Decentralized Control of Quadrotor Swarms with End-to-end Deep Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: We demonstrate the possibility of learning drone swarm controllers via large-scale multi-agent end-to-end reinforcement learning. We train policies parameterized by neural networks that are capable of controlling individual drones in a swarm in a fully decentralized manner. Our policies, trained in simulated environments with realistic quadrotor physics, demonstrate advanced flocking behaviors, perform aggressive maneuvers in tight formations while avoiding collisions with each other, break and re-establish formations to avoid collisions with moving obstacles, and efficiently coordinate in pursuit-evasion tasks. We analyze, in simulation, how different model architectures and parameters of the training regime influence the final performance of neural swarms. We demonstrate the successful transfer of the model learned in simulation to highly resource-constrained physical quadrotors performing station keeping and goal swapping behaviors. Video demonstrations are available at the project website https://sites.google.com/view/swarm-rl. Source code is available in the supplementary materials.

Keywords: Swarms, Multi-robot systems, Multi-robot learning

1 Introduction

Teams of unmanned aerial vehicles find application in a variety of tasks *e.g.* area coverage, search and rescue, reconnaissance *etc*. State of the art approaches for planning and control of drone teams rely on centralized or decentralized planners that typically require full state information and extensive computational resources to plan in advance [1]. This greatly limits their applicability since existing planning algorithms are often brittle in partially-observable environments and challenging to execute in real time on embedded hardware. Many classical methods also suffer from the curse of dimensionality as the number of possible configurations grows combinatorially with team size. In addition, kinodynamic planners typically require a precise model of drone dynamics and their interactions.

Here, we present a fully learned approach that uses a small amount of computation during execution and relies exclusively on local observations, yet can be used to create effective controllers for large-scale, swarm-like, teams. We take advantage of multi-agent deep reinforcement learning (DRL) to train quadrotor drones on hundreds of millions of environment transitions in realistic simulated environments. We find neural network architectures and observation spaces that allow our neural controllers to achieve high performance on a diverse set of tasks. Our experiments demonstrate that drone swarms controlled by our neural network policies generate highly effective, smooth, and virtually collision-free trajectories. We show simulated dynamic obstacle avoidance and recovery from external disturbances. In simulated settings, we show that our learned controllers can adapt to changes in team size, and can perform well even when the run-time swarms are significantly larger than the training scenarios. Finally, we show that our learned controllers are effective on physical drones. We demonstrate this with teams of up to 4 quadrotors successfully executing tasks under the control of learned policies. To the best of our knowledge, this is the first approach that a) demonstrates scalable coordinated flight of drone swarms in a realistic physical simulation achieved through end-to-end reinforcement learning (RL) with direct thrust control, and b) demonstrates that the learned policies transfer to physical drone teams. We view this as progress towards hardware agnostic real-world deployment of quadrotor swarms with realtime replanning capabilities.

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

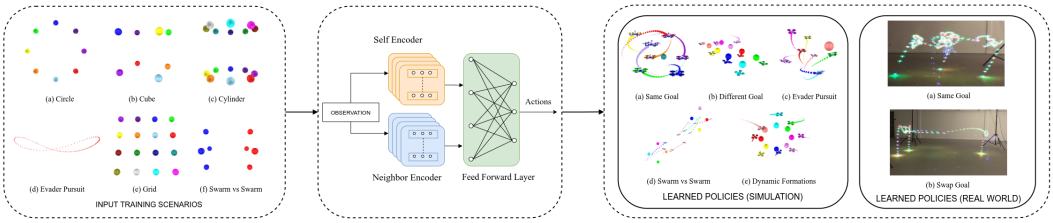

Figure 1: Drones are trained on scenarios randomly sampled from one of six pre-defined scenarios on the left. Drones employ a self and neighbor encoder that learn a mapping from a combination of proprioceptive and extereoceptive inputs to thrust outputs. The combined intermediate embeddings from each respective encoder enable policies to perform high speed, aggressive maneuvers and learn collision avoidance behaviors. Examples of these, including *e.g.* formation creation, formation swaps, evader pursuit are shown on the right both in simulation and physical experiments, demonstrating scalable simultaneous stable flight and collision avoidance.

