Evaluating the Sim-to-Real Gap of Graph Neural Network Policies for Multi-Robot Coordination

Jan Blumenkamp, Qingbiao Li and Amanda Prorok

***Abstract*— Graph Neural Networks (GNNs) are a paradigm-shifting neural architecture that facilitates the learning of complex multi-agent behaviors. Recent work has demonstrated remarkable performance in tasks such as multi-agent path planning and cooperative coverage. However, the policies derived through GNN-based learning schemes have not yet been validated in the real world on physical multi-robot systems. In this paper, we report pioneering experiments that evaluate the performance of GNN-based policies on a multi-robot coordination task that requires a team of five robots to navigate through a narrow passageway. We demonstrate that our policy, which is trained using Reinforcement Learning (RL) in simulation, is capable of being deployed to the real world with a minor drop in performance compared to simulation. These experiments promise to be the foundation for future experiments, which aim to identify sim-to-real challenges in GNNs. To that end, we investigate and characterize the impact of the communication radius and random communication message dropouts to evaluate the robustness of our framework.**

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

I. Introduction

Efficient and collision-free navigation in multi-robot systems is fundamental to advancing mobility in guiding robots from their origins to designated destinations, for example in industrial plant inspection, item retrieval in warehouses [1], and transportation with self-driving cars in smart cities [2]. Finding an optimal solution to such problems is often not tractable in polynomial time.

Recent work is actively investigating data-driven decentralized methods to approach optimal but costly algorithms by offloading the online computational burden to an offline learning procedure [3], [4], [5]. Particularly, Graph Neural Networks (GNNs) have demonstrated remarkable performance results in simulation and have been shown to generalize well to large-scale robotic teams [6], [5], [7]. However, training neural networks on real robots is a time-consuming and costly process, which yields the need to explore solutions to bridge the gap from simulation into the real world (sim-to-real) using approaches such as zero-shot transfer [8], domain adaptation [9] and domain randomization [10].

Most recent works in the field of sim-to-real have focused on the field of robot vision [11], [12], [13]. Yet, to the best of our knowledge, no work has been conducted on sim-to-real for GNNs. Current GNN models are usually trained with the assumption of synchronous communication, which is hard to ensure in real-world settings. Other challenges

All authors are with Department of Computer Science and Technology, University of Cambridge. (e-mail: {`jb2270`, `ql295`, `asp45`}`@cam.ac.uk`).

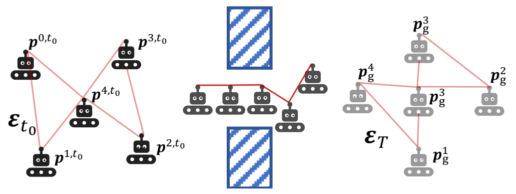


Fig. 1. A robot formation has to reconfigure to move through a narrow passage to the other side of the wall as quickly as possible.

include message dropouts, communication delays, and noise. A clear performance metric is essential to investigate the sim-to-real gap, which is difficult in other common multi-agent problems such as flocking. We suggest the following steps to tackle this problem: *(i)* First, a suitable scenario with a clear metric to evaluate the performance of deployed policies in both simulation and in reality has to be designed. *(ii)* Next, the parameters that contribute to the sim-to-real gap, but are not necessarily caused by communication, have to be isolated. *(iii)* After all variables that contribute to the sim-to-real gap but are not part of the communication component are isolated, the impact on changing parameters related to the communication has to be investigated and evaluated. *(iv)* Lastly, methods to minimize the sim-to-real gap can be developed based on these results. In this work, we report the intermediate results after step *(ii)*. In a future work, we intend to extend this work to utilize truly decentralized real-world communication to further investigate the sim-to-real gap for GNNs.

II. Related Work

Within the field of multi-robot navigation, formation control and reconfiguration remain challenging in complex, yet static, environments. For example, robots need to reconfigure the formation to pass through a narrow corridor and rejoin as a team after the passageway. Centralized approaches [14] can be deployed to compute a global and optimal path with a global knowledge of positions, destinations, and environment maps, or by setting up intermediate collision-free configurations for the team of robots. However, this method fails in scaling up to larger teams. *Decentralized* approaches provide an attractive alternative to centralized approaches, because they can reduce the computational overhead, and relax the dependence on centralized units. In prioritized planning [15], [16], each robot is assigned a unique priority, and the algorithm proceeds sequentially from the highest priority robot to the lowest priority one. However, these methods can only obtain sub-optimal solutions.

