# RAPTOR: A Foundation Policy for Quadrotor Control

Jonas Eschmann[1,2,*], Dario Albani[2], Giuseppe Loianno[1]

[1]Department of Electrical Engineering and Computer Sciences (EECS), UC Berkeley, Berkeley, CA 94720, USA.

[2]Autonomous Robotics Research Center, Technology Innovation Institute, Abu Dhabi, UAE.

[*]Corresponding author. Present Address. Email: jonas.eschmann@berkeley.edu

**Project page**: `https://raptor.rl.tools`
**Video**: `https://youtu.be/hVzdWRFTX3k`



**Humans are remarkably data-efficient when adapting to new unseen conditions, like driving a new car. In contrast, modern robotic control systems, like neural network policies trained using Reinforcement Learning (RL), are highly specialized for single environments. Because of this overfitting, they are known to break down even under small differences like the Simulation-to-Reality (Sim2Real) gap and require system identification and retraining for even minimal changes to the system. In this work, we present RAPTOR, a method for training a highly adaptive foundation policy for quadrotor control. Our method enables training a single, end-to-end neural-network policy to control a wide variety of quadrotors. We test 10 different real quadrotors from 32 g to 2.4 kg that also differ in motor type (brushed vs. brushless), frame type (soft vs. rigid), propeller type (2/3/4-blade), and flight controller (PX4/Betaflight/Crazyflie/M5StampFly). We find that a tiny, three-layer policy with only 2084 parameters is sufficient for zero-shot adaptation to a wide variety of platforms. The adaptation through In-Context Learning is made possible by using a recurrence in the hidden layer. The policy is trained through a novel Meta-Imitation Learning algorithm, where we sample 1000 quadrotors and train a teacher policy for each of them using Reinforcement Learning. Subsequently, the 1000 teachers are distilled into a single, adaptive student policy. We find that within milliseconds, the resulting foundation policy adapts zero-shot to unseen quadrotors. We extensively test the capabilities of the foundation policy under numerous conditions (trajectory tracking, indoor/outdoor, wind disturbance, poking, different propellers).**

## 1 Introduction

Bearing Vertical Take-Off and Landing (VTOL) and hovering capabilities, Multirotor Aerial Vehicles (MAVs) have become a valuable platform for many real-world applications like package delivery (*1*), infrastructure inspection and maintenance (*2*), or search and rescue (*3*). In addition to the VTOL and hovering capabilities, MAVs can be built from cheap, mass-produced Commercial Off-The-Shelf (COTS) parts and be scaled from light (tens of grams) and centimeter-sized to heavy (multiple kilograms) and meter-sized. This broad range allows for easy customization of the mechanical and electrical design for each particular application.

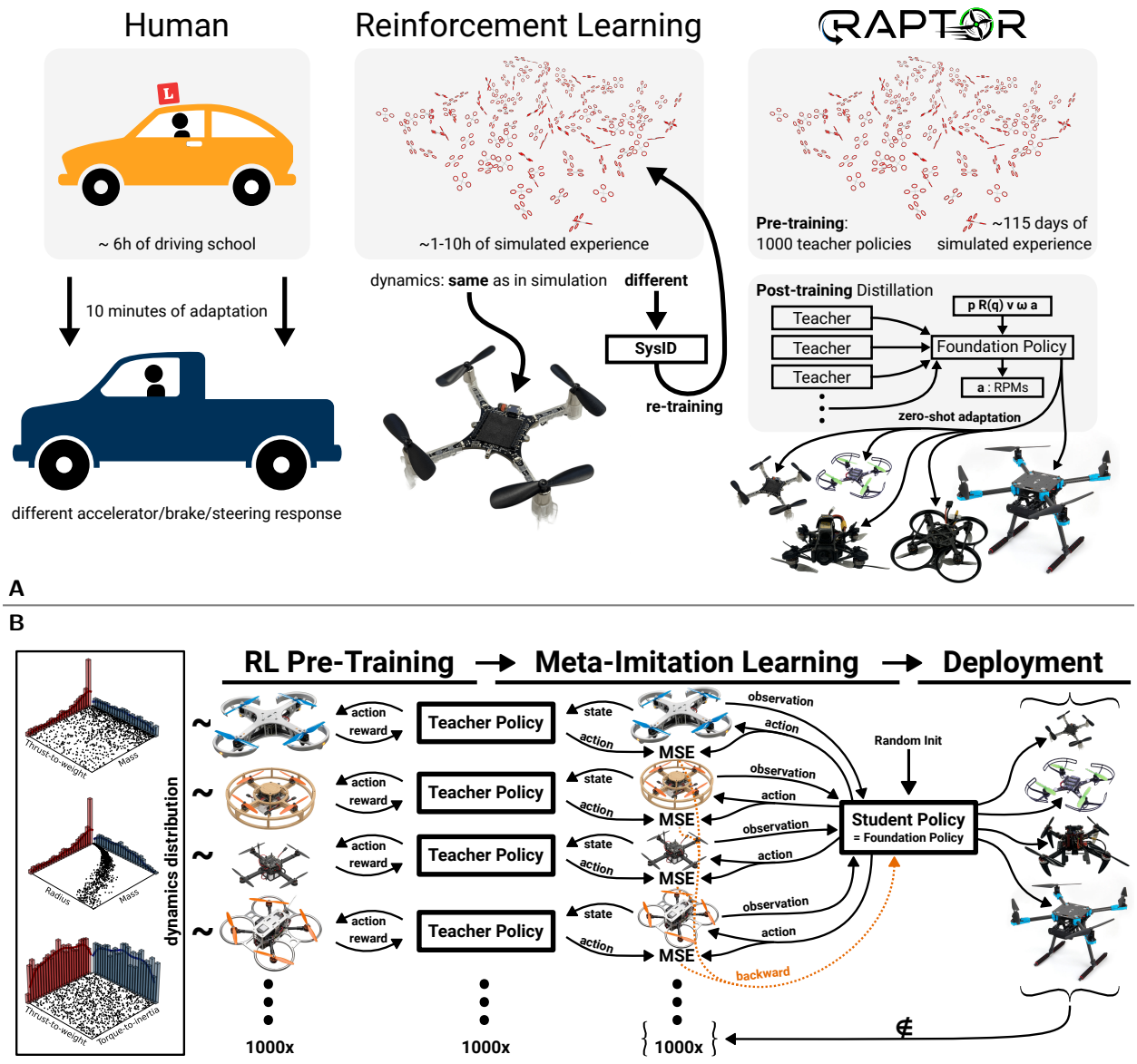



**Figure 1**: **(A) Motivation.** Comparison of the adaptation capabilities of humans, contemporary RL-based control policies and our RAPTOR method. **(B) The RAPTOR Method.** Overview over all stages of the RAPTOR architecture.

On the other hand, this variability also leads to challenges concerning the control of the platform. Changing the mechanical design and even simple modifications like switching the propeller or battery type often requires the retuning of the cascade of classical controllers that is still most widely used today. Analogously, modern, nonlinear control approaches, like Model Predictive Control (MPC) or Reinforcement Learning, heavily rely on accurate system models.

RL policies have recently gained widespread popularity for quadrotor control (*4–7*). While overcoming many challenges, the resulting policies either overfit a single dynamics model or rely on domain randomization without adaptation. Domain randomization of the dynamics parameters (*8*) is a powerful tool, but it forces fully Markovian (non-adaptive) policies to be conservative. Under domain randomization, the policy is incentivized to take conservative actions, e.g., in a state/observation that is critical for some of the quadrotors (e.g., due to thrust or angular acceleration limitations), the policy would have to take the action that saves these quadrotors, even if the current platform is agile enough to take the optimal action to continue the task.

In addition to overfitting a particular system model, in quadrotor racing, it has become standard to also overfit to a particular trajectory (e.g., a certain racetrack) (*9, 10*). This trend limits broad adoption of these methods for real-world use cases because even just changing the environment requires a full retraining of the policy.

Humans, unlike these methods, which need to be retrained from scratch for each particular platform and/or application, can adapt few-shot to new systems. An illustrative example is driving cars (cf. Figure 1A and Movie S1). Initially, humans require extensive training to be able to control a car smoothly and robustly. But when they are driving a new, unseen car, they can adapt quickly. The steering, brake, and accelerator response might be quite different from the one they have experienced before, but they usually only require a handful of tries to adapt.

With this work, we aim to devise a control policy that can adapt to unseen system dynamics using only minimal data, similar to humans. We are inspired by the recent progress on foundation models in the vision (*11*) and language domain (*12*). The main premises of foundation models are:

1. **Broad distribution**: Foundation models are trained on such a broad distribution of data that most expected queries at deployment time will be in-distribution with respect to the training data distribution.

2. **In-Context Learning**: Foundation models take causal sequences as input, facilitating In-Context Learning.

Therefore, the goal is that our policy can quickly adapt to novel systems by interacting with it and using the context/sequence of high-frequency interactions to reason about the dynamics. This can also be seen as emergent implicit system identification. The objective is to control the quadrotor, i.e., based on the observations, output low-level motor commands that achieve an objective (e.g., controlling the position). Since the optimal outputs are dependent on the system dynamics, the (recurrent) policy has to learn to implicitly identify the unobserved/latent dynamics variables on the fly (figuratively and literally).

We refer to it as emergent *implicit* system identification because it only needs to infer the parts of the quadrotor dynamics parameters that are relevant to the input/output behavior of the system. These relevant parts can, e.g., be ratios like thrust-to-weight ratio, torque-to-inertia ratio, but also aggregates of motor delays, thrust curves, etc. We never train any neural network to explicitly

reconstruct system parameters. The only training objective is performing well in terms of the reward function. This fulfills the premise 2) of foundation models.

The premise 1) of foundation models, training on such a broad distribution that all conceivable inference queries are in-distribution, is covered by designing a very broad distribution over dynamics parameters that covers virtually all realistic quadrotors. This distribution is then used to sample quadrotors to train the aforementioned adaptive policy with emergent system identification.

The main research questions are:

1. **Feasibility**: Can a recurrent, end-to-end neural network policy express the described behavior?

2. What **size** (number of parameters) does the recurrent neural network policy require to express this behavior? Can it run in hard real-time at high frequencies when deployed on **small microcontrollers**?

3. What **context window** is feasible? Recurrent neural networks are notoriously hard to train for sequences longer than $100 - 200$ steps. Will the policy *forget* the system dynamics after a short time?

4. Does the policy **generalize** to unseen quadrotors that are 1) in-distribution and 2) out-of-distribution?

5. How much **time** is required from activating the policy until it has gathered enough information to stably control the quadrotor? Is this feasible mid-flight, or would the quadrotor crash before the policy has identified the system properly?

6. Is there a trade-off between agility and adaptability?

We tackle the question of feasibility 1) by devising a method to train such a foundation policy for quadrotor control, implementing it, and testing it on a range of real-world quadrotors.

We tackle the size and inference speed question 2) by studying the scaling laws (*13*) in the student policy and by deploying the final foundation policy directly onto the microcontrollers of even the tiniest quadrotors.

We tackle the context window size question 3) by testing the context window extrapolation beyond the trained size.

We tackle the generalization question 4) by testing the policy on 9 unseen but in-distribution (in terms of thrust-to-weight ratio, torque-to-inertia ratio, motor delays, thrust curves, etc.) quadrotors in the real world. We also study the out-of-distribution performance by testing the foundation policy (a) on a quadrotor with a flexible frame, (b) by installing four different propellers (2- and 3-blade), (c) by hitting it with a tool during flight, and (d) by testing it with a quadrotor that has thrust-to-weight ratio $> 2\times$ higher than the highest one experienced during training.

We tackle the question of the number of timesteps required to infer the system dynamics 5) by studying activation response trajectories, where the policy is activated in mid-air. Here, the policy needs to probe the system and infer the dynamics of it rapidly to restore or maintain stable flight.

We tackle the question 6) about the agility-adaptability trade-off by testing the resulting foundation policy on the task of tracking trajectories from quasi-static to highly dynamic.

Answering these research questions, we provide the following main contributions:

- **RAPTOR** (**R**eal-time **A**daptive **P**olicy **T**hrough **O**nline **R**easoning): A method to train an end-to-end foundation policy for quadrotor control that can adapt to virtually any quadrotor platform zero-shot. This method consists of:
  - Design of a distribution over dynamics parameters that resemble physically plausible quadrotors.
  - A novel distillation method called Meta-Imitation Learning that condenses the behavior of 1000 Markovian teacher policies into a single adaptive/non-Markovian student policy.
  - A formal derivation of the design of the RAPTOR architecture.
- We contribute a highly efficient, open-source implementation of the aforementioned method that allows to reproduce our results even with resource-constrained, consumer-grade hardware.
- We study the scaling laws of the Meta-Imitation Learning process.
- We conduct extensive experiments (indoor and outdoor) across 10 quadrotor platforms with different flight controller setups to validate that the foundation policy resulting from our proposed method answers the stated research questions and attains the stated goals.
- We provide robust and simple means to use our resulting foundation policy for quadrotor control in different flight controller firmwares as well as simulation environments. This facilitates the reproducibility of our results and simplifies its usage as a baseline in future works of the community.

