LEARN: Learning End-to-End Aerial Resource-Constrained Multi-Robot Navigation

Darren Chiu*, Zhehui Huang*, Ruohai Ge, and Gaurav S. Sukhatme

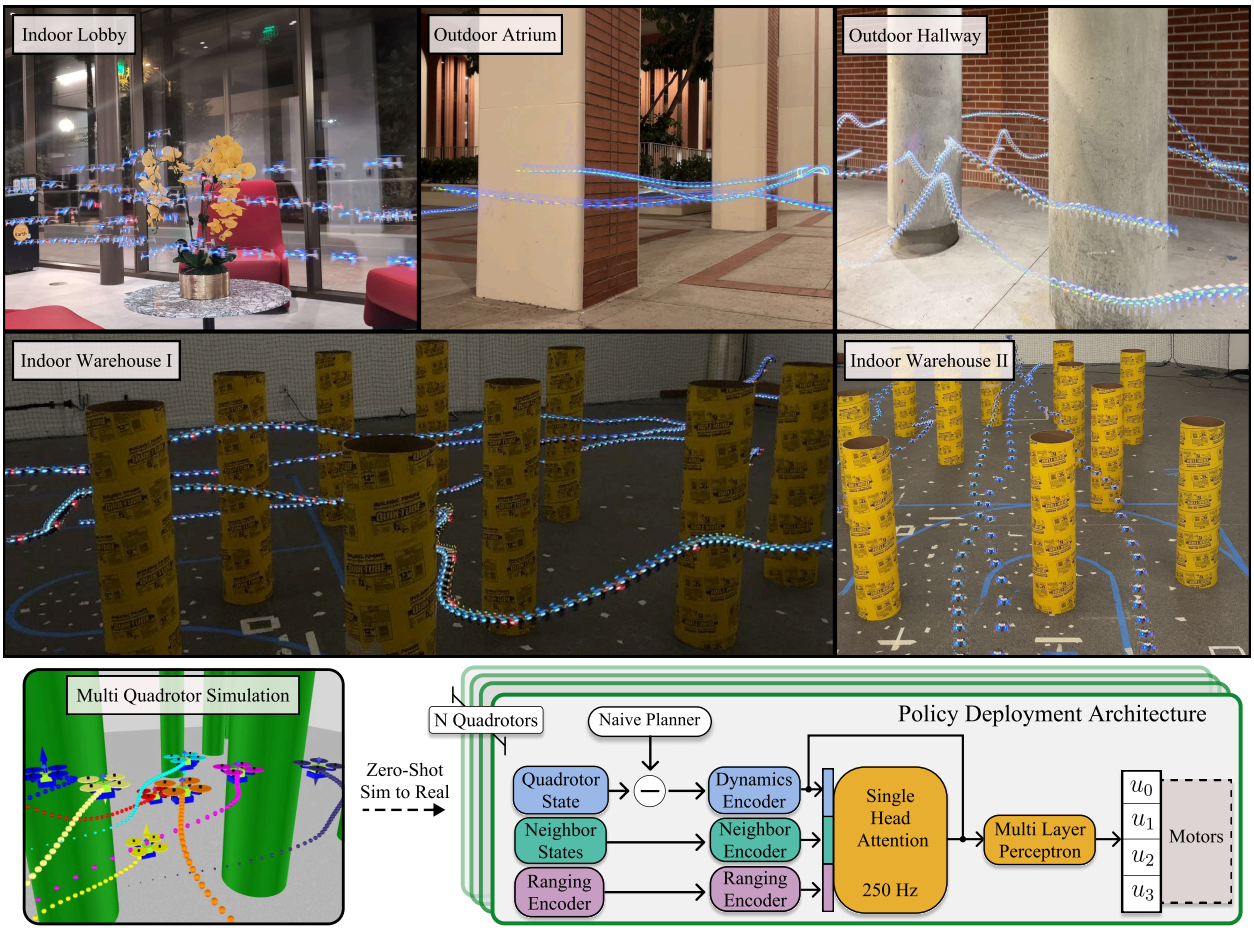

Fig. 1: LEARN is a lightweight, two-stage safety-guided reinforcement learning framework for multi-UAV navigation in cluttered indoor and outdoor spaces. All processes, including perception, localization, communication, planning, and control, run purely on an embedded single-core controller running at 168 MHz with 192 KB of RAM. A single policy is trained in simulation and duplicated across all quadrotors. During deployment, a minimum snap naive planner produces goal points for the encoder. Quadrotors obtain the two closest neighbor positions and velocities through radio; and obstacles are sensed using a low dimensional time-of-flight sensor. The policy generates individual normalized rotor thrusts that are sent directly to the motors. LEARN is zero-shot transferable to the real world with no fine-tuning. Experiments show that it scales up to 6 quadrotors in the real world and 24 in simulation.

***Abstract*—Nano-UAV teams offer great agility yet face severe navigation challenges due to constrained onboard sensing, communication, and computation. Existing approaches rely on high-resolution vision or compute-intensive planners, rendering them infeasible for these platforms. We introduce LEARN, a lightweight, two-stage safety-guided reinforcement learning (RL) framework for multi-UAV navigation in cluttered spaces. Our system combines low-resolution Time-of-Flight (ToF) sensors and a simple motion planner with a compact, attention-based RL policy. In simulation, LEARN outperforms two state-of-the-art planners by 10% while using substantially fewer resources. We demonstrate LEARN's viability on six Crazyflie quadrotors, achieving fully onboard flight in diverse indoor and outdoor environments at speeds up to $2.0m/s$ and traversing $0.2m$ gaps.**

* Denotes equal contribution. {*chiudarr, zhehuihu*}*@usc.edu*. All authors are with the University of Southern California.

TABLE I: Comparison of recent multi-robot collision avoidance and navigation methods

Year	Method	Decentr.	Async	Realistic Observation	Onboard			Resource-constrained		Learning-based	Collision avoidance in
					Localization	Perception	Computation	Yes/No?	Platform		
2018	Hönig et al. [8]	✗	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
2019	RSFC [9]	✗	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	SGBA [10]	✓	✓	✓	✓	✓	✓	✓	Crazyflie	✗	Control
2020	DMPC [11]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	Dec-RSFC [12]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	GLAS [13]	✓	✓	✗	✗	✗	✓	✓	Crazyflie	✓	Control
2021	PCAS [14]	✗	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Control
	DPDS [15]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Control
	Zhu et al. [16]	✓	✓	✗	✗	✗	✗	✗	Parrot Bebop 2	✓	Control
	EGO-Swarm [17]	✓	✓	✓	✓	✓	✓	✗	NVIDIA Xavier NX*	✗	Planning
2022	EGO-Swarm2 [4]	✓	✓	✓	✓	✓	✓	✗	NVIDIA Xavier NX*	✗	Planning
	EDG-Team [18]	✓|✗[1]	✓|✗[1]	✓	✓	✓	✓	✗	NVIDIA Xavier NX*	✗	Planning
	LSC [19]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	Ryou et al. [20]	✗	✗	✗	✗	✗	✗	✗	NVIDIA Jetson TX2*	✗	Planning
2023	LSC-DR [21]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	DLSC [22]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	RLSS [23]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	RMADER [24]	✓	✓	✗	✗	✗	✓	✗	Intel NUC10*	✗	Planning
	IMPC-OB [25]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	AMSwarm [26]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
2024	AMSwarmX [27]	✓	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	Safaoui et al. [28]	✗	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Control
	SOGM [29]	✓	✓	✗	✗	✗	✓	✗	NVIDIA Xavier NX*	✗	Planning
	DREAM [7]	✓	✓	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	HDSM [6]	✓	✗	✓	✗	✗	✗	✓	Crazyflie	✗	Planning
	CAN [30]	✓	✓	✓	✗	✗	✓	✓	Crazyflie	✓	Control
	Dec-NMPC [31]	✓	✓	✗	✗	✗	✓	✗	Qualcomm VOXL2*	✗	Control
2025	Pan et al. [32]	✗	✗	✗	✗	✗	✗	✓	Crazyflie	✗	Planning
	LEARN (Ours)	✓	✓	✓	✓	✓	✓	✓	Crazyflie	✓	Control

[1] EDG-Team switches to a centralized and synchronous planner in dense environments [6].
* The platform is customized.

***Index Terms*—Multi-Robot, Aerial Systems, Reinforcement Learning, Navigation, Collision Avoidance**

I. Introduction

Unmanned aerial vehicles (UAVs) are increasingly used in domains such as surveillance [1], search and rescue [2], and planetary exploration [3]. The physics of flight impose stringent size, weight, and power (SWaP) constraints on these platforms, making efficient system design paramount. While autonomy in UAVs has advanced significantly, many state-of-the-art navigation approaches fail to scale to resource-constrained platforms. These approaches commonly deploy high-dimensional vision-based inputs using depth cameras [4] or LiDAR sensors [5]. Furthermore, these methods assume access to ample computational resources and sufficient memory for map-building. When scaling to multiple quadrotors, a high-bandwidth communication layer is usually needed to share state information. As such, sufficient computation, memory, and communication are key assumptions that many modern decentralized multi-quadrotor navigation approaches leverage [4], [6]–[8].

However, such assumptions do not hold for resource-constrained systems. In this work, we push these limits by imposing the following hardware requirements: *(i)* single-core processors operating below 1 GHz clock speed; *(ii)* onboard memory of less than 1 MB; and *(iii)* wireless communication bandwidth below 10 MHz. As UAV platforms scale down in size, these constraints become more restrictive, precluding the use of high-power sensors and computationally demanding models. These limitations are further exacerbated in multi-robot systems, where real-time operation and decentralized decision-making must occur without reliance on a global map or reliable communication layer. Deep reinforcement learning (DRL) has emerged as a promising alternative, offering the ability to learn complex behaviors that generalize beyond predefined maps. However, its application to resource-constrained robotics remains largely unexplored given assumptions for real-world deployment.

We address two fundamental challenges that hinder the deployment of learning enabled resource-constrained robotic systems:

a) Learning End-to-end Control on Constrained Hardware: Modern deep learning models typically exceed the onboard memory capacity of nano-UAVs, preventing direct deployment. Additionally, the lack of parallel processing significantly slows inference, limiting control update rates and reducing flight stability. Prior work has shown that integrating classical planning and control techniques can improve robustness and agility in UAVs [33]. Building on this insight, we introduce a hybrid framework that incorporates planning, safety-based control, and learning to achieve highly runtime efficient decentralized navigation while maintaining a minimal computational footprint.

b) Real World Generalization: Many existing solutions assume perfect knowledge of the world [8], access to maps [13], or high-dimensional inputs [4]. In contrast, we develop a trajectory-agnostic framework that uses only low-resolution time-of-flight depth sensors, assumes no global map, and is robust to noisy observations and communication delays. Our design prioritizes simplicity, robustness, and generalization; a minimal solution for real-time multi-quadrotor navigation under extreme hardware limitations.

We present an end-to-end learned system for a swarm of nano-UAVs that can track trajectories in densely cluttered environments while avoiding collisions with obstacles and neighboring quadrotors. Our approach is designed for challenging indoor and outdoor environments where no global map or centralized planner is available. The proposed system scales to 24 quadrotors in simulation and can successfully navigate through gaps merely $0.1m$ larger than the quadrotor body. Without retraining, it achieves real world trajectory tracking with flight speeds of up to $2.0m/s$ and a mean positional error of less than 0.1 meters. Our main contributions are as follows:

1) We present LEARN, the first fully onboard, reinforcement learning-based navigation system for nano-UAV swarms in densely cluttered environments under stringent resource constraints.
2) We enhance robustness by integrating a two stage control barrier function reward into training and learn a trajectory aware policy with minimal sensing.
3) We evaluate LEARN extensively in simulation against two state-of-the-art methods, achieving 10% higher success rates with dramatically lower compute and sensing usage. We show LEARN is zero-shot deployable in indoor and outdoor environments with fully onboard deployment using the Crazyflie 2.1.

II. Related Work

In this section, we discuss existing works in three key areas: multi-robot collision avoidance and navigation, learning-based safe control, and fully onboard methods for resource-constrained robotic platforms. Table I presents a curated list of multi-robot navigation and collision-avoidance approaches that have been demonstrated in real-world settings. From this collection, we identify those methods that employ realistic obstacle perception and adopt them as our baselines. Figure 2 presents a directed graph capturing the performance relationships among these baselines as well as the additional works they compare against.

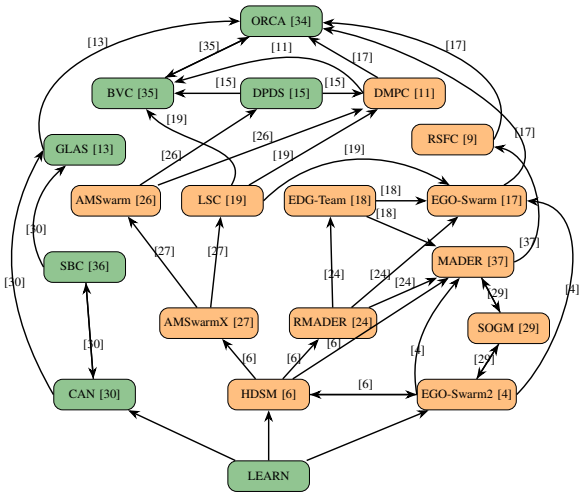

Fig. 2: **Comparison of multi-robot navigation algorithms.** We begin by reviewing studies that consider realistic obstacle perception and construct a comparison graph grounded in the algorithms examined across these works. A directed edge indicates that the source algorithm outperformed the destination algorithm in certain experiments reported in the cited paper. ■ denotes control-based methods, and ■ denotes planning-based methods. We omit SGBA [10] from the comparison since its objective is limited to collision avoidance and exploration, without addressing goal-reaching.