Learning-based methods. Rapid development and wide adoption of deep learning-based methods draw the attention of roboticists to investigate methods to find near-optimal solutions, which have been shown to be effective at robot control policies [17], manipulation [11] and multi-robot coordination [3], [18]. In the field of multi-robot coordination, learning-based decentralized framework have been actively investigated to approach near-optimal solutions given partial observation constraints. Sartoretti et al. [3] propose a hybrid learning-based method called PRIMAL for multi-agent path-finding that uses both Imitation Learning (IL) and multi-agent Reinforcement Learning (RL), which achieved high success rate in various team size in low obstacle density. However, each robot can only sense the positions of other agents within the observation range by achieving implicit coordination with only local information.

Of particular interest to us is how explicit inter-robot communication [19], [5] plays a role in accumulating information from other robots to achieve near-optimal decentralized performance. To achieve this, GNNs [20] are introduced to capture information that enables complex inter-robot coordination [6], [4], [5], [21], where each robot is considered a node in the graph and the relationship between robots is represented by an adjacency matrix as a distance-based communication graph. The essential function of GNNs is to enable explicit communication with nearby neighbors only that operate only on local data, hence resulting in fully decentralizable policies.

Sim-to-real in robotics. In robotics, conventional learning-based methods commonly have low sample efficiency which compounds the problem of learning in the real world. Vanilla deep RL algorithms are constrained by their low sample efficiency, while IL approaches required a huge amount of data on real robots, but it is costly to collect [3], [18]. To circumvent this challenge, one can leverage the power of simulation to produce large amounts of labeled data. Yet, the sim-to-real gap leads to failure when working with physical robots. This yields the need to explore solutions to bridge this gap using methods such as zero-shot learning [8], domain adaptation [9] and domain randomization [10]. These methods have demonstrated their performance in robot manipulation [11], using a robotic hand to solve Rubik's Cube [12], or in self-driving car using IL [13].

Contributions. There is a gap in the investigation of the impact of communication using GNNs on real robots. In this workshop paper, we conduct a series of experiments to evaluate how well a trained policy can be transferred from simulation to reality, without adaption, and evaluate in what cases it fails. Towards this end, we design a multi-agent navigation scenario to demonstrate how our GNNs model performs in a physical experiment:

- We set up a challenging scenario where multiple robots are navigating through a narrow passageway;
- We implement a decentralized framework based on GNNs and evaluate the trained policies on real-world multi-robot coordination tasks;
- Our results demonstrate that our model is capable in learning general knowledge of robot motion as it achieves a marginal performance drop from simulation to reality, as discussed in Sec. VI.

III. Problem Statement

We design a passage scenario to evaluate our policy for a multi-robot coordination problem. A schema of a sample setup can be seen in Fig. 1. We consider a team of N agents (i.e., $N = 5$) moving through a passageway to reconfigure on the other side of the wall and reach their goal positions in collision-free trajectories. We build a continuous environment of size $(W_{\text{env}}, H_{\text{env}})$, which is separated with a wall of thickness H_{wall} into two sections of identical area connected by a small passage of width W_{wall}.

In this environment, we assume that each robot can obtain its own state (such as global position) but can not obtain the state of other robots in the environments directly. Therefore, the inter-robot relationship 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 $\mathcal{V} = \{1, \ldots, N\}$ and the communication links between robots are represented as edge set $\mathcal{E} = \mathcal{V} \times \mathcal{V}$. If robot j is in communication range R_{com} of robot i, it is in robot i's neighborhood $j \in \mathcal{N}^{i,t}$ and robot i can obtain robot j's information via inter-robot communication.

We consider each agent to be circular with a radius of R_{agent}. At each discrete time step t, each agent i has a position $\mathbf{p}^{i,t} \in \mathbb{R}^2$, a desired velocity $\mathbf{v}_d^{i,t} \in \mathbb{R}^2$, a measured velocity $\mathbf{v}_m^{i,t} \in \mathbb{R}^2$, and a desired acceleration $\mathbf{a}_d^{i,t} \in \mathbb{R}^2$. Let $\text{clamp}(a, b, c) = \max(a, \min(b, c))$, $a \in \mathbb{R}$, $b \in \mathbb{R}$, $c \in \mathbb{R}^M$ where max and min are applied element-wise if $M > 1$. We compute the desired acceleration that is required to apply the desired velocity $\mathbf{v}_d^{i,t}$ and constrain it to $a_{\max}$ as

$$\mathbf{a}_d^{i,t} = \text{clamp}\left(-a_{\max}, a_{\max}, \frac{\mathbf{v}_d^{i,t} - \mathbf{v}_m^{i,t-1}}{\Delta t}\right) \quad (1)$$