While there have been many works on neural-network-based quadrotor control, most rely on lower-level controllers by, e.g., outputting collective thrust and body-rate setpoints and, hence, are not fully end-to-end (*4, 5, 14–17*). But recently there have also been works investigating full neural-network-based end-to-end control (*18–22*). While these approaches have attained comparable performance to classical control schemes, the control policies are each highly optimized for a single quadrotor. Changing the quadrotor requires system identification to adjust the dynamics parameters of the simulator and a full retraining of the policy. This shortcoming is being tackled by the community right now, with works that investigate better adaptability and/or generalization of neural-network policies to multiple quadrotors. In particular, (*23*) and (*6*) are the most related works to our approach.

In (*6*), the authors train a single neural-network policy that can be deployed onto two different quadrotors with thrust-to-weight ratios of $\approx 5.8$ and $\approx 11.0$. The authors demonstrate impressive agility for racing through gates. The main difference to our approach is that their policy is Markovian/stateless, while ours is adaptive.

Compared to (*6*), (*23*) is more related to our approach because the method intends to train a policy that can adapt to different quadrotors. In (*23*), the authors show deployment of their adaptive policy to two relatively similar quadrotors with thrust-to-weight ratios of 3.23 and 3.62, respectively. The biggest differences to our work are that their approach is not end-to-end. A high-level controller that outputs collective thrusts and body rates is required. Their adaptive policy receives these CTBR setpoints as an input and is not concerned with rotational mechanics, etc. Our foundation policy is a full position controller and hence, in contrast to (*23*), also understands the tilt required to build

up linear velocity and execute translations as well as the angular velocities required to execute a particular tilt. Our policy covers these major non-linear transfer functions while their policy only covers the angular rate (and thrust control) that commonly is just implemented by a simple PD controller. Additionally, our policy architecture is simpler because it does not require training two encoders that map into the same latent space. Furthermore, our policy is vastly lighter, requiring only 2084 parameters while theirs requires 114872 parameters (55× larger), even though our policy covers more complex behavior. Because their policy is so computationally intensive, we cannot run it on any of the 10 quadrotors we are using for testing without modifying the hardware to include a more powerful inference computer. Finally, the authors of (*23*) do not publish their full training code, making it hard to exactly replicate their results. In contrast, we publish the full training and inference code in an easy and future-proof way. Additionally, readers can interact with the foundation policy through the web app at https://raptor.rl.tools.

# 2 Results

Our results show that our method leads to a robust and highly adaptable foundation policy for quadrotor control. In the following, we first discuss insights from the training (pre- and post-training) using our method, as well as results from deploying the single foundation policy to multiple real-world platforms under different conditions.

## 2.1 Training

In the following we describe the training results. The methods used to attain these results are described in the Materials and Methods Section 4. As described in Section 4 and Figure 1B, our training method separates into a pre-training and post-training phase.

## 2.2 Pre-Training

In the pre-training phase, we train 1000 teacher policies that each are specialized for a single quadrotor. The quadrotors differ in their dynamics parameters, which are sampled from the distribution described in Section 4.1.

We find that the pre-training method described in Section 4.2.1 is remarkably robust and that all 1000 RL training runs reliably converge to good policies, all of which can be used for the downstream Meta-Imitation Learning. This is surprising because RL training loops are known to be unstable and hence it is common to run the training with numerous seeds and then cherry-pick the best final policy (*24–29*). In contrast, we can use the same initial seed in all 1000 cases, and we do not need to cherry-pick a different seed for each quadrotor. Note that our training runs are deterministic, i.e., a training run with the same seed always gives identical results (on the same computer). We also release the supplementary dataset Data S1 of 1000 dynamics parameters and the 1000 trained policies for each of them for further study.

The aggregated learning curve resulting from the pre-training phase can be seen in 2A. We use the episode length as the metric here because it is more robust to the variation in quadrotor dynamics than the return. More agile quadrotors, for example, achieve much higher returns than

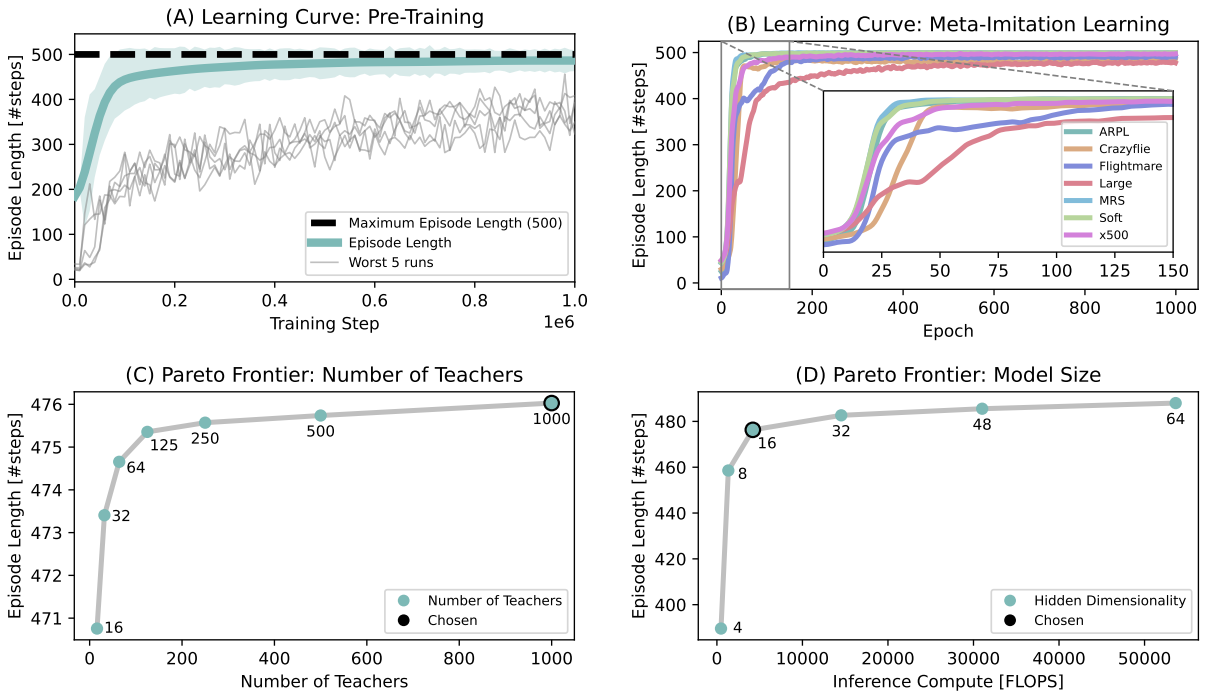



**Figure 2**: **Training Results. (A)** shows the pre-training learning curve, **(B)** shows the meta-imitation learning curve, **(C)** shows the Pareto frontier between performance and number of teachers, and **(D)** shows the Pareto frontier between performance and student/foundation policy size.

less agile quadrotors, even if both are controlled by their respective optimal policy (with respect to the same reward function).

We find that the policies already achieve good behavior after only 100000 steps of training, which is signified by the majority of policies reaching episode lengths of 400 steps or more. We observe that the main flight capabilities are attained in the first 100000 steps, and that subsequent steps refine the behavior when starting in challenging initial conditions (large tilt, large linear velocity, position close to the termination boundary, ...) as shown in the episode lengths converging to the maximum limit. We also observe that, in the later stages of the pre-training, the steady-state performance (which is not well expressed in the episode length) is also still improving.

Since, in contrast to inference, the pre-training only incurs a one-time computational cost, we decide to train each teacher policy for 1 M steps. Each training run takes 31 m on a single core (consumer laptop, AMD Ryzen 9 7945HX, 16 cores). Hence, the full pre-training takes about 34 h on a single consumer laptop (cf. Section 4.2.1 on why and how the pre-training can be parallelized and sped up).

### 2.2.1 Meta-Imitation Learning

During the Meta-Imitation Learning phase, we observe that the student quickly learns to imitate any of the 1000 teachers. In Figure 2B, we plot the test performance of the foundation policy when controlling 7 unseen quadrotors in simulation. The set of 7 quadrotors consists of 5 quadrotors listed in Figure 4 that we deploy the foundation policy to and that we have accurate system parameters

for (which is not the case for the remaining quadrotors in that table). Furthermore, we include two more quadrotors ("MRS" and "Large") that we have accurate system parameters for (from (*30*) and (*31*), respectively).

None of these test-set quadrotors appear in the 1000 pre-training quadrotors. We observe that the acquisition of good flying behavior follows a similar trajectory as a function of the epoch. For most quadrotors, the foundation policy achieves good performance early, but for the "Flightmare" and especially the "Large" model, we see continued improvement by training for longer. Eventually, after about 1000 epochs, we observe convergence and good performance on all 7 platforms, which span a large range of quadrotor dynamics, from agile to non-agile, lightweight to heavy, etc.

In Figure 2C, we study the performance of the resulting foundation policy when using the RAPTOR framework with different numbers of pre-training teachers/quadrotors. Here, the aggregate performance is shown across all 7 test quadrotors, and we observe that, even for a low number of teachers, we see that the foundation policy is able to stabilize/control its position from various and challenging initial states without crashing.

As mentioned before, the episode length is an imperfect metric for the full performance of the policy, but, nevertheless, it is more suitable than the return, which strongly varies in scale, even just between the 7 test-set quadrotors. With a training time of 1.9 h (same AMD Ryzen 9 7945HX laptop), Meta-Imitation Learning is vastly less computationally intensive than the pre-training (1.9 h vs. $1000 \times 31$ m).

The training part of the RAPTOR framework only incurs a one-time cost compared to the deployment part, which is more resource-constrained and hence more sensitive computationally. Hence, we decide to employ all 1000 teachers in the main configuration of the RAPTOR framework.

Furthermore, and with the aforementioned resource constraints in mind, we also study the scaling behavior in the model size. In Figure 2D, we can see the Pareto frontier between inference compute in terms of Floating Point Operations (FLOPS) and performance in terms of episode length. Since, in this case, a larger model size actually entails continued computational costs during deployment, we chose a relatively small model with a hidden dimensionality of 16.

This small size allows the deployment to even the tiniest microcontrollers. Even on the most resource-constrained quadrotors, the resulting foundation policy requires $< 10\%$ of the available computational power, leaving plenty of capacity for state-estimation, communication and other duties of the flight controller.

Hence, going forward (and in Figure 2A/B), we set the number of teachers to 1000 and the size of the hidden dimension in the policy to 16.

## 2.3 Emergent System Identification

To answer the stated research questions we also study the behavior of the foundation policy resulting from Meta-Imitation Learning. We find that it indeed exhibits in-context learning as can be seen in Figure 3. Here we start a quadrotor in a state where it is displaced by 2 m from the target position in x and y as well as pitched backwards by 90° (no linear or angular velocity). First of all, we can see that the policy successfully manages to recover from this initial state and to reduce the distances towards the target state in position, orientation, linear and angular velocity.

Additionally, we plot the trajectory of the hidden state and investigate if it learns something about the dynamics of the quadrotor it is interacting with. We know the ground-truth dynamics parameters of the 1000 quadrotors used for pre-training, hence we can train a linear probe (*32*) to

