End-to-End Decentralized Formation Control Using Graph Neural Network Based Learning Method

Chao Jiang [1,*], **Xinchi Huang** [2] **and Yi Guo** [2]

[1] *Department of Electrical Engineering and Computer Science, University of Wyoming, Laramie, Wyoming, USA*
[2] *Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, New Jersey, USA*

Correspondence*:
Chao Jiang
cjiang1@uwyo.edu

ABSTRACT

Multi-robot cooperative control has been extensively studied using model-based distributed control methods. However, such control methods rely on sensing and perception modules in a sequential pipeline of design, and the separation of perception and controls may cause processing latency and compounding errors that affect control performance. End-to-end learning overcomes such limitation by learning directly from onboard sensing data, and outputs control command to robots. Challenges exist in end-to-end learning for multi-robot cooperative control and previous results are not scalable. We propose in this paper a novel decentralized cooperative control method for multi-robot formation using deep neural networks, in which inter-robot communication is modeled by a graph neural network (GNN). Our method takes the LIDAR sensor data as input, and the control policy is learned from demonstrations that are provided by an expert controller for decentralized formation control. While trained with a fixed number of robots, the learned control policy is scalable. Evaluation in a robot simulator demonstrates the triangulation formation behavior of multi-robot teams with different sizes using the learned control policy.

Keywords: Distributed multi-robot control, multi-robot learning, graph neural network, formation control and coordination, autonomous robots

1 INTRODUCTION

The last decade has witnessed substantial technological advances of multi-robot systems that enabled a vast range of applications including autonomous transportation systems, multi-robot exploration, rescue and security patrols. Multi-robot systems demonstrated notable advantages over single-robot systems such as enhanced efficiency in task execution, reconfigurability and fault tolerance. Particularly, the capability of multi-robot systems to self-organize via local interaction gives rise to various multi-robot collective behaviors (e.g., flocking, formation, area coverage) to achieve team-level objectives (Guo, 2017).

A plethora of control methods has contributed to the development of multi-robot autonomy that enables complex collective behaviors of multi-robot systems. One major control design paradigm focuses on decentralized feedback control methods (Cortés and Egerstedt, 2017; Panagou et al., 2015; Bechlioulis et al., 2019) which provide provable control and coordination protocols that can be executed efficiently at run time. Control protocols are designed to compute robot actions analytically using robot's kinematic/dynamic model and communication graph that specifies the interaction connectivity for local information exchange. Such hand-engineered control and coordination protocols separate the problem into a set of sequentially executed stages including perception, state measurement/estimation, and control. However, this pipeline of stages could suffer from perception and state estimation errors which compound through the sequential individual stages (Loquercio et al., 2021; Zhang and Scaramuzza, 2018). Moreover, such pipeline introduces latency between perception and actuation as the time necessary to process perception data and compute control command accumulates (Falanga et al., 2019). Both compounding errors and latency are well-known issues that impact task performance and success in robotics. Learning-based methods, as another control design paradigm, has proven to be successful in learning control policies from data (Kahn et al., 2018; Li et al., 2020; Devo et al., 2020; Li et al., 2022). Particularly, owing to the feature representation capability of deep neural networks (DNNs), control policies can be modeled to synthesize a control command directly from a raw sensor observation. Such control policies are trained to model an end-to-end computation that encompasses the traditional pipeline of stages and their underlying interactions (Loquercio et al., 2021).

Multi-robot learning has long been an active research area (Stone and Veloso, 2000; Gronauer and Diepold, 2021). Nonetheless, it wasn't until recent years that challenges originating from real-world complexities can be handled with the advancement of deep reinforcement learning techniques. Breakthroughs on computational methods have been made to address long-standing challenges in multi-robot learning such as non-stationarity (Foerster et al., 2017; Lowe et al., 2017; Foerster et al., 2018), learning to communicate (Sukhbaatar et al., 2016; Foerster et al., 2016; Jiang and Lu, 2018), scalability (Gupta et al., 2017). Various multi-robot control problems such as path planning (Blumenkamp et al., 2022; Wang et al., 2021), coordinated control (Tolstaya et al., 2020a; Yan et al., 2022; Agarwal et al., 2020; Zhou et al., 2019; Tolstaya et al., 2020b; Kabore and Güler, 2021; Jiang and Guo, 2020) have been tackled using learning-based methods. Despite the remarkable progress in multi-robot learning, the architecture design and learning of scalable computational models that accommodate emerging information structures is still an open question. For example, it has yet to be understood what and how information should be dynamically gathered given the distributed information structure that only allows local inter-robot interaction. Recently, graph neural networks (GNNs) (Scarselli et al., 2008) were used to model the information-sharing structure between robots. A GNN can be trained to capture task-relevant information to be propagated and shared in the robot team via local inter-robot communication. GNNs have become an appealing framework for modeling distributed robot networks (Agarwal et al., 2020; Zhou et al., 2019; Tolstaya et al., 2020a,b; Wang and Gombolay, 2020; Blumenkamp et al., 2022) due to their scalability and permutation-invariance (Gama et al., 2020).

