2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

# Collision Avoidance and Navigation for a Quadrotor Swarm Using End-to-end Deep Reinforcement Learning

Zhehui Huang*, Zhaojing Yang*, Rahul Krupani, Baskın Şenbaşlar, Sumeet Batra, Gaurav S. Sukhatme

***Abstract*— End-to-end deep reinforcement learning (DRL) for quadrotor control promises many benefits – easy deployment, task generalization and real-time execution capability. Prior end-to-end DRL-based methods have showcased the ability to deploy learned controllers onto single quadrotors or quadrotor teams maneuvering in simple, obstacle-free environments. However, the addition of obstacles increases the number of possible interactions exponentially, thereby increasing the difficulty of training RL policies. In this work, we propose an end-to-end DRL approach to control quadrotor swarms in environments with obstacles. We provide our agents a curriculum and a replay buffer of the clipped collision episodes to improve performance in obstacle-rich environments. We implement an attention mechanism to attend to the neighbor robots and obstacle interactions - the first successful demonstration of this mechanism on policies for swarm behavior deployed on severely compute-constrained hardware. Our work is the first work that demonstrates the possibility of learning neighbor-avoiding and obstacle-avoiding control policies trained with end-to-end DRL that transfers zero-shot to real quadrotors. Our approach scales to 32 robots with $80\%$ obstacle density in simulation and 8 robots with $20\%$ obstacle density in physical deployment. Website: `https://sites.google.com/view/obst-avoid-swarm-rl`**

## I. Introduction

Collision avoidance in point-to-point navigation of quadrotors is an enabler for many applications, including package delivery [1], surveillance [2], warehouse stocktaking [3], and search and rescue [4]. Existing real-time trajectory planning approaches [5], [6], [7], [8], [9], [10], [11], [12], [13] are generally compute-heavy, which limits their collision avoidance ability on embedded hardware given their low reactivity. Existing classical collision-avoiding control methods [14], [15], while less compute-heavy compared with real-time trajectory planning approaches, are generally conservative, which limits their performance and scalability in complex environments.

In our prior work [16], we explored using end-to-end RL to train quadrotor teams to learn emergent cooperative behaviors and agile maneuvers in *obstacle-free environments*. However, the addition of obstacles poses a significant challenge, as the number of agent-environment interactions increases exponentially, thus destabilizing training in the early critical stages of learning. In this work, we propose several changes that not only enable learning of the same agile control policies and emergent cooperative behaviors in obstacle dense environments, but also enable learning to fly through narrow gaps and generalizes to unseen scenarios. The contributions of our work are as follows:

- To the best of our knowledge, our approach is the first *purely end-to-end DRL-based* approach that generates decentralized, low-level control policies for quadrotor swarms in obstacle-rich environments. The learned control policies allows reaching to goal positions while avoiding collisions with other quadrotors and static obstacles with a high success rate. The robots are able to fly through as small as $0.15\,\mathrm{m}$ gaps between obstacles where the radius of the quadrotors is $0.05\,\mathrm{m}$. The learned policies are zero-shot transferrable to physical quadrotor swarms. We utilize signed distance field (SDF) based obstacle observations, which are quantity and permutation invariant, and show their effectiveness in learning collision avoidance. We propose a simple but useful replay mechanism, which shows its effectiveness in training and better than prioritized level replay (PLR) [17].
- We compare our approach with state-of-the-art learning based and classical control based collision avoidance methods, and show its superior performance to the learning based and comparable performance to the classical controller using less computation time.
- We deploy our approach to compute-constrained hardware, i.e., Crazyflie 2.1, to show its applicability to real world robots. Our work is the first to provide a successful demonstration of deploying the attention mechanism on such compute-constrained hardware.

## II. Related Work

Compared with navigating multiple robots in obstacle-free environments, navigating multiple robots in static obstacle-rich environments significantly increases the complexity of the problem for both classical and learned methods [18]. Its discrete variant studied as the multi-agent path finding problem is NP-Hard for total arrival time, makespan, or distance optimization [19]. Its continuous variants for only geometric planning are known to be PSPACE-Hard [20]. We are not only interested in kinematics but also the dynamics of the underlying systems, making the problem considerably harder.

Decentralized real-time trajectory planning is utilized for multi-robot navigation in which robots plan trajectories for themselves in a receding horizon fashion avoiding collisions with each other and obstacles in the environment. Some decentralized real-time trajectory planning algorithms require position only sensing [5], [6], [7] while others rely on full state sensing [12], [13] and full feature

* Equal contribution. The authors are with the Department of Computer Science, University of Southern California, Los Angeles, CA 90089 USA (e-mail: zhehuihu@usc.edu). GSS holds concurrent appointments as a Professor at USC and as an Amazon Scholar. This paper describes work performed at USC and is not associated with Amazon.