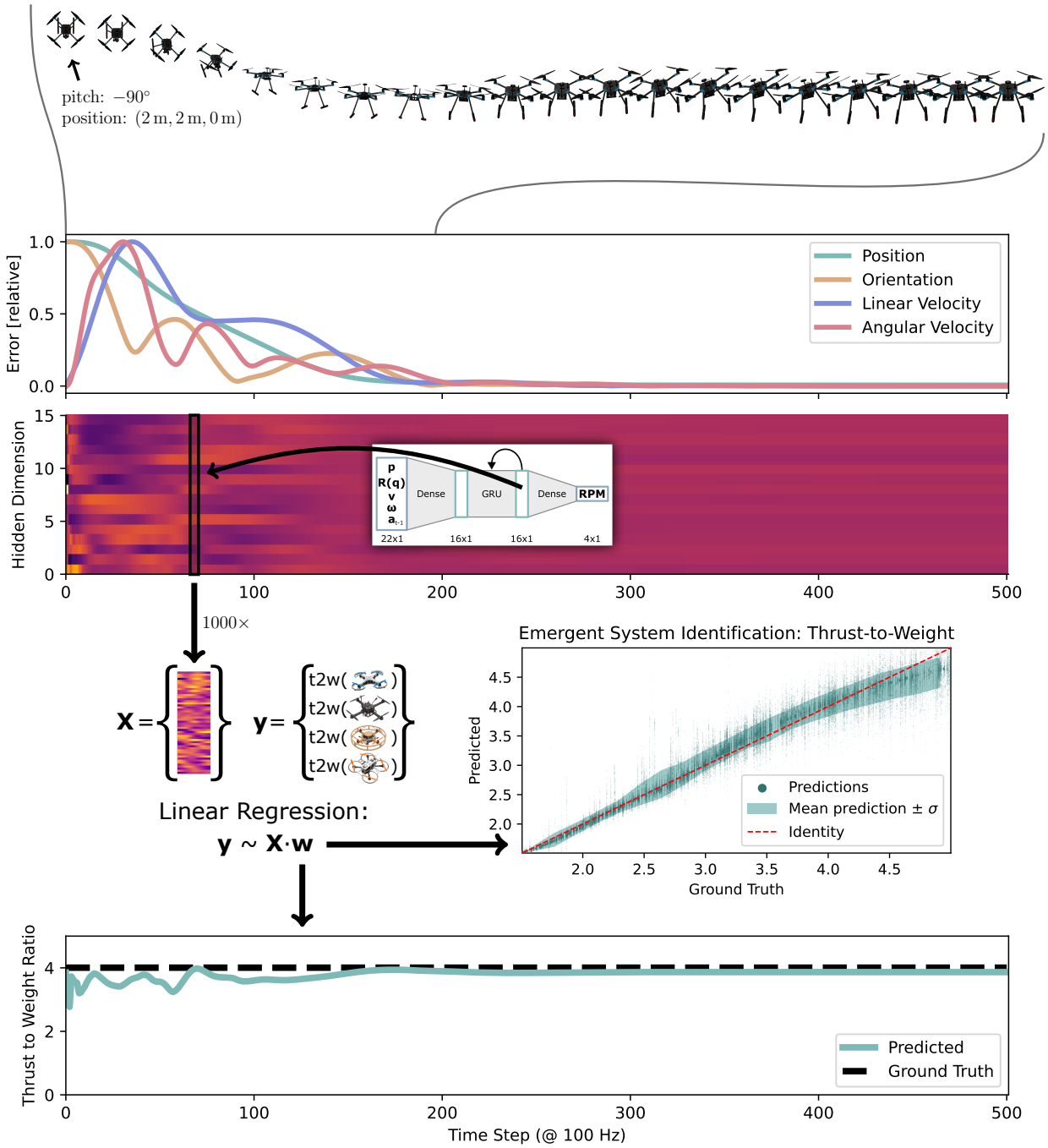



**Figure 3**: **Inference Results.** Here we show a recovery of a simulated quadrotor from an adverse initial condition using the trained foundation policy. We show the latent state of the policy throughout the trajectory and test if it performs emergent/implicit system identification by training a linear probe.

| | Humming-bird | Crazyflie | M5Stamp-Fly | Crazyflie Brushless | Meteor75 Pro | Pavo20 | SavageBee Pusher | ARPL | Soft | x500 | Gazebo | Flightmare |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Firmware** | Betaflight | Crazyflie | M5StampFly | Crazyflie | Betaflight | | | PX4 | | | | Flightmare |
| **Motors** | brushless | brushed | | | brushless | | | | | | | |
| **#Blades** | 3 | 2 | 4 | 2 | 3 | 2 | 2 | 2/3 | 3 | 2 | 2 | 2 |
| **Battery** | 1S | | | | | 2S | | 4S/6S | 4S | | - | - |
| **Weight** | 31.9 g | 35.2 g | 38 g | 42.3 g | 40.4 g | 91.8 g | 155.4 g | 801 g | 948 g | ~1.2-2.4 kg | 1.5k kg | 730 g |
| **Thrust-to-weight** | ? | ~ 1.75 | ? | ? | ? | ? | ? | ~ 3.9 | ~ 3.3 | ~ 3.2 | ~ 2.3 | ~ 12 |
| **Wheel-base** | 65 mm | 80 mm | 65 mm | 100 mm | 80.8 mm | 90 mm | 120 mm | 250 mm | 441 mm | 500 mm | 286 mm | 340 mm |
| **Configuration** | puller | | | | | pusher | | puller | | | | |
| **State-estimator** | Mahony | EKF | Madgwick | EKF | Mahony | | | EKF | | | | GT |
| **Real** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |
| **Prop-guards** | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ |
| **Symmetric** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ |
| **Flexible** | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| **Price** | $99 | $240 | $49 | $480 | $105 | $125 | $399 | not COTS | not COTS | $500 | free | free |

**Figure 4**: **Test Quadrotors.** A diverse set of 10 real and 2 simulated quadrotors that we use in the experiments.

predict, e.g., the thrust-to-weight ratio. Linear probing has become a standard test to evaluate if foundation models learn good representations (*11, 33, 34*).

As can be seen from the regression plot in Figure 3, just a linear probe can predict the thrust-to-weight ratio very well. We use an 80%-20% train-test split and the linear model achieves a Mean-Squared Error (MSE) of 0.047 and an $R^2$ of 0.949, significantly reducing the a priori uncertainty about the thrust-to-weight ratio. This shows that Meta-Imitation Learning leads to emergent implicit system identification in the latent space of the foundation policy.

While quickly going into the right range, we can also see that the estimate is improving over time, showing the in-context learning of the policy.

## 2.4 Deployment

To show the robustness and adaptability of the foundation policy that results from applying the RAPTOR framework, we deploy the foundation policy onto 10 different real quadrotors and 2 different simulators (cf. Figure 4) showing Simulation-to-Reality (Sim2Real) and Simulation-to-Simulation (Sim2Sim) transfer. We aim to strain the adaptation capabilities of the foundation policy as much as possible by testing across a wide range of parameters:

1. A quantitatively wide range of parameters:
    - Weight: 31.9 g - 2.4 kg
    - Size: 65 mm - 500 mm
    - Thrust-to-weight: ≈ 1.75 - 12
2. A qualitatively diverse set of features:
    - Flight controller: PX4, Betaflight, Crazyflie, M5StampFly

- State estimator: EKF, Mahony, Madgwick
- Motor type: brushed and brushless
- Flexible frame
- Mixing two- and three-blade propellers

Many of these quantities are (far) out-of-distribution, like a thrust-to-weight ratio of 12 ($\leq 5$ in training), a flexible frame (only rigid during training), and observations from a state estimator (Ground Truth (GT) during training).

This shows that our proposed RAPTOR framework actually produces a policy that not only generalizes to quadrotors that are in the training distribution (cf. Section 4.1) but also out-of-distribution (OOD).

Contrary to popular belief, and supporting the results in (*7*), we find that the simulation-to-reality transfer of end-to-end neural network policies is actually not hampered by the mismatch in the dynamics model, especially because the parameters can be relatively easily and accurately determined using (*31*). We find that qualitative difference matters much more. Especially for stateful policies (like the RNN used in the RAPTOR foundation policy), implementation details in the firmware that lead to delays and other artifacts have a strong impact on the simulation-to-reality deployment.

We find that the foundation policy works robustly on all platforms, but we also observe low-frequency z-axis oscillations in some of the non-EKF-based quadrotors. In the case of the Mahony (*35*) and Madgwick (*36*) filters, only the orientation is estimated by the filter and the velocity is directly fed from the motion capturing system.

We hypothesize that this leads to the z-axis oscillations due communication delays. We can reproduce the z-axis oscillations in simulation across quadrotors of different dynamics parameters by inducing a linear velocity delay of $10 - 30$ ms.

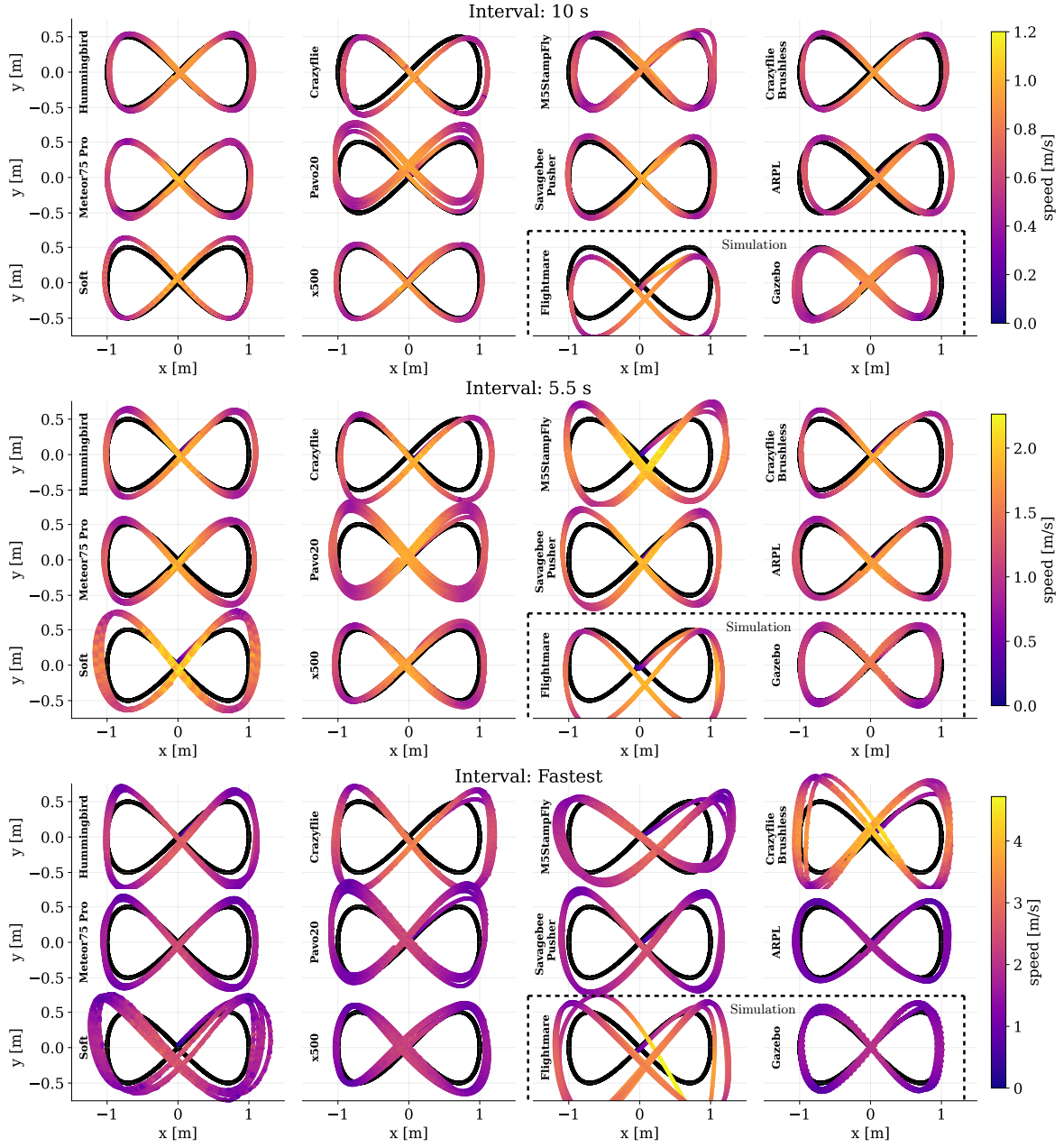
Intuitively, it makes sense that the foundation policy heavily relies on changes in the linear velocity to estimate the acceleration that is caused by the series of actions it produced previously. The policy can use these observations to estimate dynamics parameters like the thrust-to-weight ratio. Please refer to the Supplementary Materials (*37*) for the mitigation we implemented for this.

### 2.4.1 Trajectory Tracking

Besides position control (supporting "goto" workflows), trajectory tracking is an important task for real-world applications. Despite only training with random, relatively slow reference trajectories, we find that the resulting foundation policy is able to track figure-eight trajectories of varying agility well.

Figure 5 and Movie S2 show trajectory tracking of a Lissajous-based figure-eight trajectory at different intervals. We test tracking the 10 s and 5.5 s trajectories using the foundation policy on all 12 quadrotors. Each of the shown trajectories constitutes 5 consecutive full loops with an initial linear ramp-up time of 1 s (from hovering). We find that the trajectory tracking performance is good in all real-world cases.