We constrain the agents from moving into the wall or into other agents (meaning the closest point each agent can approach another agent or the wall is constrained by agents' radius R_{agent}). If the agent would collide by applying the measured velocity, we set it to zero, as follows:

$$\mathbf{v}_m^{i,t} = \begin{cases} [0, 0]^\intercal & \text{if collision} \\ \text{clamp}\left(0, v_{\max}, \mathbf{v}_m^{i,t-1} + \mathbf{a}_d^{i,t}\Delta t\right) & \text{else,} \end{cases} \quad (2)$$

Eventually, the position is updated over Δt as $\mathbf{p}^{i,t} = \mathbf{p}^{i,t-1} + \mathbf{v}_m^{i,t}\Delta t$ while constraining it to the environment dimensions $(W_{\text{env}}, H_{\text{env}})$. Note that the desired velocity is dictated by a control policy, and the measured velocity is the velocity with which the agent has moved.

Each agent is assigned a goal position $\mathbf{p}_g^i \in \mathbb{R}^2$. Let the set of agents that have reached the goal be $\mathcal{V}_g^{t=0} = \emptyset$. We update this at every time step:

$$\mathcal{V}_g^t = \mathcal{V}_g^{t-1} \cup \{i \mid i \in \mathcal{V} \wedge ||\mathbf{p}^{i,t} - \mathbf{p}_g^i|| < d_g\}. \quad (3)$$

An episode ends if $i \in \mathcal{V}_g^t \; \forall i \in \mathcal{V}$ or $t > T_{\max}$, where $T_{\max}$ is the horizon for single episode.

IV. Preliminaries

A. Graph Neural Networks

In this paper, we use the ModGNN framework [7], whose layers includes a message aggregation module (4) to transmit data and a node update module (6) to compute the output. Here, we use a single-layer GNN and $K = 2$ communication hops. In the message aggregation module, the agent i receives a set of all transmitted data from the kth-hop neighborhood at time t, as $\mathbf{y}_{(k)}^{i,t}$, which can be formulated as:

$$\mathcal{Y}_{(k)}^{i,t} = \left\{ \mathbf{y}_{(k)}^{ij,t} \;\middle|\; j \in \mathcal{N}^{i,t} \right\}. \tag{4}$$

where $\mathbf{y}_{(k)}^{ij,t}$ is the aggregated summation of the observation $\mathbf{z}$ of agent i and its kth-hop neighborhood at time t processed by f_{com} as follows:

$$\mathbf{y}_{(k)}^{ij,t} = \sum_{\mathbf{z} \in \mathcal{Y}_{(k-1)}^{j,t-1}} f_{\text{com}}(\mathbf{z}). \tag{5}$$

Note that, we use $\mathcal{Y}_{(0)}^{i,t} = \{\mathbf{x}^{i,t}\}$ when the agent i only uses local information, and $\mathcal{Z}^{i,t} = \{\mathcal{Y}_{(k)}^{i,t} | k \in [0..K]\}$ is the set of transmitted data from all neighboring agents up to K hops.

Then, the aggregated information $\mathcal{Z}^{i,t}$ is the the node input to be fed into node update module, consisting of three customized sub-modules, as follows:

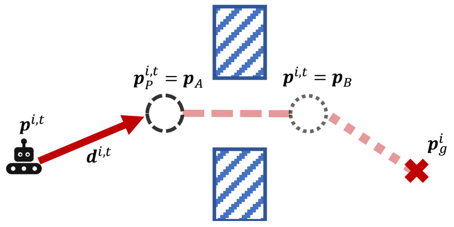
$$\mathbf{x}^{i,t} = f_{\text{final}}\left(\sum_{k=0}^{K} \left[f_{\text{mid}}^{(k)} \left(\sum_{\mathbf{z} \in \mathcal{Y}_{(k)}^{i,t}} f_{\text{pre}}^{(k)} \left(\mathcal{Z}^{i,t} \right) \right) \right] \right) \tag{6}$$

where f_{pre} combines the states from the neighbors $\mathcal{N}_i(t)$, f_{mid} aggregates information from up to $K+1$ neighborhoods, and f_{final} intersperse throughout the model from the aggregation steps to the target space.