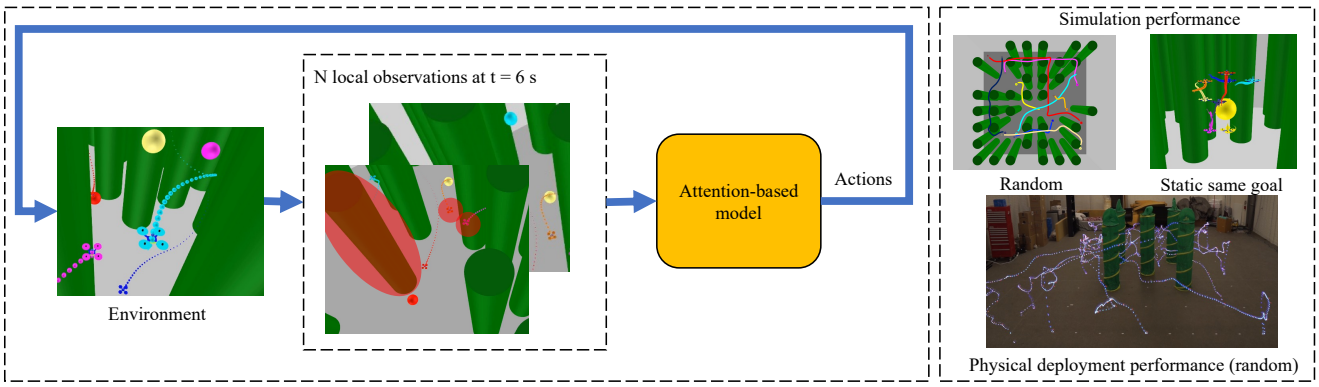



Fig. 1. **System overview.** There are $N$ robots in the environment, and the green cylinders represent obstacles. At every tick, each robot collects its own local observations from the environment and computes its actions independently *e.g.,* red shadows in the stacked local observations denote the local observations of the red robot. Our learned policy is effective in simulated trials, scales, and can be transferred to physical, severely compute-constrained quadrotors.

plan communication [9], [11]. Some explicitly account for asynchronous planning between robots [8], [10]. For dynamic feasibility, some approaches formulate the problem in a model-predictive control style [5], [8], [11], [12], while others exploit the differential flatness of the robots [6], [7], [9], [10], [13]. Since these approaches make relatively longer horizon decisions, they typically require considerable computational resources, making them inadequate for computationally limited hardware. For example, each planning iteration in MADER [9] takes $\sim 200ms$ on Intel i7 processors @4.7 GHz with access to a state-of-the-art optimization library, Gurobi, using a considerable amount of memory, and RSFC [13] takes $\sim 50ms$ per iteration on Intel i7 processors @ 3.60 GHz using CPLEX optimization studio, both of which are state-of-the-art online planning methods.

To execute on computationally limited hardware, some decentralized approaches compute only the next actions to execute at high frequency, such that if all robots compute their next action using the same approach, the system stays collision-free. ORCA [15] computes safe velocity commands, SBC [14], which is based on safety barrier certificates, computes safe acceleration commands, both of which utilize real-time optimization to compute the next actions. However, these algorithms are generally more conservative than the planners, which limits their performance and scalability in complex environments.

Utilizing learning while computing next actions has been investigated in order to tackle conservativeness short horizon approaches. GLAS [21] utilizes imitation learning to mimic a centralized planner and combines it with a safety module to make the computed actions safe. In [22], DRL is utilized for mapping observations to linear and angular velocity commands for omnidirectional robots. In [23], local neighbor observations are mapped to velocity commands using DRL. In [24], a control network is learned with a control barrier function-based safety module, in which the behavior of the integrated system is considered during training. [25] utilizes a trained policy that generates human-like commands (faster/slower), which are used by the downstream traditional trajectory optimizer, and [26] uses DRL to control flocking fixed-wing UAVs in a leader-follower setup in which control actions are velocities and roll angles. Since algorithms compute only the next actions to execute, they can run in limited capability hardware.

Using a learning-based method to train neural network policies that directly map local observations to rotor thrusts can correctly represents the true actuation limits of the quadrotor platform given there is no additional control loop needed, and potentially execute much more agile and less conservative maneuvers compared with high-level control inputs, such as desired linear velocity or future waypoints [27]. [16] proposes an end-to-end DRL-based approach to control quadrotors with thrust inputs in obstacle-free environments. The proposed approach, trained in simulation [28], can be zero-shot transferred to the real robots. However, directly applying this approach to obstacle-rich environments does not result in acceptable performance as we show in section IV. We address the limitations in [16] and show collision avoidance and team navigation in obstacle-rich environments. Our approach, based on [16], [29], inherits the features of zero-shot transferable, robust to external disturbances, can withstand harsh initial conditions, and recoverable from collisions.

## III. Method

### A. Problem Formulation

The state of the environment with $N$ robots and $M$ static obstacles at time $t$ is $(s_1^t, \ldots, s_N^t, g_1^t, \ldots, g_N^t, \mathcal{O}_1^t, \ldots, \mathcal{O}_M^t)$, where $s_i^t$ is the state of robot $i$, containing the position, linear and angular velocity, and orientation of the robot, $g_i^t$ is the goal position of robot $i$, and $\mathcal{O}_i^t$ is the state of the static obstacle $i$, containing the position of the obstacle at time $t$. The actions are rotor thrusts, which affect the states of the robots according to quadrotor dynamics [30], [31]. Our objective is to train a decentralized control policy that directly maps local observations of a robot, which mentioned in subsection III-B, to its rotor thrusts with the goal of minimizing its distance to its goal while avoiding collisions with other robots and static obstacles.

