

# A Framework for Real-World Multi-Robot Systems Running Decentralized GNN-Based Policies

Jan Blumenkamp, Steven Morad, Jennifer Gielis, Qingbiao Li and Amanda Prorok

**Abstract**—Graph Neural Networks (GNNs) are a paradigm-shifting neural architecture to facilitate the learning of complex multi-agent behaviors. Recent work has demonstrated remarkable performance in tasks such as flocking, multi-agent path planning and cooperative coverage. However, the policies derived through GNN-based learning schemes have not yet been deployed to the *real-world* on physical multi-robot systems. In this work, we present the design of a system that allows for fully decentralized execution of GNN-based policies. We create a framework based on ROS2 and elaborate its details in this paper. We demonstrate our framework on a case-study that requires tight coordination between robots, and present first-of-a-kind results that show successful real-world deployment of GNN-based policies on a decentralized multi-robot system relying on *Adhoc* communication. A video demonstration of this case-study can be found online<sup>1</sup>.

**Index Terms**—Multi-Robot Systems, Graph Neural Network, Robot Learning, Sim-to-Real

## I. INTRODUCTION

Significant effort has been invested into finding analytical solutions to multi-robot problems, balancing optimality, completeness, and computational efficiency [1], [2], [3], [4]. Data-driven approaches can find near-optimal solutions to NP-hard problems, enabling fast on-line planning and coordination, as typically required in robotics. This has thus provided alternatives for the aforementioned challenges [5], [6], [7], [8]. Graph Neural Networks (GNNs), in particular, demonstrate remarkable performance and generalize well to large-scale robotic teams for various tasks such as flocking, navigation, and control [9], [6], [10], [11], [12], [13]. In such multi-robot systems, GNNs learn inter-robot communication strategies using latent messages. Individual robots aggregate these messages from their neighbors to overcome inherently local (partial) knowledge and build a more complete understanding of the world they are operating in.

While GNN-based policies are typically trained in a centralized manner in *simulation*, and therefore assume synchronous communication, resulting policies can be executed either in a centralized or decentralized mode. Evaluating a GNN in the *centralized* mode typically requires execution on a single machine decoupled from the robots that are acting according to the policy [6], [10], [14]. This (i) introduces a single point of failure, (ii) requires all robots to maintain constant network connectivity, and (iii) introduces scalability issues due to computational complexity  $O(N^2)$  where  $N$  is the number of robots. In contrast, in the *decentralized* mode, each robot is responsible for making its own decisions. With fully decentralized evaluation, (i) there is no single point of failure, resulting in a higher fault tolerance, (ii) agents do not need to remain in network range of a router that orchestrates

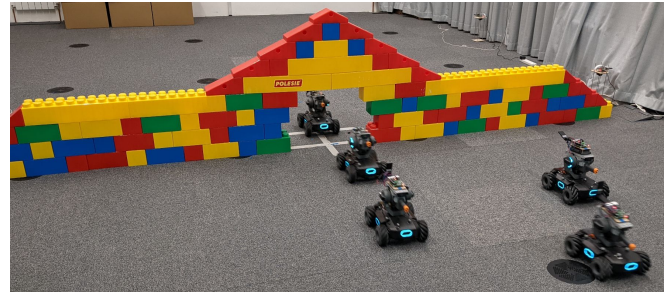


Fig. 1. We deploy a set of five DJI RoboMaster robots in a real-world setup using GNNs and Adhoc communication. The robots navigate through a narrow passageway to reconfigure on the other side, as quickly as possible.

the evaluation, and (iii) computation is parallelized across  $N$  robots, decoupling the time complexity from the number of robots.

Even though GNNs have an inherently decentralizable mathematical formulation, previous work on GNN-based multi-robot policies was conducted exclusively in centralized simulations using synchronous communication [10], [11], [9]. For practical reasons, decentralized execution is often unavoidable in the *real-world*, but it is currently unknown whether this contributes to a shift of domains, and how resulting policies are affected. Multi-robot GNNs require inter-robot communication, but real-world wireless communication is noisy, and messages can be lost or delayed, leading to significant performance loss—this is exemplified in prior work that demonstrates the need for appropriate models to overcome these challenges [15], [16], [17], [18]. Further compounding these issues, decentralized policies are typically executed asynchronously, resulting in system states not previously encountered during training.

In this paper, we provide a framework that facilitates the decentralized execution of GNN-based multi-robot policies. We present the results of a suite of real-robot experiments (see Fig. 1) to demonstrate the consequences of this decentralized execution. To that end, we introduce a taxonomy of different evaluation modes and networking configurations. Specifically, we contribute:

- 1) A ROS2-based software and networking framework for GNNs and other message-passing algorithms to facilitate operation in both simulation and the real-world, and to permit GNN-based policy execution in either a centralized or decentralized manner. We provide the source code online.<sup>2</sup>
- 2) An ablation study on several forms of execution to quantify performance shifts between centralized execution and three forms of decentralized policy execution, (i) offboard (non-local), (ii) onboard over routing in-

All authors are with Department of Computer Science and Technology, University of Cambridge. Email: {jb2270, sm2558, jag233, ql295, asp45}@cam.ac.uk.

<sup>1</sup>[youtube.com/watch?v=C0h-wLn4iO4](https://youtube.com/watch?v=C0h-wLn4iO4)

<sup>2</sup>[github.com/proroklab/ros2\\_multi\\_agent\\_passage](https://github.com/proroklab/ros2_multi_agent_passage)

frastructure, and (iii) onboard with Adhoc networking.