As can be seen from the Root Mean Square Error (RMSE) values in the bottom table, the tracking performance is in line with state-of-the-art policies (without trajectory lookahead, like the foundation policy) that are trained for a single quadrotor (*7*). The dedicated policy that has been




| Interval | Metric | Humming-bird | Crazyflie | M5Stamp-Fly | Crazyflie Brushless | Meteor75 Pro | Pavo20 | Savagebee Pusher | ARPL | Soft | x500 | Flightmare | Gazebo |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 s | RMSE [m] | 0.08 | 0.12 | 0.08 | 0.08 | 0.10 | 0.19 | **0.07** | 0.17 | 0.09 | 0.11 | 0.32 | 0.14 |
| | RMSE w/o z [m] | 0.08 | 0.12 | **0.07** | 0.08 | 0.08 | 0.19 | **0.07** | 0.16 | 0.09 | 0.10 | 0.22 | 0.13 |
| | Max velocity [m/s] | 1.02 | 1.20 | 1.03 | 1.01 | 1.11 | 1.06 | 1.10 | 1.06 | 1.09 | 1.05 | 1.04 | 0.96 |
| 5.50 s | RMSE [m] | 0.18 | 0.19 | 0.29 | **0.17** | 0.19 | 0.21 | 0.18 | 0.22 | 0.26 | 0.21 | 0.35 | 0.19 |
| | RMSE w/o z [m] | 0.17 | 0.19 | 0.28 | **0.16** | 0.17 | 0.21 | 0.18 | 0.20 | 0.26 | 0.21 | 0.23 | 0.18 |
| | Max velocity [m/s] | 2.06 | 2.06 | 2.26 | 1.97 | 2.03 | 1.97 | 2.11 | 1.96 | 2.24 | 1.95 | 1.98 | 1.62 |
| Fastest | Interval [s] | 4.5 | 3.5 | 3.5 | **3.0** | 4.5 | 4.5 | 4.5 | 4.0 | 4.5 | 4.5 | 3.5 | 4.5 |
| | RMSE [m] | 0.27 | 0.37 | 0.59 | 0.50 | 0.23 | 0.26 | 0.24 | 0.22 | 0.34 | 0.27 | 0.40 | 0.22 |
| | RMSE w/o z [m] | 0.27 | 0.32 | 0.54 | 0.38 | 0.22 | 0.26 | 0.24 | 0.20 | 0.33 | 0.26 | 0.31 | 0.20 |
| | Max velocity [m/s] | 2.99 | 3.55 | 3.16 | 4.47 | 2.57 | 2.62 | 2.70 | 1.96 | 2.97 | 2.48 | 4.72 | 2.19 |

**Figure 5**: **Trajectory Tracking Results.** Trajectory tracking results of the 10 real and 2 simulation quadrotors.

specifically trained for deployment on a Crazyflie in (*7*) reaches an RMSE tracking error of 0.17 m and 0.15 m (with and without the z-axis) for the identical 5.5 s figure-eight trajectory. The foundation policy resulting from the RAPTOR framework reaches 0.19 m and 0.19 m, respectively, on the same platform. While the tracking error of the foundation policy is slightly elevated compared to the dedicated policy, through online adaptation, the foundation policy can attain a similar performance on a plethora of other quadrotors as well.

From the RMSE difference between including the z-axis and excluding it, we can see that for all real quadrotors, the tracking error is mostly in the x-y plane. This shows that the foundation policy successfully adapts to the different thrust-to-weight ratios, battery levels, and other conditions, and adaptively cancels out the z-error.

The third cluster of trajectories in Figure 5 shows the fastest trajectory that we tested for each quadrotor. This does not necessarily mean that this is the fastest trajectory supported by the foundation policy, because, to avoid crashes, we did not push all quadrotors (especially the larger ones) beyond their limits. For the Hummingbird, Crazyflie, M5StampFly, Crazyflie Brushless, and Meteor75 Pro, we did push them beyond their (or the foundation policy's) limits and find that the shown trajectories are the most agile ones that can still be tracked with decent accuracy. When pushing, e.g., the Hummingbird or the Meteor75 Pro further, they still remain stable, but they overshoot so much that the figure-eight becomes barely recognizable.

Based on this observation, we hypothesize that for agile trajectory tracking, the foundation policy is mainly bottlenecked by the lack of trajectory lookahead in the observations. This analysis is also based on prior lookahead-free works (*7*) and recent works ablating the inclusion of lookahead (*38*).

As can be seen from the maximum velocity measurements, we push the foundation policy up to $3 - 4$ m/s in these indoor experiments.

### 2.4.2 Outdoor Tests

Additionally, we conduct outdoor tests using the x500 platform. During these tests, there was a strong wind of about 7 m/s, gusting up to 10 m/s. We test trajectories resulting in linear velocities of up to 10 m/s over ground and more than 15 m/s relative to the wind. We did not see signs of instability at these speeds, and will investigate larger speeds in future work.

Furthermore, we equip the quadrotor with up to 1.2 kg of payload (water-filled bottles), which, including the battery, is slightly above the specified maximum payload capability of the platform of 1.5 kg. This setup leads to a take-off weight of 2.4 kg, and the foundation policy is still able to control the quadrotor when hovering. With a payload of a single bottle (600 g), we can still track trajectories, as can be seen in the supplementary Movie S3.

### 2.4.3 Disturbances

We test a variety of disturbances, including hitting the quadrotors with a tool, resting the tool on top of the quadrotor while flying, as well as wind disturbances from a strong fan. Examples of these disturbances can be seen in Figure 6 and Movies S4 and S5. In Figure 6D and Movie S4, we strongly hit the quadrotor (that starts out hovering) from the bottom, leading to a strong tilt in excess of 90°. While losing altitude, the foundation policy still manages to recover the quadrotor. In Figure 6E, we rest the tool on top of a flying quadrotor, and the foundation policy quickly adapts to the added weight and does not incur a steady-state altitude deficit.

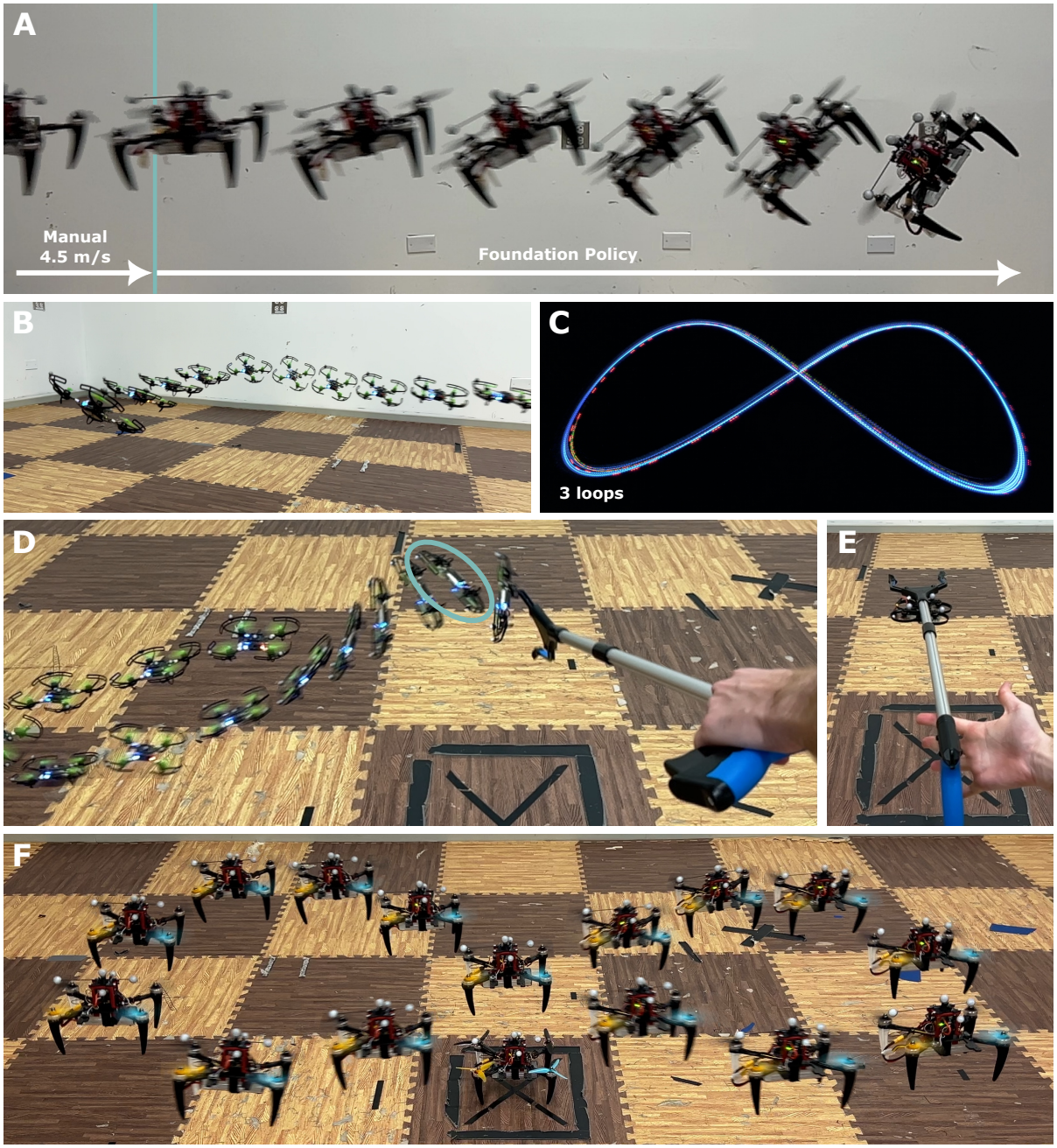



**Figure 6**: **RAPTOR Policy in Different Situations.** **(A)**: The foundation policy is activated in mid-air, starting with a linear velocity of 4.5 m/s. **(B)** Crazyflie Brushless tracking an agile trajectory. **(C)** Long exposure photo of three consecutive loops of the Crazyflie Brushless trajectory. **(D)** The foundation policy recovers from being poked and tilting > 90°. **(E)** A tool is rested onto the quadrotor flown by the foundation policy. **(F)** A quadrotor with four different propellers (2- and 3-blade) is tracking a trajectory using the foundation policy.

Additionally, we also test swapping out 1, 2, and 3 propellers with random three-blade ones (the original ones are two-blade) and find that, in any of these configurations, the foundation policy still stably controls the vehicle and is even able to track trajectories (cf. Figure 6F and Movie S6). Note that the distribution of quadrotors described in Section 4.1 covers only quadrotors with identical thrust curves for all four motors/propellers. Hence, neither the teacher policies nor the foundation policy has ever experienced differing thrust curves on the same quadrotor. Yet, we find that the foundation policy can generalize zero-shot outside of this distribution, and control the quadrotor well.

In the absence of disturbances, we find that the foundation policy yields a remarkably repeatable performance, as can be seen from the trajectories in Figure 5, which show 5 consecutive full loops each. This also answers the posed research question about the trajectory length generalization of the foundation policy. During Meta-Imitation Learning, we train the policy with sequences of 500 steps, corresponding to 5 s of flight time. During inference, we test the foundation policy with 5 consecutive iterations of, e.g., a 10 s, amounting to a trajectory length of 50 s. This shows a 10× context window size extrapolation.

In practice, we did not notice any limits on how long the policy can be activated at a time. We fly until the battery is empty (several minutes) many times over. Hence, we believe the resulting foundation policy can generalize to arbitrary trajectory lengths without degrading performance, despite only being trained on fixed-size trajectories.

Additionally, in Figure 6C, we can see the precise repeatability through a long-exposure photo of three full loops. The deviations between loops are barely recognizable. Figure 6B shows a chronophotograph of the same trajectory with the lights on.

Furthermore, we also test the recovery from aggressive initial states. An example of this is shown in Figure 6A and Movie S7, where the quadrotor is accelerated to 4.5 m/s using manual control and then the foundation policy is activated in mid-air. The policy is activated using a push button on the transmitter, and its goal (through offsetting of the observations) is to hover at the position where it was activated.

When it is activated, the hidden state is reset and the policy has to adapt zero-shot, in minimal time, to save the quadrotor.

Note that all of the previously described experiments (and more) are included in the supplementary video material, where these agile maneuvers and disturbances can be observed in motion. In Movie S2, we also show flying 6 different quadrotors (Hummingbird, Crazyflie, M5StampFly, Crazyflie Brushless, SavageBee Pusher, ARPL) from Figure 4 with 4 different firmwares and 4 different communication protocols at the same time in a tight indoor space. We find that, despite the turbulent flow created by the other quadrotors that are flying close by, the foundation policy still manages to stably control each of the quadrotors. We also accidentally fly a SavageBee Pusher (155.4 g) and a Crazyflie Brushless (42.3 g) directly beneath a hovering ARPL platform (801 g), and find that the foundation policy manages to quickly adapt and adjust to/recover from the disturbance. Furthermore, Movie S8 contains yaw step-response tests.

### 2.4.4 Simulation

To further test the out-of-distribution generalization beyond the aforementioned real-world tests including flexible frame, mixed propellers, context window extrapolation, etc., we test simulation-to-simulation transfer to the Flightmare simulator. This transfer is interesting, because the thrust-

to-weight ratio of its default quadrotor is $\approx 12$ (cf. Figure 4) and hence $> 2\times$ the upper limit of 5 of our domain randomization range. We find that the z-error is significant, but the foundation policy still controls it robustly and can track agile trajectories. This shows the remarkable robustness and out-of-distribution generalization of the foundation policy resulting from the RAPTOR method.

# 3 Discussion

In our extensive experimental evaluations, we find that our proposed RAPTOR framework produces a highly robust and versatile foundation policy that can control a broad range of quadrotors in a large variety of situations. We believe that our experiments validate the design choices in the RAPTOR architecture.

To facilitate research, we aim for the highest levels of reproducibility, publishing all training code for the benefit of the community. Furthermore, we make it extremely easy to integrate the foundation policy into existing flight controller software and simulators, such that the barrier to use it as a baseline is minimal.

