

Supplementary Materials

I. ARCHITECTURE DETAILS

A. Model architectures

In our model architecture we use exclusively feed-forward fully connected neural networks. Functions ϕ_s and ϕ_a used by all our architectures are MLPs with layers $\phi_s : (\text{inputSize}, 256), (256, 256)$ and $\phi_a : (256, 4)$ respectively. We use tanh activations after every layer except for the last layer of ϕ_a which has no activation.

B. Mean embeddings

The neighbor encoder ψ_η has layers $(\text{inputSize}, 256), (256, 256)$ with tanh activations after each layer.

C. Attention

- ψ_e : $(\text{inputSize}, 256), (256, 256)$ with tanh activations following each layer.
- ψ_h : $(256, 256), (256, 256)$ with tanh activations following each layer.
- ψ_α : $(512, 256), (256, 256), (256, 1)$ with tanh activations following the first and second layer.

II. HYPERPARAMETERS

A. Reward function

All reward coefficients except for collision penalty are multiplied by dt , which is equal to 0.01 in our experiments since we use control frequency of 100Hz. This creates a reward scheme independent of control frequency, i.e. agents receive the same reward for the same actions regardless of dt . For each collision, we only apply collision penalty on the first timestep that collision is detected. Therefore, collision penalty is independent of dt .

Coefficient name	Symbol	Value
Position reward	α_{pos}	$1.0 * dt$
Collision penalty	α_{col}	5.0
Smooth proximity penalty	α_{prox}	$10.0 * dt$
Angular velocity penalty	α_ω	$0.1 * dt$
Motor thrusts penalty	α_f	$0.05 * dt$
Rotation penalty (penalizes high pitch & roll)	α_{rot}	$1.0 * dt$

TABLE I: Reward function coefficients.

B. Training algorithm

Learning rate	10^{-4}
Discount γ	0.99
Policy initialization	Xavier uniform (Glorot and Bengio [1])
Optimizer	Adam (Kingma and Ba [2])
Optimizer settings	$\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-6}$
Gradient norm clipping	5.0
Rollout length T	128
Batch size, samples	1024
Number of training epochs	1

TABLE II: Hyperparameters for the training algorithm, APPO.

III. TRAINING DETAILS

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 [3] collects experience using 144 parallel processes, and allocates one GPU per trained policy to execute inference and backpropagation. Additionally, each experience collection process 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.

IV. PHYSICAL DEPLOYMENT

A. Ablation study

Due to the physical constraints imposed by Crazyflie 2.0, we needed to simplify our policies to meet the runtime and memory requirements of the platform. To this end, we conducted an ablation study over *deep sets* policies with smaller self-encoder and neighbor encoder, since *deep sets* architecture is more computationally efficient compared to our attention-based model. Specifically, we ablated over the following architecture sizes:

- ϕ_s : (inputSize, 32), (32, 32) ψ_n : (inputSize, 16), (16, 16)
- ϕ_s : (inputSize, 32), (32, 32) ψ_n : (inputSize, 8), (8, 8)
- ϕ_s : (inputSize, 16), (16, 16) ψ_n : (inputSize, 8), (8, 8)

Where ϕ_s is the self encoder and ψ_n is the neighbor encoder. All other hyperparameters, training procedure, and other aspects of the networks are kept the same as in simulation. Since all three models demonstrated comparable performance (albeit inferior to bigger models we used in the simulation), we chose the smallest of the three architectures, the 16x8 deep sets model.