2 Related Work

Classical methods such as RRT [2] and PRM [3] can generate collision-free piece-wise linear trajectories. These do not directly account for robot dynamics and scale poorly with team size, limiting their applicability for online re-planning. To account for the dynamics, a smooth trajectory refinement [1] or optimization [4] is performed. Kinodynamic planning methods take advantage of the known dynamics of individual robots and are more suitable for agile and aggressive flight [5]. It is also possible to convert a geometric route to a dynamically feasible trajectory [6] resulting in an agile system or to directly find dynamically feasible paths and refine them with gradient based optimization [7]. Prior learning-based approaches include [8] which estimates reachable states given the quadrotor dynamics. However, these methods require knowledge of the entire state space a priori and do not generalize well to other high dimensional environments. Additionally, kinodynamic planning is typically intractable in the multi-robot setting. Collision avoidance is particularly difficult where drones perform aggressive maneuvers. Traditional techniques use observations from neighboring drones to constrain the geometric free space or the action space of each drone. Prior work includes an n-body collision avoidance approach [9] that constrains the velocity space of a robot using velocity information from neighboring robots. Another approach utilizes pose information from neighbors to construct a Buffered Voronoi Cell, and each robot plans a collision free trajectory through its respective cell [10]. The approach in [11] minimally constrains the control space of an agent by constructing chance-constrained safety sets that account for measurement noise and disturbances present in real world environments. While these approaches guarantee collision free trajectories and are capable of running in realtime, with the exception of [11] (PrSBC), they significantly constrain the configuration space of the robot, preventing aggressive maneuvers and resulting in occasional deadlock. While PrSBC minimally constrains the control space of any given policy, it requires a model of neighboring agents in order to construct chance-constrained safety sets. Our controllers rely on a reward formulation with a high emphasis on collision avoidance and our policies are not constrained, allowing for aggressive maneuvers in a variety of rich, dynamic environments.

Neural networks trained in simulation have been used for single-agent and multi-agent quadrotor control and address shortcomings of kinodynamic planners. [12] and [13] both utilize Imitation Learning (IL) of a global centralized planner, while [13] further utilizes RL to balance the tradeoff between actions optimal to the team and individuals. [14, 15] learn controllers robust to aerodynamic interactions such as downwash. Reinforcement learning has shown promise in learning policies for UAV flight without explicitly modeling flight dynamics [16]. Deep Reinforcement Learning [17] has been used to learn minimum-time trajectory generation for quadrotors. Similar to [16] (but in a multi-agent setting) we train from scratch using DRL, and apply an end-to-end approach: the policies directly control the motor thrust. Our approach is similar to [18] which uses Soft Actor Critic to teach a single quadrotor how to fly. MARL (Multi-Agent Reinforcement Learning) has recently been applied to autonomous driving, path finding, and cooperative multi-agent control settings [19, 13, 20] and control tasks for UAV teams (*e.g.* [21] – a design for a new gym environment that models drone dynamics, taking into account aerodynamic effects, and also trains policies using RL and MARL for basic hovering and leader-following tasks). However, we are the first to train quadrotor teams with MARL on complex tasks that produce policies capable of aggressive maneuvers and scalable, cooperative behaviors, and deploy our policies on real hardware.

3 Method

3.1 Problem formulation

We consider a team of N quadrotor drones in a simulated 3D environment. Our task is to learn a control policy that directly maps proprioceptive observations of an individual drone to motor thrusts with the goal of minimizing the distances to positions in the desired formation while avoiding collisions. We analyze the case of online decentralized control, *i.e.* instead of a centralized system solving the joint trajectory optimization problem offline, we consider policies which simultaneously (and implicitly) plan trajectories and control individual quadrotors in real-time. The decentralized approach assumes no access to the global state during evaluation and scales better with the size of the swarm. In the real world, this is analogous to individual quadrotors in the swarm generating their own action sequences using only on-board computations. We demonstrate the efficacy of this approach by showing that our learned policies do indeed transfer to a physical setting. Formally, the state of the environment at time t is described by the tuple $S^{(t)} = (g_1^{(t)}, ..., g_N^{(t)}, s_1^{(t)}, ..., s_N^{(t)})$. Here $g_i^{(t)} \in \mathbb{R}^3$ are the goal locations for the quadrotors that together define the desired swarm formation at time t, and $s_i^{(t)}$ are the states of individual quadrotors. We describe the state of a single quadrotor by the tuple (p, v, R, ω), where $p \in \mathbb{R}^3$ is the position of the quadrotor relative to the goal location, $v \in \mathbb{R}^3$ is the linear velocity in the world frame, $\omega \in \mathbb{R}^3$ is the angular velocity in the body frame, and $R \in SO(3)$ is the rotation matrix from the quadrotor's body coordinate frame to the world frame. Each quadrotor is controlled by a learned policy $\pi_\theta(a^{(t)}|o^{(t)})$ which maps observations $o^{(t)}$ to Gaussian distributions over continuous actions $a^{(t)}$. To effectively maneuver and avoid collisions, the quadrotors need to be able to measure their own position and orientation in space $s_i^{(t)}$, and relative positions $\tilde{p}_{ij}^{(t)}$ and relative velocities $\tilde{v}_{ij}^{(t)}$ of their neighbors. We therefore represent each quadrotor's observations as $o_i^{(t)} = (s_i^{(t)}, \eta_i^{(t)})$, where $\eta_i^{(t)} = (\tilde{p}_{i1}^{(t)}, ..., \tilde{p}_{iK}^{(t)}, \tilde{v}_{i1}^{(t)}, ..., \tilde{v}_{iK}^{(t)})$ is a tuple containing the neighborhood information. Here $K \leq N - 1$ is the number of neighbors that each individual quadrotor can observe. We use $K \ll N$ for larger swarms to improve scalability during both training and evaluation. Simulated quadrotors, similar to their real-world counterparts, are powered by motors that spin in one direction and generate only non-negative thrust. We thus transform actions $a^{(t)} \in \mathbb{R}^4$ sampled by the policy to control inputs $f^{(t)} \in [0, 1]^4$ via $f^{(t)} = \frac{1}{2}(\text{clip}(a^{(t)}, -1, 1) + 1)$, where $f_{1...4}^{(t)} = 0$ corresponds to no thrust, and $f_{1...4}^{(t)} = 1$ corresponds to maximum thrust on motors $1...4$.