# 4 Materials and Methods

We formulate the quadrotor control problem as a Bayes Adaptive Partially Observable Markov Decision Process (BAPOMDP) (*39*) defined by the tuple $(\mathcal{S}, \mathcal{S}_0, \mathcal{D}, \mathcal{A}, \mathcal{T}, r, \mathbf{\Omega}, \mathbf{o}, \gamma, \mathbf{\Xi})$. $\mathcal{S}$ is the set of states $\mathbf{s} = \{\mathbf{p}, \mathbf{q}, \mathbf{v}, \boldsymbol{\omega}, \mathbf{a}_{t-1}, \boldsymbol{\omega}_m\}$ consisting of position, orientation, linear/angular velocity, previous action and motor states respectively. The previous action is included in the state because the reward function penalizes the change in action. $\mathcal{S}_0$ is the initial state distribution with position, orientation, linear/angular velocity uniformly sampled up to $10 \cdot l_{\text{arm}}$, 90°, 1 m/s, 1 rad/s respectively. With a probability of 10%, the initial state is overwritten with the target state (all zeros). $\mathcal{D}$ is the termination relation that includes all states outside $20 \cdot l_{\text{arm}}$ m, 2 m/s, 35 rad/s (position, linear/angular velocity respectively). $\mathcal{A}$ is the set of actions $\mathbf{a} = \{\omega_{\text{sp}_0}, \omega_{\text{sp}_1}, \omega_{\text{sp}_2}, \omega_{\text{sp}_3}\}$ (individual motor commands). The transition probabilities $\mathcal{T}$ are defined as $\text{p}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{\Xi})$ and implemented by the L2F simulator (*7*). The reward function is deterministic:

$$\text{r}(\mathbf{s}_t, \mathbf{a}, \mathbf{s}_{t+1}) = \|\mathbf{p}\|_2 + 0.2 \cdot \arccos(1 - |q_z|) + \|\mathbf{a}_t - \mathbf{a}_{t-1}\|_2 + 1.5 - 100 \cdot \mathbf{1}[\text{terminal}(\mathbf{s}_{t+1})] \quad (1)$$

We include the next state in the reward function to be able to inflict the termination penalty. The observation space $\mathcal{O}$ contains a subset of the state space $\mathbf{o} = \{\mathbf{p}, \mathbf{R}(\mathbf{q}), \mathbf{v}, \boldsymbol{\omega}, \mathbf{a}_{t-1}\}$, occluding the motor states $\boldsymbol{\omega_m}$ (not observable on most real-world platforms). $\gamma = 0.99$ is the discount factor and the domain parameters are collected in $\mathbf{\Xi} = \{m, l_{\text{arm}}, c_{f_0}, c_{f_1}, c_{f_2}, c_m, \mathbf{J} = \text{diag}(J_{xx}, J_{yy}, J_{zz}), T_{m\uparrow}, T_{m\downarrow}\}$ containing the mass, arm length, thrust-curve coefficients (zeroth, first and second order), moment coefficient, inertia matrix and rising/falling edge motor delays respectively. The difference between a BAPOMDP and a POMDP is that we can factor out the system parameters $\mathbf{\Xi}$ (which would have to be encoded into the state in a normal POMDP). This factorization allows us to implement the inductive bias that the parameters are only sampled once at the beginning of the episode and remain constant throughout.

Using this BAPOMDP framework, in the following, we provide a formal derivation of our method using a probabilistic graphical model (*40*). The full model is shown in Figure 7A. The goal is

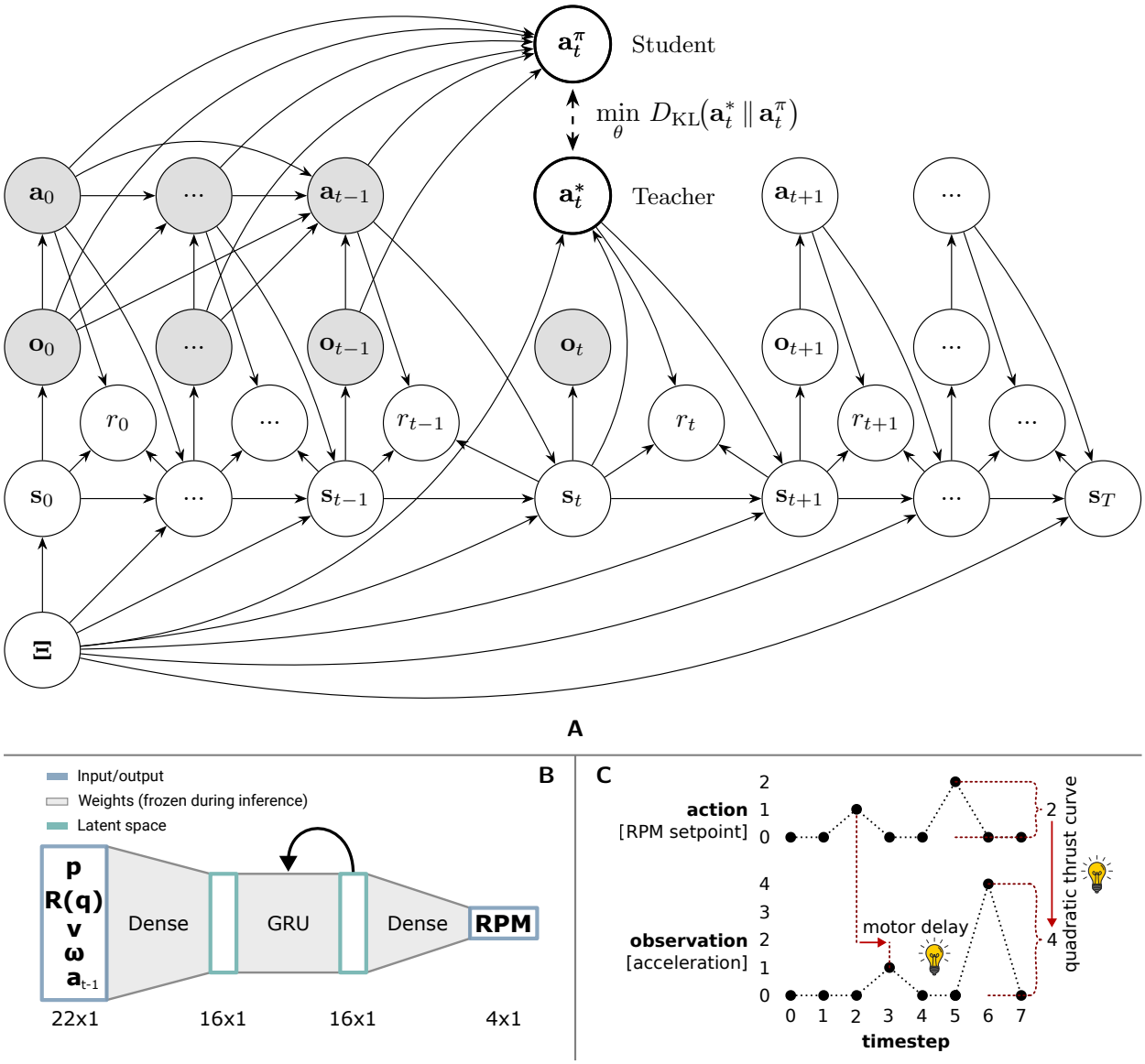



**Figure 7**: **(A)** A probabilistic graphical model (Bayesian Network) of the dynamics and control of a random quadrotor. This formal model allows us to derive the RAPTOR architecture from probabilistic principles. **(B)**: Foundation policy network architecture. **(C)**: Illustration of inferring dynamics parameters by reasoning about the observed input/output behavior of the system.

to model the decision-making at time $t$. At timestep $t$, $t$ previous (and current) observations have been observed (shaded nodes) as well as $t-1$ previous actions. We explicitly place minimal assumptions on the policy that decides the previous actions, which shows by the previous actions being causally fully connected to all previous actions and observations. By causally fully connected, we mean that an action at timestep $t$ might depend on observations $\mathbf{o}_0, \cdots, \mathbf{o}_t$ and actions $\mathbf{a}_0, \cdots, \mathbf{a}_{t-1}$. Hence, the only assumption is that the policy generating the previous trajectory is causal.

Since we want to maximize the expected discounted return (sum of rewards), we model the probability of some action $\mathbf{a}$ being the optimal action $\mathbf{a}_t^* := \mathbf{a}$. The optimal action is independent of previous states (and all other past random variables) given the current state $\mathbf{s}_t$ and the dynamics parameters $\boldsymbol{\Xi}$:

$$\mathbf{a}_t \perp\!\!\!\perp \mathbf{s}_0, \ldots, \mathbf{s}_{t-1} \mid \mathbf{s}_t, \boldsymbol{\Xi} \tag{2}$$

Due to the forward-looking nature of maximizing the discounted future returns, for completeness, we assume that future actions are also optimal. In our proposed Meta-Imitation Learning method, this distribution over the optimal action $\mathbf{a}_t^*$ is approximated by a teacher policy that is trained using RL. In our method, we train 1000 teacher policies for each of the 1000 randomly sampled quadrotors/dynamics parameters $\boldsymbol{\Xi}$. This fits into the formal model in Figure 7A because the practical, finite number of dynamics parameters/teacher policies can be viewed as a Monte Carlo approximation for the mixture model, where a unified teacher is conditioned on $\boldsymbol{\Xi}$.

Since we want to train a foundation policy that can adapt to any realistic quadrotor and does not require knowledge of identified system parameters, and due to the motor states being unobservable on most quadrotor platforms, we can observe neither of the two direct ancestors $\mathbf{s}_t$ and $\boldsymbol{\Xi}$ of the optimal action.

Using d-Separation (*41*), an algorithm to prove conditional independencies in probabilistic graphical models, we can show that not observing $\mathbf{s}_t$ and $\boldsymbol{\Xi}$ makes the distribution over the optimal action $\mathbf{a}_t^*$ dependent on other variables that could carry information about them. The most direct example is $\mathbf{o}_t$. Since $\mathbf{o}_t$ is derived from $\mathbf{s}_t$, it can carry information about $\mathbf{s}_t$ and in our case it carries almost all the information about the state, apart from the motor speeds. Hence, when $\mathbf{o}_t$ is observed, this changes the distribution over the optimal action $\mathbf{a}_t^*$. The same argument holds for $\mathbf{o}_{t-1}$ and earlier observations because, e.g., $\mathbf{o}_{t-1}$ carries information about $\mathbf{s}_{t-1}$ and $\mathbf{s}_{t-1}$ (in combination with the observable $\mathbf{a}_{t-1}$) carries all the information to infer the distribution over $\mathbf{s}_t$. This is because $\mathbf{s}_{t-1}$, $\mathbf{a}_{t-1}$, $r_{t-1}$, and $\boldsymbol{\Xi}$ form the causal Markov blanket (*41*) for $\mathbf{s}_t$. For the causal Markov blanket, random variables that lie in the future are removed. Therefore, using the d-Separation rules, we can see that the distribution over the optimal action $\mathbf{a}_t^*$ is dependent on all previous observations and actions. There is no set of observable variables that could form a Markov blanket and shield this dependency. This motivates our design decision that the foundation policy, which is the student policy from the Meta-Imitation Learning perspective, is dependent on all previous observations and actions. Intuitively, the dependence on all previous observations and actions makes sense because any of the individual observations, or, more likely, the combination of observations over time, contains information about the dynamics parameters $\boldsymbol{\Xi}$ and about the ground-truth state $\mathbf{s}$.

This is illustrated in Figure 3 and Figure 7C where we show how the relation of observations over time carries information about the dynamics parameters. By looking at the observation-action history, the policy can observe that e.g., the effect of the RPM setpoint actions is always delayed by one timestep. Note that this is a simplification by using a pure delay/dead time, while in reality the temporal relationship between RPM setpoints and motor speeds is more closely modeled as a

first-order low-pass. Furthermore, it can be observed that an RPM setpoint of 1 corresponds to an observed acceleration of 1, but an RPM setpoint of 2 corresponds to an observed acceleration of 4. Hence it can infer the curvature of the thrust curve.

Therefore, from both the theory and intuition, we can conclude that the foundation policy should be able to take past observations and actions as the input and then output the predicted distribution over optimal actions as closely as possible. Therefore, we model the action distribution of the foundation policy (student in the Meta-Imitation Learning framework) with dependencies on all previous observations and actions in Figure 7A. Even given the whole history of observations and actions, there might still be mutual information left between $\mathbf{s}_t$ and $\boldsymbol{\Xi}$. Hence, we cannot expect the distribution $\mathbf{a}_t^\pi$ to model the optimal action distribution $\mathbf{a}_t^*$ exactly.

Instead, we want to model it as closely as possible and hence phrase the problem as variational inference, where we try to minimize the Kullback-Leibler (KL) divergence between the predicted optimal action distribution and the actual optimal action distribution. Note that due to the non-linearities in the system dynamics, etc., the actual optimal action distribution is not tractable and that we approximate it by training expert teacher policies until convergence.

Please refer to the Supplementary Materials (*37*) for a full derivation of the Mean-Squared Error (MSE) training objective from Maximum Likelihood Estimation (MLE).

While the Bayesian Network in Figure 7A is a formal/mathematical probabilistic model, Figure 1B shows our proposed practical method for modeling the various conditional probability distributions that constitute it:

1. **Dynamics Distribution/Domain Randomization**: This distribution implements the $\boldsymbol{\Xi}$ node.