A. Multi-robot Collision Avoidance and Navigation

Existing multi-robot collision avoidance and navigation methods often incorporate simplifying assumptions, limiting their applicability for practical deployments, particularly on resource-constrained robotic platforms. Key restrictive assumptions prevalent in the literature include:

1) Ideal localization: neglect localization error and usually rely on external motion capture systems during deployment.
2) Unrealistic obstacle observations: assume *a priori* known environments with reliance on external motion capture systems for guidance.
3) Ideal communication: assumes no communication delays and usage of external motion capture systems to relay information.
4) Computation-rich hardware: assume computation resources are ample and usually rely on powerful onboard hardware or ground station during deployment.

Many multi-robot collision-avoidance and navigation methods are designed assuming abundant onboard resources. When deployed on platforms with limited sensing, computation, or communication capabilities, these methods typically rely on external systems, such as motion-capture setups or ground stations, to compensate.

These assumptions significantly hinder the feasibility of deployment on fully onboard, resource-constrained robots.

The first family of methods [6]–[8], [11], [15], [16], [21]–[28], [32] are comprehensively based on external systems for precise location, obstacle tracking or precomputed maps, and powerful hardware on board or ground stations for computation and communication. Although most approaches assume ideal communication, some explicitly incorporate realistic conditions such as communication delay [6], [7], [24]. In addition, although HDSM [6] relies on motion-capture systems for obstacle data during deployment, they present simulation results using a 360° LIDAR.

The second family of methods [9], [14], [20], [29], [31] still relies on external systems for high-precision localization, obstacle tracking, and precomputed maps, as well as offboard computational resources, but removes dependence on ground stations for communication. Within this group, some approaches eliminate runtime communication by precomputing trajectories for known environments [8], [9], [14], [20], while others support inter-robot state exchange over wireless networks [29], [31].

The third family of methods [13], [30] similarly relies on external motion-capture systems for robot and obstacle localization and on ground stations for communication, yet achieves computational efficiency suitable for resource-constrained deployment. GLAS [13] employs deep imitation learning with a decentralized policy comprising five multilayer perceptron (MLP) layers trained from a global planner and augmented by a safety filter to enhance collision avoidance; it runs at 40 Hz on the Crazyflie. [30] introduce an end-to-end deep reinforcement learning approach using a compact, attention-based model that maps raw observations directly to rotor thrust commands, achieving execution rates of $1kHz$ on the Crazyflie.

The fourth family of methods [4], [17], [18] eliminates reliance on external systems by using onboard sensors for robot and obstacle localization and onboard hardware for computation. Nonetheless, these approaches still require high-performance processors, high-bandwidth peer-to-peer communication networks, and sufficient memory for map storage. For example, Ego-Swarm [17], Ego-Swarm2 [4], and EDG-Team [18] leverage stereo grayscale and depth images from an Intel RealSense D430, running on an NVIDIA Xavier NX (six-core CPU, 384-core GPU, 8 GB RAM) for real-time mapping and collision avoidance.

The fifth family, exemplified by SGBA [10], removes the need for powerful onboard hardware, making it viable for fully resource-constrained platforms. SGBA uses a single VL53L1x ToF sensor and grayscale camera for localization, four additional VL53L1x sensors for omnidirectional obstacle detection, a Nordic nRF51822 radio (2 Mbps), and a 168 MHz single-core CPU with 192 KB of RAM. However, its narrow 27° field of view and simple rule-based controller are ill-suited for densely cluttered settings and goal achieving.

Inspired by SGBA's minimal-resource design [10] and the high-frequency execution of Huang et al. [30], we propose an end-to-end, decentralized deep reinforcement learning framework that empowers multiple nano-scale robots to navigate robustly in densely cluttered, unstructured environments using only onboard sensing, computation, and communication.

B. Learning With Safe Control

Traditional control theory relies on an explicit dynamics model to design controllers that guarantee stability and safety under known operating conditions. In contrast, reinforcement learning follows a data-driven paradigm, trading formal guarantees for adaptability to complex, uncertain environments [38]. Hybrid "safe-learning" approaches aim to combine these paradigms: they enforce provable safety constraints via model-based controllers or safety filters, while using learned policies to optimize performance and handle unmodeled dynamics.

We categorize existing safe-learning methods into four families, based on how safety is encoded:

The first family of methods uses learning-based policies to generate control actions and then applies existing safety filters (i.e., control barrier functions, or CBFs) to enforce theoretical guarantees [13], [39]–[41]. However, existing non-adaptive CBFs are often overly conservative [41], [42]. Hence, the second family of methods learns the safety filter itself to reduce conservatism [41], [43], [44]. Both the first and second families, however, require the safety filter to run at a high enough frequency to maintain their guarantees, contributing to the computational overhead. Moreover, they typically assume full knowledge of obstacle positions and shapes, which is an unrealistic requirement for onboard-only deployment. To address this, the third family of methods abandons explicit safety filters and instead trains a value network to predict state safety [45]. If the network determines the current state is safe, the original policy proceeds. Otherwise, a backup policy is activated. The fourth family retains the concept of safety filters but uses them only during training to shape the reward, rather than to correct actions at runtime [46]–[48]. Krasowski et al. [46] uses a constant penalty when the safety filter is activated. Wang et al. [47] and Bejarano et al. [48] scale the penalty by the magnitude of the proposed correction.

Our framework belongs to the fourth family. Unlike prior work, which assumes a single unified reward func-

TABLE II: Notation used in this paper

Notation	Definition
$*_i$	Corresponding state of the $i'th$ quadrotor
$\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{a}, \boldsymbol{j}$	Position, velocity, acceleration, and jerk. $\mathbb{R}^3$
$\boldsymbol{R}$	Rotation matrix $\in SO(3)$
ψ	quadrotor yaw
$\boldsymbol{w}, \dot{\boldsymbol{w}}$	angular velocity, angular acceleration
u_{policy}	The control action (thrusts) generated from the policy. $\mathbb{R}^4$
u_{safe}	Safe control (thrusts) generated from a Barrier Function $\mathbb{R}^4$
$\mathcal{X}_i, \mathcal{X}_{goal,i}$	Quadrotor state vector, evaluation of trajectory: $[\boldsymbol{x}\ \mathbf{v}\ \boldsymbol{\omega},\ \boldsymbol{R}] \in \mathbb{R}^{18}$.
$h_{n,j}^{quadrotor}(x,v)$	Constraint for $i'th$ quadrotor to $j'th$ quadrotor
$h_{n,k}^{obstacle}(x,v)$	Constraint for $i'th$ quadrotor to $k'th$ obstacle
π_θ	Deep neural network π with weights θ.
$\boldsymbol{e}_i^t$	Relative position input seen by policy: $\boldsymbol{x}_i^T - \boldsymbol{x}_{goal,i}^T$
$\boldsymbol{\eta}_i^t$	Relative neighbor position seen by policy: $\boldsymbol{x}_i^T - \boldsymbol{x}_j^T$
$\boldsymbol{\zeta}_i^t$	Obstacle ranging from multi zone time-of-flight. $\zeta \in [0.0, 2.0]m$. $\mathbb{R}^{4\times 8}$
$FC(x,y)$	A fully connected network with inputs x and output y.

tion, we draw inspiration from the third family to design a two-stage reward:

- *Nominal stage:* If no safety filter is triggered, the reward balances goal progression, collision avoidance, stability, energy efficiency, and smoothness.
- *Safety stage:* Once the filter triggers, we mask energy and smoothness rewards to encourage maneuvers that restore safety.

C. Resource Constrained Robots

Despite significant advances in single and multi-quadrotor navigation regardless of planning, learning, and safety-based methods, deploying these techniques on severely resource-constrained nano platforms in cluttered environments remains an open challenge. Resource-constrained robots, limited by sensing range, battery capacity, computational power, and communication bandwidth, struggle to achieve reliable autonomy. A few studies have demonstrated fully onboard solutions for tasks such as source seeking [49] and rudimentary navigation [10], but they typically assume sparsely featured or uncluttered settings.

To the best of our knowledge, our work is the first to address fully onboard perception and control for robust navigation in densely cluttered, unstructured environments using severely resource-constrained robots. We present a lightweight deep reinforcement learning framework that seamlessly integrates safety-guided learning, decentralized multi-quadrotor coordination, and onboard perception, enabling efficient and robust autonomy on nano-scale platforms.

III. Preliminaries and Background

A. Notation

In this paper, vectors are denoted by bold letters, such as $\boldsymbol{x}$, which describes the robot's state, and their time-derivatives by $\dot{\boldsymbol{x}}$. We consider the following quadrotor dynamics model:

$$\boldsymbol{a} = \boldsymbol{g} + \tfrac{1}{m}\,\boldsymbol{R}\,\boldsymbol{f}_{\text{thrust}} - \tfrac{1}{m}\,\boldsymbol{f}_{\text{drag}}, \tag{1}$$

$$\dot{\boldsymbol{\omega}} = \boldsymbol{I}^{-1}\big(\boldsymbol{\tau} - \boldsymbol{\omega} \times (\boldsymbol{I}\,\boldsymbol{\omega})\big), \tag{2}$$

$$\dot{\boldsymbol{R}} = \boldsymbol{R}\,\boldsymbol{\omega}, \tag{3}$$

where

- m and $\boldsymbol{I} \in \mathbb{R}^{3\times 3}$ are the mass and inertia matrices,
- $\boldsymbol{f}_{\text{thrust}}$ and $\boldsymbol{f}_{\text{drag}}$ are the thrust and drag force vectors,
- $\boldsymbol{\tau} \in \mathbb{R}^3$ is the total torque in the body frame, and
- $\boldsymbol{g} \in \mathbb{R}^3$ is the gravity vector.

B. Reinforcement Learning

We consider an on-policy reinforcement-learning framework based on Proximal Policy Optimization (PPO) [50]. The interaction between the agent and the environment is modeled as a finite-horizon Markov Decision Process (MDP), defined by the tuple $(\mathcal{X}, \mathcal{U}, f, r)$, where $\mathcal{X}$ and $\mathcal{U}$ denote the state and action spaces, respectively, $f(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{u}_t)$ represents the system dynamics, and $r\colon \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the scalar reward function. The policy $\pi_\theta(\boldsymbol{u}\,|\,\boldsymbol{x})$, parameterized by a deep neural network with weights θ, is optimized to maximize the expected discounted return over a horizon of length T, where $\gamma \in [0,1]$ is the discount factor:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t\, r\big(\boldsymbol{x}_t, \boldsymbol{u}_t\big)\right], \tag{4}$$

$$r(\boldsymbol{x}, \boldsymbol{u}) = r_{\text{traj}}(\boldsymbol{x}) + r_{\text{safety}}(\boldsymbol{x}, \boldsymbol{u}) + r_{\text{efficiency}}(\boldsymbol{u}) \tag{5}$$

The reward function balances trajectory-tracking objectives with safety and efficiency considerations. It comprises three components where $r_{\text{trajectory}}$ rewards accurate tracking of the reference path, r_{safety} penalizes proximity to obstacles and other quadrotors, $r_{\text{efficiency}}$ penalizes excessive control effort and enforces smoothness via higher-order derivatives of the state.

C. Control Barrier Functions

Control Barrier Functions provide a formal methodology to enforce collision constraints [51]. Given a continuously differentiable function $h\colon \mathbb{R}^n \to \mathbb{R}$ defining the safe set

$$\mathcal{S} = \{\, \boldsymbol{x} \in \mathbb{R}^n \mid h(\boldsymbol{x}) \geq 0\}, \tag{6}$$

the system remains safe so long as $h(\boldsymbol{x}) \geq 0$ at all times. Ensuring forward invariance of $\mathcal{S}$ requires

$$\dot{h}(\boldsymbol{x}, \boldsymbol{u}) + \alpha\big(h(\boldsymbol{x})\big) \geq 0, \tag{7}$$

where α is an extended class-$\mathcal{K}$ function.

To enforce safety for a given nominal controller $\boldsymbol{u}_{\text{nominal}}$, the following Quadratic Program (QP) is solved that minimally adjusts the policy control input:

$$\begin{aligned} \min_{\boldsymbol{u}_{\text{safe}}} \quad & \left\|\boldsymbol{u}_{\text{safe}} - \boldsymbol{u}_{\text{nominal}}\right\|^2 \\ \text{s.t.} \quad & \dot{h}\big(\boldsymbol{x}, \boldsymbol{u}_{\text{safe}}\big) + \alpha\big(h(\boldsymbol{x})\big) \geq 0, \\ & \boldsymbol{u}_{\text{safe}} \in \mathcal{U}. \end{aligned} \tag{8}$$

This QP guarantees that the applied input $\boldsymbol{u}_{\text{safe}}$ satisfies the safety constraint defined by h, while remaining as close as possible to the proposed control $\boldsymbol{u}_{\text{nominal}}$, which in our framework is generated by the learned policy.