B. Reinforcement Learning

We use RL to train robots to navigate a passageway as illustrated in Fig. 1 and use the same approach as detailed in [21]. We formulate a stochastic game defined by the tuple $\langle \mathcal{V}, \mathcal{S}, \mathcal{A}, P, \{R^i\}_{i \in \mathcal{V}}, \{\mathcal{Z}^i\}_{i \in \mathcal{V}}, Z, \mathcal{T}, T \rangle$, in which N agents identified by $i \in \mathcal{V}$ choose sequential actions. The environment has a true state $\mathbf{s}^t \in \mathcal{S}$. At each time step t, each agent applies action $\mathbf{a}^{i,t} \in \mathcal{A}$, forming a joint action $\mathbf{a}^t \in \mathcal{A}^N$. The state transition probability function is $P(\mathbf{s}^{t+1}|\mathbf{s}^t, \mathbf{a}^t) : \mathcal{S} \times \mathcal{A}^N \times \mathcal{S} \to [0, 1]$. We consider a partially observable setting (since agents can only communicate to agents within communication range without guarantees that the state of all agents is observable at any time), in which each agent i draws observations $\mathbf{z}^{t,i} \in \mathcal{Z}^i$ according to the observation function $Z^i(s^t) : \mathcal{S} \to \mathcal{Z}$, forming a joint observation $\mathbf{z}^t \in \mathcal{Z}^N$. Agents observe a local reward $r^{i,t}$ drawn from $R^i(\mathbf{s}^t, \mathbf{a}^{i,t}) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (i.e., it is not a global reward given to the whole team). The agents' communication topology at time step t, denoted by $\mathcal{E}_t \in \mathcal{T}$, is drawn according to $T(\mathbf{s}^t) : \mathcal{S} \to \mathcal{T}$.

Each agent learns a local policy $\pi_{\theta^i}^i(\mathbf{a}^{i,t}|\mathbf{z}^{i,t}) : \mathcal{Z} \times \mathcal{A} \to [0, 1]$ parameterized by a local set of parameters θ^i. The induced joint policy is $\boldsymbol{\pi}_\theta(\mathbf{a}^t|\mathbf{z}^t) = \prod_{i \in \mathcal{V}} \pi_{\theta^i}^i(\mathbf{a}^{i,t}|\mathbf{z}^{i,t})$ with



$\mathbf{p}_P^{i,t} = \mathbf{p}_A$
$\mathbf{p}^{i,t} = \mathbf{p}_B$
$\mathbf{p}^{i,t}$
$\mathbf{d}^{i,t}$
$\mathbf{p}_g^i$

Fig. 2. To guide each robot through the passage to its goal, we use a shaped reward we refer to as guidance reward. The path is separated into two waypoints to guide the robot towards the goal. As the robot approaches one waypoint, it moves to the next in sequence. The reward is always with respect to the next waypoint.

$\theta = \{\theta^i\}_{i \in \mathcal{V}}$. The policy of agent i can be improved through gradient ascent using the policy gradient

$$g^i = \mathbb{E}_t \left[\nabla_\theta \log \pi_{\theta^i}^i(\mathbf{a}^{i,t}|\mathbf{z}^{i,t}) \hat{A}^i(\mathbf{s}^t) \right]. \tag{7}$$

V. Approach

A. Training

The objective of each agent is to reach its goal position $\mathbf{p}_g^i$ as quickly as possible while avoiding collisions with other agents and the wall. For the agent to reach the goal, we specify two additional waypoints $\mathbf{p}_A$ and $\mathbf{p}_B$ that help each individual agent to be guided to its goal (refer Fig. 2). Let $\mathbf{p}_P^{i,t}$ be the next waypoint for agent i at time t and $\mathbf{d}^{i,t} = \mathbf{p}^{i,t} - \mathbf{p}_P^{i,t}$ the vector to the next waypoint. Note that $\mathbf{p}_P^{i,t}$ is initialized to $\mathbf{p}_A$ and will be set to the next waypoint $\mathbf{p}_B$ and eventually the goal $\mathbf{p}_g^i$ if the current waypoint is approached within a distance $\|\mathbf{d}^{i,t}\| < d_P$. The guidance reward $r_d^{i,t}$ is then defined as

$$r_d^{i,t} = \left(\frac{\mathbf{d}^{i,t}}{\|\mathbf{d}^{i,t}\|} \cdot \frac{\mathbf{v}_m^{i,t}}{\|\mathbf{v}_m^{i,t}\|} \right) \|\mathbf{v}_m^{i,t}\| \tag{8}$$

and intuitively rewards fast movements towards the next waypoints and penalizes moving away from it. The collision reward $r_c^{i,t}$ is defined as

$$r_c^{i,t} = \begin{cases} -1.5 & \text{if collision with other agent} \\ -0.25 & \text{if collision with wall} \\ 0 & \text{else} \end{cases} \tag{9}$$

and the goal-reached reward $r_g^{i,t}$ as

$$r_g^{i,t} = \begin{cases} 10 & \text{if } i \in \mathcal{V}_g^t \text{ and } i \notin \mathcal{V}_g^{t-1} \\ 0 & \text{else} \end{cases} \tag{10}$$

resulting in the total reward $r^{i,t} = r_d^{i,t} + r_c^{i,t} + r_g^{i,t}$. We found that the highly shaped reward leads to significantly better performance and faster training than a more sparse reward formulation.