In this paper, we study a multi-robot formation problem using a learning-based method to find decentralized control policies that operate on robot sensor observations. The formation problem is defined for the multi-robot team to achieve triangulation formations that constitute a planar graph with prescribed equidistant edge lengths. We use a GNN to model the inter-robot communication for learning scalable control policies. The GNN is combined with a convolutional neural network (CNN) to process sensor-level robot observations. Utilizing a model-based decentralized controller for triangulation formation as an expert control, we train the deep neural network (DNN) with a data aggregation training scheme. We demonstrate in a robot simulator that the learned decentralized control policy is scalable to different sizes of multi-robot teams while trained with a fixed number of robots.

The main contributions of this paper is the GNN-based end-to-end decentralized control for multi-robot triangulation formation. Comparing to our prior work (Jiang et al., 2019) on learning-based end-to-end control of multi-robot formation, this paper achieves decentralized scalable control policy, while our prior work (Jiang et al., 2019) adopts centralized training mechanism and the trained policy is not scalable and applies to three-robot formation only. Comparing to the recent GNN-based flocking control method (Tolstaya et al., 2020a), the triangulation formation studied in this paper imposes additional geometric constraints for multi-robot coordinated motion in comparison with the flocking behavior in Tolstaya et al. (2020a). Furthermore, our decentralized control scheme is end-to-end and takes robot LIDAR sensor data as input directly, while the method in Tolstaya et al. (2020a) takes state values of robot positions as input. As mentioned earlier, the end-to-end learning facilitates direct learning from sensor data, and can avoid potential compounding errors and latency issues commonly found in conventional design that separates perception and control in sequential stages of a pipeline.

The rest of the paper is organized as follows. Section 2 presents the model of differential-drive mobile robots, and formulates our multi-robot cooperative control problem. The GNN-based training and online control methods are described in Section 3. Robot simulation results are demonstrated in Section 4. Section 5 discusses the main difference compared to existing learning-based methods. The paper is finally concluded in Section 6.

2 PROBLEM STATEMENT

In this paper, we consider a multi-robot cooperative control problem with N differential-drive mobile robots. The kinematic model of each robot $i \in \{1, ..., N\}$ is given by the discrete-time model:

$$\begin{bmatrix} x_i(t+1) \\ y_i(t+1) \\ \theta_i(t+1) \end{bmatrix} = \begin{bmatrix} x_i(t) \\ y_i(t) \\ \theta_i(t) \end{bmatrix} + \boldsymbol{G}(t) \cdot \begin{bmatrix} u_{iL}(t) \\ u_{iR}(t) \end{bmatrix}, \tag{1}$$

where $[x_i, y_i, \theta_i]^T \in \mathbb{R}^3$ is the robot state vector that consists of the position $\boldsymbol{p}_i \triangleq [x_i, y_i]^T$ and the orientation θ_i; $\boldsymbol{u}_i \triangleq [u_{iL}, u_{iR}]^T \in \mathbb{R}^2$ is the control vector with u_{iL} and u_{iR} being the left and right motor control, respectively. The matrix $\boldsymbol{G}(t)$ is defined as:

$$\boldsymbol{G}(t) = \begin{bmatrix} \frac{\Delta T}{2}\cos\theta_i(t) & \frac{\Delta T}{2}\cos\theta_i(t) \\ \frac{\Delta T}{2}\sin\theta_i(t) & \frac{\Delta T}{2}\sin\theta_i(t) \\ -\frac{\Delta T}{l} & \frac{\Delta T}{l} \end{bmatrix}, \tag{2}$$

where ΔT is the sampling period and l is the distance between the robot's left and right wheels.