### B. Training Setup

We train our control policies in a $10m \times 10m \times 10m$ simulated room with obstacles, where the height of the obstacles is the same as the room height. We discretize the

center $8m \times 8m$ area of the room to 64 square grid cells of $1m^2$, and spawn obstacles at the centers of the cells. At the beginning of each training episode, we randomly generate obstacles with a configurable density of occupied grid cells and obstacle sizes. Following this, we spawn robots in the centers of obstacle-free grid cells at random heights between $1m$ and $3m$ with random initial orientations and velocities.

**Goal generation**: We use two goal generation methods. In the first, all robots share the same static goal position. The goal is spawned at the position within the room that is farthest away from any obstacles. The robots need to navigate around obstacles as they move toward the goal, and they need to avoid each other while staying close to the goal. In the second, each robot has an uncorrelated randomly generated goal. Robots need to navigate to the goals, while avoiding collisions with obstacles and other robots.

**Observations and actions**: The observation of robot $i$ at time t is: $o_i^t = (e_i^t, \eta_i^t, \zeta_i^t)$, where i) $e_i^t$ is robot's observation of its own state and goal, ii) $\eta_i^t$ is observation of the neighbor robots, and iii) $\zeta_i^t$ is the observations of obstacles. Specifically, $e_i^t = (p_i^t, v_i^t, R_i^t, \omega_i^t, h_i^t)$, where $p_i^t \in \mathbb{R}^3$ is the position of the robot relative to its goal position, $v_i^t \in \mathbb{R}^3$ is its linear velocity in the world frame, $R_i^t \in SO(3)$ is the rotation matrix from the body frame to the world frame, $\omega_i^t \in \mathbb{R}^3$ is its angular velocity in the body frame, and $h_i^t \in \mathbb{R}$ is the altitude of robot in the room. $\eta_i^t = (\tilde{p}_{i1}^t, \ldots, \tilde{p}_{iK}^t, \tilde{v}_{i1}^t, \ldots, \tilde{v}_{iK}^t)$, where $\tilde{p}_{ij}^t \in \mathbb{R}^3$ and $\tilde{v}_{ij}^t \in \mathbb{R}^3$ are the position and velocity of the robot relative to its $j$-th nearest neighbor robot, and $K \leq N-1$ is the number of neighbors that the robot can sense. The obstacle observations are based on the idea of a SDF [32]. The obstacle observations $\zeta_i^t$ have 9 values, which represent the distance to the closest obstacles scaled to a 3×3 cells and discretized into uniformly spaced cells with the pre-defined resolution, 0.1m in our setting. The obstacle observations are quantity and permutation invariant, which can support an arbitrary number of obstacles. The action of robot $i$ at time t is $a_i^t \in [0,1]^4$, corresponding to the thrust levels at each of the four rotors. We transform $a_i^t$ to thrusts $f_i^t$ linearly such that 0 is no thrust, and 1 is maximum thrust.

**Reward function**: We extend the reward function proposed in [16] for inter-robot collision avoidance in order to provide obstacle avoidance behavior as well. The reward function for each robot $i$ consists three main components: $r_i^t = r_{i,\text{dist}}^t + r_{i,\text{col}}^t + r_{i,\text{control}}^t$. $r_{i,\text{dist}}^t = -\alpha_{\text{dist}} \left\| p_i^t \right\|_2$ encourages robot $i$ to minimize its relative distance to the goal. $r_{i,\text{col}}^t = -\alpha_{\text{ocol}} \mathbb{1}_{\text{ocol}}^t - \alpha_{\text{rcol}} \mathbb{1}_{\text{rcol}}^t - \alpha_{\text{rclose}} \sum_{j=1}^{K} \max\left(1 - \left\| \tilde{p}_{ij}^t \right\|_2 / d_{\text{rclose}}, 0\right)$ penalizes i) robot-obstacle collisions, ii) inter-robots collisions, and iii) approaching to within $d_{\text{rclose}}$ distance of other robots. $\mathbb{1}_{\text{rcol}}^{(t)}$ and $\mathbb{1}_{\text{ocol}}^{(t)}$ are indicator functions, which are equal to 1 when robots collide with other robots or obstacles, respectively. We only penalize every collision once even though the duration of each collision in the simulation is bigger than one step. $r_{i,\text{control}}^t = -\alpha_{\text{floor}} \mathbb{1}_{\text{floor}}^t - \alpha_\omega \left\| \omega_i^t \right\|_2 - \alpha_f \left\| f_i^t \right\|_2 + \alpha_{\text{orient}} R_{i,33}^t$ penalizes robot $i$ for i) crashing with the floor, having ii) high angular velocity, iii) high control effort, and iv) big rotation angle relative to the $z$ axis in the world frame. All symbols starting with $\alpha$ and $d$ are hyperparameters.