The observation $\mathbf{z}^{i,t}$ consists only of local information, specifically the absolute position $\mathbf{p}^{i,t}$ and the relative goal position $\mathbf{p}_g^i - \mathbf{p}^{i,t}$, which are concatenated to a single vector so that $\mathbf{z}_t^i \in \mathbb{R}^4$. Note that the wall is always located at the same absolute position, and our policy would easily generalize to a variable passage position. The desired velocity is the policy's action output $\mathbf{v}_d^{i,t} = \mathbf{a}^{i,t}$.

For each episode, the initial positions and the goal positions are initialized randomly. We train with $N = 5$ agents in a $+$-shaped formation both for start and goal positions. The relative positions of the agents to each other are always identical, but the starting point as well as rotation of the initial and goal positions are randomized according to a uniform distribution chosen so that the initial positions will never be outside the environment boundaries and never in conflict with the wall while covering the whole area on both sides of the wall.

B. Implementation

We choose f_{com} to subtract incoming messages from its own state, where f_{pre} and f_{pre} consists of four fully connected MLP with 64 neurons each and f_{mid} of four layers with 128 neurons each. We use the same approach as described in [21] to train our model (PPO with local rewards for each agent). During training, we train with infinite acceleration $a_{\max} = \infty$, which means that the desired velocity is directly applied to the position if doing so does not lead to a collision. We attempted to train with acceleration constraints, but found that the policy performed poorer in the real world compared to training without such constraints. We set the radius of each agent to $R_{\text{agent}} = 0.3$ m, which is significantly larger than the real robots, but we found that increasing this value to a larger value counteracts training without acceleration constraints. We set the width of the passage to $W_{\text{wall}} = 1.0$ m and the thickness to $H_{\text{wall}} = 0.5$ m. We train with a communication radius of $R_{\text{com}} = 2.0$ m and run the control loop at 20 Hz, resulting in $\Delta t = 1/20$ s.

VI. Experiments

A. Metrics

We use two metrics to evaluate the performance of our model in simulation and real world.

The success rate is the fraction of episodes in which no failure occurred, a failure being a collision with the wall or between any two given agents at any point in time. This means only if there are no collisions at all, an episode will be counted as success. A robot is considered having collided with another robot if the distance between any two robots is smaller than $d_{\min} = 0.29$ m or if the distance between any robot and the wall is smaller than $d_{\min} = 0.16$ m. For the purposes of this evaluation, we decrease the robot radius to $R_{\text{robot}} = 0.12$ m to reflect the size of the real robot.

The makespan is the time it takes for the last agent to approach its goal to a distance smaller than d_g ($\mathcal{V}_g^i = \mathcal{V}$). We consider the distribution of makespans over all episodes as well as the mean makespan. We count non-successful episode runs as makespans of duration $T_{\max} = 25$ s.

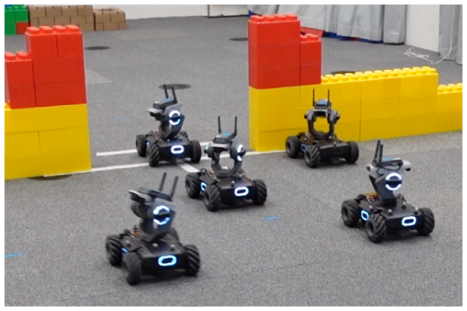


Fig. 3. Five RoboMaster robots reconfiguring their formation to individually move through the passage.

B. Robots

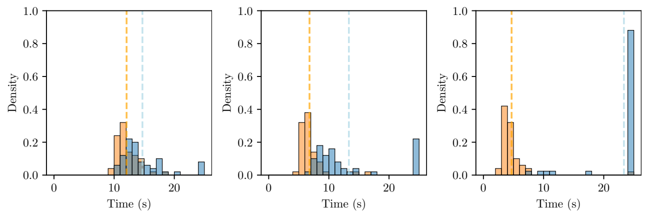
Our real-world experiments were conducted using five DJI RoboMaster S1, which is able to move holonomically using Mecanum wheels. The robot is 32 cm $\times$ 24 cm in size and can move with a lateral velocity of up to 3.5 m/s and with an angular velocity of up to 600 °/s [22], making it an attractive platform for evaluating aggressive ground maneuvers. Note that the RoboMaster S1 is not officially supported through DJI's Python API and requires undocumented work to be interfaced. We currently use DJI's API to control all robots through a local wireless network from a central controller, but it is generally possible to control each robot locally, for example with a Raspberry Pi in a mesh network configuration.