We assume that each robot is equipped with a LIDAR sensor to detect neighboring robots. The LIDAR measurements are transformed to an occupancy map, denoted by $\boldsymbol{o}_i(t)$, serving as the robot's local observation. The proximity graph of the robot team is defined as a Gabriel graph (Mesbahi and Egerstedt, 2010), denoted as $\mathcal{G} = (V, E)$, where $V = \{v_1, ..., v_N\}$ is the set of vertices corresponding to the robots located at $\boldsymbol{p}_1, ..., \boldsymbol{p}_N \in \mathbb{R}^2$ and E is the set of edges. The "line" connecting the vertices $v_i, v_j \in V$, $i \neq j$, is said to be an edge if and only if the circle of diameter, $\|\boldsymbol{p}_i - \boldsymbol{p}_j\|$, containing both vertices v_i and v_j does not contain any vertex in its interior. An example of a valid and an invalid edge of a Gabriel graph can be seen in Fig. 1. Robots i and j associated with vertices v_i and v_j, respectively, are said to be neighbors and

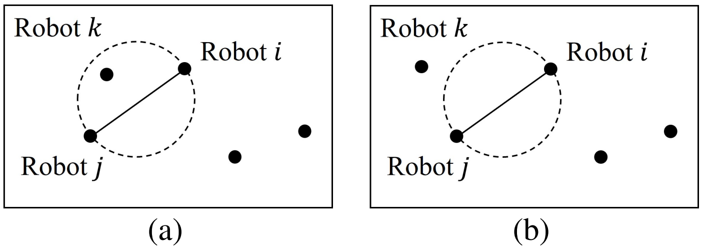


Figure 1. Gabriel graph: (a) robot i and j are not neighbors as robot k exists in the circle whose diameter is defined by the distance between robots i and j; (b) robot i and j are valid neighbors as there are no other robots in the circle.

can communicate if $\{v_i, v_j\} \in E$. Note that the proximity graph $\mathcal{G}$ is time-varying as a robot's neighbors vary when they move around.

The objective of the cooperative control is to find a decentralized control protocol for each robot such that, starting from any initial positions that there's at least one robot within the neighborhood of each robot (i.e., the initial proximity graph $\mathcal{G}$ is connected), the group of robots achieves triangulation formations with a prescribed inter-robot distance, d^*, for all pairs of robots $\{v_i, v_j\} \in E$. That is, $\|\boldsymbol{p}_i - \boldsymbol{p}_j\| \rightarrow d^*$ as $t \rightarrow \infty, \forall (v_i, v_j) \in E$.

To address the formulated multi-robot coordination problem, we propose a learning-based method to find a decentralized and scalable control policy that can be deployed on each robot. A GNN in conjunction with a CNN will be used as the parameterized representation of the control policy. The neural network policy is decentralized in the sense that only local information obtained by each robot is used to compute a control action. We show in simulation experiments that, owing to the scalability of the GNN representation, the learned control policy is scalable in that once trained with a given number of robots, the policy is applicable to different sizes of robot team with the team size remaining unchanged during operation. In the next section, we introduce the architecture and training of the neural network control policy.

3 METHOD

3.1 Overview of Learning-based Cooperative Control

The overview of the proposed learning-based multi-robot cooperative control is shown in Fig. 2. The decentralized control policy is parameterized by a DNN consisting of a CNN, a GNN, a multi-layer perceptron (MLP) network, and a fully-connceted (FC) network as shown in the dashed box. The CNN extracts task-relevant features from an occupancy map obtained by the robot's own onboard LIDAR sensor. The features from the robot's local observation are communicated via the GNN which models the underlying communication for information propagation and aggregation in the robot network. Given the features aggregated locally via the GNN, the MLP and FC layers compute a robot control command as the final output. The DNN policy can be expressed by

$$\boldsymbol{u}_i = \boldsymbol{\pi}(\boldsymbol{o}_i; \mathcal{G}, \boldsymbol{\Theta}), \forall i \in \{1, ..., N\}. \tag{3}$$

To compute a control action $\boldsymbol{u}_i$, the policy (3) uses each robot's own observation $\boldsymbol{o}_i$ and the local information aggregated from current neighboring robots determined by the proximity graph $\mathcal{G}$ that the GNN has access to. $\boldsymbol{\Theta}$ is the tensor of parameter of the DNN which is tuned during policy training. During online control, the DNN policy computes robot control end-to-end through a feed-forward pass in a