## II. RELATED WORK

In this section, we first review related multi-robot systems testbeds and frameworks. Our survey includes centralized frameworks as well as decentralized methods that either use machine-learning based approaches or communication. We emphasize that none of these methods combine learning-based methods and communication. Lastly, we review the related work on robotic communication frameworks and standards to evaluate an appropriate choice for our use-case.

**Multi-Robot Systems Testbeds.** Remotely accessible mobile and wireless sensor testbeds are in high demand both in research and industry. Mobile Emulab [19] and CrazySwarm [20] were developed as centrally controlled real-world multi-robot research platforms. As decentralized platforms gained popularity, roboticists developed a variety of systems for small footprint robot swarms, including Robotarium [21], Micro-UAV [22] or IRIS [23] to large scale platforms such as HoTDeC [24]. These platforms provide testbeds for decentralized control and communication. However, none of these systems utilize machine-learning-based policies, and only few learning-based methods have demonstrated real-world experiments [5]. Although work at the intersection of machine-learning and multi-robot control shows remarkable performance [5], [6], [10], [25], [26], [27], little work has been done to show how to make these methods practical (i.e., real-world). Of particular interest is how explicit inter-robot communication [9], [10], [28] plays a role in accumulating information from other robots. A recent study investigates the robustness of decentralized inference of binary classifier GNNs in wireless communication systems [29], but their work is limited to simulation and does not focus on communication contention and latency. These learning-based multi-agent platforms and multi-robot frameworks are either restricted to simulation [6], [10], [26], rely on centralized evaluation [5], [14], or are only evaluated in simulated experiments for decentralized wireless communication. There is a gap between simulation-based testbeds and testbeds that facilitate the deployment of policies derived from machine-learning methods to the real-world.

**Robotic Communications Frameworks.** Communications between agents and controllers is a ubiquitous requirement on experimental robotics platforms, either for experimental control or operational messaging. For these functions, the IEEE 802.11 (commonly WiFi) and 802.15 protocol suites are commonly used [30], with various communications frameworks are overlaid on top of these low-level technologies (e.g. RTPS, MQTT [23] or standard IP [24]). Whatever the specific technology, the underlying protocol suites and the nature of wireless communication set fundamental limitations [31] on available messaging rates when multiple agents are communicating in a decentralized manner. Multiple strategies exist that attempt to maximize protocol performance under specific conditions [32], [33], including dynamic centralization using homogeneous agents [34]. Despite these strategies, the performance of these systems at scale remain poorly tested in real-world robotics systems, which often entail unexpected overheads [35].

## III. PRELIMINARIES

In this section, we review the formalization of GNNs as well as the basic functionalities of Robot Operating System (ROS), the software library that we build on.

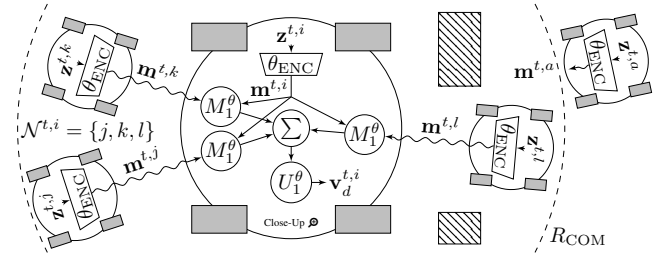


Fig. 2. The robots form a graph based on their separation and communication range  $R_{COM}$ . They leverage communication via latent messages  $m^{t,i}$  generated from local observations  $z^{t,i}$  propagated over graph edges (wireless Adhoc communication links) to overcome the partial observability of the workspace. To solve this task, we utilize and deploy GNN-based policies that aggregate messages of robots within the local neighborhood  $N^{t,i}$  and compute a local action.

### A. Graph Neural Networks

A multi-robot system can be defined as graph  $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ , where each robot is represented as a node in the node set  $\mathcal{V} = \{1, \dots, n\}$ . The inter-robot relationships are represented as edge set  $\mathcal{E} = \mathcal{V} \times \mathcal{V}$  with edge features  $e^{t,j,i} \in \mathcal{E}$  at each discrete time step  $t$ . If robot  $j$  is in communication range  $R_{COM}$  of robot  $i$ , it is in robot  $i$ 's neighborhood  $j \in N^{t,i}$  and robot  $i$  can emit a message  $m^{t,i}$  that is broadcast to its neighbors.

Neural message passing [36] updates the hidden state  $h_k^{t+1,i}$  of each robot  $i$  for each neural network layer  $k$  using the message function  $M$  and the vertex update function  $U$  according to

$$h_k^{t+1,i} = U_k^\theta \left( h_{k-1}^{t,i}, \sum_{j \in N^{t,i}} M_k^\theta \left( h_{k-1}^{t,i}, h_{k-1}^{t,j}, e^{t,j,i} \right) \right), \quad (1)$$

where  $U$  and  $M$  are functions with learnable parameters  $\theta$ . The decentralized evaluation is explained in Fig. 2. Although centralized formulations also exist, according to (1), evaluating a GNN is a fully decentralizable operation depending only on received messages and local information.

### B. ROS and ROS2

ROS is a set of open-source libraries for messaging, device abstraction, and hardware control [37]. ROS generates a peer-to-peer graph of processes (*Nodes*), communicating over edges (*Topics*). ROS requires a master node to connect to all other nodes, preventing its use in fully decentralized systems. ROS2 is a redesign of ROS that solves the master node issue, enabling completely decentralized systems [38]. Many popular frameworks have not migrated from ROS to ROS2, preventing their use in fully decentralized multirobot systems. Our software infrastructure leverages ROS2 to create fully independent agents.