We follow the formulation of the barrier certificate presented in [36], where the state of the system is a concatenated vector of all robot states defined as pairwise interactions. We consider only the 2 closest neighbors in a range of $2.0m$.

$$\begin{aligned} h_{i,j}^{quadrotor}(x, v) = & \sqrt{2(\alpha_i + \alpha_j)(||\Delta \boldsymbol{x}_{i,j}|| - D_s)} \\ & + \frac{\Delta \boldsymbol{x}_{i,j}^T}{||\Delta \boldsymbol{x}_{i,j}||} \Delta v_{i,j} \end{aligned} \tag{9}$$

Where α denotes the maximum acceleration for each quadrotor and D_s is the distance that must be enforce between a pair of quadrotors. Likewise, obstacle constraints are defined as other quadrotors (of spherical constraints) with no control authority i.e. $\alpha_j = 0$.

$$\begin{aligned} h_{i,k}^{obstacle}(x, v) = & \sqrt{2(\alpha_i)(||\Delta \boldsymbol{x}_{i,k}|| - (\frac{D_s}{2} + r_k))} \\ & + \frac{\Delta \boldsymbol{x}_{i,k}^T}{||\Delta \boldsymbol{x}_{i,k}||} \Delta v_{i,k} \end{aligned} \tag{10}$$

Additional constraints are also added for the room boundaries, such as walls, floors and ceiling.

IV. Problem Statement

We consider a 3D environment with N homogeneous quadrotors and an unknown number of static obstacles with arbitrary positions and dimensions. Each quadrotor has no prior map of its surroundings and relies exclusively on onboard resources: power-constrained sensors for localization and perception, single-core, sub-gigahertz processors with less than one megabyte of memory for computation, and peer-to-peer communication limited to megabit-per-second bandwidth. Under these stringent hardware constraints, our objective is to design a decentralized navigation scheme that enables all N quadrotors to reach their designated goals within a specified time limit while avoiding collisions with both obstacles and other quadrotors.

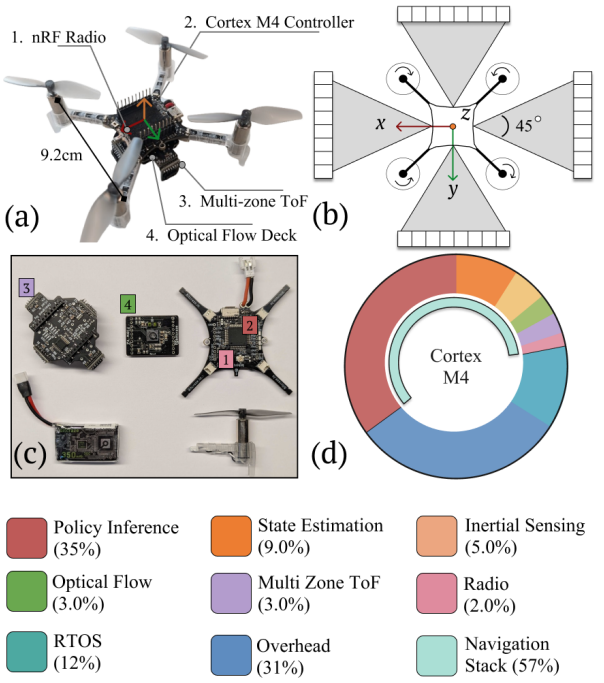

Fig. 3: **Hardware System.** The Crazyflie platform used in the real world (a) and in simulation. The quadrotor is equipped with a set of 4 VL53L5CX sensors that each provide an 8×8 depth image (b). Each quadrotor is $9.2cm$ in size and weighs merely $47g$. We utilize the onboard nRF51822 radio to communicate neighbor positions and velocities. (c) shows the individual components where number and color denotes the corresponding compute component and (d) the breakdown of compute usage.

V. Method

The navigation problem described in Section IV becomes increasingly intractable as the number of quadrotors, obstacles, and planning horizon grows. Solving it using only onboard, resource-constrained hardware introduces several challenges: i) Perception and localization uncertainty due to sensor noise, limited field of view, and environmental uncertainties; ii) Limited data exchange and communication delay arising from bandwidth constraints; iii) Trade-off between optimality and solution times owing to limited onboard computational resources. To overcome these challenges, this section introduces the hardware design considerations and then proceeds to the RL framework design. We believe for reliable real world results, both hardware and algorithmic considerations must be made.

A. Hardware Design

We use the Bitcraze Crazyflie 2.1, a widely adopted commercial off-the-shelf (COTS) nano-UAV platform

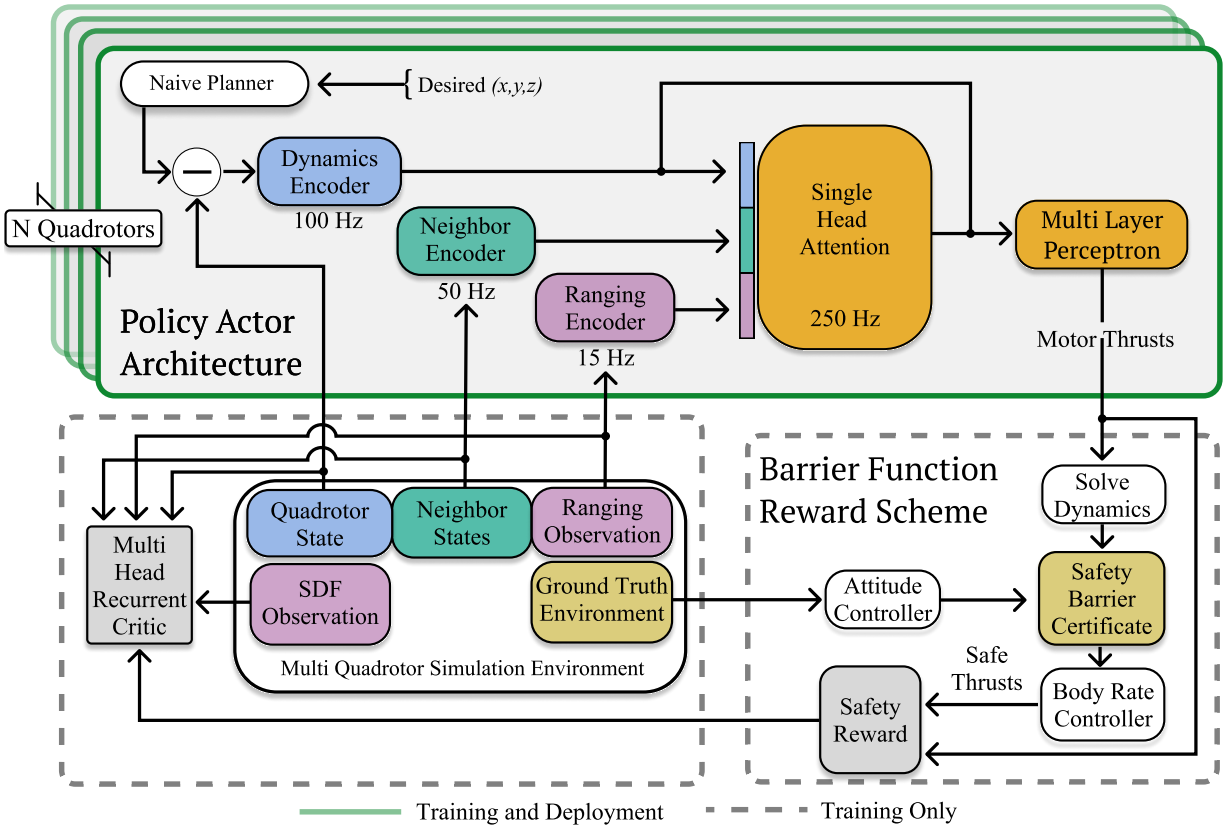

Fig. 4: **Method Overview.** The training framework incorporates a two stage safety based reward function using Safety Barrier Certificates [36]. An asynchronous actor critic architecture is used where the critic observes a signed distance field (SDF) and employs a recurrent multi-headed attention architecture. At the beginning of each episode we generate a minimum snap trajectory. The trajectory is evaluated at each controller step to generate a 13 dimension goal point which is subtracted from the current state. The green denotes what is deployed on hardware and gray for purposes of training only. The same policy is both trained and deployed across all quadrotors.

for our experiments. Onboard computation is performed by an STM32F405 ARM Cortex-M4 microcontroller running at 168 MHz, with 192 KB of RAM and 1 MB of flash. For perception, we employ the custom expansion board from Niculescu *et al.* [52], which carries four VL53L5CX multi-zone ToF sensors. Each VL53L5CX delivers either a 4×4 depth map at $30Hz$ or an 8×8 depth map at $15Hz$. For localization, we use the Optical Flow Deck v2. Velocity and altitude measurements are fused via an extended kalman filter to produce localization estimates for each quadrotor. Figure 3(b) shows the field of view of 4 VL53L5CX sensors.

B. Algorithm Overview

We propose a reinforcement learning (RL)-based framework that integrates trajectory planning and safety-based control to yield a fully learned, end-to-end navigation policy. The policy takes as observation the quadrotor's states, ranging data, and neighboring positions to output normalized thrusts. Unlike conventional learning-based methods that directly utilize the final goal to form observations [13], [30], our approach employs a naive trajectory planner to generate smooth goal points at each timestep. These intermediate goals then serve as input for constructing the observation space. The key idea is that the trajectory generation does not account for obstacles and other quadrotors, instead, the policy is trained to modify the controls online to account for naivety. During training, we leverage privileged information for two components: a recurrent multi-head attention critic and a safety filter based on control barrier functions [36]. The safety filter is activated through a two stage training process. When training from scratch, the filter provides no useful guidance for learning how to fly. Therefore, the barrier function based reward is applied only in the latter stages of training to improve collision performance. In the following section, we discuss scenario design (Section V-C), trajectory generation (Section V-D), and the reinforcement learning framework (Section V-E).

C. Scenario design

The training environment is configured as an $8\,\mathrm{m} \times 8\,\mathrm{m} \times 3\,\mathrm{m}$ room containing multiple static obstacles extending from the floor to the ceiling. At the start of each episode, a central obstacle region measuring either $6\,\mathrm{m} \times 4\,\mathrm{m}$ or $4\,\mathrm{m} \times 6\,\mathrm{m}$ is selected with equal probability. This region is discretized into 24 square grid cells, each with an area of $1\,\mathrm{m}^2$. Cylindrical obstacles with diameters randomly chosen between $0.2\,\mathrm{m}$ and $0.85\,\mathrm{m}$ are placed within these cells. Each obstacle is positioned so that any point on it is at least $0.075\,\mathrm{m}$ away from the cell boundaries, ensuring a minimum clearance of $0.15\,\mathrm{m}$ between obstacles and resulting in at least $0.05\,\mathrm{m}$ of traversable free space for the quadrotors. The obstacle density, defined as the fraction of occupied grid cells, is randomly selected between 0.1 and 1.0 for each episode. Quadrotors are initialized at random positions along one $4\,\mathrm{m}$-side of the obstacle region, with their goal positions randomly assigned along the opposite side. Two goal assignment strategies are used: (i) all quadrotors share the same goal position, or (ii) each quadrotor is assigned a unique goal position.

During training, we choose not to terminate episodes upon collisions. We believe that the collision interaction is important for learning navigation behaviors, especially with a safety filter based guidance. Therefore, collisions between quadrotors and obstacles are explicitly simulated with randomized interactions. When two quadrotors collide, each quadrotor's velocity direction is reversed relative to their mutual positions, and random noise is added to capture realistic collision dynamics. The speeds of both quadrotors are then reduced by $20 - 80\%$ of their pre-collision values. Additionally, each quadrotor's angular velocity vector is reassigned: a random direction is sampled uniformly in 3D, and the magnitude is sampled uniformly, $\|\omega_i\| \sim \mathcal{U}(10\pi, 20\pi)$. For quadrotor to obstacle collisions, we reverse the velocity direction with respect to the obstacle center. In this case, the new angular velocity direction is also random in 3D, with the magnitude sampled as $\|\omega_i\| \sim \mathcal{U}(\frac{\pi}{2}, \pi)$ radians per second to balance the performance between passing through narrow gaps and collision avoidance.

D. Trajectory Generation

We adopt the minimum-snap trajectory generation method introduced by Mellinger et al. [53] for trajectory generation. At the start of each episode, we randomly sample initial and final positions for all N quadrotors and compute a naive, minimum-distance straight–line trajectory, ignoring obstacles. Each trajectory is then cast as a piecewise polynomial under the minimum-snap formulation. At every timestep, we evaluate these polynomials to extract the current goal state, $\mathcal{X}_{goal} :=$ $[\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{\omega}, \boldsymbol{R}]$, which indicates position, velocity, angular velocity, and rotation matrix.