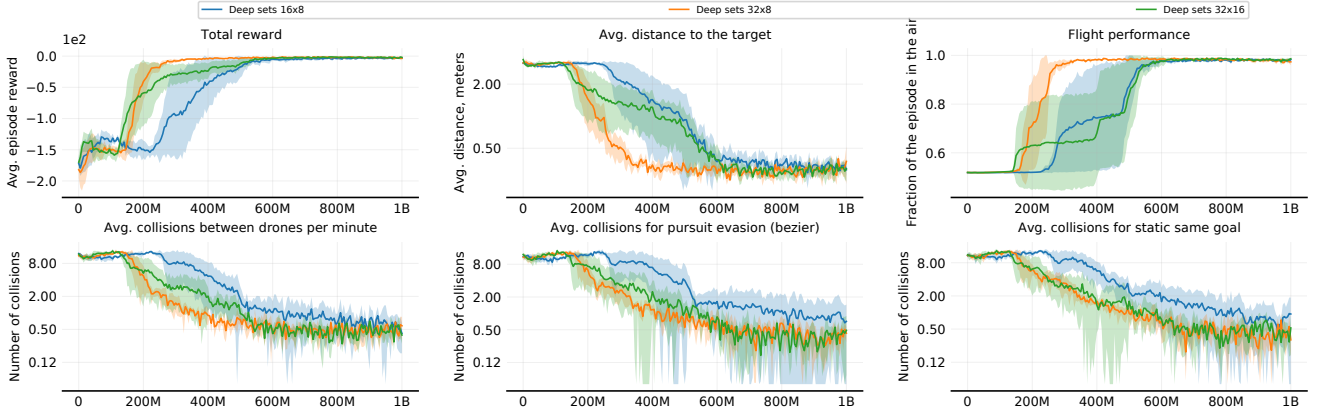


Fig. 1: Ablation study over smaller *deep sets* network architectures. In the legend, 16x8 corresponds to the model with the hidden size 16 for the self encoder and hidden size 8 for the neighbor encoder. Likewise, 32x16 and 32x8 correspond to models with hidden size 32 in the self-encoder and 16 and 8 respectively in the neighbor encoder.

B. Experiment details

We conducted experiments on four training scenarios from simulation: *same static goal*, *same dynamic goal*, *swap goals*, and *dynamic geometric formations*. The first two scenarios are based on four drones and the last two scenarios are based on eight drones. A video of the results of these experiments has been included in our [website](#).

C. Physical baseline comparison

We repeated the "swap goals" experiment with an algorithm from [4]: a PID controller combined with Buffered Voronoi Cells for generating safe regions around the quadrotors. Figure 2 demonstrates the significant difference between the behavior of two methods. Our neural controller quickly reacts to the change in target position, reaching speeds up to 4 m/s and acceleration up to 7 m/s². Spikes on the velocity plot show that it takes only 1-1.5 seconds to react to the two goal swapping events. Controller in [4] is a lot more conservative: the drones slowly drift to their targets at 1 m/s and each time the goals are swapped it takes the controller several seconds to complete the maneuver.

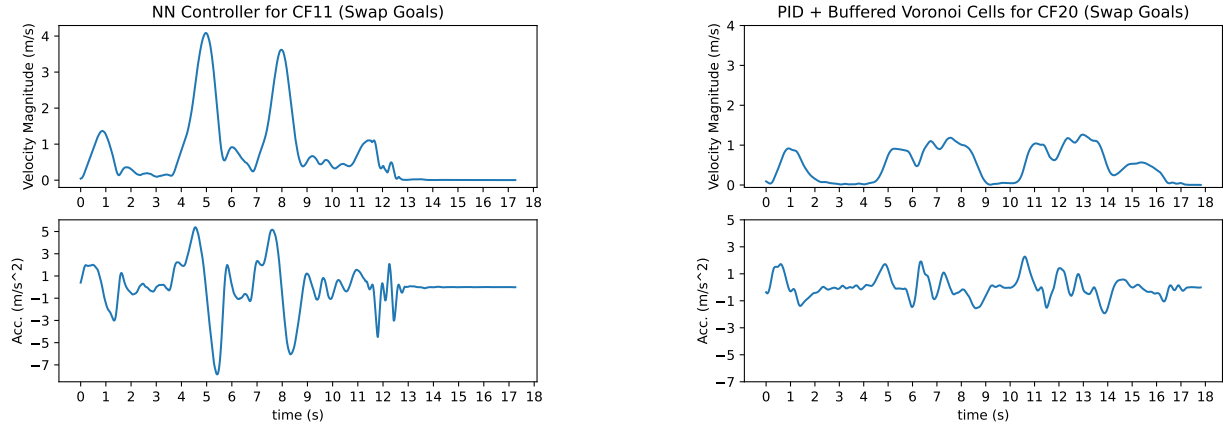


Fig. 2: "Goal Swap" physical experiment. (Left) Our policy: the quadrotors reach high velocity and acceleration to quickly react to the changes in target positions. (Right) PID + Buffered Voronoi Cells: The quadrotors take longer to complete the task and use much more conservative maneuvering.

REFERENCES

- [1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010.
- [2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
- [3] Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *ICML*, 2020.
- [4] Dingjiang Zhou, Zijian Wang, Saptarshi Bandyopadhyay, and Mac 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>.