}$]	1.909e-8	±0.0

A. Observations, Actions, and Rewards

An observation $\boldsymbol{o}_t$ obtained from the environment at time t consist of (i) a history of previous states and applied actions and (ii) the future reference along the trajectory. Specifically, the state information contains a history of length $H = 10$ of the z-position of the platform, its velocity, attitude represented as rotation matrix, and bodyrates. Even though the simulator internally uses quaternions, we pass attitude as rotation matrix to the networks to avoid discontinuities [28]. The reference information consists of a sequence of length $R = 10$ of future relative position, velocity, and bodyrates as well as the full rotation matrix of the reference. The position and velocity components of the reference states are expressed as the residual from the current state of the quadrotor. All observations are normalized before passing them to the networks.

Since the value network is only used during training time, it can access privileged information about the environment that is not accessible to the policy network. Specifically, this privileged information contains the mass and inertia biases applied during randomization, as well as the sampled drag coefficients and the additive gravity bias. An overview of the observation provided to the policy and value network is given in Table I. The value network and the policy network share the same architecture but have different parameters. The state and reference information are encoded by two separate fully-connected neural networks with 3 hidden layers with 64 neurons each. The encodings are then concatenated and fed to a final multilayer perceptron with two layers of 128 neurons each.

We use a dense shaped reward formulation to learn the task of agile trajectory tracking. The reward r_t at timestep t is given by

$$\begin{aligned} r_t = &-(\boldsymbol{x}_t - \boldsymbol{x}_{\text{ref},t})^\top \boldsymbol{Q}(\boldsymbol{x}_t - \boldsymbol{x}_{\text{ref},t}) \\ &-(\boldsymbol{u}_t - \boldsymbol{u}_{\text{ref},t})^\top \boldsymbol{R}(\boldsymbol{u}_t - \boldsymbol{u}_{\text{ref},t}) - r_{\text{crash}} \ , \end{aligned} \tag{5}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are diagonal matrices, $\boldsymbol{x}_t$ the full state of the quadrotor, $\boldsymbol{u}_t$ the applied action, $\boldsymbol{x}_{\text{ref},t}$ and $\boldsymbol{u}_{\text{ref},t}$ their respective references, and r_{crash} is a binary penalty that is only active when the altitude of the platform is negative, which also ends the episode.

TABLE III: Training hyperparameters.

Hyperparameter	Value
γ (discount factor)	0.98
Actor learning rate	3e-4
Critic learning rate	3e-4
Entropy regularization	1e-2
ε (importance ratio clipping)	0.2

B. Policy Learning

All control policies are trained using Proximal Policy Optimization (PPO) [27]. PPO has been shown to achieve state-of-the-art performance on a set of continuous control tasks and is well suited for learning problems where interaction with the environment is fast. Data collection is performed by simulating 50 agents in parallel. At each environment reset, every agent samples a new trajectory from the set of training trajectories and is initialized with bounded perturbation at the start of the trajectory.

Inspired by prior work on simulation to reality transfer, we perform randomization of the dynamics of the platform during training time and apply Gaussian noise to the policy observations. Specifically, we randomize mass, inertia, aerodynamic drag, and thrust variations of the quadrotor.

C. Training Details

The policies are trained in a simulated quadrotor environment implemented using TensorFlow Agents. The nominal quadrotor parameters such as mass and inertia are identified from the real platform and are summarized in Table II together with the amount of randomization applied at training time. Training hyperparameters specified in Table III.

During trajectory tracking, the agent receives at each timestep a reward that penalizes tracking error and deviation from the reference action as laid out in Eq. (5). The matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ have nonzero elements only on the diagonal. Specifically, we use $\boldsymbol{Q} = \text{diag}\{0.1 \cdot \mathbf{1}_{3\times1}, 0.02 \cdot \mathbf{1}_{9\times1}, 0.002 \cdot \mathbf{1}_{3\times1}, 0.01 \cdot \mathbf{1}_{3\times1}\}$ and $\boldsymbol{R} = \text{diag}\{0.001 \cdot \mathbf{1}_{4\times1}\}$. The episode is terminated when the quadrotor crashes (i.e. $p_z \leq 0.0$) with a reward of $r_{\text{crash}} = -500$.