## IV. APPROACH

Our framework can be separated into software and networking infrastructure. In this section, we first explain our software framework. Our framework is capable of running policies in a fully decentralized asynchronous Adhoc mode, but for the purpose of an experimental ablation analysis, we identify a range of sub-categories with different degrees of decentralization.

Specifically, we introduce the four modes: *Centralized* (fully centralized evaluation), *Offboard* (asynchronous evaluation on a central computer), *Onboard o/Infra* (decentralization using existing centralized networking infrastructure) and

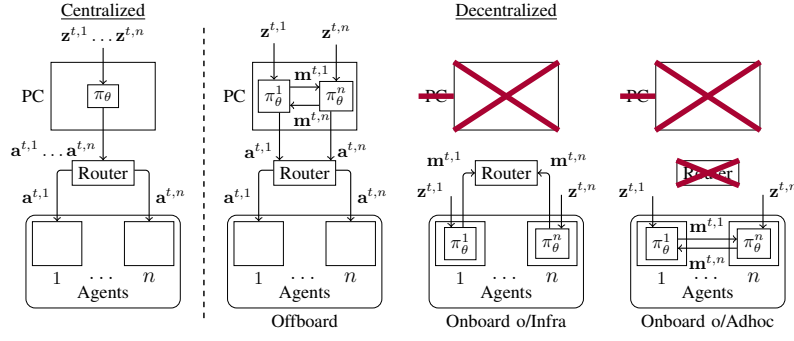


Fig. 3. The framework configurations used in our experiments. The ROS2 infrastructure is either centralized or decentralized, with varying degrees of decentralization depending on the network setup. We refer to these four configurations as Centralized, Offboard, Onboard over Infrastructure, and Onboard over Adhoc. Observations  $\mathbf{z}^{t,1} \dots \mathbf{z}^{t,n}$  feed into centralized policy  $\pi_\theta$  or local policies  $\pi_\theta^1 \dots \pi_\theta^n$  to produce actions  $\mathbf{a}^{t,1} \dots \mathbf{a}^{t,n}$  for agents  $1 \dots n$ . Local policies consist of a GNN and pass messages  $\mathbf{m}^{t,1} \dots \mathbf{m}^{t,n}$  to communicate. In the centralized case, a single policy produces actions for all agents at once in a synchronized manner. For Offboard, local policies run asynchronously, exchanging messages over localhost. The PC is removed for Onboard o/Infra, moving inference onto the robot computers. Onboard o/Adhoc is fully decentralized – the agents forgo the router and communicate directly using Adhoc networking.

Onboard o/Adhoc (full decentralization using Adhoc communication networks), as visualized in Fig. 3. We describe the networking considerations that allow ROS2 to be used for decentralized Adhoc communication between agents.

#### A. ROS2 Infrastructure

Our multi-agent ROS2 infrastructure (see Fig. 4) allows us to run both simulated and real-world agents concurrently, over multiple episodes, in centralized or decentralized mode, and without human intervention (facilitated through automated resets). An episode is one instance of one experimental trial and a reset is a scenario-specific resetting operation, e.g., requiring robots to move to initial positions. These two actions are repeated for a set number of iterations and different initial states. Our infrastructure follows the Reinforcement Learning (RL) paradigm of delineating the *agent* from the *world*.

1) *Agent*: The agent receives raw sensor data and emits motor commands. The agent is composed of the cache/filter, policy, and control nodes. The cache/filter node uses sensor information  $\hat{\mathbf{z}}^{t,i}$  to determine neighboring agents  $j \in \mathcal{N}^{t,i}$  within the specified communication radius. It caches neighborhood messages  $\mathbf{m}^{t,j}$  and sensor information  $\mathbf{z}^{t,i}$  over  $\Delta t$  for the policy. The policy node wraps a trained policy  $\pi_\theta^i$ . It receives the observation  $\mathbf{z}^{t,i}$  and messages  $\mathbf{m}^{t,j}$  and emits a message  $\mathbf{m}^{t,i}$  and action  $\mathbf{a}^{t,i}$ . The action feeds into the control node, which emits motor commands  $\mathbf{v}^{t,i}$ .

2) *World*: The world is everything external to the agent. The world can be either *real*, *simulated*, or a mix of both. In the real-world, an external system like GPS or motion capture produces state estimates for the agents. In the simulated world, a rigid-body dynamics simulator receives agent control commands and moves the agents in simulation accordingly. All sim-to-real abstraction is contained within the world, so the agents are unaware if they are operating in the real-world or the dynamics simulator.

The state server is a state machine that coordinates asynchronous episode execution and resets between independent agents. It enables back-to-back episodes and large-scale experimental data collection. It records agent heartbeats, then broadcasts a global operating mode and initial conditions. Agents use the global operating mode to determine if they should reset or execute the policy.

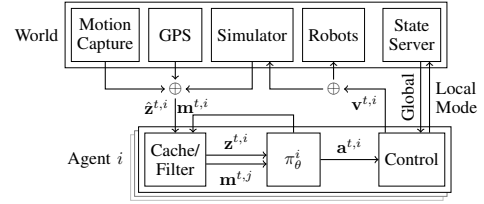


Fig. 4. Our ROS2 architecture is composed of the *world* and *agents*. There is one agent  $i = 1$  in the centralized case, and multiple  $i \in \{1 \dots n\}$  in the decentralized case. The agents receive sensor information  $\hat{\mathbf{z}}_t$  from either the motion capture system, GPS, or simulator. The aggregator combines sensor information with messages  $\mathbf{m}^{t,i}$  to produce observation  $\mathbf{z}^{t,i}$  and neighborhood messages  $\mathbf{m}^{t,j}; j \in \mathcal{N}^{t,i}$ , for the policy  $\pi_\theta^i$  to generate action  $\mathbf{a}^{t,i}$ . The control node converts the action into velocity commands  $\mathbf{v}^{t,i}$ . In simulation mode, control drives the simulator instead of the robot wheel motors. The state server orchestrates termination, resets, and operational mode syncs during sequential episodes. This system allows us to run agents in simulation and the real-world concurrently, over multiple episodes, and without any human intervention.