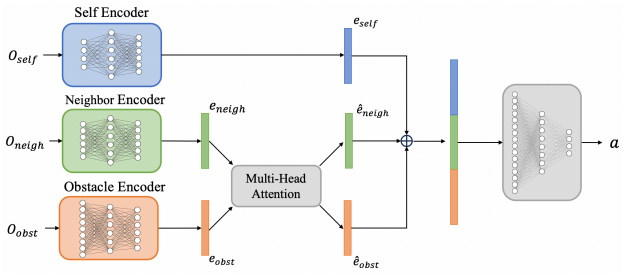


Fig. 2. **Model architecture**

**Reinforcement learning algorithm**: We use the asynchronous version of the decentralized, independent proximal policy optimization [33]. Specifically, we use the implementation from Sample Factory [34] to train our control policies.

### C. Model architecture

Our model architecture is a combination of MLPs and a multi-head attention module [35], shown in Figure 2. We use three two-layer MLPs as encoders to process the self, neighbor, and obstacle observations and obtain the corresponding embeddings separately. We then fuse these three embeddings. The multi-head attention module is used to prioritize the importance between neighbor and obstacle embedding: $(\hat{e}_{\text{neigh}}, \hat{e}_{\text{obst}}) = f_{\text{attn}}(e_{\text{neigh}}, e_{\text{obst}})$. Finally, we concatenate $e_{\text{self}}, \hat{e}_{\text{neigh}}, \hat{e}_{\text{obst}}$ as the final embedding and feed it into a two-layer MLP to obtain the actions $a_i^t$.

### D. Replay Buffer

We observe a substantial number of collisions within each episode. However, given the time-extended trajectory, these tend to be dispersed and attenuated. To amplify collision events and enhance collision aversion toward obstacles and other robots, once a collision is detected, we append the environment state 1.5 s before the collision happens to a replay buffer. We save multiple environment states from the same episode if there are multiple collisions during that episode. During training, we use a simple curriculum learning method. We define the replay rate $\alpha_r$, the probability of replaying one of the previous episodes. Alternatively, we generate a new episode with probability $1 - \alpha_r$. At the end of each episode, we check the number of times each state in the replay buffer is replayed. If the replay count of an environment state has exceeded a maximum replay threshold, we assume that the episode starts at this environment state is too difficult to learn for the current policy and we remove it from the replay buffer. This way, all episodes in the buffer are learnable with the current policy.

## IV. Experiments and Results

We evaluate the effectiveness of our learned controller by i) ablating its important parts and showing that all parts are required for its effectiveness, ii) investigating reward function, iii) conducting scaling experiments to show its applicability to highly cluttered environments, iv) comparing it to two state-of-the-art baselines, and v) transferring it to

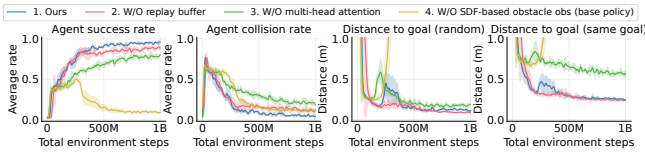



Fig. 3. **Ablation study:** We remove components one-by-one in order to show the necessity of various parts.

real quadrotors and showing zero-shot sim-to-real transfer. vi) investigating generalizability. The base setting for experiments reported in this section is 8 robots where each robot is able to sense 2 nearest neighbors, $20\%$ obstacle density and $0.6m$ obstacle size in a $10m \times 10m \times 10m$ room. We train our experiments across 4 different seeds.

We use five metrics during our evaluations. **Success rate** is the ratio of robots that reach their goal without collisions, **collision rate** is the ratio of robots that collide with other robots or obstacles at least once during the whole episode, **distance to goal** is the average distance (across all robots) to the goal in the final second of the episode, **flight distance** is the average flight distance (across all robots) throughout the whole episode, and **inference time** is the total inference time from observation collection to emitting actions.

### A. Ablation study

We conduct an ablation study to investigate the impact of the new components and present the results in Figure 3.

**Replay mechanism**: Removing the replay mechanism results in a noticeable drop in performance. For the collision rate plot, we find that without the replay buffer, the collision rate increases from $0.05$ to $0.12$. Typically when a collision occurs, the collision itself lasts for only one to two timesteps out of a 1500 timestep episode. By storing collision events in a buffer and clipping the episode around the collision event, we essentially force the RL algorithm to focus on learning better actions to reduce the collision penalty by artificially reducing the sparsity of the collision reward. To further demonstrate the effectiveness of our replay mechanism, we compare our method with a popular curriculum learning approach, prioritized level replay (PLR) [17]. The success rate plot in Figure 4 shows our replay mechanism can learn policies that are $5\%$ higher in success rate than the policies trained with PLR. Our hypothesis is that utilizing a potential score function (L1 value loss) in PLR encourages faster convergence to the target location *and* collision avoidance, which are not always aligned goals. In contrast, our replay mechanism solely focuses on collision avoidance.