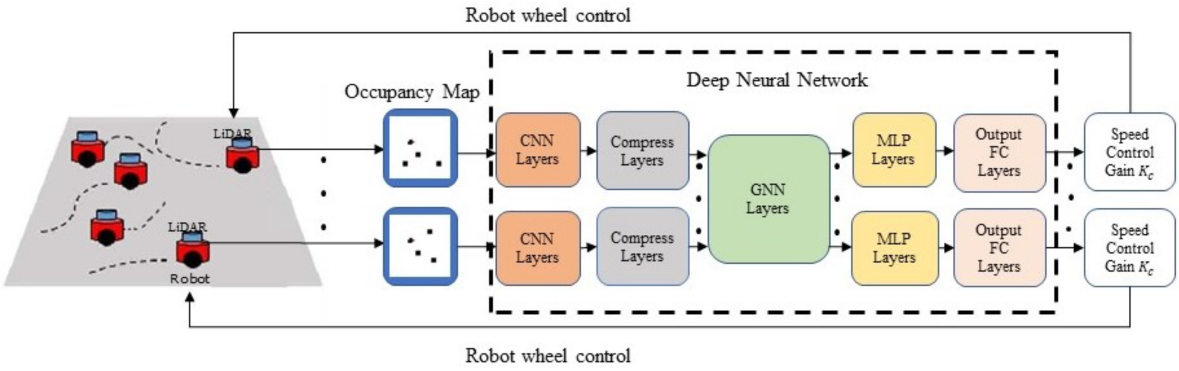


Figure 2. The overall diagram of the end-to-end GNN-based decentralized formation control.

decentralized manner. It's worth noting that the GNN block shown in Fig. 2 represents data exchange within the entire robot team through local communication and does not signify a central communication unit. The computation of GNN is decentralized as each robot aggregates local information from its neighbors only. More details of the computation of GNN are presented in Section 3.2.

We train the policy (3) via learning from demonstrations (LfD), and a model-based controller is used as an expert controller to provide demonstration data. The data set is composed of pairs of robot observation $\boldsymbol{o}_i$ and expert control action $\boldsymbol{u}_i^*$ associated with that observation. The policy training then amounts to finding the optimal parameters $\boldsymbol{\Theta}$ that minimize the following loss function (for a single data sample):

$$\mathcal{L}(\boldsymbol{\Theta}) = \frac{1}{2}\|\boldsymbol{\pi}(\boldsymbol{o}_i;\mathcal{G},\boldsymbol{\Theta}) - \boldsymbol{u}_i^*\|^2. \tag{4}$$

The loss function measures the difference between the neural network controller's output given by $\boldsymbol{\pi}(\boldsymbol{o}_i;\mathcal{G},\boldsymbol{\Theta})$ and the expert controller's output $\boldsymbol{u}_i^*$ computed in the same system state from which the observation $\boldsymbol{o}_i$ is obtained. Minimizing the loss function encourages the policy to imitate the control strategies of the model-based decentralized controller.

3.2 Graph Neural Network

The feature vector $\mathbf{x}_i \in \mathbb{R}^F$ extracted by the CNN on each robot i will be communicated to its neighbors by one-hop communication via the GNN. The GNN architecture adopted in this paper is the aggregation GNN (Gama et al., 2018). Each layer of the GNN performs a graph convolution that aggregates information from neighboring robots. The information aggregation with k-hop communication in the robot network at time step t creates a signal:

$$\mathbf{Z}(t) = \Big[\mathbf{X}(t), \boldsymbol{S}\mathbf{X}(t), \boldsymbol{S}^2\mathbf{X}(t), ..., \boldsymbol{S}^k\mathbf{X}(t)\Big] \in \mathbb{R}^{N\times F(k+1)},$$

where $\mathbf{X}(t) = [\mathbf{x}_1(t), \mathbf{x}_2(t), ..., \mathbf{x}_N(t)]^T \in \mathbb{R}^{N\times F}$ is the collection of feature vectors of all robots and $\boldsymbol{S}(t) \in \mathbb{R}^{N\times N}$ is the Graph Shift Operator (GSO) (Gama et al., 2018). The GSO is defined as a local linear operation applied to graph signals (e.g., the feature vector $\mathbf{x}_i(t)$) (Gama et al., 2018). Specifically, the f-th element of the feature vector for robot i after applying the GSO with one-hop (i.e., $k = 1$) communication is given by:

$$[\boldsymbol{S}\mathbf{X}(t)]_{if} = \sum_{j=1}^{N} [\boldsymbol{S}]_{ij}\, [\mathbf{X}(t)]_{jf}\,. \tag{5}$$

The GSO is associated with the graph structure, and in our problem it is defined as the adjacency matrix, i.e., $[\boldsymbol{S}]_{ij} = 1$ if robot i and j are neighbors, otherwise $[\boldsymbol{S}]_{ij} = 0$.