V. Experiments

We design our experimental setup to investigate the influence of the choice of action space on flight performance. Specifically, we design our experiments to answer the following research questions: (i) How is the peak control performance in situation of perfect model identification affected by the actuation model? (ii) How does the choice of action space affect the robustness against model mismatch? (iii) What is the impact of the choice of action space on training data requirement?

We evaluate the performance of all policies on a set of test trajectories of varying agility, spanning from a hover trajectory up to a racing trajectory [29] that requires to perform accelerations beyond 3g to track. All test trajectories are within the distribution of training trajectories and are feasible, i.e. they do not exceed the platform limits. Table IV shows the key metrics of all test trajectories.

A. Simulation Experiments

In a set of controlled experiments in simulation, the tracking performance of each policy is investigated. We compare performance with respect to average positional tracking error. Experiments are performed on the test trajectories in two settings: (i) in the *Nominal* setting, the test environment perfectly matches the training environment; (ii) in the *Model Mismatch* setting, the environment at test time is different from the training environment. Specifically, we use in setting (ii) a quadrotor simulation that was identified from real flight data and uses blade-element momentum theory to accurately model the aerodynamic forces acting on the platform [11]. We also apply a control delay of 20 ms to simulate wireless communication latency. Note that we can only use this simulation at test time, since it is computationally too expensive to run it at training time.

While setting (i) is focused on the maximum possible performance achievable by a method and its training data requirement, setting (ii) investigates the robustness of policies against model mismatch. All policies tested in setting (i) have been trained specifically for the nominal environment without any randomization, while the policies tested in setting (ii) have been trained on a distribution of environments as explained in Section IV-B. We also compare against two state-of-the-art classical control approaches: MPC-SRT represents an optimization-based controller [30] that directly controls at individual rotor thrust level, while MPC-CTBR makes use of a low-level controller. All learned policies are run at a constant frequency of 50 Hz, while the traditional controllers are executed at 100 Hz.

Nominal Model. Table V shows the results of the experiments in the *Nominal* setting (i). SRT and CTBR policies perform comparable in this setting, with CTBR marginally outperforming on slower trajectories, while SRT performs slightly better on the more aggressive maneuvers. These results confirm previous findings from experiments in the domain of 2D locomotion [15]: policies that operate in concert with an underlying low-level controller outperform end-to-end policies. The policies that produce linear velocity commands (LV) perform inferior especially for agile maneuvers. This can be explained by the fact that the action space of linear velocity commands does not correctly represent the dynamic constraints of a quadrotor platform, which leads to a reduced maneuverability. This result extends the findings of [15] and shows that more abstraction does not necessarily lead to better performance. Compared to the learned policies, the traditional control approaches (MPC-SRT, MPC-CTBR) perform significantly better in the *Nominal* setting. This results is expected, as the system dynamics implemented in the MPC exactly match the simulated dynamics of the platform. We still provide these results to allow a comparison with traditional control approaches.

TABLE IV: Maxima of velocity, mass-normalized collective thrust and bodyrates of the test trajectories.

Trajectory	$\|\mathbf{v}\|_{\max}[\mathrm{m\,s^{-1}}]$	$c_{\max}[\mathrm{m\,s^{-2}}]$	$\|\boldsymbol{\omega}\|_{\max}[\mathrm{rad\,s^{-1}}]$
Hover	0.0	9.81	0.0
RandA	3.87	12.68	1.27
RandB	6.36	13.54	1.52
RandC	8.92	14.52	1.93
RaceA	10.48	16.18	5.74
RaceB	11.97	24.94	8.37
Split-S	12.40	26.35	6.11
RaceC	14.22	33.04	11.56

TABLE V: Average positional tracking error in centimeter on each test trajectory in case of no model mismatch. The table reports results for learned policies (SRT, CTBR, LV), and traditional approaches (MPC-SRT, MPC-CTBR). Results report mean and standard deviation for 10 trained policies.