#### B. Communications Networks

Our evaluations consider four different configurations, as summarized in Fig. 3, which take the form of variable execution locations (i.e., offboard vs onboard) for policies, and the networks used for messaging between agents and policy execution points. Centralized and Offboard run policies on an external computer, with the remaining two *Onboard* configurations running them on robots' computers. These varied modes allowed us to separate sources of error and performance drop during evaluation.

Our framework uses two wireless communications methods over the various configurations. Both use 802.11, with the first being an *Infrastructure mode* network, and the second being *Adhoc mode*. We selected 802.11 in preference to other Adhoc capable wireless standards, such as IEEE 802.15 due to achievable data rates and compatibility with IP-based networking.

1) *Infrastructure Mode*: This mode is characterized by a central access point being responsible for managing the network's functions. For Centralized and Offboard configurations only agent actions are sent, which is easily handled by the network. For the Onboard o/Infra configuration, agents forward messages to one another using this message, with observations from the agent location system sharing the network. Finally, in the Onboard o/Adhoc configuration, it

handles only the delivery of agent location observations. The implications of each of these modes are discussed further in Sec. V-A.

2) *Adhoc mode*: We use this network mode only in the Onboard o/Adhoc mode, where it handles messages between agents. Physically, this network is supported by distinct wireless transceivers carried with each agent, allowing fully decentralized operation. This network takes the form of an 802.11n IBSS, which means that no agent has any special priority access to the wireless medium. Note that this is *not* a mesh network, as there is no facility for multi-hop communications.

3) *ROS2 Middleware*: Communications with agents exclusively use ROS2 provided middleware for message passing, specifically the eProxima Fast-DDS implementation of RTPS. Due to the fact that we use dynamic agent discovery rather than setting explicit communications routes, an agent-based firewall is deployed to block RTPS messaging traffic from using the incorrect network interface.

## V. NETWORK INFRASTRUCTURE

We investigate the effects of networking, ROS2, and Fast DDS settings on performance. During evaluations, our primary metric is the probability of delays between packet transmission and reception for message transmission rates between 10 and 500 per second; where the optimal case is all messages delivered with no delay.<sup>3</sup>

We carry out all network-specific experiments on a platform of five Raspberry Pis spaced 2m to 10m away from each other, in the same lab environment described in Sec. VI.

a) *Multicast vs Unicast*: We use the eProxima DDS RTPS implementation [39], which defaults to using unicast (one-to-one) communications between publishers and subscribers. This allows reliable transport protocols; however when multiple subscribers are active, the publisher will send duplicate messages as many times as there are subscribers, leading to exponential increases in messaging rates with increasing agent counts. The alternative is to use *Multicast*, where each publisher sends only one wireless broadcast for each message. The drawback is that neither reliable transport protocols or the 802.11 hardware based re-transmission mechanism can be utilised, reducing the odds of a given message being delivered.

b) *802.11 hardware retries*: When using 802.11, as the number of competing nodes goes up, the probability of any given packet surviving transmission goes down. This is because if two or more nodes transmit at the same time, both packets are lost, and there is no coordination mechanism. For unicast messages, the lack of an acknowledgement from the receiver will cause re-transmission attempts up to a limit. This limit defaults to 7, and we evaluate the performance of 1, 3, 5 and 7 in our testing. We focus upon lower settings than default because these reduce contention.

c) *Wireless adapter selection and channel bandwidth*: The data rate of the network is dependent upon the distance between participants, transmit power, receiver sensitivity, 802.11 version and channel bandwidth. The final configuration used the Netgear A6210 adapter, based upon the MediaTek MT7612U chipset, and a 40MHz channel width. The adapter was selected because it runs a recent 802.11 version (802.11ac) and had Linux driver compatibility with

IBSS mode. This adapter has a maximum transmit power of 18dbm; this is sufficient for ranges as high as 20m between robots even in the presence of interference from neighbouring 802.11 traffic, though communications over 200m were possible in quiet environments.

d) *Reliability*: When RTPS-based reliability is enabled through the ROS2 configuration, subscribers notify publishers when messages are not received as expected through different mechanisms. One of these is the use of positive acknowledgements by subscribers, which allows publishers to re-transmit when messages are lost, but causes subscribers to generate additional packets.

## A. Results

a) *Multicast and 802.11 Retries*: Fig. 5 displays unicast-only performance with dashed lines denoting default 802.11 and RTPS settings at 200 messages per second. With default settings, only 44% of all messages are delivered, along with consistently higher delays. Even at messaging rates as low as 20 per second, delays remain highly variable and can exceed the interval between policy executions. We found the reduced latency of multicast operation generally performs better than unicast at similar rates despite lower packet delivery rates. Reducing 802.11 hardware retries to one reduces latency using RTPS unicast defaults (Fig. 5).

Overall, using multicast, an 802.11 hardware retry setting of one and using RTPS's reliability mechanism results in the most favourable performance, delivering approximately 84% of messages within 20 ms in the Onboard o/Adhoc setup, with the remainder being lost. Fig. 5 highlights the relative latency stability of the chosen scheme where packets are either delivered at low latency, or fail to be delivered at all.

b) *Reliability*: We found disabling positive acknowledgements reduces messaging delays due to the reduction in network contention.

c) *Applicability*: Any adapter conforming to the same 802.11 revision with similar transmit powers and antennas should perform similarly to the presented results. Achievable inter-robot range is limited by wireless interference. If an increased number of robots are contending for airspace, the total number of packets per second achievable will reduce.