Each row i of $\mathbf{Z}(t)$, denoted by $\mathbf{z}_i \in \mathbb{R}^{F(k+1)}$, is a local signal representing the information vector aggregated on the i-th robot. The local signal $\mathbf{z}_i$ is then convolved with a bank of FG filters, denoted by $\mathbf{h} \in \mathbb{R}^{F(k+1)\times G}$, to produce an output feature vector: $\mathbf{y}_i = \boldsymbol{\sigma}_{\text{gnn}}\big(\mathbf{h}^T\mathbf{z}_i\big) \in \mathbb{R}^G$, where $\boldsymbol{\sigma}_{\text{gnn}}(\cdot)$ is a point-wise nonlinear activation function. The elements of $\mathbf{h}$ represent the learnable filter weights which are shared by all robots. The local feature vector, $\mathbf{y}_i$, is fed into the MLP and FC layers of each robot's local policy to compute an robot control command. More details of the aggregation GNN used in this paper can be found in Gama et al. (2018). Note that we specifically use one-hop communication (i.e., $k = 1$) and select the the number of GNN layers to be one (i.e., the graph convolution operation $\boldsymbol{\sigma}_{\text{gnn}}(\cdot)$ is only performed once per time step) to reduce the communication load at each time step.

It is worth mentioning that our proposed policy model inherits the scalability property of the GNN. The scalability of GNNs stems from their permutation equivariance property and stability to changes in the topology (Gama et al., 2020). These properties allow GNNs to generalize the signal processing protocol learned at local nodes to every other node with similar topological neighborhood.

3.3 Policy Training

3.3.1 Expert Controller

The model-based controller (Mesbahi and Egerstedt, 2010) for multi-robot triangulation formation problem was employed as the expert controller to provide training data. The expert controller achieves triangulation formations by minimizing the potential function associated with robots i and j, i.e.,

$$U_{ij} = \frac{1}{2}(\|\boldsymbol{p}_i - \boldsymbol{p}_j\| - d^*)^2, \forall\{i,j\} \in E. \tag{6}$$

The potential function takes on its minimum at the prescribed inter-robot distance, d^*. Assuming single-integrator dynamics of the robots, i.e., $\dot{\boldsymbol{p}}_i = \mathbf{v}_i$, the control law is given by

$$\mathbf{v}_i = -K_c \sum_{j\in\mathcal{N}_i} \frac{\|\boldsymbol{p}_i - \boldsymbol{p}_j\| - d^*}{\|\boldsymbol{p}_i - \boldsymbol{p}_j\|} \cdot (\boldsymbol{p}_i - \boldsymbol{p}_j), \tag{7}$$

where robot j belongs to the neighbors of robot i, $\mathcal{N}_i$, defined by the Gabriel graph and K_c is the control gain. When the inter-robot distance is greater than d^*, the controller exerts attractive force through the positive weight $\frac{\|\boldsymbol{p}_i-\boldsymbol{p}_j\|-d^*}{\|\boldsymbol{p}_i-\boldsymbol{p}_j\|}$. When the inter-robot distance is smaller than d^*, the controller repels the robots away from each other as the weight becomes negative. At convergence, the neighboring robots form triangulations with the distance d^*.

The control input, $\mathbf{v}_i$, computed by the expert controller for the single-integrator model is converted to the motor control of the differential-drive robot model, $\boldsymbol{u}_i$, using coordinate transformation method (Chen et al., 2019). The transformation is given by

$$\begin{bmatrix} u_{iL} \\ u_{iR} \end{bmatrix} = \begin{bmatrix} \sin\theta_i + \frac{l}{2c}\cos\theta_i & \sin\theta_i - \frac{l}{2c}\cos\theta_i \\ -\sin\theta_i + \frac{l}{2c}\cos\theta_i & \sin\theta_i + \frac{l}{2c}\cos\theta_i \end{bmatrix} \cdot \mathbf{v}_i, \tag{8}$$

where l is defined in (2) and $c = l/2$. Then the differential-drive robot (1) can be controlled by the expert controller after transformation.