E. Reinforcement Learning Framework

In this section, we discuss i) The observation and action space (Section V-E1), ii) reward function (Section V-E2), and iii) model architecture (Section V-E2d),

1) Observation and Action Space: We train our policy using PPO [50], with asynchronous network architectures between the actor and critic. At each timestep t, quadrotor i receives observation $\boldsymbol{o}_{i,a}^t = (\boldsymbol{e}_i^t, \boldsymbol{\eta}_i^t, \boldsymbol{\zeta}_i^t)$, while the critic receives: $\boldsymbol{o}_{i,c}^t = (\boldsymbol{e}_i^t, \boldsymbol{\eta}_i^t, \boldsymbol{\xi}_i^t)$. Here, $\boldsymbol{e}_i^t$ denotes the quadrotor's observation of its own state and goal, $\boldsymbol{\eta}_i^t$ represents observations of neighboring quadrotors, and $\boldsymbol{\zeta}_i^t$ (actor, time of flight) and $\boldsymbol{\xi}_i^t$ (critic, signed distance field) denote observations of obstacles.

a) Self and Goal Observation: The quadrotor's own state and goal observation is defined as $\mathbf{e}_i^t = (\mathbf{r}\mathbf{x}_i^t, \mathbf{r}\mathbf{v}_i^t, \mathbf{r}\mathbf{R}_i^t, \boldsymbol{r\omega}_i^t, \mathbf{v}_i^t, \boldsymbol{\omega}_i^t)$ where $\mathbf{r}\mathbf{x}_i^t \in \mathbb{R}^3$ is the position of the quadrotor relative to its goal, $\mathbf{r}\mathbf{v}_i^t \in \mathbb{R}^3$ is the quadrotor's linear velocity relative to the goal's velocity, $\mathbf{r}\mathbf{R}_i^t \in \mathrm{SO}(3)$ is the quadrotor's orientation relative to the goal's orientation, $\boldsymbol{r\omega}_i^t \in \mathbb{R}^3$ is the angular velocity of the quadrotor relative to the goal's angular velocity, $\mathbf{v}_i^t \in \mathbb{R}^3$ is the quadrotor's linear velocity, and $\boldsymbol{\omega}_i^t \in \mathbb{R}^3$ is the quadrotor's angular velocity.

b) Neighbor Observations: The observation of neighboring quadrotors is given by $\eta_i^t = (\tilde{\boldsymbol{x}}_{i,1}^t, \ldots, \tilde{\boldsymbol{x}}_{i,K}^t, \tilde{\boldsymbol{v}}_{i,1}^t, \ldots, \tilde{\boldsymbol{v}}_{i,K}^t)$, where $\tilde{\boldsymbol{x}}_{i,j}^t \in \mathbb{R}^3$ and $\tilde{\boldsymbol{v}}_{ij}^t \in \mathbb{R}^3$ denote, respectively, the position and velocity of the j-th nearest neighbor relative to quadrotor i. Where, $K \leq N - 1$ is the number of neighbors with which the quadrotor can communicate.

c) Obstacle Observations: For the actor, obstacle observations $\boldsymbol{\zeta}_i^t$ are constructed from hardware measurements. Each quadrotor is equipped with four ToF sensors, each producing an 8×8 depth map with a 45° field of view. As all obstacles are static and extend from floor to ceiling, we condense each 8×8 map along the vertical axis by taking the middle row across the z-dimension, resulting in a 4×8 representation across all sensors. For the critic, we follow the approach in [30]: $\boldsymbol{\xi}_i^t$ is a 3×3 grid of minimum distances to the nearest obstacles, scaled and discretized to a pre-defined resolution ($0.1m$ in our experiments). This formulation presents a permutation invariant critic with privileged environment information, helping the policy reason against moving in directions of sensor blind spots.

d) Action Space: The policy outputs an action $\boldsymbol{u}_i^t \in [0, 1]^4$ for quadrotor i at time t, specifying the normalized thrust levels for each of the four rotors. These are linearly mapped to physical thrust values, where 0 corresponds to zero thrust and 1 corresponds to maximum thrust.

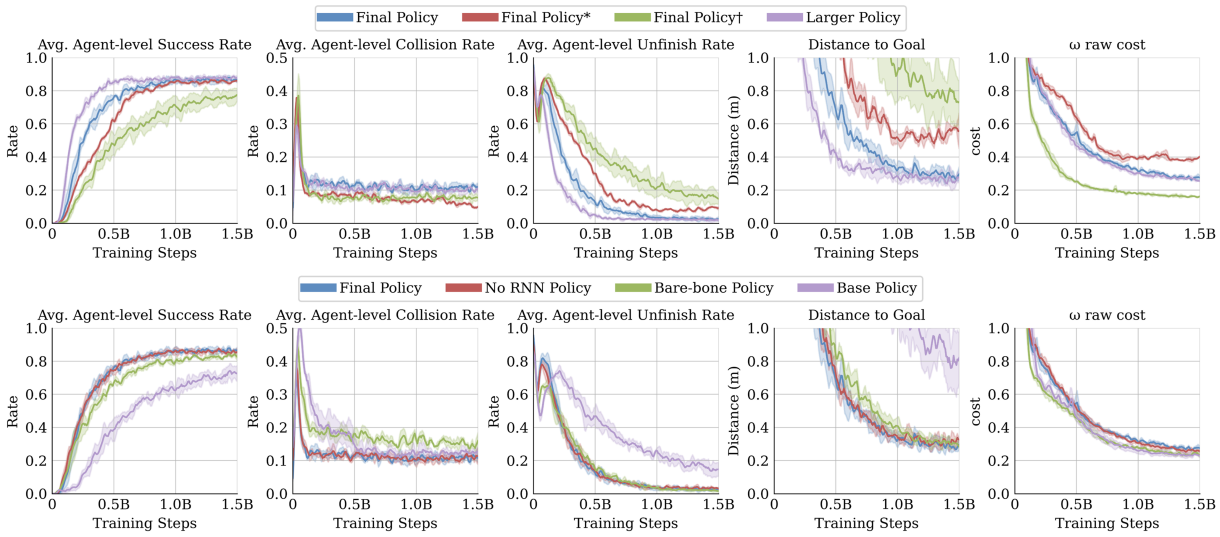

Fig. 5: **Training Curves.** PPO training curves for different model variants are shown above. The final policy shown by the blue curve is the one used for all experiments, including real world and simulation. * is the final policy trained using Population Based Training [54]. † denotes a version where the safety reward is for all steps (single stage). The larger policy employs a hidden size of 64, as opposed to 32 in main experiments. The base policy from [30] is also compared as a baseline.

2) Reward Function: The total reward is formulated as three core objectives: accurately tracking a trajectory ($r_{\text{trajectory}}$), maintaining a safe distance between objects and other quadrotors (r_{safety}), and minimizing unnecessary control efforts ($r_{\text{efficiency}}$):

$$r(\boldsymbol{x}, \boldsymbol{u}) = r_{\text{trajectory}}(\boldsymbol{x}) + r_{\text{safety}}(\boldsymbol{x}, \boldsymbol{u}) + r_{\text{efficiency}}(\boldsymbol{u})$$

a) Trajectory Tracking: The trajectory tracking reward encourages the quadrotor to track a full state trajectory ($\mathbf{x} = [\boldsymbol{x},\ \mathbf{v},\ \boldsymbol{\omega},\ \mathbf{R}]$). A hybrid position penalty is applied with the following formulation: for distances greater than $0.2m$, we apply a clipped Euclidean cost; for distances within $0.2\,\mathrm{m}$, we apply a smooth exponential decay to encourage convergence. Rotation misalignment is penalized by comparing only the relative yaw. A spin penalty is also included to suppress high angular velocities such as wobbling behavior and promotes smoother flight.

b) Safety: The safety term r_{safety} penalizes crashes, low-altitude flight, and proximity to constraint boundaries. To discourage flight in regimes where ground effect and localization errors become significant, a fixed crash penalty is applied when the quadrotor is detected to be on the floor, and a continuous penalty is added if the altitude falls below $0.2m$. We additionally incorporate safety-guided control (SBC) costs. These include a thrust disagreement penalty, which measures the deviation between the policy's output and the SBC provided thrusts, scaled by the distance to constraint boundaries (i.e. $h^{\text{quadrotor}_{n,j}}$ and $h^{\text{obstacle}_{n,k}}$. A boundary proximity cost penalizes the quadrotor when it nears the edge of the safe set given that a solution to eq 8 exists. In the case the optimization is infeasible, a no-solution penalty is triggered to penalize actions that brought the quadrotors into this configuration. To ensure that safety takes precedence in high-risk states, costs relevant to efficiency are temporarily down-weighted when SBC penalties are active. This formulation allows us to balance the explorative freedom of the RL framework with the guidance from a *when needed* safety filter.

c) Efficiency: The efficiency term $r_{\text{efficiency}}$ minimizes unnecessary control effort by penalizing the L_2 norm of the rotor thrust vector. The final reward is computed as the negative weighted sum of all cost components, scaled by the simulation timestep, $\Delta t = 0.005$.

d) Model Architecture: The full architecture is shown in Figure 4. The encoders are designed to process quadrotor dynamics, neighbor, and obstacle observations in a lightweight fashion. Each observation mode is embedded via independent and asynchronous MLPs, followed by a shared single head attention mechanism and feedforward action generation. The actor and critic networks share the overall structure but differ in the obstacle encoder where privileged critic information, a signed distance field, is used. Furthermore, the critic employs a recurrent architecture and larger multi-head attention mechanism, as opposed to the memory-less single-head attention used for the actor.

TABLE III: **Ablation Studies.** We evaluate different variations of the policy training along with ablated components. denotes performance gains from the ablation, where shows the trade-off.

Model Type	Scenario	Quadrotor-level ↑	Overall ↑	Incomplete ↓	Q–Q coll. ↓	Q–O coll. ↓	Avg. Dist. ↓	Avg. Vel. ↑
LEARN	Straight Line	0.975	0.880	0.0 ± 0.0	0.020 ± 0.069	0.020 ± 0.058	12.447 ± 0.226	0.495 ± 0.007
	Swap Goal	0.843	0.480	0.240 ± 0.431	0.100 ± 0.184	0.083 ± 0.112	13.395 ± 0.721	0.526 ± 0.024
LEARN*	Straight Line	0.622	0.061	0.939 ± 0.242	0.005 ± 0.036	0.064 ± 0.109	14.921 ± 0.442	0.587 ± 0.017
	Swap Goal	0.605	0.020	0.940 ± 0.240	0.026 ± 0.077	0.087 ± 0.105	16.889 ± 0.637	0.553 ± 0.017
LEARN†	Straight Line	0.857	0.367	0.571 ± 0.500	0.0 ± 0.0	0.025 ± 0.051	12.985 ± 0.282	0.428 ± 0.001
	Swap Goal	0.824	0.306	0.633 ± 0.487	0.010 ± 0.050	0.043 ± 0.070	13.459 ± 0.421	0.444 ± 0.015
LEARN‡	Straight Line	0.673	0.020	0.980 ± 0.143	0.005 ± 0.036	0.051 ± 0.080	12.574 ± 0.235	0.494 ± 0.008
	Swap Goal	0.610	0.040	0.960 ± 0.198	0.038 ± 0.095	0.135 ± 0.131	13.871 ± 0.566	0.445 ± 0.019
No SBC Policy[1]	Straight Line	0.806	0.327	0.061 ± 0.242	0.117 ± 0.185	0.148 ± 0.176	13.032 ± 0.730	0.489 ± 0.018
	Swap Goal	0.650	0.14	0.060 ± 0.240	0.260 ± 0.206	0.213 ± 0.181	14.879 ± 1.124	0.450 ± 0.184
No RNN Policy[2]	Straight Line	0.962	0.755	0.041 ± 0.200	0.010 ± 0.050	0.026 ± 0.057	12.267 ± 0.261	0.479 ± 0.011
	Swap Goal	0.853	0.340	0.04 ± 0.198	0.058 ± 0.111	0.118 ± 0.111	13.555 ± 0.616	0.434 ± 0.019
Bare-bone Policy[3]	Straight Line	0.969	0.755	0.102 ± 0.306	0.0 ± 0.0	0.018 ± 0.044	12.370 ± 0.147	0.477 ± 0.011
	Swap Goal	0.785	0.3	0.18 ± 0.388	0.115 ± 0.159	0.125 ± 0.145	14.028 ± 0.785	0.437 ± 0.021
Larger Policy[4]	Straight Line	0.947	0.755	0.022 ± 0.149	0.022 ± 0.071	0.047 ± 0.104	12.226 ± 0.190	0.484 ± 0.006
	Swap Goal	0.858	0.340	0.0 ± 0.0	0.035 ± 0.087	0.123 ± 0.127	13.430 ± 0.629	0.442 ± 0.019
Base Policy[5]	Straight Line	0.798	0.184	0.633 ± 0.487	0.0 ± 0.0	0.076 ± 0.075	14.027 ± 0.647	0.434 ± 0.016
	Swap Goal	0.825	0.200	0.520 ± 0.505	0.004 ± 0.033	0.094 ± 0.103	14.008 ± 0.676	0.434 ± 0.022

* Denotes final policy trained using population based training [54].
† Denotes policy trained using only single stage training.
‡ Denotes a harsh collision interaction within simulation.
[1] Denotes without SBC, with RNN Critic.
[2] Denotes without RNN in critic, with SBC.
[3] Denotes without SBC and without RNN in the critic.
[4] Denotes a hidden size of 64.
[5] Denotes the policy used in [30].

e) Sim-to-real Deployment: As we do not have access to a precise model that will distinctly capture N different quadrotors, we apply perturbations to the dynamics, observations, and actions. For deployment, this is sufficient in generalizing to a large group, without needing re-identification across specific quadrotors.