2. **RL Pre-Training**: We use RL to train teacher policies that act as an oracle for the optimal action distribution node $\mathbf{a}_t^*$. Since we train 1000 specialized actors, the $\mathbf{a}_t^*$ node is implemented by a mixture policy, where the selected teacher is dependent on the dynamics parameters $\boldsymbol{\Xi}$, which is also well characterized by the Bayesian Network through the dependence of $\mathbf{a}_t^*$ on $\boldsymbol{\Xi}$.

3. **Meta-Imitation Learning / Student Policy**: This models the partially observable action distribution node $\mathbf{a}_t^\pi$ and is implemented by a recurrent policy, which takes a history of observations and actions as input.

4. **Deployment**: We deploy the foundation policy onto different, unseen, real-world quadrotors. We assume that the dynamics of most real quadrotors are in-distribution with respect to the distribution over dynamics parameters $\mathrm{p}(\boldsymbol{\Xi})$.

In the following, we describe these modules of the RAPTOR architecture in more detail.

## 4.1 Domain Randomization

We want the resulting foundation policy to be able to control a wide range of quadrotors. The RAPTOR philosophy is to employ radically wide domain randomization over $\boldsymbol{\Xi}$ and take advantage of emergent meta-learning (*42*) to produce a foundation policy that can adapt to unseen quadrotors zero-shot. To facilitate this, we need to design a distribution over realistic quadrotors that assigns a sufficient amount of probability mass to real-world quadrotors.

We are mainly concerned with the mass, geometric dimensions, inertia, thrust curves, torque coefficients, and motor delays, since these capture the most important parts of the quadrotor dynamics. These quantities are correlated in non-linear ways, making it intractable to directly formulate the joint distribution in analytical form.

Hence, we factorize the distribution based on physical properties. By formulating the distribution in this factorized way, we can use efficient ancestral sampling to sample new quadrotors. Please refer to the supplementary Figure S1 for a graphical model corresponding to this factorization and ancestral sampling scheme. Due to this scheme, we prevent having to resort to heavy sampling mechanisms like Markov Chain Monte Carlo (MCMC), and can sample the root (shaded) quantities from simple, independent uniform distributions. Even though the marginal distributions are independent, the computed nodes are dependent on multiple inputs and are correlated in a physically plausible way.

Please refer to the Supplementary Material (*37*) for the detailed equations that establish the physically plausible correlations.

## 4.2 Training Methodology

After establishing a realistic distribution over quadrotors, the question arises on how to devise a foundation control policy that can adapt to any one of them. We initially experimented using end-to-end RL with a single recurrent policy and critic, in the spirit of meta-RL (*43, 44*), but we did not see signs of convergence, and the training was very time-intensive due to the sequential nature of training Recurrent Neural Networks (RNNs).

Since we knew that by combining (*7*) and (*31*), we can train good individual RL policies for a wide range of quadrotors, and due to the dependencies derived from the probabilistic model, we made the design choice to factorize the architecture into a pre-training and post-training/Meta-Imitation Learning stage. This architectural division is also inspired by the common practice of splitting the training of language and vision foundation models into pre- and post-training (*45*).

### 4.2.1 Pre-Training

In the pre-training phase, we take 1000 quadrotors sampled from the distribution described in Section 4.1 and train a dedicated expert policy for each of them by creating an independent MDP for each set of sampled domain parameters $\Xi$. We also make the ground-truth states directly observable because this expert policy does not need to be deployed on hardware. Since we are not constrained by the deployment onto hardware, we can overparameterize it to aid the training (*46*). We use three layers with a hidden dimensionality of 64, making each teacher policy $> 3\times$ larger than the condensed foundation policy in terms of parameters. The training pipeline is adapted from (*7*) with five modifications:

1. Switching from TD3 to SAC because we observed slightly more robust training dynamics in SAC.

2. Training for longer to ensure convergence for all quadrotors.

3. Adjusting the reward function, adding a penalty for termination and for the action derivative.

4. Removing the curriculum because we found that the changes to the reward function stabilize the training without the need for a curriculum.

5. Ground-truth motor RPM states. The teacher policies are never deployed in reality, so instead of feeding a proprioceptive action history to account for the unobservable motor states as in (*7*), the teachers can directly observe the ground-truth motor states. This also makes the actor-critic architecture symmetric.

We do these modifications to trade off wall-clock training time for highly reliable training dynamics, and we found them to yield high-quality teacher policies in all 1000 cases without cherry-picking random seeds and without requiring case-by-case modifications, even though the dynamics of the quadrotors vary drastically as described in Section 4.1.

The teachers observe the states fully $\mathbf{o}_{\text{teacher}} = \{\mathbf{p}, \mathbf{R}(\mathbf{q}), \mathbf{v}, \boldsymbol{\omega}, \mathbf{a}_{t-1}, \boldsymbol{\omega}_m\}$.

We train using a position control objective where the quadrotor is initialized in a random state (e.g., up to 90° tilt) and the policy has to navigate it back to the origin with zero yaw while also minimizing linear/angular velocity and action changes. To prepare the policy for trajectory tracking, we also train with the objective of tracking a relatively slow trajectory sampled from a second-order Langevin process. The Supplementary Materials (*37*) contain additonal details about the motivation for and the implementation of the distribution over reference trajectories.

### 4.2.2 Meta-Imitation Learning

After training 1000 teacher policies, we would like to distill all of their behaviors into a single student foundation policy. From the perspective of each teacher policy, there are no hidden/latent parameters because each teacher policy can assume that it always interacts with the same quadrotor $\boldsymbol{\Xi}$. However, in aggregate, and from the perspective of the student policy, the parameters of the quadrotor $\boldsymbol{\Xi}$ it is interacting with are not observable and need to be inferred as described in Section 4. Due to the variable number of past steps (cf. Figure 7A), we design a Gated Recurrent Unit (GRU) (*47*)-based foundation policy architecture as displayed in Figure 7B.

The relatively small hidden dimensionality of 16 is justified by the scaling experiments in Section 2.3. Due to the recurrence, the policy can theoretically ”access” all the previous observations and actions.

We refer to our proposed algorithm as Meta-Imitation Learning because the student not only needs to learn to recreate the teachers' outputs from a different set of inputs/sensors, but also must learn to perform inference about the current MDP that it is acting in. Each quadrotor constitutes a separate MDP because the transition function varies based on the random dynamics parameters $\boldsymbol{\Xi}$. We currently only tackle the case where the dynamics parameters change, but in the future, we will also incorporate, e.g., changes in the reward function.

Our proposed method is conceptually similar to the DAgger algorithm (*48*) but differs in two key ways, firstly in the aforementioned requirement for meta-learning and secondly in that we perform on-policy data collection (after warm-up) and learning while DAgger is strictly off-policy. Figure 8 shows our full proposed algorithm consisting of the sampling of 1000 random quadrotors and the two main learning phases: pre- and post-training.

In the post-training phase, we distill the combined behaviors of the 1000 teachers into the student foundation policy. We train it for 1000 epochs and solely use on-policy data after a warm-up (using teacher rollouts) of 10 epochs. The task of the student foundation policy, characterized by

```
P ← [ ] ;            // List of dynamics parameters
// Init foundation policy weights
θ ← init_weights() ;
Π* ← [ ] ;           // List of teacher checkpoints
for i ← 1 to 1000 do
    Ξ ← sample_dynamics_parameters();
    P.append(Ξ);
end
for i ← 1 to 1000 do
    Ξ ← P[i];
    // Train for 1e6 environment steps
    // Returns the best checkpoint
    θ* ← run_SAC(Ξ);
    Π*.append(θ*);
end
for epoch ← 1 to 1000 do
    T ← [ ] ;                // List of trajectories
    for i ← 1 to 1000 do
        Ξ ← P[i];
        θ* ← Π*[i] ;              // Teacher weights
        for j ← 1 to 10 do
            if epoch ≤ 10 then
                // Warmup using teacher
                τ ← sample_trajectory(Ξ, θ*);
            end
            else
                // On-policy sampling
                τ ← sample_trajectory(Ξ, θ);
            end
            τ*_a ← forward(θ*, τ);
            T.append((τ, τ*_a));
        end
    end
    B ← shuffle_into_batches(T);
    foreach (X, Y) ∈ B do
        Y_pred ← forward(θ, X);
        L ← MSE(Y_pred, Y);
        L.backward();
        θ ← adam_step(θ);
    end
end
```

**Figure 8**: **Meta-Imitation Learning Algorithm.**

its parameters $\boldsymbol{\theta}$, is to predict the teachers' motor commands as closely as possible just based on the history of observations and (its own actions), without knowing the teacher or dynamics parameters at hand.

As shown in Figure 7A, this forces the policy to infer the parameters of $\Xi$ that are relevant for the input/output behavior of the system. This meta-learning using in-context reasoning is the central part of our proposed method. Additionally, we propose the use of on-policy imitation learning, where we neither use the actions of the teachers during rollout (in contrast to the $\beta$ trade-off in (*48*)) nor use trajectories from past policies. We use on-policy imitation learning because full dataset aggregation as in DAgger (*48*) is infeasible due to memory constraints, and we also find it to learn faster and better policies.

## 4.3 Computational Aspects

Computationally, the separation into pre-training and post-training/meta-imitation learning is a major advantage of the RAPTOR framework. This decouples the time/compute-intensive pre-training and renders it "embarrassingly parallel" (*49*). This can be seen from Figure 1B, where the teacher training processes are independent. Hence, we can horizontally scale out the number of training processes over multiple processors and/or machines, linearly speeding the pre-training up to the ceiling, which is the 31 m duration of each training run (at 1000 cores in parallel). In contrast to LLM pre-training, where the gradients have to be communicated between all nodes at every training step, in RAPTOR, pre-training is entirely independent and communication/synchronization is only required after pre-training, in the distillation/meta-imitation learning phase, which is about three orders of magnitude less computationally intensive than the pre-training.

# References and Notes


1. G. Li, X. Liu, G. Loianno, Human-Aware Physical Human–Robot Collaborative Transportation and Manipulation With Multiple Aerial Robots. *IEEE Transactions on Robotics* **41**, 762–781 (2025), doi:10.1109/TRO.2024.3502508.

2. A. Ollero, *et al.*, The AEROARMS Project: Aerial Robots with Advanced Manipulation Capabilities for Inspection and Maintenance. *IEEE Robotics and Automation Magazine* **25** (4), 12–23 (2018), doi:10.1109/MRA.2018.2852789.

3. M. Tranzatto, *et al.*, CERBERUS in the DARPA Subterranean Challenge. *Science Robotics* **7** (66), eabp9742 (2022), doi:10.1126/scirobotics.abp9742, `https://www.science.org/doi/abs/10.1126/scirobotics.abp9742`.

4. Y. Song, A. Romero, M. Müller, V. Koltun, D. Scaramuzza, Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics* **8** (82), eadg1462 (2023), doi:10.1126/scirobotics.adg1462, `https://www.science.org/doi/abs/10.1126/scirobotics.adg1462`.

5. E. Kaufmann, *et al.*, Champion-level drone racing using deep reinforcement learning. *Nature* **620** (7976), 982–987 (2023), doi:10.1038/s41586-023-06419-4.

6. R. Ferede, T. Blaha, E. Lucassen, C. De Wagter, G. C. de Croon, One Net to Rule Them All: Domain Randomization in Quadcopter Racing Across Different Platforms. *arXiv preprint arXiv:2504.21586* (2025).

7. J. Eschmann, D. Albani, G. Loianno, Learning to Fly in Seconds. *IEEE Robotics and Automation Letters* **9** (7), 6336–6343 (2024), doi:10.1109/LRA.2024.3396025.

8. X. B. Peng, M. Andrychowicz, W. Zaremba, P. Abbeel, Sim-to-Real Transfer of Robotic Control with Dynamics Randomization, in *IEEE International Conference on Robotics and Automation (ICRA)* (2018), pp. 3803–3810, doi:10.1109/ICRA.2018.8460528.

9. A. Loquercio, *et al.*, Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics* **36** (1), 1–14 (2019).

10. D. Hanover, *et al.*, Autonomous drone racing: A survey. *IEEE Transactions on Robotics* **40**, 3044–3067 (2024).

11. A. Radford, *et al.*, Learning Transferable Visual Models From Natural Language Supervision, in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila, T. Zhang, Eds. (PMLR), vol. 139 of *Proceedings of Machine Learning Research* (2021), pp. 8748–8763, `https://proceedings.mlr.press/v139/radford21a.html`.

12. T. Brown, *et al.*, Language models are few-shot learners. *Advances in neural information processing systems* **33**, 1877–1901 (2020).

13. J. Kaplan, *et al.*, Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

14. E. Kaufmann, L. Bauersfeld, D. Scaramuzza, A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight, in *International Conference on Robotics and Automation (ICRA)* (2022), pp. 10504–10510, doi:10.1109/ICRA46639.2022.9811564.

15. R. Zhang, D. Zhang, M. W. Mueller, Proxfly: Robust control for close proximity quadcopter flight via residual reinforcement learning. *arXiv preprint arXiv:2409.13193* (2024).

16. J. Heeg, Y. Song, D. Scaramuzza, Learning quadrotor control from visual features using differentiable simulation. *arXiv preprint arXiv:2410.15979* (2024).