3.3.2 Policy Training with DAgger

The DNN policy was trained via learning from demonstration, and a model-based controller was used to provide expert demonstration data. In order to obtain a model sufficiently generalizable to unseen states at test time, we used a Data Aggregation (DAgger) training framework (Ross et al., 2011). The idea behind this is that an empty data set is gradually "aggregated" by data samples with states visited by a learning policy and actions given by the expert. To this end, we picked the learning policy with a probability $(1-\beta)$ to execute a control action at each time step of collecting data samples during training. The probability β was initialized to 1 and decayed by a factor of 0.9 after every 50 episodes.

The policy training with DAgger is outlined in Algorithm 1. As training progresses, the data set $\mathcal{D}$ was aggregated with data samples in the form of $\big(\boldsymbol{o}_i(t), \boldsymbol{S}(t), \boldsymbol{u}_i^*(t)\big)$. Since computing a control input by the model at time t requires the robot's local observation $\boldsymbol{o}_i(t)$ and the information aggregated through the GSO, we recorded $\boldsymbol{S}(t)$ along with the observation-action pair, $\boldsymbol{o}_i(t)$ and $\boldsymbol{u}_i^*(t)$, to create a data sample. In each training episode, mini-batches of size B were sampled from the data set $\mathcal{D}$ to train our model by backpropagating the mini-batch gradient of the loss calculated by (4).

Algorithm 1 Policy Training with DAgger

Require: Observation $\boldsymbol{o}_i(t)$, graph shift operator $\boldsymbol{S}(t)$, expert control action $\boldsymbol{u}_i^*(t)$ at each time step t
Ensure: DNN policy $\boldsymbol{\pi}(\boldsymbol{o}_i; \mathcal{G}, \boldsymbol{\Theta})$

Initialize data set $\mathcal{D} \leftarrow \emptyset$
Initialize policy parameter $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}_0$
Initialize $\beta \leftarrow 1$
for episode $e = 1$ to E **do**
  Initialize robot state $[x_i, y_i, \theta_i], \forall i \in \{1, ..., N\}$
  for time step $t = 1$ to T **do**
    for robot $i = 1$ to N **do**
      Query an expert control $\boldsymbol{u}_i^*(t) \leftarrow \boldsymbol{\pi}^*\big(\boldsymbol{s}_i(t)\big)$
      Get sample $\big(\boldsymbol{o}_i(t), \boldsymbol{S}(t), \boldsymbol{u}_i^*(t)\big)$
      Choose a policy $\boldsymbol{\pi}_i \leftarrow \beta\boldsymbol{\pi}^* + (1-\beta)\boldsymbol{\pi}$
    end for
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{\big(\boldsymbol{o}_i(t), \boldsymbol{S}(t), \boldsymbol{u}_i^*(t)\big)\}_{i=1}^N$
    Execute policy $\boldsymbol{\pi}_i, \forall i \in \{1, ..., N\}$, to advance the environment
  end for
  for $n = 1$ to K **do**
    Draw mini-batch samples of size B from $\mathcal{D}$
    Update policy parameters $\boldsymbol{\Theta}$ by mini-batch gradient descent with loss (4)
  end for
  Update $\beta \leftarrow 0.9\beta$ if $\mathrm{mod}(e, 50) = 0$
end for
return learned policy $\boldsymbol{\pi}(\boldsymbol{o}_i; \mathcal{G}, \boldsymbol{\Theta})$

3.4 Online Cooperative Control

At test time i.e., online formation control, a local copy of the learned policy $\boldsymbol{\pi}(\boldsymbol{o}_i; \mathcal{G}, \boldsymbol{\Theta})$ was deployed on each robot as a decentralized controller. At each time step, the local policy received an occupancy map and a control action was calculated in a feed-forward pass. One-hop communication was performed between neighboring robots i and j, for which $\{v_i, v_j\} \in E$, to aggregate information in a decentralized way. Note that the communication graph is defined in Section 2 as a Gabriel graph and shown in Fig. 1. The online cooperative control is outlined in Algorithm 2.

Algorithm 2 Online Cooperative Control

Require: Occupancy map $\boldsymbol{o}_i(t)$
Ensure: Robot control action $\boldsymbol{u}_i(t)$

```
Initialize robot state [x_i, y_i, θ_i], ∀i ∈ {1, ..., N}
for time step t = 1 to T do
    for robot i = 1 to N do
        Obtain an occupancy map o_i(t)
        Aggregate information locally by applying (5) via one-hop communication
        Compute a control action u_i(t) ← π(o_i; G, Θ)
    end for
    Execute the control action u_i, ∀i ∈ {1, ..., N} to advance the environment
end for
```

4 EXPERIMENT RESULTS

4.1 Simulation Environment

The robot control simulation was conducted in the robot simulator, CoppeliaSim (from the creators of V-REP). We choose a team of P3-DX mobile robots, each of which has a Velodyne VLP 16 LiDAR sensor used to obtain LiDAR data and then converted to occupancy maps. The LiDAR sensors were set to a sensing range of 10 meters. The size of the occupancy map created from the sensory reading was 100 pixels $\times 100$ pixels, making the granularity of the occupancy maps 0.1 meters/pixel. The robot simulator was controlled via various Python scripts, as the simulator API can be accessed via local data communication to and from the client Python program.