C. Experimental Setup

For each experiment we conduct, we run 50 episodes in simulation and on real robots, using identical initial and goal positions. The goal positions of the previous episode are then used as the initial positions of the next episode. We use a motion capture system for real-time indoor positioning of the robots. The detected poses are used to run the individual robot policies on a central computer, and the actions generated from the policies are then transmitted to each robot. This way, the only sim-to-real gap we encounter is due to the motion dynamics of the robots and potentially delays in the control and communication loop. The motion capture system is also used to reset the robots accurately to their initial positions after each episode has finished. We use ROS2 to interface all components of this system.

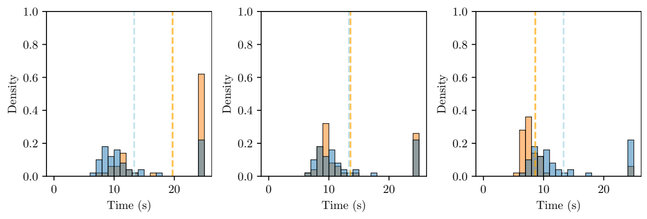
D. Isolating Parameters

Although we run the robots' control policies on a centralized computer, we encounter a performance gap between simulation and reality nonetheless. This gap can be seen in Fig. 4 and Tab. I, where we constrain the maximum velocity $v_{\max}$ to three different values in evaluation. We can clearly see that the success rate of the policy in simulation is not affected at all, as the makespan keeps decreasing from 12.7 s to 4.7 s. In contrast, the success rate of real robots decreases from 94% to 12%, while the mean makespan increases to 23.4 s. Therefore, the performance for $v_{\max} = 1.0$ m/s is a good compromise between success rate (78%) and



(a) $v_{\max} = 0.5$ (b) $v_{\max} = 1.0$ (c) $v_{\max} = 2.0$

Fig. 4. We evaluate the makespan for sim (orange) and real (blue) for different $v_{\max}$ in evaluation and $a_{\max} = \infty$ and mark the mean makespan as dashed line. Since the simulation was trained with $v_{\max} = 2.0$, $v_{\max}$ only decreases the makespan without decreasing the success rate, while the real robots are affected much more.



(a) $a_{\max} = 1.0$ (b) $a_{\max} = 1.2$ (c) $a_{\max} = 2.0$

Fig. 5. With $v_{\max} = 1.0$, we constrain the acceleration in sim (orange) and thus to close the gap between the two different distributions, as shown in Fig. 4.

makespan ($13.3s$) in reality, and we choose it for all further experiments.

TABLE I
EVALUATION STATISTIC OF SIM AND REAL EXPERIMENTS AS v_{max} CHANGES (FIG. 4) IN EVALUATION.

	$v_{\max}$	0.5	1.0	2.0
Sim	Success Rate	100%	100%	100%
	Mean Makespan	12.1 s	6.73 s	4.7 s
Real	Success Rate	94%	**78%**	12%
	Mean Makespan	14.7 s	**13.3** s	23.4 s

We tried to introduce an acceleration constraint $a_{\max}$ during training, but found that this leads to significantly worse real-world results. Instead of bringing the real-world performance closer to simulation, we therefore bring the simulation performance closer to the real world by adding acceleration constraints during evaluation. This can be seen in Fig. 5 and Tab. II. The results show that the performance is most similar for $a_{\max} = 1.2$ m/s^2, which we choose for all further experiments.

TABLE II
EVALUATION STATISTIC OF SIM AND REAL EXPERIMENTS AS VARYING ACCELERATION CONSTRAINT ($a_{\max}$, FIG. 5) GIVEN $v_{max} = 1$.

	Sim			Real
$a_{\max}$	1.0	1.2	2.0	
Success Rate	38%	**74%**	96%	**78%**
Mean Makespan	19.7 s	**13.6** s	8.6 s	**13.3** s

E. Experiments

After isolating parameters that contribute to a notable gap in performance between simulation and real world and changing the parameters in simulation so that this gap is minimized, we perform a series of experiments involving the communication performance of the GNN. Firstly, we vary the communication radius to $R_{\text{com}} \in \{3.0, 0.5, 0.0\}$ to evaluate the impact on the GNN performance. Furthermore, we randomly drop out individual messages between agents with a probability of $D \in \{25\%, 50\%\}$.

The results can be seen in Fig. 6 and Tab. III. The baseline is the same as Fig. 5b, with $R_{\text{com}} = 2.0$ m and $D = 0\%$. The trajectory plot (second row) shows agents moving out of the way for other agents and then moving through the passage. The position distribution plot (third row) is symmetric as all goal positions are successfully reached. The plots showing minimum distance $d_{\min}$ to other agents and the wall over vertical (fourth row) and horizontal (fifth row) position indicate agents approaching each other and the wall more closely if they are closer to the passage.