The model is deployed completely onboard the MCU that drives the Crazyflie. As the MCU only contains one core, the implementation is done *in-series* with a Real Time Operating System (RTOS). Since we train the policy with asynchronous observation updates i.e. observations come in at a different frequencies, we forward pass encoders *asynchronously* on hardware. To accomplish this, each encoder is implemented as a RTOS task with their respective frequencies, as denoted in Figure 4. In order to protect the information during encoder propagation, components of the model are wrapped in a semaphore. Obstacle observations ($\boldsymbol{\zeta}_i$) are captured using a buffering scheme to ensure data completeness. First, the raw unfiltered data is queried from the sensor and placed into a buffer at a rate of $15hz$. When new data is placed into the buffer, we clip ($[0m, 2.0m]$), transform ($\mathbb{R}^{4\times8\times8} \rightarrow \mathbb{R}^{4\times8\times1}$) and apply an exponential filter to ready the data for the obstacle encoder. It is important to note that we do not simulate ray tracing of neighboring quadrotors in training. As a result ranging in the real world is clipped to $2.0m$ if the field of view is intersecting another quadrotor. Neighbor observations ($\boldsymbol{\eta}_i$) are exchanged using the onboard radio in an asynchronous fashion. Each quadrotor broadcasts its position and velocity at $50hz$. When another quadrotor receives this data, it compares the received position with the two nearest stored positions. The two closest received positions (and their respective velocities) are used to propagate the neighbor encoder. The entire pipeline leaves a computational overhead of 30% to the microcontroller. Surrounding literature [55] points out that this is the upper limit to maintain the scheduling deadlines (i.e. encoder update rates) that we train the model on.

VI. Results

We validate our system extensively in both simulation (Section VI-B) and in real world experiments (Section VI-C), to show the performance in four main attributes: i.) navigation through dense obstacle environments in simulation and the real world ii.) how learning-based methods perform against traditional motion planning iii.) the robustness of the overall system and iv.) experimental insights in key points of the proposed framework. We present two definitions of success rate: quadrotor-level and overall. We define quadrotor level as the fraction of quadrotors who arrived within distance δ of the goal collision free. The overall success rate is defined as the fraction of entire runs where all quadrotors arrived within distance δ of the goal collision free. In both cases, we use a threshold distance of $\delta = 0.1m$. The incomplete rate is found by taking the fraction of runs where *all* quadrotors arrive within δ of the goal and were collision free. The remaining statistics (quadrotor collisions, distance, velocity, acceleration, jerk) were all found as a fraction over the number of quadrotors.

TABLE IV: **Quantitative Comparisons.** Performance comparison across methods, scenarios, and planner versions. All rates are fractions over $N = 50$ trials with 8 quadrotors; collision rates, distances, speeds are $\mu \pm \sigma$. Q-Q coll. denotes quadrotor to quadrotor collision fractions, whereas quadrotor to obstacle fractions are denoted by Q-O coll. The swap goal task becomes significantly more difficult due to intersections between quadrotors and obstacles.

Method	Scenario	Quadrotor-level ↑	Overall*↑	Incomplete*↓	Q–Q coll. ↓	Q–O coll. ↓	Avg. Dist. ↓	Avg. Vel. ↑	Avg. Acc. ↓	Avg. Jerk ↓
LEARN	*Straight Line*	0.975	0.880	0.000 ± 0.000	0.020 ± 0.069	0.020 ± 0.058	12.447 ± 0.226	0.495 ± 0.007	0.057 ± 0.011	0.126 ± 0.054
	Swap Goal	0.843	0.480	0.240 ± 0.431	0.100 ± 0.184	0.083 ± 0.112	13.395 ± 0.721	0.526 ± 0.024	0.081 ± 0.021	0.206 ± 0.132
ES2 [4]	*Straight Line*	0.950	0.780	0.180 ± 0.388	0.020 ± 0.035	0.120 ± 0.075	13.689 ± 1.330	0.924 ± 0.162	0.063 ± 0.035	0.205 ± 0.411
	Swap Goal	0.860	0.440	0.280 ± 0.453	0.042 ± 0.098	0.074 ± 0.112	13.022 ± 1.442	0.855 ± 0.204	0.083 ± 0.046	0.384 ± 0.500
HDSM [6]	*Straight Line*	0.765	0.300	0.160 ± 0.370	0.215 ± 0.195	0.027 ± 0.068	13.022 ± 0.196	0.414 ± 0.011	0.057 ± 0.010	0.271 ± 0.041
	Swap Goal	0.735	0.340	0.040 ± 0.198	0.271 ± 0.249	0.002 ± 0.016	14.921 ± 1.062	0.127 ± 0.013	0.058 ± 0.011	0.273 ± 0.044

* Denotes that the statistic was calculated across runs, whereas others were calculated across quadrotors.

We perform evaluations in two navigation scenarios: *Straight Line* and *Swap Goal* each involving 8 quadrotors and $N = 50$ randomized trials. Obstacles are modeled as cylinders with radius uniformly sampled in the $[0.35, 0.85]m$ range and a minimum inter-obstacle gap of $0.25m$. At each trial, quadrotors must either traverse a long corridor (Straight Line) or swap goals across the y-axis (Swap Goal), while avoiding both obstacles and inter-quadrotor collisions. No global map is provided and each quadrotor observes only a fixed radius around it.

A. Ablation Studies

We compare eight variants of LEARN to isolate the contributions of the safety-guided barrier framework, the privileged RNN critic, and network capacity. We also investigate the choice of single stage training and a harsher collision simulation (a wider sampling bound for collision dynamics). Table III shows that our full model combining a two stage safety reward scheme, recurrent multi-headed critic, and a compact attention network yields the best or near-best quadrotor-level and overall success rates, with minimal collisions. Removing SBC causes a dramatic rise in obstacle collisions and a drop in overall success (from 0.48 to 0.14 in the Swap Goal task), underscoring its role in guiding safe exploration. Omitting the RNN critic has a milder effect (overall success falls to 0.34), indicating that memory aids navigation but is secondary to safety guidance. The minimal bare-bones policy (no SBC, no RNN) still succeeds in straight-line tasks but largely fails under goal-swap complexity. Simply increasing model size (with a hidden size of $H = 64$) degrades performance, confirming that training strategies becomes more important than number of tunable parameters. Finally, the Base Policy, representing a version without architectural refinements [30], under-performs across the board.

B. Simulation Experiments

Our approach is compared against Ego-Swarm 2 (ES2) [4] and HDSM [6] within a simulation environment. We choose the baselines given the comparisons described by Figure 2. It is important to note that these two baselines present results in the real world, but are not deployable on a resource-constrained platform. Further, they rely on high dimensional sensing such as depth cameras [4] or LiDARs [6] to generate voxel based maps and trajectory sharing schemes. Our approach on the other hand, does not rely on map generation and uses only a lightweight communication framework.

1) Comparison against Motion Planning:

a) Quantitative Results: Table IV provides a detailed breakdown of metrics. In the *Straight Line* scenario, LEARN achieves both the highest overall success rate (0.88) and the highest quadrotor-level success rate (0.975) while completing the task fully. While ES2 performs similarly in terms of the quadrotor-level success rate, the planner fails to complete 18% of trials i.e. 18% contained at least one quadrotor not reaching the goal or colliding. In comparison to HDSM, our method reported a higher quadrotor to obstacle collision rate, but was able to trade-off by showcasing a much lower quadrotor to quadrotor collision rate. The overall distance traveled for our method is also the lowest against the two planners, supporting the previous qualitative claim on using the minimum distance planner. Our method also demonstrates the smoothest flight, given the lowest jerk.

The *Swap Goal* scenario was conducted to simultaneously stress the obstacle and quadrotor navigation abilities. ES2 achieves the highest quadrotor-level success rate (0.860), likely due to their trajectory sharing schemes allowing for denser navigation. Despite the amount of information exchanged between quadrotors in ES2, LEARN only falls behind by 0.015. In terms of overall success rates, the policy outperforms ES2 by 5%, suggesting that while our method relies on much more limited information it still yields effective multi quadrotor navigation. Most importantly, we do not rely on building a voxelized representation of the environment, such as in HDSM. From these complexities, the HDSM planner suffers the most in the *Swap Goal* scenario. The results show high quadrotor to quadrotor collision rates (0.27) and the lowest overall success rate 0.34. We believe this is due to the way the

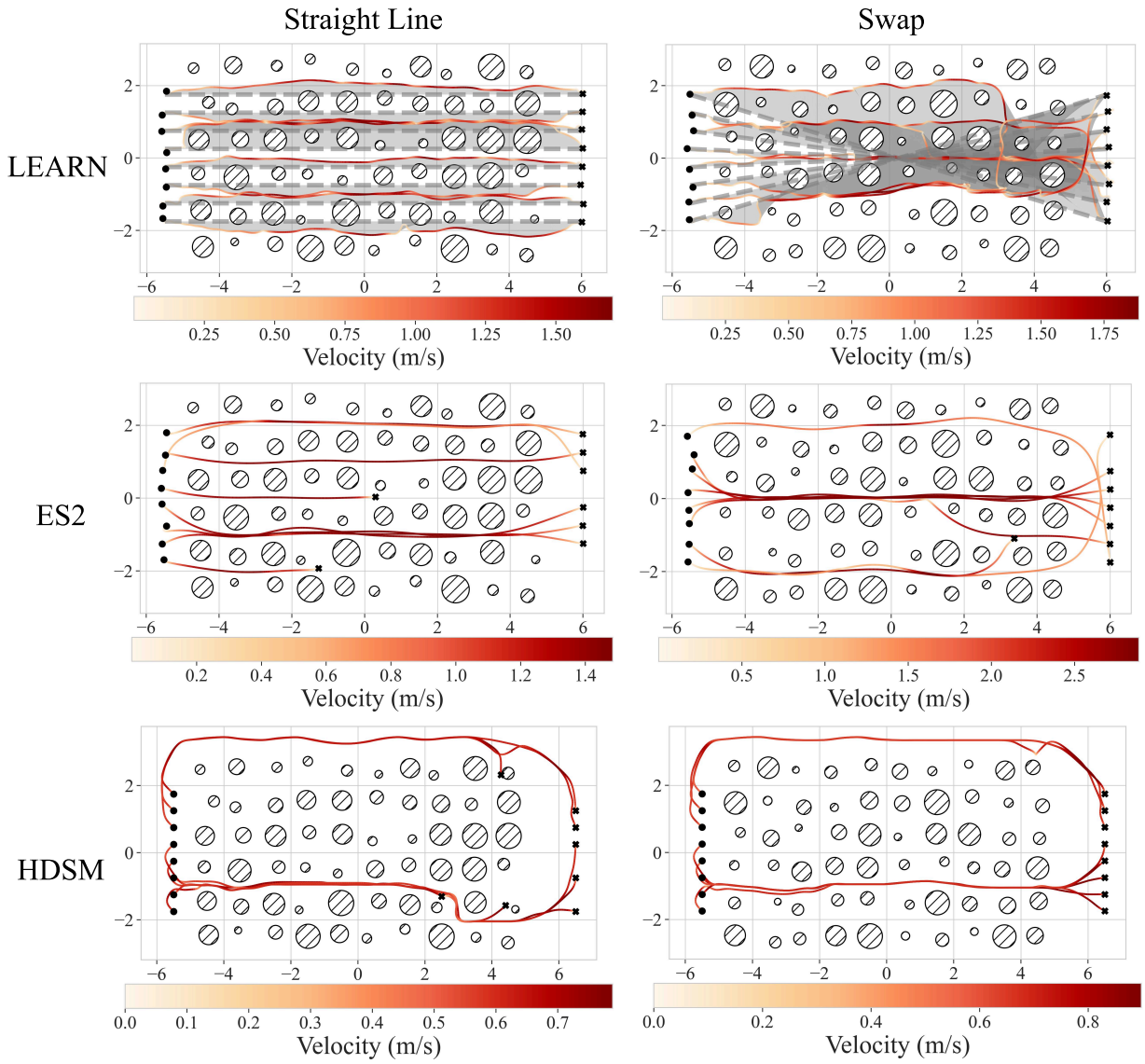

Fig. 6: **Trajectory Evaluation.** Comparison between two state of the art multi quadrotor motion planning frameworks ES2 [4] and HDSM [6], and LEARN in a long corridor type navigation task. The naive planned trajectories are shown as dotted lines. The velocity-colored trajectories are provided for a qualitative comparison. While LEARN has no objective function, the defined reward function often shapes how the resulting navigation path behaves. Our experiments show that LEARN successfully navigates more obstacle configurations with comparably less collisions than the two baseline comparisons.

synchronous planning is implemented, which especially suffers in tighter corridor navigation. When examining flight speeds, it is important to note that LEARN travels significantly slower than ES2. This occurs because our training environment randomly plans trajectories using a desired velocity sampled from a normal distribution, $\mathcal{N} \sim (\mu = 0.5, \sigma = 0.1)$. Thus, to remain well within the distribution, the planned trajectories used in benchmarks were fixed at $v_{desired} = 0.5$. We chose a normal distribution centered at $v = 0.5$ to accommodate for the limits of the hardware platform. To fly faster, one can train the policy on higher velocity trajectories as done in [56] to achieve higher flight speeds with a suitable platform.