**Multi-head attention**: Removing the multi-head attention model results in a further performance drop. In Figure 3, the success rate drops from $0.88$ to $0.79$, the collision rate increases from $0.12$ to $0.20$, distance to goal (random) increases from $0.09m$ to $0.18m$, and distance to goal (same goal) increases from $0.24m$ to $0.57m$. The attention mechanism enables the agents to prioritize certain agent-agent and agent-obstacle interactions over others, e.g., according to their distances or collision courses. Without this mechanism, agents equally weigh other objects, which result in low performance.

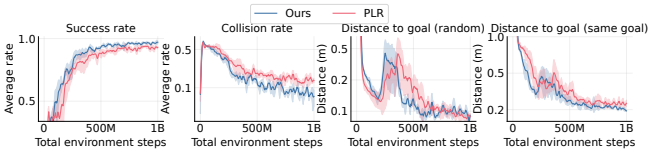



Fig. 4. Comparison of our replay strategy with prioritized level replay.

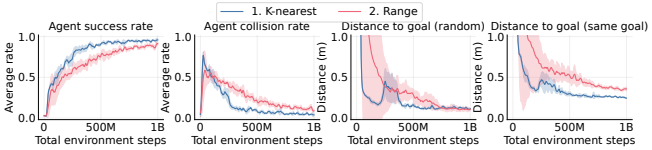



Fig. 5. Comparison of K-nearest neighbor observations with range-based neighbor observations. $K = 2$ and range = $4m$.

**SDF-based obstacle observations**: Switching from the compact and scalable SDF-based obstacle representation to a L-nearest obstacles obstacle representation, we get the policy proposed in [16] for inter-robot and dynamic obstacle collision avoidance, which results in even further performance drop. We find that without the SDF representation, the success rate drops from $0.79$ to $0.10$, representing a *nearly 70*$\%$ reduction in performance and a divergence in training. Distance to goal (random) and distance to goal (same goal) plots convey similar trends. L-nearest obstacle representation, although share the same idea as neighbor robots representation, does not work for flying through gaps between obstacles. We hypothesize that learning collision avoidance is different from learning to fly through gaps. To avoid collisions, robots only need to stay away from neighbor robots or obstacles. However, to fly through gaps, the robots need to precisely estimate their feasible moving spaces. L-nearest obstacle observations, with reduced representational capacity compared to the proposed SDF-based representations, results in a lower ability to navigate obstacle-dense environments, especially when the number of obstacles exceeded the number of nearest-neighbor encoders.

**Range-based neighbor observations**: We compare our K-nearest neighbor observations with range-based neighbor observations, which sense all neighbor robots in the pre-defined range. To support range-based representations, instead of using a two-layer MLP, we use the attention model proposed in [16] to deal with varying length of neighbor observations. In Figure 5, if we replace K-nearest neighbor observations with range-based neighbor observations, the success rate drops from $0.95$ to $0.89$, the collision rate increases from $0.05$ to $0.11$, and the distance to goal (same goal) increases from $0.26m$ to $0.36m$. We hypothesize that although range-based observations can provide more information to make robots to perform better in theory, with a more complicated architecture, the learning process becomes more challenging.

### B. Analyzing reward functions

To analyze if our reward function is designed properly, we use the critic encoder in our training algorithm, IPPO, to evaluate the state value (V-value) by investigating the relation between position and V-value. For better visualization, we simplify the position from 3D to 2D and set the velocity of all robots to $0$. For each robot, we change its position and change

the relative positions to its K-nearest neighbor robots accordingly given the pre-defined resolution. And we use the critic encoder to calculate the V-value at every position in the pre-defined range to build the V-value map. Figure 6 shows the critic encoder is reasonably well in estimating the V-value.

### C. Scaling

*1) Number of robots:* We investigate the scalability of our approach while keeping the sensed number of neighbor robots at 2. The results in Figure 7 show that our policies can scale up to 32 robots without a significant decrease in success rate in a $10m \times 10m \times 10m$ room. With a larger room, we hypothesize that our policies can scale to a greater number of robots.

*2) Number of sensed neighbor robots:* We fix the number of robots at 32 and set the number of sensed neighbor robots to $1, 2, 6, 16$ and 31. The results in Figure 8 show our policies work even with one neighbor that can be sensed, but sensing two neighbor robots stably provides the best performance. A larger number of neighbors does not help performance and even decreases performance, such as the 31 neighbor. We attribute this to the significant increase in the input dimension, increasing the hardness of the learning problem.

*3) Obstacle density:* We investigate the ability of our policies to scale to large obstacle density with the obstacle size fixed at 0.6m. The results in Figure 9 show the robustness of our policies to obstacle density scaling; our method can scale up to $80\%$ obstacle density.

*4) Obstacle size:* We investigate the ability of our policies to scale to large obstacle size with the obstacle density fixed at $80\%$. The results in Figure 10 show our policies can scale up to 0.85m obstacle size.

### D. Baseline comparison

We compare our approach with a state-of-the-art safety barrier certificates-based method SBC [14] and a state-of-the-art learning-based method GLAS [21] in simulations.