3.2 Simulation

We train and evaluate our control policies in simulated environments with realistic quadrotor dynamics. We adopt a simulation engine with drone dynamics [16], and augment it to support quadrotor swarms. Virtual quadrotors are modeled after agile nano quadrotors Crazyflie 2.0 [22] – the platform on which we show physical demonstrations. A key feature of this engine is the elaborate model of hardware imperfections previously shown to facilitate zero-shot sim-to-real transfer of stabilizing policies for single quadrotors [16]. Non-ideal motors are modeled using motor-lag and thrust noise and the simulator provides imperfect observations, with noise injected into position, orientation, and velocity estimations. Noise parameters are estimated from data collected on real Crazyflie 2.0 quadrotors. While motor and sensor noise create a challenging learning environment, they are instrumental to prevent overfitting to otherwise unrealistic idealized conditions of the simulator. To facilitate the emergence of collision-avoidance behaviors in quadrotor swarms, in addition to modeling the dynamics, we also simulate collisions between individual quadrotors. Creating a realistic collision model is a difficult task; in the real world, the outcome of the collision depends on many factors, such as the rigidity and materials of the quadrotor frames, whether the blades of two colliding quadrotors touch, *etc*. Instead of modeling these complex processes, we adopt a simple randomized collision model. When the collision between quadrotors is detected, we briefly apply random force and torque to both quadrotors with opposite signs, preserving linear and angular momenta.

3.3 Training setup

For training we use a policy gradient RL algorithm. In our setup, the learning algorithm updates the parameters θ of the policy $\pi_\theta(a^{(t)}|o^{(t)})$ to maximize the expected discounted sum of rewards

$\mathbb{E}_{\pi_\theta}\left[\sum_{t'=1}^{T}\gamma^{t'-t}R(S^{(t')},a^{(t')})\right]$. For our experiments we choose Proximal Policy Optimization (PPO) [23]. In particular, we use the implementation from the high-throughput asynchronous RL framework "SampleFactory" [24] which supports multi-agent learning and enables large scale experiments on limited hardware. In each training episode, the goal of every quadrotor in the swarm is to reach its designated position in the formation while avoiding collisions with the ground and with other quadrotors in the swarm. In order to provide rich and diverse training experiences, we train our policies in a mixture of scenarios featuring a variety of geometric 2D and 3D formations. Further training details are in Section 4. The reward function optimized by the RL algorithm for quadrotor i consists of three major components: $r^{(t)} = r^{(t)}_{\text{pos}} + r^{(t)}_{\text{col}} + r^{(t)}_{\text{aux}}$. Here, $r^{(t)}_{\text{pos}} = -\alpha_{\text{pos}}\left\|p^{(t)}_i\right\|_2$ rewards the quadrotors for minimizing the distance to their target locations. $r^{(t)}_{\text{col}}$ is responsible for penalizing collisions between quadrotors and is defined as: $r^{(t)}_{\text{col}} = -\alpha_{\text{col}}\mathbb{1}^{(t)}_{\text{col}} - \alpha_{\text{prox}}\sum_{j=1}^{K}\max\left(1-\frac{\left\|\tilde{p}^{(t)}_{ij}\right\|_2}{d_{\text{prox}}},0\right)$. The indicator function $\mathbb{1}^{(t)}_{\text{col}}$ is equal to 1 for timesteps where a new collision involving the i-th quadrotor is detected. The second term represents the smooth proximity penalty with linear falloff distance d_{prox}. Here $\left\|\tilde{p}^{(t)}_j\right\|_2$ is the distance between centers of mass of i-th and j-th quadrotors, and d_{prox} in our experiments is double the size of the quadrotor frame, which encourages them to keep minimal distance in tight formations. In addition to the first two terms $r^{(t)}_{\text{pos}}$ and $r^{(t)}_{\text{col}}$ that convey our main objective we also adopt an auxiliary reward function similar to [16] to facilitate the initial learning of stabilizing controllers: $r^{(t)}_{\text{aux}} = -\alpha_\omega\left\|\omega^{(t)}_i\right\|_2 - \alpha_f\left\|f^{(t)}_i\right\|_2 + \alpha_{\text{rot}}R^{(t)}_{33}$ penalizing high angular velocity, high motor thrusts, and large rotations about horizontal (x- and y-) axes respectively.