## VI. CASE STUDY: NAVIGATION THROUGH PASSAGE

We showcase the capabilities of our framework in a case-study requiring tight coordination between multiple mobile robots. We consider a team of  $n = 5$  agents that start in a cross-shaped formation and need to move through a narrow passage to reconfigure on the other side of the wall, as seen in Fig. 1. The robots are required to reach their goal positions through collision-free trajectories. Each robot only has knowledge of its own position and goal (i.e., does *not* directly observe the other robots), and is trained to leverage a GNN-based communication strategy to share this local information with neighbors to find the fastest collision-free trajectory to its respective goal. An image of this setup can be seen in Fig. 1 and a video demonstration is available online<sup>1</sup>. We briefly explain the training and provide the code with implementation details online.<sup>4</sup>

<sup>3</sup>Approximated in the Centralized and Offboard configurations

<sup>4</sup>[github.com/proroklab/rl\\_multi\\_agent\\_passage](https://github.com/proroklab/rl_multi_agent_passage)



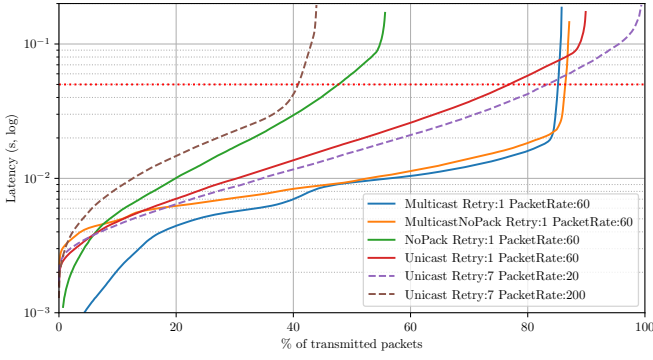


Fig. 5. CDF of network evaluation results. y-axis is the delay between one ROS node sending a message via RTPS and the destination receiving it. Four results show Unicast and Multicast performance with and without (NoPack) positive acknowledgements, using only one 802.11 retry at 60 messages per second (approximately the load during episode execution). For context, an additional two results are included which show Unicast performance using seven 802.11 retries at 20 and 200 messages per second, which are default settings. Results are from an 802.11 Adhoc setup using 5 agents sending messages to every other agent, such that the total rate of messages sent is as specified. The dotted line at 50ms indicates our approximate acceptable limit for latency, above which there is a risk that messages will arrive later than the intended model execution round.

a) *Environment*: At each discrete time step  $t$ , each agent  $i$  has a position  $\mathbf{p}^{t,i}$ , a desired velocity  $\mathbf{v}_d^{t,i}$ , a measured velocity  $\mathbf{v}_m^{t,i}$ , and a desired acceleration  $\mathbf{a}_d^{t,i}$ . We approximate each agent to be circular and implement a simple holonomic motion model that integrates acceleration-constrained velocities into positions. Collisions between agents and the wall result in an immediate stop of the agent. Note that the desired velocity is dictated by the control policy and the measured velocity is current true velocity of the agent. Each agent is assigned a goal position  $\mathbf{p}_g^i$ . An episode ends if all agents have reached their goal or after the episode times out.

b) *Reward*: We train agents using RL. The objective of each agent is to reach its goal position  $\mathbf{p}_g^i$  as quickly as possible while avoiding collisions. We use a shaped reward that guides individual agents to their respective goal positions as quickly as possible while penalizing collisions.

c) *Observation and Action*: The observation  $\mathbf{z}^{t,i}$  consists of locally available information, specifically the absolute position  $\mathbf{p}^{t,i}$ , the relative goal position  $\mathbf{p}_g^i - \mathbf{p}^{t,i}$ , as well as a predicted position  $\mathbf{p}^{t,i} + \mathbf{v}^{t,i}$ . The desired velocity is the policy's action output  $\mathbf{v}_d^{t,i} = \mathbf{a}^{t,i}$ . We constrain acceleration and velocity to  $a_{\max} = 1\text{m/s}^2$  and  $v_{\max} = 1.5\text{m/s}$ .

d) *Model*: As the model for the policy  $\pi_\theta^i$  we use the GNN introduced in Sec. III-A. The number of layers is constrained by our communication framework. Since more layers result in multiple rounds of communication exchanges at the same time step, we set  $k = 1$ . Each message is an encoding of the observation so that  $\mathbf{m}^{t,i} = \mathbf{h}_0^{t,i} = \theta_{\text{ENC}}(\mathbf{z}^{t,i})$ . We define our message function and vertex update function as  $M_1^\theta(\mathbf{h}_0^{t,i}, \mathbf{h}_0^{t,j}, \cdot) = \theta_{\text{GNN}}(\mathbf{h}_0^{t,i} - \mathbf{h}_0^{t,j})$  and  $U_1^\theta(\cdot, x) = \theta_{\text{ACT}}(x)$ . Furthermore, we include self-loops and thus consider agent  $i$  as part of its own neighborhood so that  $\mathcal{N}^{t,i} = \mathcal{N}^{t,i} \cup \{i\}$ . The output of the GNN is the desired velocity  $\mathbf{v}_d^{t,i} = \mathbf{a}^{t,i} = \mathbf{h}_1^{t,i}$ .  $\theta_{\text{ENC}}$ ,  $\theta_{\text{GNN}}$  and  $\theta_{\text{ACT}}$  are learnable Multilayer Perceptrons (MLPs). We use the same approach as described in [13] to train our model using PPO with local rewards for each agent.

e) *Experimental Setup*: In total, we run a series of six different real-world experiments for the four modes

(Fig. 3) to demonstrate the capabilities and performance of our framework and two additional experiments to demonstrate the robustness of our policy against changes to the communication radius in the real-world. In addition to using a set communication radius of  $R_{\text{COM}} = 2\text{m}$ , we (i) run the policy in a fully connected communication topology, and (ii) run the policy in a noisy communication topology by modeling the communication range as a Gaussian with a mean of  $R_{\text{COM}} = 2\text{m}$  and a standard deviation of  $0.5\text{m}$  (the policy is trained with  $R_{\text{COM}} = 2\text{m}$ ).