17. J. Xing, I. Geles, Y. Song, E. Aljalbout, D. Scaramuzza, Multi-task reinforcement learning for quadrotors. *IEEE Robotics and Automation Letters* (2024).

18. S. Gronauer, M. Kissel, L. Sacchetto, M. Korte, K. Diepold, Using simulation optimization to improve zero-shot policy transfer of quadrotors, in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE) (2022), pp. 10170–10176.

19. R. Ferede, G. de Croon, C. De Wagter, D. Izzo, End-to-end neural network based optimal quadcopter control. *Robotics and Autonomous Systems* **172**, 104588 (2024).

20. R. Ferede, C. De Wagter, D. Izzo, G. C. De Croon, End-to-end reinforcement learning for time-optimal quadcopter flight, in *2024 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE) (2024), pp. 6172–6177.

21. L. Balandi, P. Robuffo Giordano, M. Tognon, Acceleration-Based Inner-Loop Control and MPC for Aerial Robots: Advantages and Drawbacks, in *European Robotics Forum* (Springer) (2025), pp. 75–80.

22. S. M. Hegre, W. Rehberg, M. Kulkarni, K. Alexis, A Neural Network Mode for PX4 on Embedded Flight Controllers. *arXiv preprint arXiv:2505.00432* (2025).

23. D. Zhang, *et al.*, A Learning-Based Quadcopter Controller With Extreme Adaptation. *IEEE Transactions on Robotics* **41**, 3948–3964 (2025), doi:10.1109/TRO.2025.3577037.

24. P. Henderson, *et al.*, Deep reinforcement learning that matters, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32 (2018).

25. Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, J. Ba, Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *Advances in neural information processing systems* **30** (2017).

26. H. P. Van Hasselt, A. Guez, M. Hessel, V. Mnih, D. Silver, Learning values across many orders of magnitude. *Advances in neural information processing systems* **29** (2016).

27. W. C. Lewis II, M. Moll, L. E. Kavraki, How much do unstated problem constraints limit deep robotic reinforcement learning? *arXiv preprint arXiv:1909.09282* (2019).

28. K. Clary, E. Tosch, J. Foley, D. Jensen, Let's play again: Variability of deep reinforcement learning agents in atari environments. *arXiv preprint arXiv:1904.06312* (2019).

29. R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, M. Bellemare, Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems* **34**, 29304–29320 (2021).

30. T. Baca, *et al.*, The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles. *Journal of Intelligent & Robotic Systems* **102** (1), 26 (2021).

31. J. Eschmann, D. Albani, G. Loianno, Data-Driven System Identification of Quadrotors Subject to Motor Delays, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2024), pp. 8095–8102, doi:10.1109/IROS58592.2024.10801441.

32. G. Alain, Y. Bengio, Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644* (2016).

33. A. Dosovitskiy, *et al.*, An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

34. M. Oquab, *et al.*, Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193* (2023).

35. R. Mahony, T. Hamel, J.-M. Pflimlin, Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on automatic control* **53** (5), 1203–1218 (2008).

36. S. O. Madgwick, *et al.*, An efficient orientation filter for inertial and inertial/magnetic sensor arrays (2010).

37. Materials and methods are available as supplementary material.

38. P. Kunapuli, J. Welde, D. Jayaraman, V. Kumar, Leveling the Playing Field: Carefully Comparing Classical and Learned Controllers for Quadrotor Trajectory Tracking, in *Proceedings of Robotics: Science and Systems* (Los Angeles, United States of America) (2025).

39. S. Ross, B. Chaib-draa, J. Pineau, Bayes-adaptive pomdps. *Advances in neural information processing systems* **20** (2007).

40. D. Koller, N. Friedman, *Probabilistic graphical models: principles and techniques* (MIT press) (2009).

41. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA) (1988).

42. OpenAI, *et al.*, Solving Rubik's Cube with a Robot Hand (2019).

43. J. X. Wang, *et al.*, Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763* (2016).

44. Y. Duan, *et al.*, RL2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779* (2016).

45. J. Achiam, *et al.*, Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

46. M. Belkin, D. Hsu, S. Ma, S. Mandal, Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences* **116** (32), 15849–15854 (2019).

47. K. Cho, *et al.*, Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

48. S. Ross, G. Gordon, D. Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning, in *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (JMLR Workshop and Conference Proceedings) (2011), pp. 627–635.

49. C. Moler, Matrix computation on distributed memory multiprocessors. *Hypercube Multiprocessors* **86** (181-195), 31 (1986).

50. J. Eschmann, D. ALBANI, G. Loianno, RAPTOR: A Foundation Policy for Quadrotor Control (2025), doi:10.5281/zenodo.17096679, `https://doi.org/10.5281/zenodo.17096679`.

51. Supplementary Code and Data Repository, Github: rl-tools/raptor, `https://github.com/rl-tools/raptor`.

52. Project Page, Static Website, `https://raptor.rl.tools/`.

53. Supplementary Video, YouTube, `https://youtu.be/hVzdWRFTX3k`.

54. S. Sarkka, A. Solin, J. Hartikainen, Spatiotemporal Learning via Infinite-Dimensional Bayesian Filtering and Smoothing: A Look at Gaussian Process Regression Through Kalman Filtering. *IEEE Signal Processing Magazine* **30** (4), 51–61 (2013), doi:10.1109/MSP.2013.2246292.

55. S. Särkkä, A. Solin, *Applied stochastic differential equations*, vol. 10 (Cambridge University Press) (2019).

56. C. Berner, *et al.*, Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

# Acknowledgments

We thank professor Van Anh Ho and Quang Ngoc Pham for letting us test the foundation policy on the soft quadrotor.

**Funding:** This work was supported in part by the National Science Foundation (NSF) CAREER program under Grant 2145277, and in part by the Defense Advanced Research Projects Agency (DARPA) Young Faculty Award under Grant D22AP00156-00.

**Author contributions:** J.E. formulated the main ideas, implemented them, conducted the experiments and wrote the paper. D.A. and G.L. provided supervision and guided the direction during all phases of the project and helped writing the paper.

**Competing interests:** The authors declare that they have no competing interests.

**Data and materials availability:** The code and dataset of sampled quadrotor dynamics as well as the trained teacher checkpoints are available from https://doi.org/10.5281/zenodo.17096679 (*50*) and the Git repository https://github.com/rl-tools/raptor (*51*). Please also refer to the project page at https://raptor.rl.tools (*52*) where an interactive simulation can be used to test the RAPTOR policy. The full-length and quality video can also be accessed at https://youtu.be/hVzdWRFTX3k (*53*).

## Supplementary materials

Figure S1
Movie S1 to S8
Data S1

# Supplementary Materials for
# RAPTOR: A Foundation Policy for Quadrotor Control

Jonas Eschmann*, Dario Albani, Giuseppe Loianno
*Corresponding author. Email: jonas.eschmann@berkeley.com

**This PDF file includes:**



**Other Supplementary Materials for this manuscript:**

# Materials and Methods

## Linear Velocity Feedback Delay Mitigation

We counteract the linear velocity delay experienced by the non-EKF-based platforms by overlaying a simple accelerometer integral that is grounded through an exponential decay. This can also be interpreted as a convolution of the accelerometer data with an Infinite Impulse Response (IIR) filter. Before the convolution, the gravity is subtracted by applying the current orientation estimate to the accelerometer data.

$$\mathbf{a}_g(t) = \mathbf{R}^T(t)\mathbf{a}(t) - \mathbf{g} \tag{S1}$$

$$\mathbf{v}_a(t) = \int_0^t \mathbf{a}_g(\tau) e^{-\frac{t-\tau}{T}} d\tau \tag{S2}$$

We use the following discrete approximation to implement the filter:

$$\alpha = e^{\frac{\Delta t}{T}} \tag{S3}$$

$$\mathbf{v}_a^t = \alpha \mathbf{v}_a^{t-1} + \mathbf{a}_g^t \Delta t \tag{S4}$$

We find that this mitigation significantly reduces the z-axis oscillations on the non-EKF-based platforms.

### Sampling Quadrotors

We establish a physically plausible posterior distribution over quadrotors through the following relations:

$$r_{\text{t2w}} \sim \text{Uniform}(1.5, 5) \tag{S5}$$

$$m_{\min} = 0.02, \quad m_{\max} = 5 \tag{S6}$$

$$m = s^3, \quad s \sim \text{Uniform}(\sqrt[3]{m_{\min}}, \sqrt[3]{m_{\max}}) \tag{S7}$$

$$f(\omega_{m_i}) = c_{f_0} + c_{f_1}\omega_{m_i} + c_{f_2}\omega_{m_i}^2 \tag{S8}$$

$$\omega_{m_i} \in [0, 1], \ \sum_i C_{f_i} = 1, \ C_{f_0} = 0.038, \ C_{f_1} = 0.154, \ C_{f_2} = 0.987 \tag{S9}$$

$$T = r_{\text{t2w}} \cdot 9.81 \cdot m \tag{S10}$$

$$c_{f_i} = C_{f_i} \frac{T}{4} \tag{S11}$$

$$\tag{S12}$$

The idea is to first sample the thrust-to-weight ratio $r_{\text{t2w}}$ and mass $m$ to create the thrust curve. The thrust curve consists of the coefficients $C_{f_0}, C_{f_1}, C_{f_2}$ for the constant, linear and quadratic term respectively. To relate the thrust-to-weight ratio to the thrust curve we require the mass. To sample the mass we need to consider that it grows cubically with the size (arm length, assuming constant density). If we were to uniformly sample the mass, we would disproportionately bias the distribution towards large quadrotors. Instead, we pose that quadrotors are rather uniformly distributed in size so we uniformly sample a scale $s$ that we map back cubically into the desired

mass range $m \in [m_{\min}, m_{\max}]$ to get a realistic distribution over masses. Note that w.l.o.g., we normalize the motor effort setpoints $\omega_{m_i}$ and the baseline thrust curve coefficients $C_{f_i}$. This allows us to simply scale the baseline thrust curve shape (taken from the Crazyflie (*31*)) to reflect the sampled thrust-to-weight ratio. Note: The baseline thrust curve coefficients $C_{f_i}$ are capitalized while the coefficients $c_{f_i}$ of the sampled thrust curve $f$ are lower-case.

Due to different design considerations, the mass-size ratio $\frac{\sqrt[3]{m}}{l_{\text{arm}}}$ is not always constant so we establish the mass-size ratio of the Crazyflie $R_{ms}$ as the base value and sample variations around that.

$$R_{ms} = \frac{\sqrt[3]{m_{\text{crazyflie}}}}{l_{\text{arm,crazyflie}}} \approx 7.90 \tag{S13}$$

$$u \sim \mathcal{N}(-0.1, 0.1) \tag{S14}$$

$$s_{ms} = \begin{cases} \frac{1}{1-u} & \text{if } u < 0 \\ 1 + u & \text{if } u \geq 0 \end{cases} \tag{S15}$$

$$r_{ms} = \frac{\sqrt[3]{m}}{l_{\text{arm}}} \stackrel{!}{=} s_{ms} R_{ms} \tag{S16}$$

$$l_{\text{arm}} = \frac{\sqrt[3]{m}}{s_{ms} R_{ms}} \tag{S17}$$

$$\tag{S18}$$

We intended to use a reciprocal deviation $s_{\text{ms}}$ of $\pm 10\%$ but accidentally used a normal instead of a uniform distribution. In a post-hoc analysis we find that the resulting distribution has a mean and standard deviation of $\approx 7.24 \pm 0.66$. This incidentally fits the empirical distribution of mass-size deviations of the quadrotors in Figure 4 with a mean and standard deviation of $\approx 7.66 \pm 1.93$ better than the intended distribution of just 10% uniform reciprocal deviations. For future works we advise to investigate directly sampling from a normal distribution with a wider standard deviation that is closer to the empirical standard deviation of the actual quadrotors.

The sampled mass-size ratio allows us to compute the arm length $l_{\text{arm}}$ which, in combination with the assumption that the quadrotor is in a symmetric X configuration, defines the geometry.

Finally, we need to sample the inertia based on the previously sampled quantities:

$$r_{\text{t2i}} \sim \text{Uniform}(40, 1200) \tag{S19}$$

$$\tau = T \cdot \sqrt{2} \cdot l_{\text{arm}} \tag{S20}$$

$$J_{xx} = J_{yy} = \frac{\tau}{r_{\text{t2i}}} \tag{S21}$$

$$J_{zz} = \frac{J_{xx} + J_{yy}}{2} \cdot 1.832 \tag{S22}$$

$$\tag{S23}$$

For sampling the inertia, we introduce another ratio, the torque-to-inertia ratio $r_{\text{t2i}}$. We collect quadrotor dynamics parameters from the literature and find a realistic randomization range to be between 40 and 1200. After sampling the torque-to-inertia ratio, we can use the previously sampled

thrust-to-weight, mass and size (arm length) parameters to calculate the x and y inertia. To get the z inertia, we apply the rule of (*31*).