3.4 Model architectures

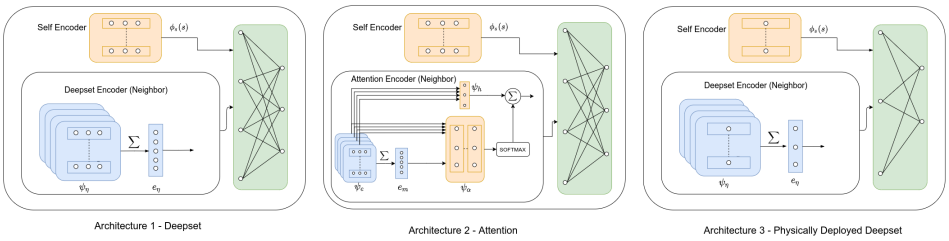

Figure 2: Detailed neural network architectures. Left: Deepsets architecture used in simulation. Middle: Attention architecture used in simulation. Right: Smaller deepsets architecture deployed on Crazyflie2.0.

During training and evaluation, the quadrotors' actions a are sampled from a parametric stochastic policy $\pi_\theta(a|o)$. Here, and in the following sections, we omit time indices and quadrotor identity for simplicity where possible. We represent π_θ with a Gaussian distribution, $a \sim \pi_\theta(a|o) \stackrel{d}{=} \mathcal{N}(\mu_a, \sigma^2)$, where the mean $\mu_a \in \mathbb{R}^4$ is a function of the quadrotor's observation at time t parameterized by a feed-forward neural network, and the variance $\sigma^2 \in \mathbb{R}$ is a single learned parameter independent of the state. In order to compute the distribution means we embed the state of the quadrotor and its neighborhood before regressing: μ_a: $e_s = \phi_s(s)$, $e_\eta = f_\eta(s,\eta)$ $\mu_a = \phi_a(e_s, e_\eta)$. Here e_s and e_η are the embedding vectors that encode each quadrotor's own state and the state of its neighborhood respectively, ϕ_s and ϕ_a are fully-connected neural networks, and f_η is the *neighborhood encoder*. We analyze two types of neighborhood encoders: deep sets encoder and attention-based encoder (Section 3.5). The value function V_π uses the same architecture as the policy, except that it regresses a single deterministic value estimate instead of the parameters of the action distribution. We do not share weights between π and V_π models.

3.5 Neighborhood encoder

Deep sets. The task of the encoder f_η is to generate a compact and expressive representation e_η of each quadrotor's local neighborhood in the swarm. Since the individual quadrotors are agnostic to

the identity of their neighbors, this representation has to be permutation invariant. In addition, scale invariance is a desirable property since the size K of the observable neighborhood can fluctuate over time, *i.e*. when a sufficient number of quadrotors in the formation move beyond the sensor range. The deep sets architecture proposed by Zaheer et al. [25] has these required properties. We apply the same learned transformation ψ_η to the observed features of each quadrotor in the neighborhood, after which a permutation-invariant aggregation function is applied to neighbor embeddings e_j. We calculate the mean of neighbor embeddings to achieve scale invariance: $e_j = \psi_\eta(\tilde{p}_{ij}, \tilde{v}_{ij})$, $e_\eta = \frac{1}{K}\sum_{j=1}^{K} e_j$. Not all neighbors are equally important for trajectory planning and decision making. For example, distant and stationary drones are less likely to influence a drone's behavior compared to closely located and fast-moving neighbors. In other words, a stationary neighbor 10 m away has a lower priority compared to one that is only 1 m away and moving towards the viewing quadrotor. The mean operation in the deep sets encoder does not allow it to convey this relative importance of different neighbors, which motivates a more sophisticated encoder architecture.

Attention-based. The attention mechanism [26] provides a natural way to express the relative importance of individual neighbors. To design the attention-based neighborhood encoder we use an architecture similar to [27], adapted for quadrotors in 3D space. The current quadrotor's state s_i and the neighbor observations $(\tilde{p}_{ij}, \tilde{v}_{ij})$ for the j-th neighbor are used to compute the attention weights in the following manner: $e_j = \psi_e(s_i, \tilde{p}_{ij}, \tilde{v}_{ij}),\ e_m = \frac{1}{K}\sum_{j=1}^{K} e_j,\ \alpha_j = \psi_\alpha(e_j, e_m)$. Here e_j are the embedding vectors of individual neighbors, e_m is the summary of the whole neighborhood, and ψ_e and ψ_α are fully-connected neural networks. We then use the softmax operation over α_j to compute the attention scores which sum up to 1. The neighborhood embedding is thus produced as $e_\eta = \sum_{j=1}^{K} \text{Softmax}(\alpha_j)\psi_h(e_j)$, where ψ_h represents an additional hidden layer. Both in deep sets and attention encoders, we used multi-layer perceptrons (MLPs) with 256 neurons and $\tanh$ activations. Additional details are provided in the supplementary materials.