To collect a statistically significant amount of data, we generate  $E = 16$  episodes, each with a different set of random start and goal positions, and repeat each episode for each experiment  $K = 12$  times, resulting in  $K \cdot E$  episodes in the training environment (simulation) and on real robots. We use customized DJI RoboMaster robots equipped with Raspberry Pi's that locally run policies. The robots are provided with state information as explained in Sec. IV-B.1.

## A. Results

We use two metrics to evaluate the performance of our model in simulation and real-world. The *success rate* is the fraction of collision-free episodes for which all robots reached their goal. The *makespan* is the time it takes for the last agent to reach its goal. For both metrics, episodes with wall or inter-agent collisions are excluded. Inter-agent collisions are defined as two agents approaching each other closer than  $0.32\text{m}$ . We compare to a simulation baseline, for which the policy is evaluated during training conditions. We show distributions of makespans and positions in Fig. 6 and show quantitative results in Tab. I.

The Centralized case reflects the performance gap caused by dynamic constraints that are not considered in simulation. Since the GNN is evaluated synchronously, communication is not affected by real-world effects. The makespan is about 1.5 times worse and the success rate 5.7 percentage points (pp) worse than in simulation.

The Offboard mode evaluates the GNN asynchronously across different processes on the same physical computer. Compared to the Centralized mode, it features *asynchronous* evaluation but little to no inter-process communication delays, resulting in slightly worse performance of 4.2 pp and worse median makespan of  $0.2\text{s}$  wrt the Centralized mode.

The Onboard o/Infra mode moves the decentralized GNN from a central computer to the on-board computers of each individual robot and therefore adds communication delays caused by wireless routing and contention. We notice a decrease in performance of 26.5 pp in terms of success, and a deterioration of  $1.0\text{s}$  of median makespan w.r.t. the Offboard mode. Onboard o/Adhoc mode improves the performance by 10.9 pp, with a similar median makespan. This can be attributed to less contention.

Setting  $R_{\text{COM}} = \infty$  results in an identical median makespan and a slight decrease in performance of 3.6 pp. This decrease is expected due to out-of-distribution neighborhoods that never occur during training (while the agents are typically fully connected in the start and the beginning of each episode, they are not when moving through the passage). When adding noise to the communication range, the success rate drops by another 4.2 pp (or 7.8 pp wrt the Onboard o/Infra mode) and  $0.9\text{s}$  median makespan.

The second and third row in Fig. 6 visualize distributions over positions. The second row shows that the distribution of