As we train the policy with $R_{com} = 2.0\ m$, increasing the communication range (columns 2-4 in Fig. 6) in the inference stage to 3.0 m leads to a slightly worse performance both in simulation and reality. This means that the communication range is an important part of the control policy, and increasing it does not necessarily lead to an improved performance. Decreasing the communication range to 0.5 m and eventually disabling all communication leads to a significant drop in performance. This is most obvious when looking at rows three and four, which indicate a clear shift in distribution towards smaller distances between robots.

A similar shift in distribution can be seen as more messages are randomly dropped out (columns 5-6 in Fig. 6). The performance for dropping out 50% of all messages is approximately similar to decreasing the communication range to 0.5 m.

It can be observed that the distributions between sim and real generally look similar, but the policy run on real robots generally performs better than the policy in simulation. This is likely caused by other inaccuracies in the simulation as well as our simplified dynamics model. For example, we assume the robot to be circular, while it is actually rectangular. Further fine-tuning of the acceleration constraint or using a physics simulator will likely resolve this discrepancy.

VII. DISCUSSION AND FURTHER WORK

Conclusion. We conducted pioneering experiments by deploying a learning-based framework based on a GNN to navigate a multi-robot system through a narrow passageway with a constrained communication range. We compared the performance of our trained policy in both simulation and physical experiments with similar environment configurations. We also studied the impact of communication radius between robots, the probability of communication dropout at the inference stage. These results showed only a small gap between the performance of our trained policy in simulation and that in reality.