4 Experiments and results

We train our control policies in an episodic manner, in diverse randomized scenarios with static and dynamic formations. Our virtual experimental facility is a $10 \times 10 \times 10$ m room. At the beginning of each episode, we randomly spawn the quadrotors in a 3 m radius around the central axis of the room, at a height between 0.25 and 2 m. We randomly initialize the orientation, linear and angular velocities of the quadrotors to facilitate learning or robust recovery behaviors. In order to provide a diverse and challenging training environment, we procedurally generate scenarios of different types, listed below.

Static formations. The target formation is fixed throughout the episode. The formation takes various geometric shapes *e.g*. 2D grid, circle, cylinder, and cube (Fig. 1). The separation r between goals in the formation is chosen randomly. A special case $r = 0$ where the goal locations for all quadrotors coincide demands very dense configurations with high probabilities of collisions. We refer to this task as the *same goal* scenario.

Dynamic formations. Here we additionally modify the separation between the quadrotors, as well as the position of the formation origin. The examples of dynamics that we explored include gradually shrinking the inter-quadrotor separation over time, as well as randomly teleporting the formation around the room. In order to train policies to avoid head-on collisions at high speed, we also include scenarios where the swarm is split into two groups. We then swap the target formations of quadrotors in these two groups several times per episode, which requires two teams of quadrotors to fly 'through' each other. This task is referred to as *swarm-vs-swarm* scenario.

Evader pursuit. For the evader pursuit task, the quadrotor team is given a shared goal that moves according to some policy. The goal simulates an evader that the team must pursue. There are two configurations for this task that each employ a different evader trajectory parameterization. In the first configuration, the trajectory of the evader is parameterized by a 3D Lissajous curve, which produces a figure eight trajectory situated in the x-y plane that spans the room.

We train on a single 36-core server with four RTX2080Ti GPUs. A typical experiment includes four parallel training runs, each starting from different random weight initializations. The training system [24] collects experience using 144 parallel processes, and allocates one GPU per trained policy to execute inference and backpropagation. Additionally, each experience collection process

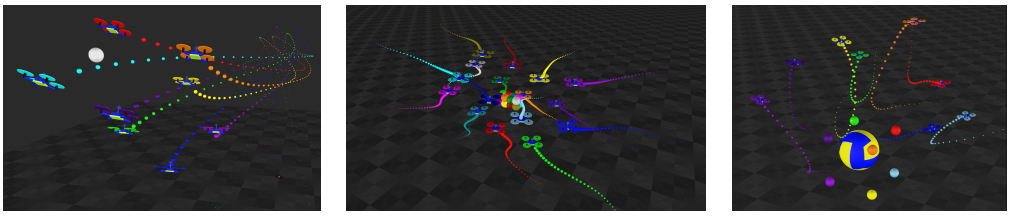

Figure 3: (Left) Evader pursuit, (Middle) $N = 16$ quadrotors in a dense formation after fine-tuning (see Section 4.3), and (Right) A swarm breaking formation to avoid a collision with an obstacle. Videos of the learned swarm behaviors in different scenarios are at: `https://sites.google.com/view/swarm-rl`.

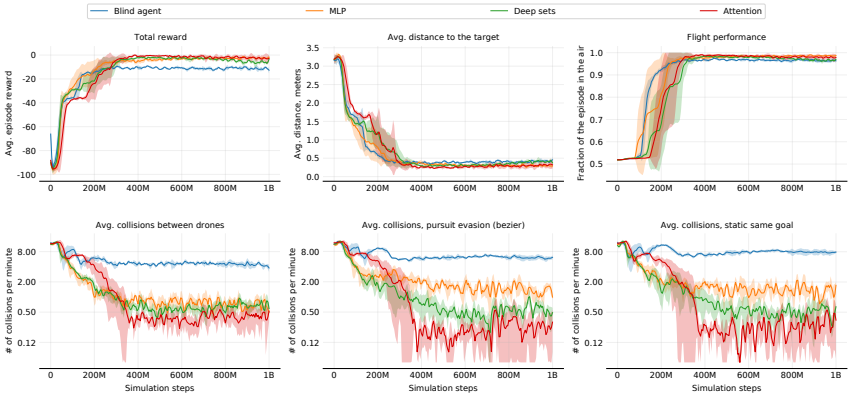

Figure 4: Comparison of different model architectures for $N = 8$ quadrotors with $K = 6$ neighbors visible to each. For each architecture we show mean and standard deviation of four independent training runs.