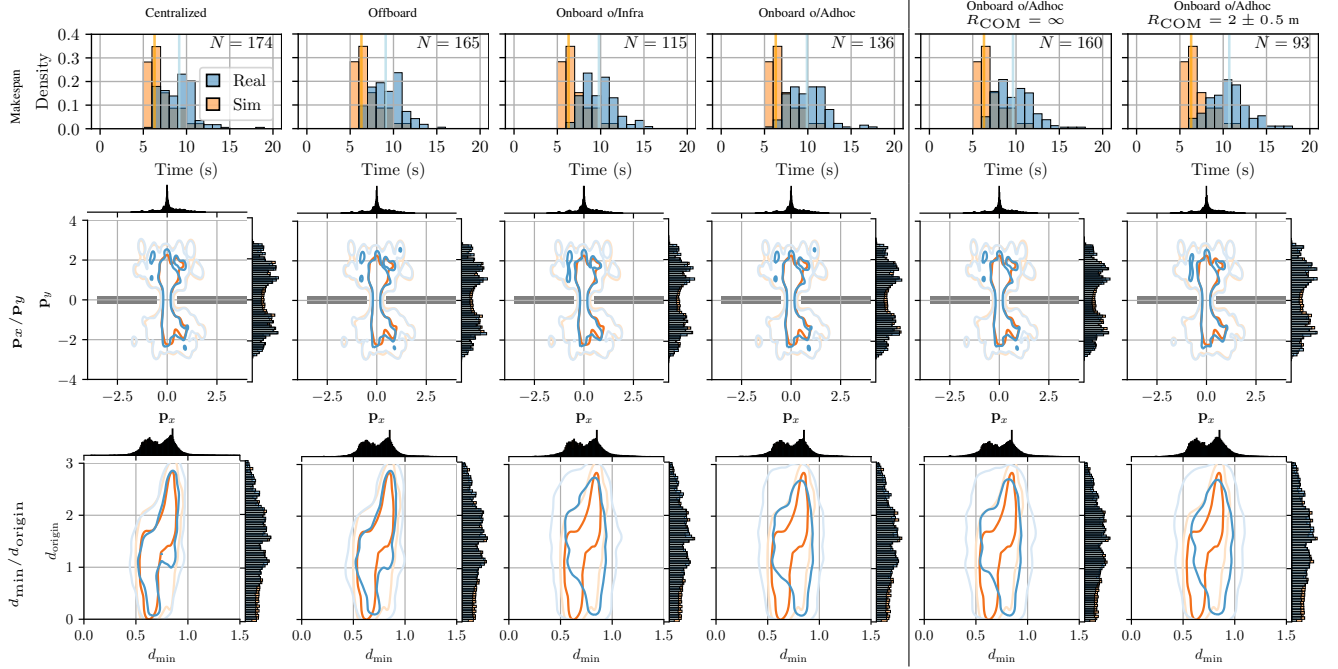


Fig. 6. We visualize a variety of makespan and position distributions over the six experiments we conducted. The columns show the data of the centralized simulation baseline in orange and the data of the corresponding real-world experiment as labeled in the column headers in blue. For each experiment, we run a total of 192 episodes with 16 different start and goal positions. The last column compares the Onboard o/Adhoc experiment with a simulation evaluated with communication delays. The first row shows the probability densities of makespans of successful episodes (episodes that did not result in a collision with the wall and for which all robots reached their goal, indicated with  $N$ ). The median makespan is indicated with a dashed line. The second row shows the distribution of positions, indicating the position of the wall and the passage. The third row shows the distribution of minimum distances between robots at each time step  $d_{\min}$  and distance from the origin or passage  $d_{\text{origin}}$ .

TABLE I  
OVERVIEW OF PERFORMANCE METRICS FOR ALL CASE STUDY EXPERIMENTS.

	simulation	Centralized	Offboard	Onboard o/Infra	Onboard o/Adhoc	Onboard o/Adhoc $R_{\text{COM}} = \infty$	Onboard o/Adhoc Noise
Success Rate	95.8%	90.1%	85.9%	59.4%	70.3%	66.7%	62.5%
Median Makespan	6.3 s	9.1 s	9.1 s	9.8 s	9.9 s	9.6 s	10.7 s

absolute positions over all experiments are consistent, even when comparing to the centralized simulation. In the third row, we compare the distribution of distance to the origin (or the passage)  $d_{\text{origin}}$  over minimum distance between agents  $d_{\min}$ . While the simulation and real distributions are overlapping in the Centralized and Offboard mode, there is a noticeable discrepancy in all Onboard modes, especially for small  $d_{\min}$ , for which  $d_{\text{origin}}$  is shifted towards higher values, indicating that the robots are further away from each other when close to the passage, which can be attributed towards slower reaction times caused by communication delays.

We run an additional simulation that evaluates the GNN in a decentralized mode with communication delays. We observed that for higher delays, the success rates dropped significantly, while the makespan decreased much less notably. The distribution of  $d_{\min}$  over  $d_{\text{origin}}$  shifted slightly towards the real-world distribution. This indicates that the shift in makespan we observe is mostly due to robot dynamics, and real-world communication latency causes the agents to be less responsive and therefore to collide.

## VII. DISCUSSION AND FURTHER WORK

This work is the first to demonstrate the real-world deployment of a GNN-based policy to a fully decentralized real-world multi-robot system using ROS2 and an Adhoc communication network. We performed a suite of exper-

iments that discuss the selection of suitable networking settings, and subsequently presented results on a real-world scenario requiring tight coordination amongst robots. Our results showed that our framework allows for the successful deployment of our control policy in an Adhoc configuration, albeit with a performance that is 22 pp worse in terms of success rate and 9 pp worse in terms of median makespan wrt the centralized mode.

Even though the deployment of our scenario was successful, we reported a degradation of performance when moving from simulation to the real world, which can be attributed to real-world effects such as communication delays. In the future, we plan to use our software framework to validate novel mechanisms that are robust to communications-specific domain shifts and thus aid in closing the sim-to-real gap for GNNs. We believe that the presented framework will facilitate the deployment of robot systems into more complex environments and the unstructured outdoors, potentially leveraging more complex networking architectures such as mesh networks and on-board sensing.

## ACKNOWLEDGMENT

J. Blumenkamp acknowledges the support of the ‘Studienstiftung des deutschen Volkes’ and an EPSRC tuition fee grant. We gratefully acknowledge the support of ARL grant DCIST CRA W911NF-17-2-0181, the EPSRC (grant EP/S015493/1), and ERC Project 949940 (gAia). The support of Arm is gratefully acknowledged.