The computer used to simulate the results has an Intel i7 12900K, 12 core CPU that ran at 3.6 GHz. The GPU used for rendering and neural network training and testing was an NVIDIA Titan Xp GPU. The PyTorch framework handled the GNN implementation as well as computations for training and testing the neural network.

4.2 DNN Implementation

The implementation details of the neural network architecture shown in Fig. 2 are given in Table 1. The CNN layers were composed of multiple convolutional blocks. The input size of the first convolutional block was set to $(1, 100, 100)$ to fit the size of the occupancy map. The extracted features from the input by the CNN were flattened into a vector of size $(1, 18432)$, which was further compressed by the compression block into a feature vector whose size was $(1, 128)$. That is, the dimension F of $\mathbf{x}_i$ was set to 128. The compressed feature vector was fed to the GNN block and communicated to neighboring robots. The GNN consisted of 1 graph convolution layer that produced a new feature vector of the same size with the input. That is, the dimension G of $\mathbf{y}_i$ was set to 128. Muliple MLP blocks took as input the feature vector and output the robot control action whose dimension was $(1, 2)$.

Table 1. Blocks and parameters of the DNN.

Layer Block	Input Size	Output Size
Convolutional Max Pool Block 1	(1,100,100)	(32,50,50)
Convolutional Block 2	(32,50,50)	(32,50,50)
Convolutional Max Pool Block 3	(32,50,50)	(64,25,25)
Convolutional Block 4	(64,24,24)	(128,12,12)
Convolutional Max Pool Block 5	(128,12,12)	(128,12,12)
Feature Compression Block 1	(1,18432)	(1,128)
GNN Block 1	(1,128)	(1,128)
MLP Block 1	(1,128)	(1,128)
MLP Block 2	(1,128)	(1,128)
Output MLP Block 3	(1,128)	(1,2)

4.3 System Parameters and Performance Metrics

The desired triangulation formation was set as $d^* = 2$ m. We set $k = 1$ in the k-hop communication. The initial conditions of robot positions were randomly generated in a circle of radius 5m, and the initial orientation of each robot was randomly chosen from $[0, 2\pi]$. The distance l between the robot's left and right wheels is 0.331 m.

We trained the DNN on a team of five robots. To evaluate the performance of the trained model, we tested it on different numbers of robots ranging from $N = 4$ to $N = 9$. We define the formation error between neighboring robots i and j at time t as: $\mathcal{E}_{i,j}(t) = |\|\boldsymbol{p}_i(t) - \boldsymbol{p}_j(t)\| - d^*|$. The group formation error at any time t is defined as: $\mathcal{E}_g(t) = \frac{1}{N}\sum_{j\in\mathcal{N}_i} \mathcal{E}_{i,j}(t)$.

During training with $N = 5$, the data collection period of each training episode (i.e., lines 6-14 in Algorithm 1) ran for at most 200 s, and a data point was collected every 0.05 s. We terminated the simulation if the temporal average of $\mathcal{E}_g(t)/d^*$ over the most recent 20 s was smaller than 5%. Note that when we chose the speed control gain K_c in the expert control (7), there was a tradeoff between the converging speed and the steady-state error. A large K_c makes the system converges to the triangulation formation faster, but may cause the system to oscillate around the equilibrium. We chose an adaptive control gain K_c in (7) to be 1 initially, and then after $\mathcal{E}_g(t)/d^* < 0.05$, K_c is decreased to slow down the robots as they are close to each other.