runs simulation in 4 independent environments with $N = 8$ drones in each, which amounts to $144 \times 4 \times 8 = 4608$ quadrotors modeled simultaneously. The drones are trained in an episodic setting, each episode lasting 16 seconds of subjective time. With physics integration frequency of 200Hz and control frequency of 100Hz, this amounts to 3200 physics steps and 1600 drone actions per episode. The training system reaches a throughput of 6.1×10^4 samples per second and trains four policies on 10^9 samples each in under 19 hours. Thus a single training session amounts to roughly 15 months of simulated quadrotor flight. This highlights the importance of the fast simulated environment since collecting the equivalent amount of experience in the real world would be prohibitive.

4.1 Model architecture study

We compare the training performance with different neighborhood encoders on scenarios with $N = 8$ quadrotors (see Fig. 4) and a fixed number $K = 6$ of visible neighbors. In addition to the architectures described in Section 3.5 we train two baselines. The first baseline is a *blind* quadrotor, for which we remove the neighborhood encoder f_η entirely. While blind quadrotors manage to get close to their targets, they are completely unable to avoid collisions with other quadrotors. The second baseline uses a plain multi layer perceptron, which concatenates of neighbor observations as input. This encoder does not enforce permutation invariance. This baseline policy fails to avoid collisions in most scenarios, suggesting that a permutation-invariant architecture is needed. The difference between the attention and deep sets encoders is most prominent in tasks that require dense swarm configurations, such as the *same goal* scenario. In addition, quadrotors with the deep sets encoder do not get as close to their target locations, sacrificing formation density to minimize collisions. Since the attention-based architecture demonstrated the best performance in both goal reaching and collision avoidance, we use it in all further simulated experiments.

4.2 Attention weights study

We investigate the results of training an attention-based architecture for encoding relative scores of neighboring drones. We ask whether the attention mechanism learns to assign higher attention scores to neighbors that are closer and whose velocity vector points towards the current agent. In addition, we investigate to what degree distance and velocity individually affect the scores. We

modify the *swarm-vs-swarm* scenario to contain two teams of two drones whose goals are 1 m apart and situated in the same horizontal plane. The drones are allowed to settle at their respective goals following which the goals are swapped. We take a snapshot of the experiment and record the softmax attention weights for each drone. In addition, we manually set the relative velocities of all neighbors to $(0, 0, 0)$ for each drone and pass the modified observations to the attention encoder, which provides information on how velocity and distance individually affect the attention weights. The results (Fig. 5) show that the red quadrotor assigns the highest attention weight of 0.61 to the blue quadrotor, which is on a collision course with it. Similarly, the blue quadrotor assigns the highest weight of 0.57 to its red neighbor. For the gray and green quadrotors, all neighbors are assigned a roughly equal weighting, with neighbors closer in distance having slightly higher weights. When the relative velocity observations are manually set to 0 and fed to the attention encoder, we observe a drastically different distribution of attention weights. – much more uniform than when true relative velocity observations are passed, with closer neighbors being assigned slightly higher weights. From this, we can conclude that neighboring quadrotors with small relative distance and high relative velocity vectors in the direction of the viewer are being prioritized over drones that are further away with velocities in other directions. Furthermore, drones with relative velocities pointing towards the viewer seem to be prioritized higher than drones that are closer in distance but with velocity vectors away from the viewer, implying that velocity is considered more important than distance.

4.3 Scaling

We investigate the ability of our policies to scale to larger swarm sizes with no additional training and with fine-tuning. With fine-tuning, the baseline policies trained on 8 drones are trained for an additional 500 million steps with the target number of quadrotors *i.e.* the baseline policies are copied and tuned separately to 16, 32, and 48 quadrotors. The results (Fig. 6) show that without additional fine-tuning, the number of collisions per minute per quadrotor increases with the number of quadrotors, but fine-tuning has a significant positive effect. Even with high swarm sizes (up to 48 drones), the quadrotors are capable of staying below one collision per minute in challenging and dynamic environments. This implies that our approach can take an existing policy trained with only a handful of quadrotors and scale up to much larger swarm sizes without re-training from scratch.