Figure 7 plots success rates against traversability bins, where traversability is defined as the average unobstructed ray distance over N samples normalized against quadrotor radius ($r = 0.046m$ for all cases) [57]. In the multi quadrotor case, we propose a modified multi

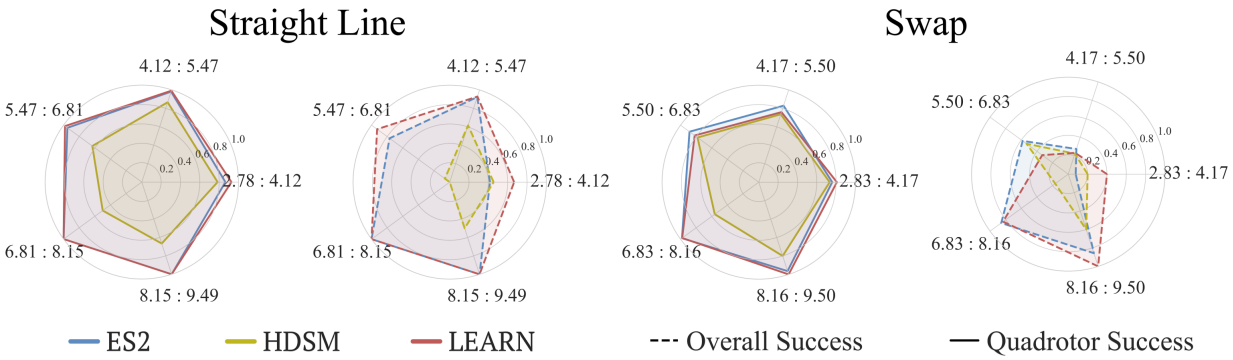

Fig. 7: **Traversability.** Performance breakdowns across the multi-quadrotor traversability metric (eq. 11). The metric measures the environment difficulty against the quadrotor size ($r = 0.046$ for all cases) and number of quadrotors. Each radar plot shows the range of the traversability metric as an edge. The solid lines indicate quadrotor-level success rates, whereas the dotted show overall success. A higher traversability indicates easier environments; lower traversability equates to more difficult.

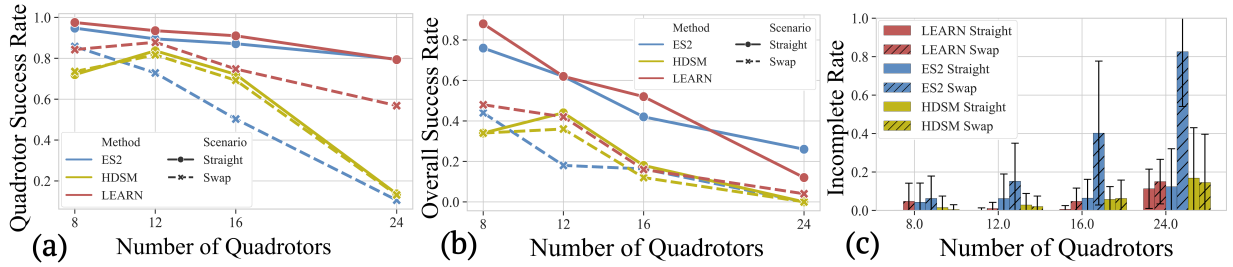

Fig. 8: **Scaling Experiments.** We conduct experiments that stress the scalability against baselines. Our results present both the overall and quadrotor level success rates, along with the fraction of quadrotors who did not reach the goal (unfinished rate). We find that the motion planning baselines are unable to complete the given environments for larger number of quadrotors as compared to LEARN.

quadrotor traversability metric:

$$\mathcal{T} = \frac{1}{j \cdot N \cdot r} \sum_{i}^{N} s_i \tag{11}$$

where j denotes the number of quadrotors and s_i the maximum ray distance (assuming full field of view). As traversability decreases (i.e. smaller sampled ray distances), all methods exhibit lower success rates and subsequently higher collision rates. However, our method degrades more efficiently: it sustains high quadrotor-level and overall success even at low traversability values. In contrast, ES2 and HDSM see much steeper drops in success and sharp rises in collisions as the environment becomes more cluttered. However, we note that ES2 maintains marginally higher performance in swap goal cases due to their denser sharing scheme.

b) Qualitative Results: Figure 6 shows the *Straight Line* scenario, where all quadrotors must traverse from $-x$ to $+x$ through an obstacle-dense field. Qualitatively, we find that the trajectories realized in all methods studied are visually smooth. LEARN shows a more direct traversal strategy that respects quadrotor to quadrotor distances. We believe that the minimum distance naive trajectory (the dotted line in Figure 6) is the main contributor to the efficient traversal behavior. In comparison, ES2 exhibits more conservative behaviors where quadrotors tend to slow down and maintain a larger distance around obstacles. HDSM demonstrates the most conservative maneuvers that loop around to completely avoid the obstacle field.

Figure 6 highlight the *Swap Goal* scenario, where quadrotors must traverse from $-x$ to x after mirroring goal positions across the y-axis. LEARN showcases more complex behaviors, where quadrotors find respective openings in the obstacle field to avoid quadrotor to quadrotor collisions. ES2 and HDSM both showcase similar behaviors by simplifying into a single-line formation. This is likely due to the constraints imposed by the planning process that forces quadrotors to plan into the largest gap. This may become cumbersome when dealing with larger quadrotor numbers and also can result in a higher likelihood of quadrotor to quadrotor collisions. Given these trajectories, we find that our method is able to distribute the quadrotors in a sparse and safe fashion, while preventing obstacle collisions.

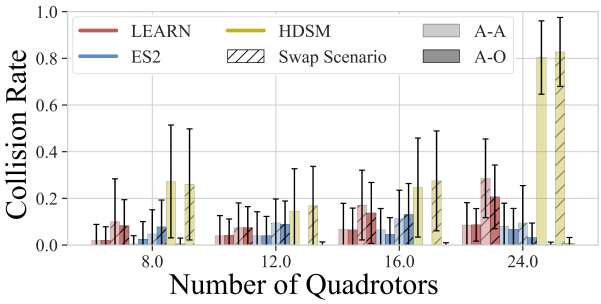

Fig. 9: **Collision Scaling.** Breakdown of collision rates as the number of quadrotors scale upwards. Our method, LEARN, becomes subject to higher collision rates in comparison to ES2 when 16 quadrotors are deployed.

2) Scaling: To evaluate the scalability and robustness of our navigation policy, we conducted experiments across group sizes of 8, 12, 16, and 24 quadrotors in both the *Straight Line* and *Swap Goal* scenarios (Figure 8). The quadrotor level success rates are reported in Figure 8(a), and the overall success rates in Figure 8(b). In the *Straight Line* task, our method exhibits strong quadrotor-level success rates, declining only from 0.975 (8 quadrotors) to 0.882 (24 quadrotors). The overall success rate, shows a sharper drop from 0.88 to 0.16. Despite this, the incomplete rate remains low and well-controlled, increasing from 0% at 8 quadrotors to just 0.103 at 24 quadrotors. This indicates that the majority of failures are due to collisions rather than inability to reach the goal. Furthermore, in the presence of a collision the policy is able to regain stability in order to reach the desired goal positions (given by the low incomplete rates). In contrast, ES2 and HDSM exhibit significant degradation at earlier quadrotor counts; for instance, HDSM's incomplete rate reaches 0.3 at 8 quadrotors and doubles at 24 quadrotors. ES2 follows a similar relationship, doubling from a 0.2 incomplete rate when 8 quadrotors are deployed to 0.465 at 24 quadrotors. When examining the *Swap Goal* scenario, our method's quadrotor-level success begins at 0.865 (8 quadrotors) and decreases to 0.635 (24 quadrotors), while the overall success rate declines more steeply from 0.480 to 0.06. The incomplete rate correspondingly increases from 0.2 to 0.5, highlighting the complexity of path-crossing. Importantly, even under these difficulties, our method maintains bounded collision rates (Figure 9), with quadrotor-quadrotor collisions increasing sub-linearly $(0.089$ to $0.281)$, demonstrating the policy's ability to maintain robustness. In comparison, HDSM and ES2 become bottlenecked by constraints (i.e. inter-quadrotor distance or minimum obstacle distances). The ES2 framework becomes stalled is unable to begin solving for the trajectory at 24 quadrotors, leading to the much lower collision rates. On the other hand, HDSM exhibits larger degradation in the quadrotor to quadrotor collision rates. We believe that the hyperplane separation used in the planner becomes too naive when deploying larger numbers of quadrotors. This is especially emphasized when the task involves swapping in obstacle and quadrotor rich environments.

A key differentiator lies in the communication and planning frameworks: both ES2 and HDSM rely on explicit trajectory sharing, where quadrotors broadcast their planned paths to start a joint-planning process. This approach introduces substantial bandwidth requirements, quadrotor to quadrotor temporal coupling, and excessive amounts of information, making the systems brittle in bandwidth-limited or high traffic situations. Our method, requires only exchanging instantaneous state vectors ($[x, \dot{x}]$) yielding a communication scheme that is both computationally and bandwidth efficient. Despite this minimalism, our method maintains higher robustness to increasing quadrotors counts, particularly in decentralized settings where full trajectory negotiation is difficult to deploy. Another reason comes from how the policy is trained. During training, the reward is a sum of *all* quadrotors' performance; whereas ES2 and HDSM solve their path planning *independently*. This causes an RL based approach to have higher overall success rates i.e. the policy is trained to maximize the rewards of all quadrotors not just individual. Despite this, the policy is deployed in an asynchronous fashion.

These results collectively illustrate that our attention-based policy exhibits favorable scalability properties. The model trades off between completeness and collision rates as group density increases, outperforming classical planners in scenarios where computation, sensing, and dense communication become bottlenecks.

3) Robustness to Communication Delays: We investigate the policy's robustness to reduced communication frequencies. Table V shows the two success rates, quadrotor to quadrotor collisions (Q-Q coll.), and quadrotor to obstacle collisions (Q-O coll.). We conduct over 50 trials where the state exchange rate of the environment is reduced. We observe that the performance of the policy in the straight line scenario remains above 95% for all trials. Whereas, quadrotor to quadrotor and obstacle collision rates remain below 4%. In the *Swap Goal* experiments, the performance is similarly consistent where the success rates remain above 85%. These results confirm that our policy tolerates significantly lower update rates, making it especially well-suited for nano-sized quadrotors with limited communication bandwidth. Given that our quadrotors travel at under 1.0 m/s on average, even infrequent state exchanges suffice to maintain collision free trajectories; however, abrupt velocity changes (for example, following an ob-

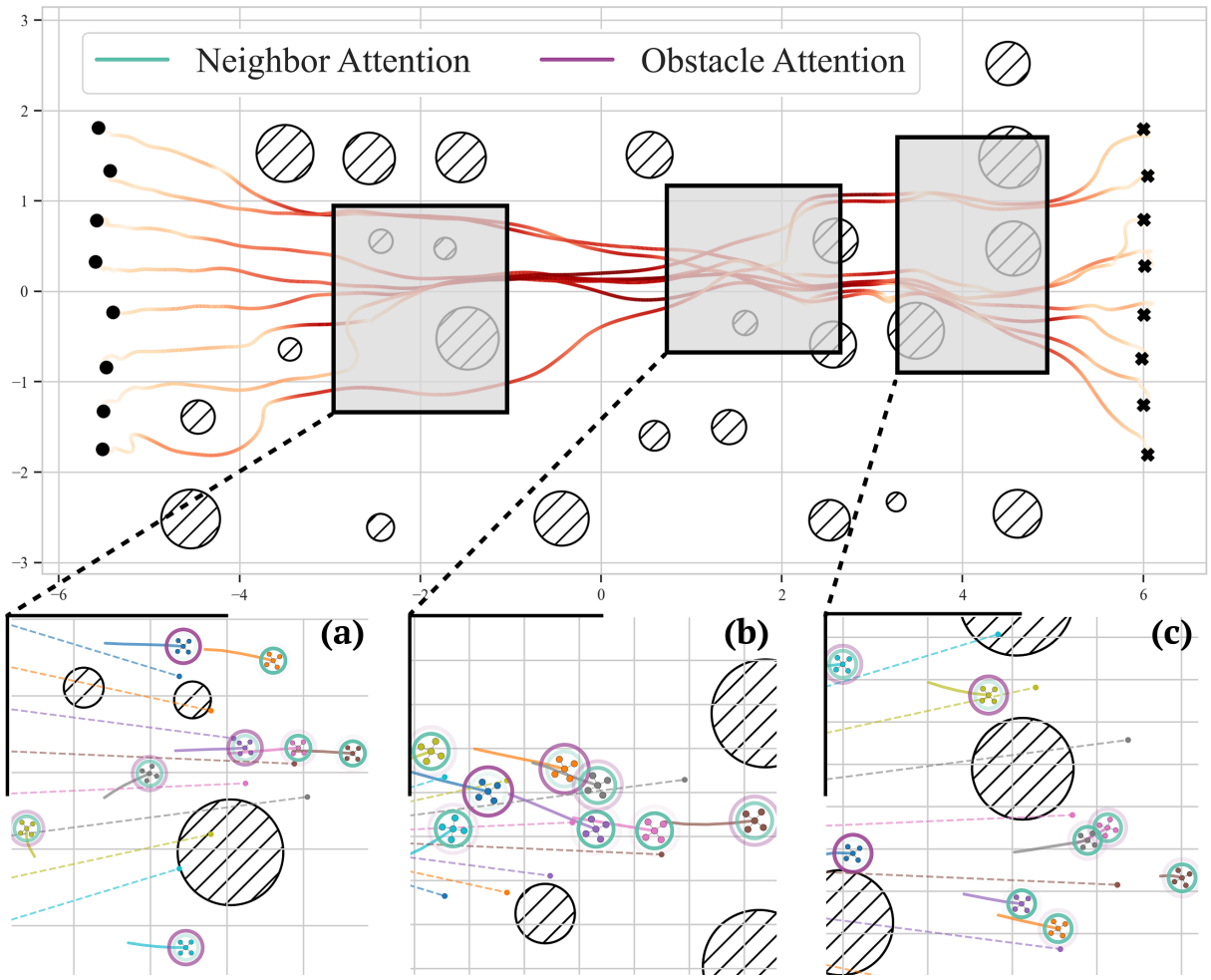

Fig. 10: **Attention Analysis.** We analyze the attention mechanism on a swap goal environment. The softmax outputs for the single head attention are shown as rings around each quadrotor. The softmax represents how much emphasis the policy is placing on each of the encodings, such as balancing neighbor and obstacle observations. A bolder color represents more emphasis, while a softer color shows less importance placed. The planned naive trajectory is shown for the current time step, and a trail of the past states is plotted for reference.

stacle impact) could, at very low frequencies, lead to sudden increases in collision risk. We also believe that sharing longer horizon trajectories (common to motion planning methods), become much less reliable in low communication settings. As message exchanges become infrequent, the shared trajectories become stale much faster. This can often lead to undesirable relationship between communication and performance.