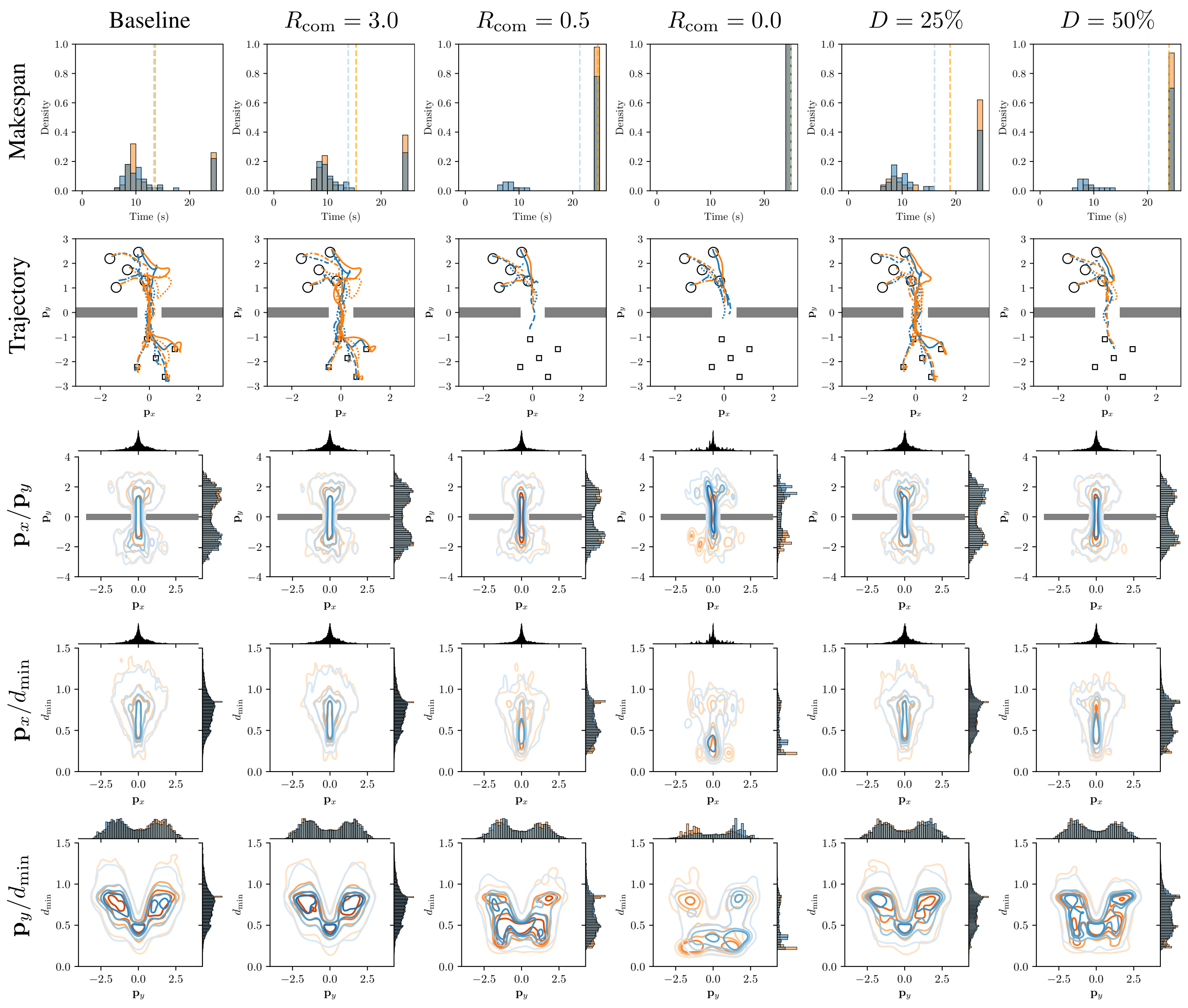


Fig. 6. Sim (orange) to real (blue) comparison. We artificially constrain the communication radius (R_{com}, columns 2-4) and randomly drop messages (columns 5-6) with different dropping probability (D) to analyze the impact of these parameters in sim (orange) and real (blue). The rows show the makespan and the dashed lines the mean makespans, a single sample trajectory (starts: circles), the distributions of positions and the distribution of minimum distance between any two agents at every time step over position.

TABLE III

PERFORMANCE OF THE POLICY IN SIMULATION AND REAL-WORLD EXPERIMENTS EVALUATED BY SUCCESS RATE AND MEAN MAKESPAN WITH DIFFERENT COMMUNICATION RADIUS (R_{com}) AND RANDOMLY DROP MESSAGES WITH A CERTAIN PROBABILITY (D) WHEN $v_{max} = 1m/s$.

	Experiment	Baseline	$R_{\text{com}} = 3.0$	$R_{\text{com}} = 0.5$	$R_{\text{com}} = 0.0$	$D = 25\%$	$D = 25\%$
Sim	Success Rate	74%	62%	2%	0%	38%	6%
	Mean Makespan	13.6 s	15.5 s	24.7 s	25.0 s	18.9 s	24.0 s
Real	Success Rate	78%	74%	22%	0%	58%	30%
	Mean Makespan	13.3 s	13.9 s	21.3 s	25.0 s	16.0 s	20.2 s

In the future, we will study the impact of some extreme communication scenarios in our framework, including delay in communication and asynchronous communication frequency. We also introduce a physics engine (i.e., PyBullet) to train our policy with explicit physical constraints (such as inertia and the resulting acceleration constraints). Most importantly, we are interested in developing a truly decentralized system by attaching an on-board device (i.e., Raspberry Pi) and compass sensor on each robot to align the coordinate frame of the robots to be independent of the motion capture system and set up a mesh communication router-less network to further evaluate the impact on sim-to-real.

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. A. Prorok was supported by the Engineering and Physical Sciences Research Council (grant EP/S015493/1) and ERC Project 949940 (gAIa). We appreciate the helpful discussions with Steven Morad about the intricacies of ROS 2 and his help with setting it up.

References

[1] J. Enright and P. R. Wurman, "Optimization and coordinated autonomy in mobile fulfillment systems." in *Automated Action Planning for Autonomous Mobile Robots*, 2011, pp. 33–38.

[2] A. Prorok and V. Kumar, "Privacy-preserving vehicle assignment for mobility-on-demand systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2017, pp. 1869–1876.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner, "Darla: Improving zero-shot transfer in reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1480–1490.

[9] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 627–12 637.

[10] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2017, pp. 23–30.

[11] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *IEEE International Conference on Robotics and Automation*. IEEE, 2017, pp. 3389–3396.

[12] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[13] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *International Conference on Robotics and Automation*. IEEE, 2019, pp. 8248–8254.

[14] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, 2013.

[15] 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.

[16] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 1928–1935.

[17] A. Rajeswaran, K. Lowrey, E. Todorov, and S. Kakade, "Towards generalization and simplicity in continuous control," in *Advances in Neural Information Processing Systems*. Curran Associates Inc., 2017, p. 6553–6564.

[18] S. Liu, L. Wen, J. Cui, X. Yang, J. Cao, and Y. Liu, "Moving forward in formation: A decentralized hierarchical learning approach to multi-agent moving together," *arXiv preprint arXiv:2011.02373*, 2020.

[19] 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.

[20] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, "Graphs, convolutions, and neural networks: From graph filters to graph neural networks," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 128–138, 2020.

[21] J. Blumenkamp and A. Prorok, "The emergence of adversarial communication in multi-agent reinforcement learning," *Conference on Robot Learning*, 2020.

[22] DJI, "RoboMaster S1 Specs," https://www.dji.com/uk/robomaster-s1/specs, 2021, [Online; accessed 21/04/22].