4.4 Obstacle avoidance

To demonstrate the flexibility of our approach, we experiment with a harder version of the environment: we introduce a spherical obstacle that moves through the quadrotor formations at random angles multiple times throughout the episode. At the beginning of each episode, we randomly sample the obstacle size, as well as its velocity and the parameters of its trajectory. To incorporate obstacles into our training protocol, we augment the quadrotor observations $o_i^{(t)}$ to contain the information about the obstacle state $\zeta_i^{(t)}$. This information includes the radius $\hat{r}$ of the obstacle, as well as its position $\hat{p}_i^{(t)}$ and velocity $\hat{v}_i^{(t)}$ relative to the i-th quadrotor. We process the obstacle observations with an additional MLP ϕ_o to produce the obstacle embedding e_o, which is then used in conjunction with the neighborhood encoder to generate the action distributions (see Section 3.4 for details): $e_o = \phi_o(\zeta)\,, \quad \mu_a = \phi_a(e_s, e_\eta, e_o)\,.$ The collision physics and the penalties are modeled in the same way as for quadrotor-vs-quadrotor collisions (Sections 3.2 and 3.3). Fig. 7 shows the training performance in obstacle avoidance scenarios. Despite increased complexity, we achieve performance comparable to the baseline experiment, keeping dense formations close to the target locations.

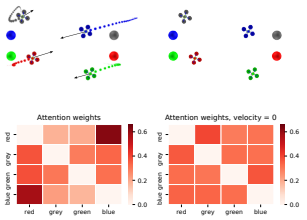

Figure 5: Attention weights between drones. Left: original velocities, right: velocities set to 0.

# agents (N)	Collisions per minute	Distance to target, m	Collisions per minute +finetuning	Distance to target, m +finetuning
8 (Baseline)	0.20	0.21	0.20	0.20
16	1.0	0.27	0.20	0.17
32	5.2	0.37	0.32	0.33
48	8.55	0.41	0.76	0.31

Figure 6: Scaling up the baseline attention policy to a larger number of quadrotors with and without fine-tuning. The number of collisions and average distance to the target are averaged over 20 episodes.

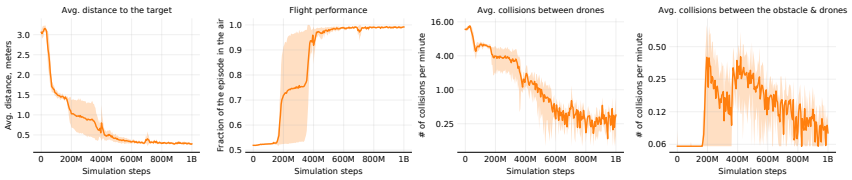

Figure 7: Training performance in the scenario with obstacles. Our quadrotors learn how to maintain dense swarm formations while avoiding collisions with moving external obstacles.

Figure 8: Time lapse of demonstrations with four drones swapping goals (left) and hovering about a shared static goal (right).

5 Physical Deployment

We deployed our policies on the Crazyflie2.0, a small, lightweight, open source quadrotor platform. We tested with four Crazyflie quadrotors on two training scenarios: *static shared goal* and *swap goals*. For the shared goal scenario, the drones are placed on the ground in a square formation 1 m apart. For the swap goals scenario, the drones are placed in a rectangular formation and split into two teams. The teammates are placed 1 m apart, and the two teams are 3 m apart (Fig. 8). This platform is potentially well-suited for large scale experiments, however it is compute constrained (a single quadrotor has a microcontroller running at 168MHz and 1Mb of memory), presenting challenges for our policies for footprint and runtime. To address these challenges, we retrained a deep sets multi-agent policy with only 16 neurons in the hidden layers of the self encoder and 8 neurons in the hidden layers of the neighbor encoder. Notably, these smaller policies performed very well on the Crazyflie platform as illustrated in Fig. 8. We use a Vicon system to receive position updates of neighbor drones at 100Hz. Our controller runs at 500Hz and reuses data from the latest available Vicon update. We apply a low pass exponential filter to all neighbor position measurements to reduce noise. This filter inevitably introduces small delays in the position updates of neighboring drones. For the goal swapping scenario, this is not an issue since the drones have plenty of time to react to neighbors that are on a collision course. However, for the static same goal scenario, we qualitatively observe that noisy and delayed neighbor pose updates exacerbate the collision avoidance problem. We plan to address this issue in the future by training a policy with a smooth collision penalty that has a larger fall off distance.

6 Conclusion

Our results demonstrate that drones trained with deep reinforcement learning can achieve strong goal-reaching and collision avoidance performance across a diverse range of training scenarios with realistic quadrotor dynamics. We present evidence of successful swarm behavior in simulation as well as demonstrating the transfer of policies learned in simulation to the Crazyflie platform, on which we are able to demonstrate successful trials on multiple tasks. Our policies learn how to control quadrotors from scratch, without the use of pre-tuned controllers. This makes the method presented in this paper model-agnostic, *i.e.* we can learn policies for quadrotors with different physical parameters (such as mass, size, inertia matrix, thrust) by re-running the training in the updated simulator. In contrast with classical planning methods, our approach does not introduce any constraints on the velocity or acceleration and allows the controller to take advantage of full capabilities of the quadrotor. This enables agile flight with aggressive maneuvers. Our pursuit-evasion experiments are the most representative of this. In order to stay close to the fast evader, our simulated quadrotors exceed speeds of 7 m/s and reach accelerations up to 1.7 g. Permutation and scale-agnostic model architectures used in our method allow us to switch between different team sizes. We found that after fine-tuning, our policies can control swarms of up to 48 members without a significant increase in computation burden per quadrotor or significant increase of collisions.