SBC is a method using safety barrier certificates, implemented in conjunction with a controller [30] that computes safe accelerations to reach the goal position given the current state and then converts the acceleration into thrusts. SBC takes the state of all the neighbors and obstacles within a certain radius of the robot. Given this information and the required acceleration from the controller, SBC outputs a safe acceleration command to direct the robot to the goal, which is then converted intro direct thrust control by the controller. In GLAS, each robot takes the observation of neighbor robots and obstacles within a sensing radius as the input and outputs velocity in $xy$ plane at the next timestep. Since GLAS is implemented in 2D, we set the spawning point and the goal of each robot with the same z-value for a fair comparison. We also use the same controller as above to transform GLAS output into direct thrusts.

In Table I, we compare with GLAS and SBC in the environment with $8$ robots and $20\%$ obstacle density. $\uparrow$ represents higher value refers better performance, and $\downarrow$ represents lower value refers better performance. Table I shows our method outperforms GLAS by $22\%$ in terms of success rate and 3 times faster in inference time. Our method is comparable to SBC in success rate and collision rate while being 4x faster in inference speed than SBC.

TABLE I

BASELINE COMPARISON

| Method | Success rate ↑ | Collision rate ↓ | Flying distance(m) ↓ | Inference time(ms) ↓ |
|---|---|---|---|---|
| GLAS | 0.75 | 0.25 | 4.4 (2D) | 15 |
| SBC | 0.99 | 0.01 | 7.1 (3D) | 21 |
| **Ours** | 0.97 | 0.03 | 5.3 (3D) | 5 |

TABLE II

TRAIN FROM SCRACH VS POLICY DISTILLATION

| Method | Success rate ↑ | Collision rate ↓ | Distance to goal(m) ↓ |
|---|---|---|---|
| From Scratch | 0.88 | 0.04 | 0.43 |
| Policy Distillation | 0.72 | 0.28 | 0.31 |

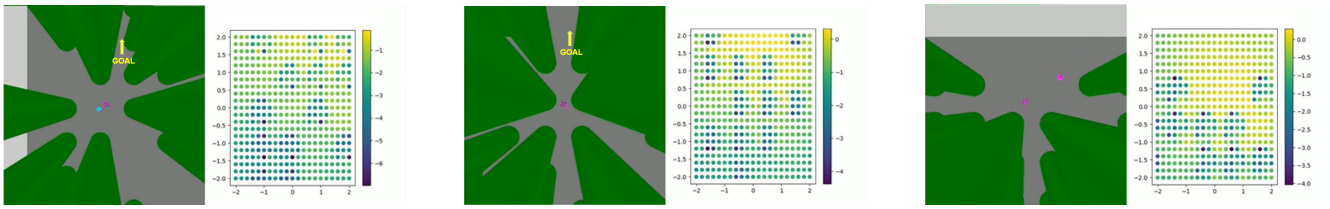
We further compare our approach with these two baselines in more complex environments by fixing the number of robots at 32, and comparing with different obstacle density and obstacle size. Figure 11 shows that our polices are insensitive to obstacle density and obstacle size, while SBC is sensitive to obstacle size, and GLAS is sensitive to obstacle density and does not work when obstacle density is $80\%$ and obstacle size $\geq$ 0.6m. When the obstacle size is 0.6m, our policies are comparable to SBC regardless of obstacle density. However, in the environment with 32 robots and $80\%$ obstacle density, when the obactacle size increases to 0.8m, our policies outperform SBC. In this case, if two obstacles are positioned adjacent to each other, there is a gap of only 0.2m between them, and the diameter of drone itself is 0.1m. When we further increase the obstacle size even larger to 0.85m (resulting in the narrowest gap between obstacles measuring only 0.15m), the results in Figure 11 illustrates that our approach maintains its effectiveness, while SBC falls short. This highlights that our policy is better at flying through complex environments with narrow gaps.

### E. Generalization

We evaluate our control policies in two unseen scenarios, pursuit evasion and swap goals. In pursuit evasion scenario, all robots pursuit a same goal, and the trajectory of the goal is based on Bézier curve. In swap goals scenario, all robots swap their goals after a random period of time. We evaluate the performance of our control policies over 20 episodes in the environment with 8 robots and $20\%$ obstacle density. The success rate of pursuit evasion scenario is $0.83$, and the success rate of swap goals scenario is $0.85$.

### F. Physical deployment

We use Crazyflie 2.1, a quadrotor platform as our testbed for physical deployment. We use Vicon system for localization with the frequency of 100 Hz, and each quadrotor's controller runs at 1000 Hz. For obstacle mapping, we use a list to store the location of static obstacles. Quadrotors generate and use local SDFs using the obstacle list online. Given the compute constraints of Crazyflie 2.1 (168 MHz CPU and 192 Kb RAM), we decrease the model size shown in Figure 2. We set the hidden size of self encoder, neighbor encoder, obstacle encoder, and attention layers to 10. Further, we replace multi-head attention with single head to reduce the number of




(a) Before navigating through obstacles (b) Navigating through obstacles (c) After navigating through goal

Fig. 6. **V-Value Maps.** (a), (b), (c) shows the moment before, during, and after the pink quadrotor flying through obstacles. In each subfigure, the left part is a top-down view of the environment, and the right part is a V-value map which is calculated given different positions. The blue sphere in (a) is the goal of other robots. The pink sphere in (c) is the goal of the pink robot.