	SRT	CTBR	LV	MPC-SRT	MPC-CTBR
Hover	1.0±0.2	**0.6±0.2**	7.0±1.6	**0.1**	0.2
RandA	1.5±0.2	**0.9±0.1**	15.4±3.0	**0.2**	0.3
RandB	2.4±0.2	**1.6±0.1**	61.5±21.0	**0.2**	0.4
RandC	3.0±0.3	**2.0±0.2**	85.7±11.5	**0.2**	0.4
RaceA	**5.0±1.2**	**5.0±1.0**	121.1±25.8	**0.3**	1.3
RaceB	7.1±1.8	**6.9±1.5**	170.2±16.3	**0.7**	3.0
Split-S	**3.5±0.4**	6.6±1.1	92.1±20.8	**1.0**	2.1
RaceC	**9.2±3.2**	12.3±2.2	197.9±38.1	**1.2**	3.9

TABLE VI: Average positional tracking error in centimeter on each test trajectory obtained in a quadrotor simulator based on blade-element momentum theory with a control delay of 20 ms. Results report mean and standard deviation for 10 trained policies.

	SRT	CTBR	LV	MPC-SRT	MPC-CTBR
Hover	11.3±4.5	**0.6±0.5**	6.7±2.0	1.0	**0.5**
RandA	12.0±4.0	**1.2±0.5**	17.8±1.4	3.3	**1.1**
RandB	14.4±2.4	**2.2±0.8**	57.0±12.0	7.0	**2.0**
RandC	17.6±5.9	**2.6±0.8**	78.9±13.4	8.5	**2.6**
RaceA	crash	**5.6±1.7**	144.0±20.1	12.6	**4.8**
RaceB	crash	**10.0±4.0**	171.4±17.0	crash	**6.3**
Split-S	crash	**6.9±2.6**	83.8±9.7	**11.3**	11.4
RaceC	crash	**14.9±5.5**	176.7±22.0	crash	**7.5**

Model Mismatch. Table VI shows the results of the *Model Mismatch* scenario. Controllers that directly specify single rotor thrusts exhibit a significant reduction in performance, especially for agile trajectories: SRT has a significantly higher tracking error for slow trajectories and often crashes on the faster maneuvers; MPC-SRT also has higher tracking error and even crashes on RaceB and RaceC. We report *crash*, as soon as one policy crashes on the maneuver. The CTBR policies (as well as MPC-CTBR) are less affected by the model mismatch and can still execute all maneuvers with a modest increase in tracking errors. The LV policies show a smaller sensitivity to the model mismatch, but are still consistently outperformed by the CTBR policies on all trajectories.

Training Data Requirement. Figure 3 depicts the learning curves of all policies in case of no domain randomization (left) and with domain randomization (right). All policies have been trained for a total of 50M environment interactions. The learning curves also show the robustness of CTBR and LV to changes in the platform dynamics, we even observed that training with a randomized platform accelerates learning in the early stages of training. In contrast, the learning curves of SRT in case of domain randomization initially exhibit a high variance, train slower, and converge to a final performance substantially lower than in case of no domain randomization.