4) Attention Analysis: To motivate the usage of an attention mechanism for multi quadrotor navigation, we explore the softmax outputs generated from the single head attention model (purple block in Figure 4). Figure 10 shows the progression as the policy tracks the trajectory, avoids obstacles, and negotiates neighboring collisions for a swap goal scenario. Subfigures (a-c) represent snapshots where the attention softmax is shown as rings around each quadrotor. The softmax value represents the amount of *attention* that is being placed for the corresponding inputs (neighbor or obstacle). We choose to exclude the dynamics encoding by re-normalizing against the neighbor and obstacle inputs. This is due to the observation that the dynamics attention is always prevalent and represented in attention. As we train from scratch i.e. the policy must first learn to first fly and track trajectories, this forces the policy to always attend to the dynamics. Figure 10(a) portrays the group as it navigates the beginning of the obstacle field. The yellow quadrotor in the back must first yaw to ensure that there is free space before steering around the obstacle. The balance between the green and red rings shows that

TABLE V: **Effects of Communication Delays.** The effects of slower state exchange rates on the RL policy for the straight and swap scenarios ($N = 50$). All experiments use 8 quadrotors.

Scenario	**Frequency** (hz / ms)	**Quadrotor-level**	**Overall**	**Q–Q coll.**	**Q–O coll.**
Straight	$50 \approx 20$	0.975	0.880	0.080	0.120
	$45 \approx 22.222$	0.974	0.878	0.015	0.025
	$35 \approx 28.571$	0.966	0.837	0.020	0.017
	$25 \approx 40$	0.961	0.837	0.031	0.028
	$15 \approx 66.667$	0.974	0.857	0.015	0.012
	$5 \approx 200$	0.977	0.878	0.010	0.020
Swap	$50 \approx 20$	0.864	0.510	0.089	0.081
	$45 \approx 22.222$	0.859	0.408	0.076	0.092
	$35 \approx 28.571$	0.877	0.489	0.089	0.079
	$25 \approx 40$	0.877	0.530	0.066	0.089
	$15 \approx 66.667$	0.869	0.510	0.0841	0.076
	$5 \approx 200$	0.885	0.489	0.061	0.059

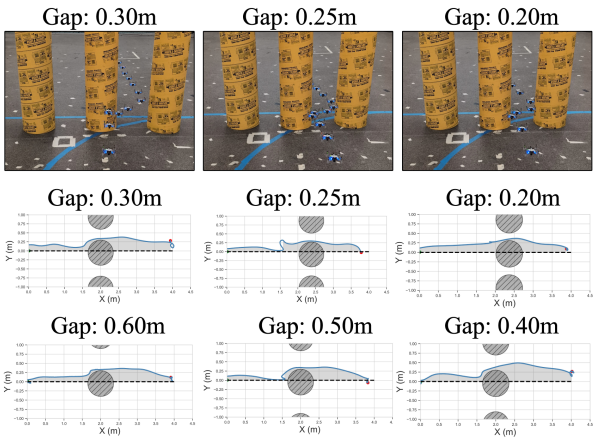

Fig. 11: **Gap Test.** LEARN is tasked with flying through decreasing gap sizes. During training, we limit the environment generation to only produce gap sizes as small as $0.15m$. The real world deployment shows reliable flight down to gaps as small as $0.2m$. This experiment uses the same policy, but with ablated neighbor encoders.

the quadrotor is still mindful of others that are just making it past the obstacle, whereas the quadrotors in the lead must negotiate their positions in the obstacle free space. In Figure 10(b) we see that the group is able to balance well between both of the encoded inputs as the naive trajectories overlap. In this case, the relative positions between the quadrotor and the current trajectory evaluation (x_{goal}) plays an important role. When the evaluated goal is in an area of high neighbor traffic (seen in the cyan, yellow, and pink quadrotors), the policy must focus on lagging or overshooting the desired goal to prevent collisions. On the contrary, if the evaluated goal point is in the direction of obstacles (orange, blue) the policy must focus on finding an obstacle free path. Furthermore, we find exciting the ranging sensors (i.e. $\exists \boldsymbol{\zeta}_i < 2.0 \forall\, i$) does not simply equate to immediately attending to the obstacle encoder. This is exhibited in the purple quadrotor of snapshot Figure 10(b). The trajectory velocity is pointing parallel to the obstacle i.e. continuing the straight line, but the quadrotor's current proximity to other neighbors forces it to focus on keeping a safe distance within the group. Finally, in Figure 10(c) we observe that the lower right group clears the obstacles and must negotiate the remaining free space among themselves. The three remaining quadrotors must still focus on the obstacles in front, yet are conscious of the others in front, such as the cyan quadrotor.

C. Hardware Experiments

We validate our policy's real-world performance using a fleet of six Crazyflie 2.1 nano quadrotors equipped with VL53L5CX time-of-flight sensors [58]. All processing, including sensor fusion, state estimation, trajectory planning, policy inference, and communication, is performed onboard, demonstrating feasibility on highly resource-constrained platforms. Experiments are conducted in both indoor warehouse environments and outdoor settings, employing cylindrical obstacles to test the policy's real-world generalization. Goal positions are provided through the Crazyswarm ROS environment [59], while execution, sensing, and collision avoidance rely entirely on each quadrotor's onboard compute.

1) Obstacle Avoidance: We evaluate our policy in the real world by first tasking a single quadrotor to navigate through progressively narrower gap sizes. Below we show the trajectories of a single quadrotor flying through gaps ranging from sizes of $0.6m$ to $0.2m$, whereas the planned trajectories are shown in the dotted. The policy consistently maintains a close yet safe distance to the seen obstacles, allowing it to tightly navigate corridors. This demonstrates that the trajectory tracking reward and obstacle collision penalty work together well for this scenario. As such, our experiments indicate that we are able to reliably fly through a $0.2m$ gap, roughly 2 times the quadrotor diameter.

2) Multi Quadrotor Navigation: Finally, we conduct experiments by evaluating trajectory tracking, obstacle avoidance, and neighbor avoidance together. We evaluate our models in 6 randomly generated obstacle configurations in an indoor warehouse and various outdoor situations. A composite image of the systems deployed trajectories are shown in Figure 12. The difficulty of the tested environment increases with (a) being the easiest and (e) being the hardest. We group obstacles together in Figure 12(f) to show that the deployed model does not overfit to a cylindrical obstacle size. In this case the obstacle sizes range from 0.35m to 1.0m with linear diagonals. Our framework exhibits near zero collision rates with respect to both obstacles and other quadrotors. The recorded trajectories for (e) and (f) are shown in Figure 12. The policy remains close to the planned trajectory

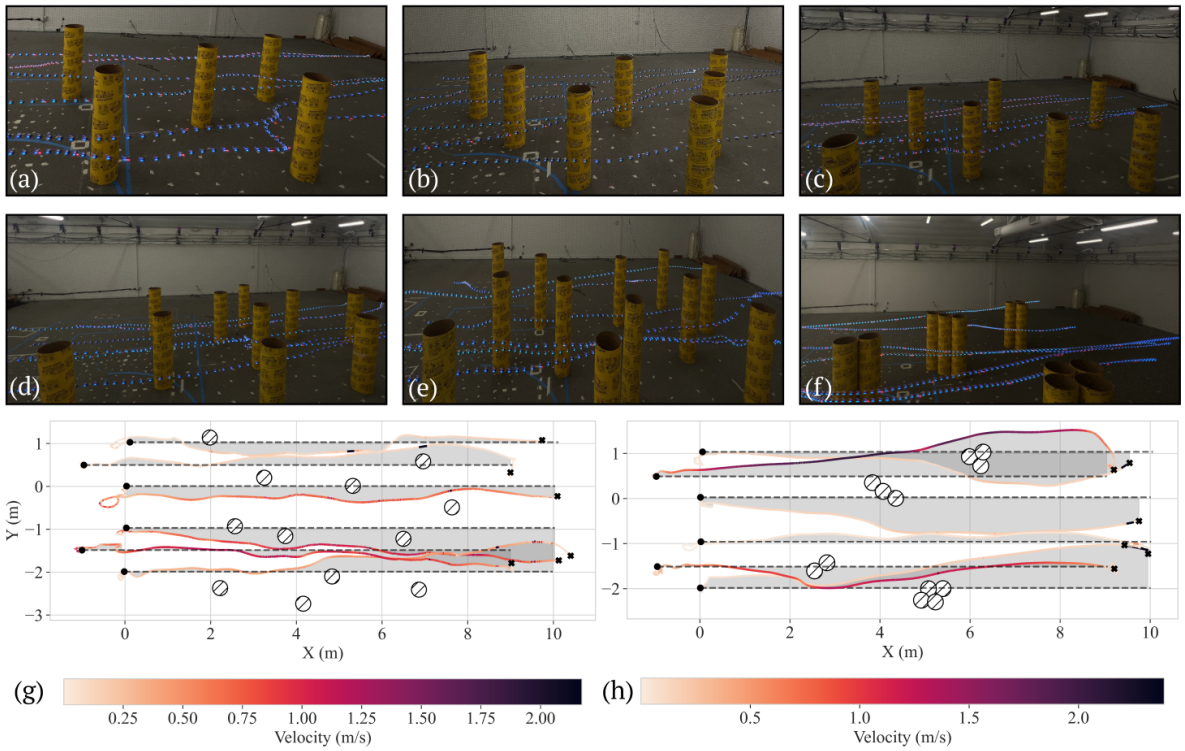

Fig. 12: **Deployment Test.** Various real world deployment configurations from easy (a) to difficult (e). All individual obstacles are $0.61m$ in diameter. (f) shows that the model is able to transfer to randomized obstacle sizes. The group is tasked with tracking a $10m$ long straight line trajectory that intersects a field of obstacles. (g) represents the recorded trajectory for deployment (e). (h) is the recorded trajectory for deployment (f).

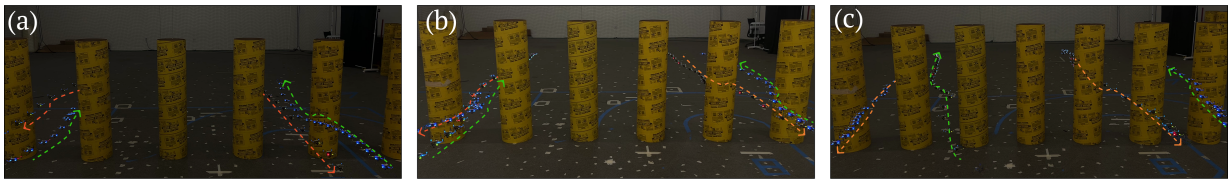

Fig. 13: **Unilateral Swap.** A unilateral swap configuration. Quadrotors begin in a square and must swap positions with the quadrotor directly opposite of them. As the gap size of the obstacles grow smaller, the policy chooses to separate diverge quadrotor trajectories. (a) shows a gap size of $0.5m$, where (b) and (c) show gap sizes of $0.4m$ and $0.3m$ respectively.

until an obstacle is seen within the field of view.

a) Goal Swapping: We also evaluate the policy's ability in a symmetric goal-swapping task. Four quadrotors are initialized in a square, each required to reach the location directly opposite their starting position. In wider gap scenarios (e.g. $0.5m$), the quadrotors naturally select the most direct shared gap, minimizing deviation from their planned straight-line paths, even if that means multiple quadrotors simultaneously pass through the same corridor. This indicates that the policy can implicitly reason about observed obstacle locations to share space without collisions.

As the gap narrows to $0.3m$, the behavior shifts: quadrotors actively select different paths by deviating farther from their planned trajectories. This emergent behavior showcases the policy's capacity to adapt its aggressiveness in response to environmental conditions.

To further assess policy deployment, we introduce a bilateral swap scenario, where quadrotors must cross along diagonals of the square resulting in head-on trajectories through the center of the obstacle wall. This is a significantly harder configuration due to tighter interactions in path intersection and reduced clearance near the center. We observe that the policy is still capable of executing the bilateral swap reliably in $0.5m$ gaps. Each quadrotor adjusts its path by slowing down or

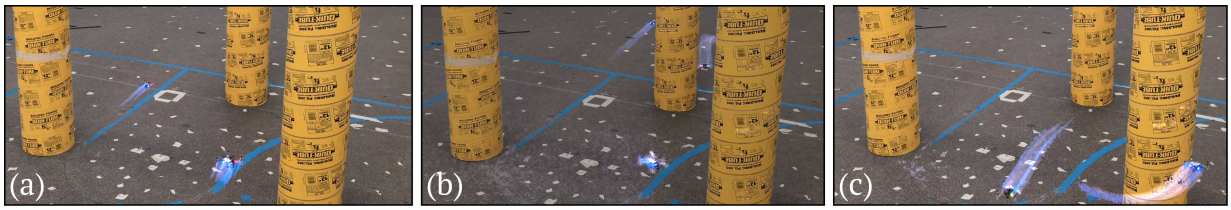

Fig. 14: **High Conflict Negotiation.** The quadrotors are faced with intersecting trajectories with the presence of obstacles. The right outgoing quadrotor waits for the incoming to pass before swerving to the right to dodge.