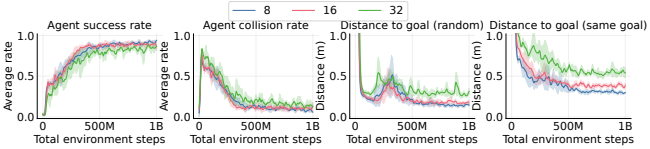



Fig. 7. Experiments with varying number of robots.

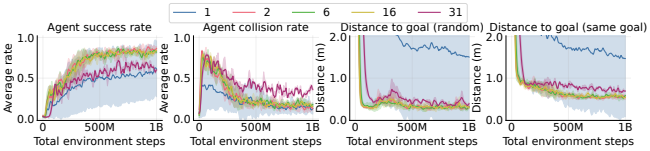



Fig. 8. Experiments with varying sensible number of neighbor robots.

operations needed. Through these operations, we end up with a model with 1820 network parameters ($7KB$ of memory), which runs at 0.35 ms on-board. To train a model that can be deployed on Crazyflie 2.1, we use two methods, training from scratch and using policy distillation [36]. We compare their performance with the same model architecture in the environment of 8 robots and 20% obstacle density, and evaluate the performance over 20 episodes. The results are listed in Table II. The model trained from scratch demonstrates superior performance in both success rate and collision rate to the model utilizes policy distillation As we prioritize on the collision avoidance capabilities, we use the model trained from scratch for physical deployment. Figure 12 shows the model used for the physical deployment has comparable collision avoidance performance to the model used in the simulation, albeit at a cost of a higher average distance to goal.

## V. Limitations and Future Work

**More complex environments**: Our method only considers static obstacles. This assumption limits the ability of control robots flying in more unstructured environments. In the future, we will add dynamic obstacles into the environment.

**Push towards onboard**: Our current localization and obstacle detection is not based on on-board sensors. In the future, we will explore utilizing on-board sensors for localization and obstacle detection. Besides, investigating the smallest model can be used for physical deployment [37] is a interesting direction.

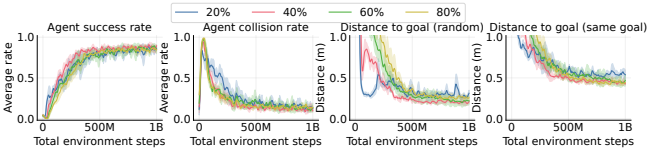



Fig. 9. Experiments with varying obstacle density.

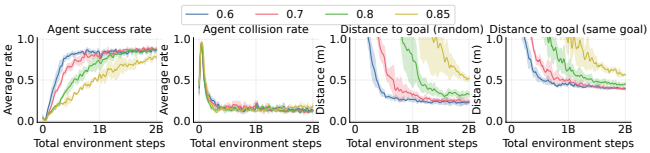



Fig. 10. Experiments with varying obstacle size.

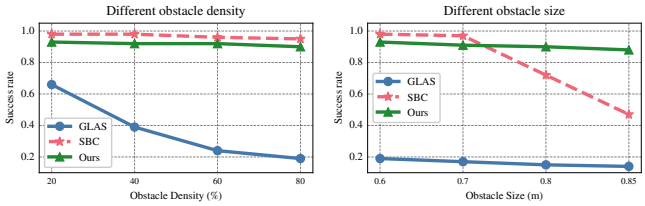
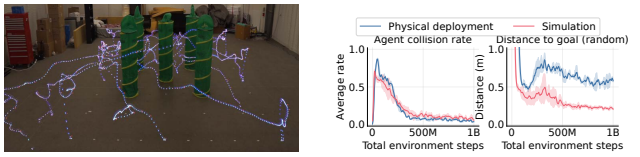



Fig. 11. Baseline comparison: different obstacle densities and obstacle sizes. Obstacle size= $0.6m$ (left). Obstacle density=80% (right).




(a) Time lapse of 8 robots flying to their goals in an area with 5 obstacles (b) Comparison between simulation and physical deployment.

Fig. 12. **Physical deployment.** The smaller model used in the physical deployment has same collision avoidance performance but with higher distance to goal compared to the simulation model.

**Lack of safety and stability guarantees**: Although our approach shows promising performance in collision avoidance and stability, it is not guaranteed. Investigating how to design hybrid methods which combine learning-based methods with classical methods that have safety and stability guarantees is an promising direction.

## VI. Conclusion

In this work, we propose an end-to-end decentralized control policy to control a robot swarm. The policy is trained with RL wherein each robot minimizes the distance to its specified goal while avoiding collisions. We demonstrate that the learned policy transfers zero-shot to the real world on the highly-constrained Crazyflie 2.1 quadrotor platform. We make three major improvements to prior work: replay mechanism, multi-head attention, sdf-based representation, resulting in high performance in task completion. Our policies scale to 32 robots, in simulation at 80% obstacle density. Our method achieves comparable collision avoidance and task completion rates to SBC with 4x faster inference speed and performs considerably better than GLAS. In future work, we plan to investigate more complex environments, explore long horizon planning problems, and make our policy safety-guaranteed.