References

[1] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian. Trajectory planning for quadrotor swarms. *IEEE Trans. Robotics*, 34(4):856–869, 2018.

[2] K. Solovey, O. Salzman, and D. Halperin. Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. *Int. J. Robotics Res.*, 35(5):501–513, 2016.

[3] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Robotics Res.*, 30(7):846–894, 2011. doi:10.1177/0278364911406761. URL https://doi.org/10.1177/0278364911406761.

[4] J. Tordesillas and J. P. How. MADER: trajectory planner in multi-agent and dynamic environments. *CoRR*, abs/2010.11061, 2020. URL https://arxiv.org/abs/2010.11061.

[5] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, pages 2520–2525. IEEE, 2011. doi:10.1109/ICRA.2011.5980409. URL https://doi.org/10.1109/ICRA.2011.5980409.

[6] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics research*, pages 649–666. Springer, 2016.

[7] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.

[8] R. Allen and M. Pavone. A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance. In *AIAA Guidance, Navigation, and Control Conference*, page 1374, 2016.

[9] J. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In C. Pradalier, R. Siegwart, and G. Hirzinger, editors, *Robotics Research - The 14th International Symposium, ISRR 2009, August 31 - September 3, 2009, Lucerne, Switzerland*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. Springer, 2009. doi:10.1007/978-3-642-19457-3_1. URL https://doi.org/10.1007/978-3-642-19457-3_1.

[10] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager. Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells. *IEEE Robotics Autom. Lett.*, 2(2):1047–1054, 2017. doi:10.1109/LRA.2017.2656241. URL https://doi.org/10.1109/LRA.2017.2656241.

[11] W. Luo, W. Sun, and A. Kapoor. Multi-robot collision avoidance under uncertainty with probabilistic safety barrier certificates. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/03793ef7d06ffd63d34ade9d091f1ced-Abstract.html.

[12] B. Rivière, W. Hönig, Y. Yue, and S. Chung. GLAS: global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning. *IEEE Robotics Autom. Lett.*, 5(3):4249–4256, 2020. doi:10.1109/LRA.2020.2994035. URL https://doi.org/10.1109/LRA.2020.2994035.

[13] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset. PRIMAL: pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics Autom. Lett.*, 4(3):2378–2385, 2019. doi:10.1109/LRA.2019.2903261. URL https://doi.org/10.1109/LRA.2019.2903261.

[14] G. Shi, W. Hönig, Y. Yue, and S. Chung. Neural-swarm: Decentralized close-proximity multirotor control using learned interactions. pages 3241–3247, 2020. doi:10.1109/ICRA40945.2020.9196800. URL https://doi.org/10.1109/ICRA40945.2020.9196800.

[15] G. Shi, W. Hönig, X. Shi, Y. Yue, and S. Chung. Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions. *CoRR*, abs/2012.05457, 2020. URL https://arxiv.org/abs/2012.05457.

[16] 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 *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 59–66. IEEE, 2019. doi:10.1109/IROS40897.2019.8967695. URL https://doi.org/10.1109/IROS40897.2019.8967695.

[17] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza. Autonomous drone racing with deep reinforcement learning. *CoRR*, abs/2103.08624, 2021. URL https://arxiv.org/abs/2103.08624.

[18] G. Moraes Barros and E. Luna Colombini. Using soft actor-critic for low-level uav control. *arXiv e-prints*, pages arXiv–2010, 2020.

[19] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *CoRR*, abs/1610.03295, 2016. URL http://arxiv.org/abs/1610.03295.

[20] J. K. Gupta, M. Egorov, and M. J. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In G. Sukthankar and J. A. Rodríguez-Aguilar, editors, *Autonomous Agents and Multiagent Systems - AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers*, volume 10642 of *Lecture Notes in Computer Science*, pages 66–83. Springer, 2017. doi:10.1007/978-3-319-71682-4_5. URL https://doi.org/10.1007/978-3-319-71682-4_5.

[21] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig. Learning to fly - a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. *CoRR*, abs/2103.02142, 2021. URL https://arxiv.org/abs/2103.02142.

[22] J. Förster. System identification of the crazyflie 2.0 nano quadrocopter. BA Thesis, ETH Zurich, 2015.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[24] A. Petrenko, Z. Huang, T. Kumar, G. Sukhatme, and V. Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *ICML*, 2020.

[25] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola. Deep sets. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3391–3401, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[27] C. Chen, Y. Liu, S. Kreiss, and A. Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6015–6022. IEEE, 2019.