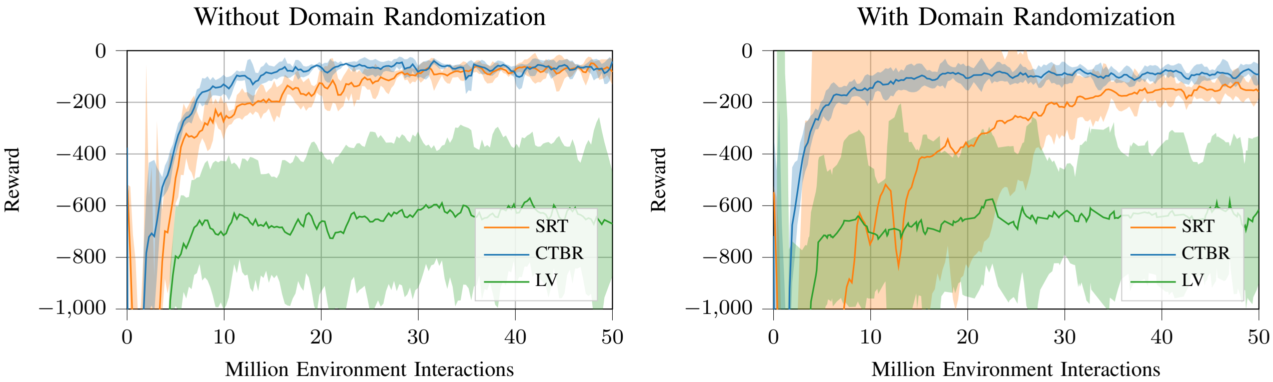

Fig. 3: Learning curves of policies trained without (left) and with (right) domain randomization. All policies are trained for a total of 50M environment interactions. Learning curves show mean performance and standard deviation computed over all trained policies.

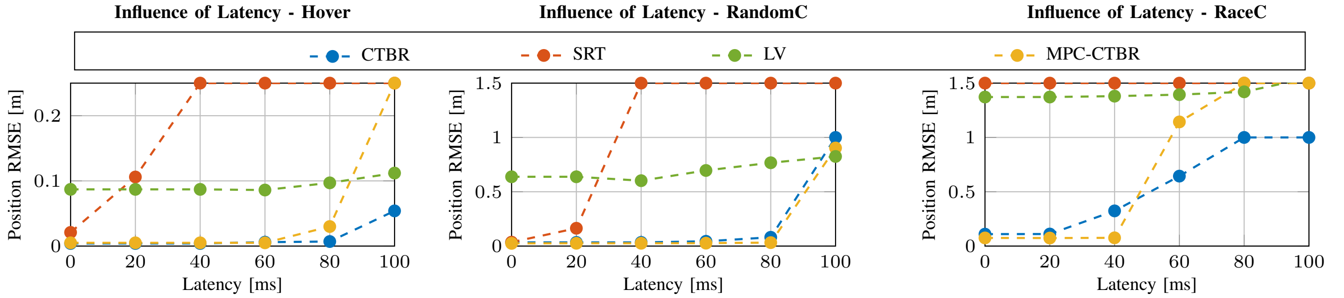

Fig. 4: Sensitivity to control delay on three trajectories of increasing agility. The results show that policies that operate on single rotor thrust (SRT) are less robust against control delay.

Influence of Delay. Our experiments show that the performance of the tested control policies varied significantly in case of unknown control delay. Figure 4 shows that policies that operate at higher abstraction levels such as LV or CTBR are less sensitive to such delay. Furthermore, accurate identification of control delay is more important for agile trajectories; while hover is possible for CTBR without a noticeable decrease in performance for latencies up to 60 ms, the same latency leads to a crash on the racing trajectory.

B. Real World Experiments

We assess the performance of different control policies when deployed on a real quadrotor platform. As in the simulation experiments, we execute a set of trajectories and compare tracking performance between the methods presented in Section IV. We encourage the reader to watch the supplementary video to understand the dynamic nature of these experiments.

The results of the real world experiments are shown in Table VII. Due to its significant sensitivity to control delays and a communication latency of 60 ms imposed by the real system, the SRT policies could not be deployed. The CTBR policies instead manage to fly unseen maneuvers on the real platform despite the control delay. The LV policies transfer to the real platform as well, but CTBR significantly outperforms on agile trajectories. Compared to the results in the BEM simulator, tracking errors are higher in the real world mainly due to unmodelled effects such as varying battery voltage, imperfect motor thrust mappings, and torque imbalances due to imperfect mass distribution. Throughout the tested trajectories, the CTBR policies reach accelerations of up to 3g and speeds beyond $45\,\mathrm{km\,h^{-1}}$, which outperforms the previous state of the art in learning-based quadrotor control by a factor of 3 in terms of speed.