## REFERENCES


[1] R. D'Andrea, "Guest editorial can drones deliver?" *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 3, pp. 647–648, 2014.

[2] A. V. Borkar, S. Hangal, H. Arya, A. Sinha, and L. Vachhani, "Reconfigurable formations of quadrotors on lissajous curves for surveillance applications," *European Journal of Control*, vol. 56, pp. 274–288, 2020.

[3] H. Liu, Q. Chen, N. Pan, Y. Sun, Y. An, and D. Pan, "Uav stocktaking task-planning for industrial warehouses based on the improved hybrid differential evolution algorithm," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 1, pp. 582–591, 2021.

[4] H. A. F. Almurib, P. T. Nathan, and T. N. Kumar, "Control and path planning of quadrotor aerial vehicles for search and rescue," in *SICE Annual Conference 2011*, 2011, pp. 700–705.

[5] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, online collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1047–1054, 2017.

[6] B. Şenbaşlar, W. Hönig, and N. Ayanian, "Robust trajectory execution for multi-robot teams using distributed real-time replanning," in *Distributed Autonomous Robotic Systems (DARS)*, 2019, pp. 167–181.

[7] ——, "RLSS: real-time, decentralized, cooperative, networkless multi-robot trajectory planning using linear spatial separations," *Autonomous Robots*, 2023.

[8] B. Şenbaşlar and G. S. Sukhatme, "Asynchronous real-time decentralized multi-robot trajectory planning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 9972–9979.

[9] J. Tordesillas and J. P. How, "MADER: Trajectory planner in multiagent and dynamic environments," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 463–476, 2022.

[10] K. Kondo, J. Tordesillas, R. Figueroa, J. Rached, J. Merkel, P. C. Lusk, and J. P. How, "Robust MADER: Decentralized and asynchronous multiagent trajectory planner robust to communication delay," *arXiv preprint arXiv:2209.13667*, 2022.

[11] C. Luis, M. Vukosavljev, and A. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 2020.

[12] X. Wang, L. Xi, Y. Chen, S. Lai, F. Lin, and B. M. Chen, "Decentralized mpc-based trajectory generation for multiple quadrotors in cluttered environments," *Guidance, Navigation and Control*, vol. 01, no. 02, p. 2150007, 2021.

[13] J. Park and H. J. Kim, "Online trajectory planning for multiple quadrotors in dynamic environments using relative safe flight corridor," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 659–666, 2021.

[14] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 661–674, 2017.

[15] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed Autonomous Robotic Systems: The 10th International Symposium*, 2013, pp. 203–216.

[16] S. Batra, Z. Huang, A. Petrenko, T. Kumar, A. Molchanov, and G. S. Sukhatme, "Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 576–586.

[17] M. Jiang, E. Grefenstette, and T. Rocktäschel, "Prioritized level replay," in *International Conference on Machine Learning*. PMLR, 2021, pp. 4940–4950.

[18] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu *et al.*, "Swarm of micro flying robots in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm5954, 2022.

[19] J. Yu and S. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 27, no. 1, 2013, pp. 1443–1449.

[20] J. Hopcroft, J. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem"," *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, 1984.

[21] B. Riviere, W. Hönig, Y. Yue, and S.-J. Chung, "Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 2020.

[22] S. Feng, B. Sebastian, and P. Ben-Tzvi, "A collision avoidance method based on deep reinforcement learning," *Robotics*, vol. 10, no. 2, p. 73, 2021.

[23] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 285–292.

[24] Y. Cui, L. Lin, X. Huang, D. Zhang, Y. Wang, W. Jing, J. Chen, R. Xiong, and Y. Wang, "Learning observation-based certifiable safe policy for decentralized multi-robot navigation," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 5518–5524.

[25] H. Hua and Y. Fang, "A novel learning-based trajectory generation strategy for a quadrotor," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2022.

[26] C. Yan, C. Wang, X. Xiang, K. H. Low, X. Wang, X. Xu, and L. Shen, "Collision-avoiding flocking with multiple fixed-wing uavs in obstacle-cluttered environments: A task-specific curriculum-based madrl approach," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2023.

[27] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 10 504–10 510.

[28] Z. Huang, S. Batra, T. Chen, R. Krupani, T. Kumar, A. Molchanov, A. Petrenko, J. A. Preiss, Z. Yang, and G. S. Sukhatme, "Quadswarm: A modular multi-quadrotor simulator for deep reinforcement learning with direct thrust control," *arXiv preprint arXiv:2306.09537*, 2023.

[29] 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)*. IEEE, 2019, pp. 59–66.

[30] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2520–2525.

[31] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 5420–5425.

[32] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance transforms of sampled functions," *Theory of computing*, vol. 8, no. 1, pp. 415–428, 2012.

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

[34] 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 *International Conference on Machine Learning*. PMLR, 2020, pp. 7652–7662.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[36] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," *arXiv preprint arXiv:1511.06295*, 2015.

[37] S. Hegde, Z. Huang, and G. S. Sukhatme, "Hyperppo: A scalable method for finding small policies for robotic control," *arXiv preprint arXiv:2309.16663*, 2023.