Furthermore, we can independently sample the moment constant as well as the motor delays because they do not correlate strongly with any of the other variables in our experience:

$$c_m \sim \text{Uniform}(0.005, 0.05) \tag{S24}$$
$$T_{m\uparrow} \sim \text{Uniform}(0.03, 0.1) \tag{S25}$$
$$T_{m\downarrow} \sim \text{Uniform}(0.03, 0.3) \tag{S26}$$

**Sampling Trajectories**

During pre-training, the policies are Markovian and only consider the current observation (plus previous actions). This means that (in the absence of aerial drag-forces from linear velocity) trajectory tracking and position control appear identical to the policy because we can just feed the position and linear velocity error w.r.t. the trajectory as observations and achieve good trajectory tracking (*7*). In the case of a stateful, non-Markovian policy as with the recurrent foundation policy, this is not the case anymore. If we add the dynamics of the trajectory itself to the dynamics of the quadrotor through the error-state observation, the trajectory of error-state observations does not appear like a quadrotor anymore.

A simple example would be a reference trajectory with a jump in linear velocity. From the perspective of a stateful policy that has only been trained on position control (no trajectory dynamics in the observation space) this abrupt change in linear velocity appears like a quadrotor with an unrealistically high thrust-to-weight ratio, unrealistically fast angular response and/or even a world with much larger than 1 G gravity. The latter happens if e.g., the velocity step is downwards in z and the quadrotor is not upside down or if the action inputs have been low/zero.

To counter-act this, we train the expert and student policies using a simple probabilistic mixture model of trajectories. With 50% probability the task is just tracking the null-trajectory (position control, going back to the origin from any initial state). In the other 50% the task is to track randomly sampled trajectories.

We would like to cover a wide variety of possible reference trajectories. Trajectories are vector-valued functions of time so we need to design a broad distribution over functions. Here we take inspiration from the Gaussian Process (GP) community, which has been concerned with designing prior distributions over functions since its inception. Additionally, we require the sampling of reference trajectories to not slow down the simulation too much. Hence we choose to sample the reference trajectories from a second-order Langevin process. A second-order Langevin process corresponds to a GP with a certain structure in the kernel (depending on the parameters of the Langevin process) (*54, 55*) but we can easily sample it incrementally while simulating the quadrotor dynamics.

Based on the results described in Section 2.4.1, we find that this approach works well for moderately agile trajectory tracking and even generalizes from second-order Langevin random walks to cyclical Lissajous trajectories. But we acknowledge that, while the second-order Langevin process covers the space of smooth functions relatively well, many trajectories with real-world use-cases like e.g., step-functions/responses are not covered or exponentially unlikely (e.g., cyclical trajectories).

### Meta-Imitation Learning Objective

We do not perform variational inference in a case-by-case basis but instead in an amortized fashion, where the inference itself is conducted by the recurrent neural network policy at test time. This amortization makes the inference very compute efficient and allows us to deploy the foundation policy onto even the tiniest microcontrollers while meeting real-time constraints at high frequencies.

For tractability of the KL divergence we assume the action distributions are parameterized Gaussians:

$$p(\mathbf{a}_t^* \mid \mathbf{s}_t, \boldsymbol{\Xi}) \approx \mathcal{N}(\mathbf{a}_t^*; \boldsymbol{\pi}^*(\mathbf{s}_t, \boldsymbol{\Xi}), \mathbf{I}) \tag{S27}$$

$$\boldsymbol{\pi}^*(\mathbf{s}_t, \boldsymbol{\Xi}) := \boldsymbol{\pi}_{\boldsymbol{\Xi}}^*(\mathbf{s}_t) \tag{S28}$$

$$p(\mathbf{a}_t^\pi \mid \mathbf{o}_0, \ldots, \mathbf{o}_t, \mathbf{a}_0, \ldots, \mathbf{a}_{t-1}) \approx \mathcal{N}(\mathbf{a}_t^\pi; \boldsymbol{\pi}(\mathbf{o}_0, \ldots, \tag{S29}$$

$$\mathbf{o}_t, \mathbf{a}_0, \ldots, \mathbf{a}_{t-1}), \mathbf{I}) \tag{S30}$$

$$= \mathcal{N}(\mathbf{a}_t^\pi; \boldsymbol{\pi}(\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}), \mathbf{I}) \tag{S31}$$

Where $\boldsymbol{\pi}_{\boldsymbol{\Xi}}^*$ is one of the 1000 teacher policies trained for the particular quadrotor/set of dynamics parameters $\boldsymbol{\Xi}$ and $\boldsymbol{\pi}$ is the student foundation policy.

We can see that the input shape of the foundation policy $\boldsymbol{\pi}$ is dependent on the number of previous steps in the episode. Due to the sequential nature of the inputs, we choose a recurrent neural network architecture for three main reasons:

- **Computational Efficiency**: We require computational efficiency for direct deployment on compute-constrained microcontrollers. Recurrent Neural Networks (RNNs) are $O(1)$ in the history length $N$, each incremental step only requires a fixed amount of compute at inference time. In contrast, Convolutional Neural Networks (CNN) and attention are $O(\log(N))$ and $O(N)$ for each step, respectively.

- **Context Window Extrapolation**: At inference time we would like the policy to fly for longer time than the context window used during training. During training the context is usually limited to not slow down the process too much. CNNs, by design, have a limited context window that cannot be extended. Similarly attention, being a set-to-set mapping, requires position embeddings which complicate context window extrapolation.

- **Successful Use in Related Works**: E.g. (*42*) and (*56*) have used recurrent policies successfully.

We find that a surprisingly small (in terms of number of parameters and compute requirements) three-layer recurrent neural network is sufficient to express the desired behavior described in Section 1. Figure 7B shows the architecture which contains a dense input layer, Gated Recurrent Unit (GRU) layer and a dense output layer. The initialization of the recurrent state (value at step 0) is all zeros and we feed back the previous output of the policy as an input. Due to the small hidden dimensions

the foundation policy only has 2084 parameters:

$$P = P_{\text{input}} + P_{\text{GRU}} + P_{\text{output}} \tag{S32}$$
$$P_{\text{input}} = 22 \cdot 16 + 16 = 368 \tag{S33}$$
$$P_{\text{GRU}} = 16 \cdot 16 \cdot 3 \cdot 2 + 16 \cdot 3 \cdot 2 + 16 = 1648 \tag{S34}$$
$$P_{\text{output}} = 16 \cdot 4 + 4 = 68 \tag{S35}$$
$$P = 2084 \tag{S36}$$

We train the foundation policy using Meta-Imitation Learning where it acts as the student. During Meta-Imitation Learning, we want to adjust the students weights to minimize the KL divergence (also known as relative entropy) between the predicted optimal action distribution by the student and the optimal action distribution (approximated by the particular teacher for the current system dynamics):

$$D_{\text{KL}}\left(\mathbf{a}_t^* \parallel \mathbf{a}_t^\pi\right) = \mathop{\mathbb{E}}_{\mathbf{a}_t^* \sim p(\mathbf{a}_t^* | \mathbf{s}_t, \boldsymbol{\Xi})} \left[ \log \frac{p(\mathbf{a}_t^* \mid \mathbf{s}_t, \boldsymbol{\Xi})}{p(\mathbf{a}_t^\pi = \mathbf{a}_t^* \mid \mathbf{o}_{0:t}, \mathbf{a}_{0:t-1})} \right] \tag{S37}$$
$$[\ldots] = \log p(\mathbf{a}_t^* \mid \mathbf{s}_t, \boldsymbol{\Xi}) \tag{S38}$$
$$- \log p(\mathbf{a}_t^\pi = \mathbf{a}_t^* \mid \mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}) \tag{S39}$$
$$\tag{S40}$$

Hence we want to find the log probabilities of the action distributions that we approximated as Gaussians before:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-k/2} \det(\boldsymbol{\Sigma})^{-1/2} \tag{S41}$$
$$\cdot \exp\left( -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\mathsf{T} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right) \tag{S42}$$
$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{I}) = C \cdot \exp\left( -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\mathsf{T} (\mathbf{x} - \boldsymbol{\mu}) \right) \tag{S43}$$
$$= C \cdot \exp\left( -\frac{1}{2} \|\mathbf{x} - \boldsymbol{\mu}\|_2^2 \right) \tag{S44}$$

Since we are aiming at quadrotors, both, the numerator and denominator multivariate Gaussian are 4 dimensional and the constants cancel.

$$[\ldots] = \log p(\mathbf{a}_t^* \mid \mathbf{s}_t, \boldsymbol{\Xi}) \tag{S45}$$
$$- \log p(\mathbf{a}_t^\pi = \mathbf{a}_t^* \mid \mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}) \tag{S46}$$
$$\boldsymbol{\mu}_T := \boldsymbol{\pi}^*(\mathbf{s}_t, \boldsymbol{\Xi}) \tag{S47}$$
$$\boldsymbol{\mu}_S := \boldsymbol{\pi}(\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}) \tag{S48}$$
$$[\ldots] = \log C - \frac{1}{2} \|\mathbf{a}_t^* - \boldsymbol{\mu}_T\|_2^2 \tag{S49}$$
$$- \log C + \frac{1}{2} \|\mathbf{a}_t^* - \boldsymbol{\mu}_S\|_2^2 \tag{S50}$$
$$\tag{S51}$$

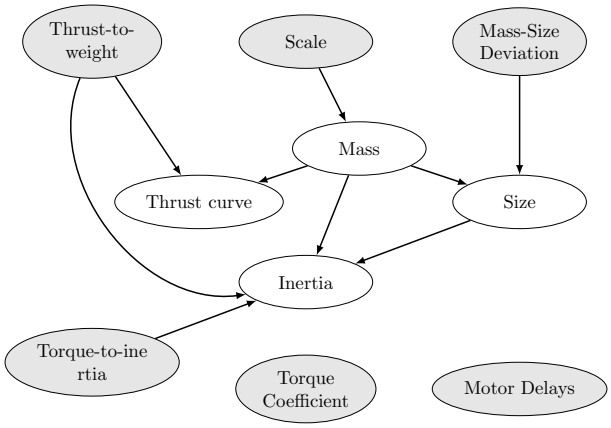



**Figure S1**: **Probabilistic Graphical Model for Ancestral Sampling of Quadrotors**

We can split the second squared norm:

$$\begin{aligned}
&\|\mathbf{a}_t^* - \boldsymbol{\mu}_S\|_2^2 \\
&= \|(\mathbf{a}_t^* - \boldsymbol{\mu}_T) + (\boldsymbol{\mu}_T - \boldsymbol{\mu}_S)\|_2^2 \\
&= \|(\mathbf{a}_t^* - \boldsymbol{\mu}_T)\|_2^2 + 2(\mathbf{a}_t^* - \boldsymbol{\mu}_T)^\mathsf{T}(\boldsymbol{\mu}_T - \boldsymbol{\mu}_S) + \|(\boldsymbol{\mu}_T - \boldsymbol{\mu}_S)\|_2^2
\end{aligned}$$

Plugging back into Eq. S44:

$$[\ldots] = (\mathbf{a}_t^* - \boldsymbol{\mu}_T)^\mathsf{T}(\boldsymbol{\mu}_T - \boldsymbol{\mu}_S) + \frac{1}{2}\|(\boldsymbol{\mu}_T - \boldsymbol{\mu}_S)\|_2^2$$

We remember that we take the expectation over $[\ldots]$ in Eq. S37 and that we assume that the teacher is an unbiased estimator for the optimal action ($\mathbb{E}\left[\mathbf{a}_t^* - \boldsymbol{\mu}_T\right] = 0$):

$$D_{\mathrm{KL}}\big(\mathbf{a}_t^* \,\|\, \mathbf{a}_t^{\boldsymbol{\pi}}\big) = \frac{1}{2}\|(\boldsymbol{\mu}_T - \boldsymbol{\mu}_S)\|_2^2 \tag{S52}$$

$$= \frac{1}{2}\|(\boldsymbol{\pi}^*(\mathbf{s}_t, \boldsymbol{\Xi}) - \boldsymbol{\pi}(\mathbf{o}_{0:t}, \mathbf{a}_{0:t-1}))\|_2^2 \tag{S53}$$

Hence, under the mild assumption of unit standard deviations, we can conduct the Meta-Imitation Learning in a principled manner by minimizing the relative differential entropy between the (approximated) optimal action distribution and the predicted optimal action distribution by the student. In practice, we implement this by using Eq. S52, which is identical to the Mean-Squared Error (MSE), as the loss function and by optimizing the student policy's parameters $\boldsymbol{\theta}$ using gradient descent.

**Caption for Movie S1. Motivation and Introduction**

**Caption for Movie S2. Trajectory Tracking Experiments**

**Caption for Movie S3. Outdoor Experiments**

**Caption for Movie S4. Disturbance Experiments: Poking**

**Caption for Movie S5. Disturbance Experiments: Wind**

**Caption for Movie S6. Disturbance Experiments: Different Propellers**

**Caption for Movie S7. Agile Recovery / Rapid In-Context Learning Experiments**

**Caption for Movie S8. Yaw Response Experiments**

**Note** All these movies are combined in `https://youtu.be/hVzdWRFTX3k`

**Caption for Data S1. Data and Code** This package (*50*) contains all training and inference code as well as the complete pre-training data (including dynamics parameters and checkpoints of the 1000 quadrotors) and the resulting foundation policy. Please refer to the readme.txt for instructions.