TABLE VII: Positional tracking error in centimeter on a set of test trajectories executed in the real world. Results report mean and standard deviation for 5 trained policies.

	CTBR	LV	MPC-CTBR
Hover	**4.4±1.4**	6.2±2.0	3.0
RandA	**8.1±1.0**	60.0±16.8	8.0
RandB	**8.6±0.8**	87.0±30.3	8.0
RandC	**47.8±9.9**	134.8±19.6	14.0
Circle	**31.8±4.4**	170.7±11.6	25.0
Lemniscate	**26.8±4.4**	189.5±13.7	16.0
Racing	**53.0±9.2**	200.8±14.5	20.0

VI. Conclusion

We presented a comparison of learning-based controllers for agile quadrotor flight. We compared policies that specify individual rotor thrusts, collective thrust and bodyrates, and linear velocity commands. While all tested policy types were able to learn a universal flight controller, they differed strongly in terms of peak performance and robustness against dynamics mismatch. We identified that policies producing collective thrust and bodyrates exhibit strong resilience against dynamics mismatch and transfer well between domains while retaining high agility. This work can serve as guideline for future work on learning-based quadrotor control by identifying the control input modality that is best suited for agile flight.

References

[1] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2016.

[2] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep Drone Acrobatics," in *Robotics: Science and Systems (RSS)*, 2020.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, 2016.

[4] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," in *Science Robotics*, 2021.

[5] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robot. Autom. Lett.*, 2017.

[6] 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 *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2019.

[7] M. Muller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Teaching uavs to race: End-to-end regression of agile controls in simulation," in *Eur. Conf. Comput. Vis. Workshops (ECCVW)*, 2018.

[8] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodr´iguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robot. Autom. Lett.*, 2016.

[9] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robot. Autom. Lett.*, 2018.

[10] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *Robotics: Science and Systems (RSS)*, 2018.

[11] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "Neurobem: Hybrid aerodynamic quadrotor model," in *Proceedings of Robotics: Science and Systems*, 2021.

[12] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, "Learning dexterous in-hand manipulation," *Int. J. Robot. Research*, 2020.

[13] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," in *Robotics: Science and Systems (RSS)*, 2017.

[14] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. IEEE, 2017.

[15] X. B. Peng and M. van de Panne, "Learning locomotion skills using deeprl: Does the choice of action space matter?" in *Proceedings of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 2017.

[16] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," in *Conference on Robot Learning (CoRL)*, 2018.

[17] 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. Autom. Lett.*, 2021.

[18] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2019.

[19] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2021.

[20] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robot. Autom. Lett.*, 2019.

[21] C.-H. Pi, K.-C. Hu, S. Cheng, and I.-C. Wu, "Low-level autonomous control and tracking of quadrotor using reinforcement learning," *Control Engineering Practice*, 2020.

[22] C.-H. Pi, W.-Y. Ye, and S. Cheng, "Robust quadrotor control through reinforcement learning with disturbance compensation," *Applied Sciences*, 2021.

[23] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Transactions on Cyber-Physical Systems*, 2019.

[24] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors—a modular gazebo mav simulator framework," in *Robot Operating System (ROS)*. Springer, 2016.

[25] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robot. Autom. Lett.*, 2017.

[26] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robot.* Springer, 2018.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv e-prints*, 2017.

[28] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2019.

[29] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, 2021.

[30] G. Torrente, E. Kaufmann, P. Foehn, and D. Scaramuzza, "Data-driven MPC for quadrotors," *IEEE Robot. Autom. Lett.*, 2021.