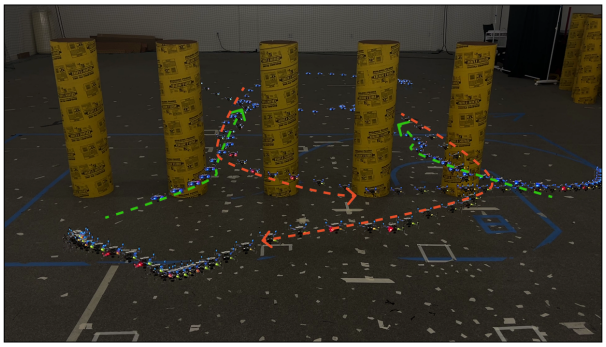

Fig. 15: **Bilateral Swap.** A bilateral swap configuration is shown where quadrotors are tasked with swapping goals along the diagonals. We find that the policy does not travel in the direction of the blind spots, despite the naive trajectory being planned through the diagonals of the area.

shifting laterally to avoid collisions through the constrained portion of the space. Notably, these behaviors emerge without explicit coordination logic or trajectory handshaking. Furthermore, the hardware experiments show that the policy learns to only travel in directions within the sensing field of view. This stems from the explicit lateral movements, despite the planned trajectory crossing through the middle. These results reinforce the learned controller's capability for real-time, decentralized negotiation in high-conflict scenarios even under obstacle restricted environments. We emphasize that the obstacle configuration is unknown *a priori* making the swap more difficult.

b) High-Conflict Scenario: Finally, Figure 14 shows a challenging real-world deployment scenario involving three quadrotors navigating through a triangular arrangement of obstacles. The quadrotors are initially positioned on intersecting paths, one approaching from the incoming direction, and two departing along outgoing trajectories. At first, the left outgoing quadrotor travels through the obstacle formation (a). The incoming quadrotor must then select a maneuver that allows it to bypass the central obstacle without obstructing the outbound paths of the two others. As it begins its approach, the right outgoing quadrotor preemptively stops tracking the trajectory (b), creating space for the incoming quadrotor. When the incoming has decided to move, the right outgoing then dodges to the right of the obstacle (c). This interaction unfolds without a centralized controller, prior communication or a prebuilt map; instead, each quadrotor operates using only instantaneous state exchanges and local ranging sensing. This scenario highlights our approach's capability for agile, autonomous conflict resolution in tight, spatially constrained conditions, for multi quadrotor navigation.

VII. Limitations

While our results demonstrate the feasibility of deploying a fully onboard learned navigation policy for multi-robot coordination, several limitations remain.

First, the policy relies on local sensing and short-horizon reasoning, which can lead to suboptimal behavior in environments requiring long-term foresight—such as dead-ends, maze-like corridors, or traps requiring backtracking. Without global map information or memory mechanisms, the system must rely on reactive adjustments, which may not suffice in adversarial or partially observable settings. As our method works at the control level, future work can extend by integrating a relatively lightweight motion planner, where the neural controller is trained to correct for potential mistakes. This would integrate both a long horizon decision making mechanism with our short horizon lightweight observation conditioned controller.

Second, while the system is trained to generalize across a range of obstacle configurations, it assumes a relatively static environment during deployment. Rapidly moving obstacles, strong wind disturbances, or unpredictable third-party agents (e.g., humans or other robots) may challenge the policy's reactivity. One can include disturbances and faster moving objects within training to improve the controller.

Addressing these limitations will involve integrating richer sensor fusion, predictive modeling, memory-aware policies, and adaptive communication strategies—opening pathways for more robust and scalable decentralized multi-robot navigation. However, given the

current limits of the platform we believe that our method pushes the hardware constraints to its limit.

VIII. Conclusion

We present an end-to-end learned navigation system designed for real-time deployment on resource-constrained nano quadrotors. Our approach integrates a lightweight attention-based neural network with minimal depth sensing, enabling fully onboard perception, planning, and control without reliance on external infrastructure or centralized coordination.

Through extensive simulation and hardware experiments, we demonstrated that our policy can reliably perform multi-quadrotor navigation, obstacle avoidance, and formation tracking in both unknown environments. The system maintains low collision rates even in dense scenarios, and generalizes across a wide range of obstacle configurations and quadrotor formations. Our results indicate marginally better performance over state of the art motion planners, despite only using a fraction of compute. Notably, we showed that the learned controller can handle high-conflict maneuvers including goal-swapping, bilateral negotiations, and conflict resolution without explicit communication or global planning. These results underscore the feasibility of using compact, learned attention models for decentralized multi-quadrotor navigation in highly constrained settings. Future work will explore scaling to larger teams, incorporating more diverse sensor modalities, and integrating memory for long-horizon reasoning in dynamic environments.

References

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *Ieee Access*, vol. 7, pp. 48 572–48 634, 2019.

[2] B. Mishra, D. Garg, P. Narang, and V. Mishra, "Drone-surveillance for search and rescue in natural disaster," *Computer Communications*, vol. 156, pp. 1–10, 2020.

[3] M. Hassanalian, D. Rice, and A. Abdelkefi, "Evolution of space drones for planetary exploration: A review," *Progress in Aerospace Sciences*, vol. 97, pp. 61–105, 2018.

[4] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, and F. Gao, "Swarm of micro flying robots in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm5954, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm5954

[5] M. Y. Arafat, M. M. Alam, and S. Moh, "Vision-based navigation techniques for unmanned aerial vehicles: Review and challenges," *Drones*, vol. 7, no. 2, p. 89, 2023.

[6] C. Toumieh and D. Floreano, "High-speed motion planning for aerial swarms in unknown and cluttered environments," *IEEE Transactions on Robotics*, vol. 40, pp. 3642–3656, 2024.

[7] B. Şenbaşlar and G. S. Sukhatme, "Dream: Decentralized real-time asynchronous probabilistic trajectory planning for collision-free multi-robot navigation in cluttered environments," *IEEE Transactions on Robotics*, 2024.

[8] W. Hönig, J. A. Preiss, T. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 856–869, 2018.

[9] J. Park, J. Kim, I. Jang, and H. J. Kim, "Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 434–440.

[10] K. N. McGuire, C. D. Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, p. eaaw9710, 2019. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.aaw9710

[11] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 604–611, 2020.

[12] 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, 2020.

[13] 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. 5, no. 3, pp. 4249–4256, 2020.

[14] E. Soria, F. Schiano, and D. Floreano, "Predictive control of aerial swarms in cluttered environments," *Nature Machine Intelligence*, vol. 3, no. 6, pp. 545–554, 2021.

[15] ——, "Distributed predictive drone swarms in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 73–80, 2021.

[16] H. Zhu, F. M. Claramunt, B. Brito, and J. Alonso-Mora, "Learning interaction-aware trajectory predictions for decentralized multi-robot motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2256–2263, 2021.

[17] X. Zhou, J. Zhu, H. Zhou, C. Xu, and F. Gao, "Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments," in *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2021, pp. 4101–4107.

[18] J. Hou, X. Zhou, Z. Gan, and F. Gao, "Enhanced decentralized autonomous aerial robot teams with group planning," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9240–9247, 2022.

[19] J. Park, D. Kim, G. C. Kim, D. Oh, and H. J. Kim, "Online distributed trajectory planning for quadrotor swarm with feasibility guarantee using linear safe corridor," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4869–4876, 2022.

[20] G. Ryou, E. Tal, and S. Karaman, "Cooperative multi-agent trajectory generation with modular bayesian optimization," *arXiv preprint arXiv:2206.00726*, 2022.

[21] J. Park, I. Jang, and H. J. Kim, "Decentralized deadlock-free trajectory planning for quadrotor swarm in obstacle-rich environments," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1428–1434.

[22] J. Park, Y. Lee, I. Jang, and H. J. Kim, "Dlsc: Distributed multi-agent trajectory planning in maze-like dynamic environments using linear safe corridor," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3739–3758, 2023.

[23] B. Şenbaşlar, W. Hönig, and N. Ayanian, "Rlss: real-time, decentralized, cooperative, networkless multi-robot trajectory planning using linear spatial separations," *Autonomous Robots*, vol. 47, no. 7, pp. 921–946, 2023.

[24] 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," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1687–1693.

[25] Y. Chen, C. Wang, M. Guo, and Z. Li, "Multi-robot trajectory planning with feasibility guarantee and deadlock resolution: An

obstacle-dense environment," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2197–2204, 2023.

[26] V. K. Adajania, S. Zhou, A. K. Singh, and A. P. Schoellig, "Amswarm: An alternating minimization approach for safe motion planning of quadrotor swarms in cluttered environments," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1421–1427.

[27] ——, "Amswarmx: Safe swarm coordination in complex environments via implicit non-convex decomposition of the obstacle-free space," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 555–14 561.

[28] S. Safaoui, A. P. Vinod, A. Chakrabarty, R. Quirynen, N. Yoshikawa, and S. Di Cairano, "Safe multi-agent motion planning under uncertainty for drones using filtered reinforcement learning," *IEEE Transactions on Robotics*, 2024.

[29] S. Wu, G. Chen, M. Shi, and J. Alonso-Mora, "Decentralized multi-agent trajectory planning in dynamic environments with spatiotemporal occupancy grid maps," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 7208–7214.

[30] Z. Huang, Z. Yang, R. Krupani, B. Şenbaşlar, S. Batra, and G. S. Sukhatme, "Collision avoidance and navigation for a quadrotor swarm using end-to-end deep reinforcement learning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 300–306.

[31] M. Goarin, G. Li, A. Saviolo, and G. Loianno, "Decentralized nonlinear model predictive control for safe collision avoidance in quadrotor teams with limited detection range," *arXiv preprint arXiv:2409.17379*, 2024.

[32] L. Pan, Y. Wang, and N. Ayanian, "Hierarchical trajectory (re) planning for a large scale swarm," *arXiv preprint arXiv:2501.16743*, 2025.

[33] A. Romero, Y. Song, and D. Scaramuzza, "Actor-critic model predictive control," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 14 777–14 784.

[34] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research: The 14th International Symposium ISRR*. Springer, 2011, pp. 3–19.

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

[36] 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.

[37] 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, 2021.

[38] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 411–444, 2022.

[39] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 3387–3395.

[40] 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)*. IEEE, 2022, pp. 5518–5524.

[41] W. Xiao, T.-H. Wang, R. Hasani, M. Chahine, A. Amini, X. Li, and D. Rus, "Barriernet: Differentiable control barrier functions for learning of safe robot control," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 2289–2307, 2023.

[42] N. Csomay-Shanklin, R. K. Cosner, M. Dai, A. J. Taylor, and A. D. Ames, "Episodic learning for safe bipedal locomotion with control barrier functions and projection-to-state safety," in *Learning for dynamics and control*. PMLR, 2021, pp. 1041–1053.

[43] Z. Gao, G. Yang, and A. Prorok, "Online control barrier functions for decentralized multi-agent navigation," in *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 2023, pp. 107–113.

[44] O. So, Z. Serlin, M. Mann, J. Gonzales, K. Rutledge, N. Roy, and C. Fan, "How to train your neural control barrier function: Learning safety filters for complex input-constrained systems," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 11 532–11 539.

[45] T. He, C. Zhang, W. Xiao, G. He, C. Liu, and G. Shi, "Agile but safe: Learning collision-free high-speed legged locomotion," *arXiv preprint arXiv:2401.17583*, 2024.

[46] H. Krasowski, J. Thumm, M. Müller, L. Schäfer, X. Wang, and M. Althoff, "Provably safe reinforcement learning: Conceptual analysis, survey, and benchmarking," *arXiv preprint arXiv:2205.06750*, 2022.

[47] X. Wang, "Ensuring safety of learning-based motion planners using control barrier functions," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4773–4780, 2022.

[48] F. P. Bejarano, L. Brunke, and A. P. Schoellig, "Safety filtering while training: Improving the performance and sample efficiency of reinforcement learning agents," *IEEE Robotics and Automation Letters*, 2024.

[49] B. P. Duisterhof, S. Li, J. Burgués, V. J. Reddi, and G. C. H. E. de Croon, in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, 2021, p. 9099–9106. [Online]. Available: https://doi.org/10.1109/IROS51168.2021.9636217

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

[51] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European control conference (ECC)*. Ieee, 2019, pp. 3420–3431.

[52] V. Niculescu, T. Polonelli, M. Magno, and L. Benini, "Nanoslam: Enabling fully onboard slam for tiny robots," *IEEE Internet of Things Journal*, 2023.

[53] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 2520–2525.

[54] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population based training of neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1711.09846

[55] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[56] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.

[57] C. Nous, R. Meertens, C. De Wagter, and G. Croon, "Performance evaluation in obstacle avoidance," 10 2016, pp. 3614–3619.

[58] H. Müller, V. Niculescu, T. Polonelli, M. Magno, and L. Benini, "Robust and efficient depth-based obstacle avoidance for autonomous miniaturized uavs," *IEEE Transactions on Robotics*, 2023.

[59] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm." IEEE Press, 2017, p. 3299–3304. [Online]. Available: https://doi.org/10.1109/ICRA.2017.7989376