## REFERENCES

- [1] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *European Conference on Artificial Intelligence*, 2014, pp. 961–962.
- [2] J. P. Van Den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 430–435.
- [3] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *International Joint Conference on Artificial Intelligence*. Barcelona, Catalonia, Spain: AAAI Press, 2011, pp. 668–673.
- [4] W. Wu, S. Bhattacharya, and A. Prorok, "Multi-Robot Path Deconfliction through Prioritization by Path Prospects," *IEEE International Conference on Robotics and Automation*, 2020, arXiv: 1908.02361.
- [5] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.
- [6] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar, "Graph policy gradients for large scale robot control," in *Conference on Robot Learning*, vol. 100. PMLR, 2020, pp. 823–834.
- [7] Q. Li, W. Lin, Z. Liu, and A. Prorok, "Message-aware graph attention networks for large-scale multi-robot path planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5533–5540, 2021.
- [8] L. Zhou, V. D. Sharma, Q. Li, A. Prorok, A. Ribeiro, and V. Kumar, "Graph neural networks for decentralized multi-robot submodular action selection," *arXiv preprint arXiv:2105.08601*, 2021.
- [9] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Conference on Robot Learning*, vol. 100. PMLR, 2020, pp. 671–682.
- [10] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.
- [11] R. Kortvelesy and A. Prorok, "ModGNN: Expert policy approximation in multi-agent systems with a modular graph neural network architecture," in *IEEE International Conference on Robotics and Automation*. IEEE, 2021.
- [12] S. D. Morad, S. Liwicki, and A. Prorok, "Graph convolutional memory for deep reinforcement learning," *arXiv preprint arXiv:2106.14117*, 2021.
- [13] J. Blumenkamp and A. Prorok, "The emergence of adversarial communication in multi-agent reinforcement learning," *Conference on Robot Learning*, 2020.
- [14] J. Blumenkamp, Q. Li, and A. Prorok, "Evaluating the sim-to-real gap of graph neural network policies for multi-robot coordination," in *IEEE International Conference on Robotics and Automation Real World Swarms Workshop*, 2021.
- [15] M. Calvo-Fullana, D. Mox, A. Pyattaev, J. Fink, V. Kumar, and A. Ribeiro, "Ros-netsim: A framework for the integration of robotic and network simulators," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1120–1127, 2021.
- [16] J. M. Gregory, J. N. Twigg, and J. R. Fink, "Enabling autonomous information-gathering and self-recovery behaviors in partially-known, communication-constrained environments," in *IEEE International Symposium on Safety, Security, and Rescue Robotics*. IEEE, 2016, pp. 92–99.
- [17] J. Stephan, J. Fink, V. Kumar, and A. Ribeiro, "Concurrent control of mobility and communication in multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1248–1254, 2017.
- [18] J. Stephan, J. Fink, B. Chawrow, A. Ribeiro, and V. Kumar, "Robust routing and multi-confirmation transmission protocol for connectivity management of mobile robotic teams," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 3753–3760.
- [19] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile emulab: A robotic wireless and sensor network testbed," in *IEEE International Conference on Computer Communications*. IEEE, 2006, pp. 1–12.
- [20] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "CrazySwarm: A large nano-quadcopter swarm," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 3299–3304.
- [21] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: A remotely accessible swarm robotics research testbed," in *IEEE International Conference on Robotics and Automation*. IEEE, 2017, pp. 1699–1706.
- [22] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.
- [23] J. A. Tran, P. Ghosh, Y. Gu, R. Kim, D. D'Souza, N. Ayanian, and B. Krishnamachari, "Intelligent robotic iot system (iris) testbed," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2018, pp. 1–9.
- [24] A. Stubbs, V. Vladimerou, A. T. Fulford, D. King, J. Strick, and G. E. Dullerud, "Multivehicle systems control over networks: a hovercraft testbed for networked and decentralized control," *IEEE Control Systems Magazine*, vol. 26, no. 3, pp. 56–69, 2006.
- [25] A. Prorok, J. Blumenkamp, Q. Li, R. Kortvelesy, Z. Liu, and E. Stump, "The holy grail of multi-robot planning: Learning to generate online-scalable solutions from offline-optimal experts," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2022.
- [26] B. Wang, Z. Liu, Q. Li, and A. Prorok, "Mobile robot path planning in dynamic environments through globally guided reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6932–6939, 2020.
- [27] M. Zhou, J. Luo, J. Villella, Y. Yang, D. Rusu, J. Miao, W. Zhang, M. Alban, I. Fadar, Z. Chen, A. C. Huang, Y. Wen, K. Hassanzadeh, D. Graves, D. Chen, Z. Zhu, N. Nguyen, M. Elsayed, K. Shao, S. Ahilan, B. Zhang, J. Wu, Z. Fu, K. Rezaee, P. Yadmellat, M. Rohani, N. P. Nieves, Y. Ni, S. Banijamali, A. C. Rivers, Z. Tian, D. Palenicek, H. bou Ammar, H. Zhang, W. Liu, J. Hao, and J. Wang, "Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving," in *Conference on Robot Learning*, 11 2020.
- [28] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016, pp. 2137–2145.
- [29] M. Lee, G. Yu, and H. Dai, "Decentralized inference with graph neural networks in wireless communication systems," *arXiv preprint arXiv:2104.09027*, 2021.
- [30] A. Frotzsch, U. Wetzker, M. Bauer, M. Rentschler, M. Beyer, S. Elspass, and H. Klessig, "Requirements and current solutions of wireless communication in industrial automation," in *IEEE International Conference on Communications Workshops*, 2014, pp. 67–72.
- [31] S.-c. Wang and A. Helmy, "Performance limits and analysis of contention-based IEEE 802.11 MAC," in *IEEE Conference on Local Computer Networks*, 2006, pp. 418–425.
- [32] V. Bhargava and N. Raghava, "Improve collision in highly dense wifi environment," in *IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems*, 2018, pp. 1–5.
- [33] J. Gielis and A. Prorok, "Improving 802.11p for delivery of safety-critical navigation information in robot-to-robot communication networks," *IEEE Communications Magazine*, vol. 59, no. 1, pp. 16–21, 2021.
- [34] H. Kang, H. Kim, and Y. M. Kwon, "Recen:resilient manet based centralized multi robot system using mobile agent system," in *IEEE Symposium Series on Computational Intelligence*, 2019, pp. 1952–1958.
- [35] T. Kronauer, J. Pohlmann, M. Matthe, T. Smejkal, and G. Fettweis, "Latency analysis of ROS2 multi-node systems," 2021.
- [36] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1263–1272.
- [37] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al., "Ros: an open-source robot operating system," in *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.
- [38] Open Robotics, "ROS 2 Design," <https://design.ros2.org/>, 2021, [Online; accessed 12/09/21].
- [39] eProsima, "Fast-DDS," <https://github.com/eProsima/Fast-DDS>, 2020, [Online; accessed 14/09/21].