During testing, we consider the multi-robot system *converged* if the group formation error (i.e., the temporal average of the group formation error $\mathcal{E}_g(t)/d^*$ over the most recent 20 s) is smaller than 5% or 10%. We define three performance metrics as follows:

1. *Success rate*: $Rate = n_{success}/n$, is the proportion of successful cases over total number of tested cases n. A simulation run is considered successful if it converges before the end of the simulation. We present success rate with 5% and 10% tolerance in Table 2.
2. *Converge time*: $T_{converge}$ is the time when a simulation run converges. That is, the first time that the temporal average of the group formation error over the most recent 20 s reaches the 5% threshold and then keeps decreasing.
3. *Group formation error* defined in percentage: $\overline{\mathcal{E}}_g/d^*$, where $\overline{\mathcal{E}}_g$ is defined as the temporal average of $\mathcal{E}_g(t)$ over the last 20 s prior to the end of the simulation.

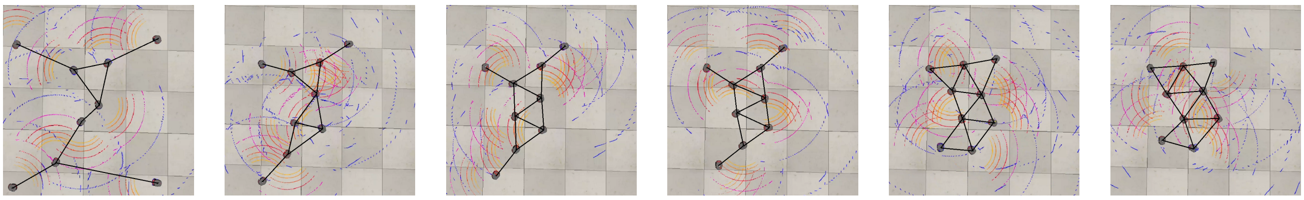
4.4 Training

We trained the DNN using a five-robot team by running Algorithm 1. The data collection period of each training episode (i.e., lines 6-14) ran for at most 200 s, where a data point was collected every 0.05 s. The probability value β for picking between the neural network controller and the expert controller at every time step started at $\beta = 1$ in episode $e = 1$, and was updated every episode with the formula for episode e as $\beta_e = 0.9^{\lfloor \frac{e}{50} \rfloor}$ where $\lfloor \cdot \rfloor$ represents the floor operator. Thereby, β decayed by a factor of 0.9 every 50 training episodes. The loss function used was a mean squared error loss between the expert control that was stored and the control that was returned by the learned model. During training, the RMSprop optimizer (Mustapha et al., 2020) was used. The learning rate was set to 0.0001 and the size of mini-batch, B, used to calculated gradient was set to 16. After training for 200 episodes, the weights were saved for testing.

4.5 Testing Results

After training the neural network model using a five-robot team, we tested our end-to-end decentralized formation control for robot teams of varying sizes by running Algorithm 2. Random initial conditions were used for robot start positions. We demonstrate the empirical and statistical results below.

The snapshots of nine robots achieving triangulation formation in the CoppeliaSim simulator are shown in Fig. 3. The solid black lines represent the formation achieved at different time steps. The testing results for different robot team sizes, $N = 5, 6, 7, 8, 9$, are shown in Fig. 4 from (a) to (e), respectively. The time histories of inter-robot distance (i.e., $d_{ij}(t), i = 1, \ldots, N, j \in \mathcal{N}_i$), and the trajectory of each robot are shown in the top and the bottom of the figures, respectively. We can see that the robot team achieves the desired triangulation formation and maintains the desired neighboring distance $d^* = 2$ m. More simulation results for robot team sizes can be found in the supplementary video file submitted together with this paper.



$t = 0$ s $t = 10$ s $t = 20$ s $t = 30$ s $t = 40$ s $t = 50$ s

Figure 3. Snapshots of online formation control of nine robots in CoppeliaSim simulator. The colored arcs visualize the LiDAR scanning.

To evaluate *scalability*, we ran testing experiments with 100 different robot initial conditions for each of the robot-team sizes from 4 to 9. The success rate for different robot-team sizes are shown in Table 2. We can see that the success rate achieves 100% for any team size with the 10% tolerance (i.e., the group formation error $\overline{\mathcal{E}}_g/d^*$ is smaller than 10% as defined in Section 4.3). With the 5% tolerance threshold, the success rate decreases as the number of the robots increases. This is due to the fact that when robots get closer to the desired formation, small motion uncertainties cause oscillations of trajectories, and the oscillations persist more when the number of robots increases. This phenomenon can be mitigated by reducing the speed control gain K_c further after the robots reach around the desired formation. However, tuning this control parameter is tedious and is by trial and error. The success rate reported in this table were obtained using one set of K_c. Thus, we can see that our method possesses good scalability, that is, while trained with a 5-robot team, the DNN policy can be applied to different sizes of robot team.

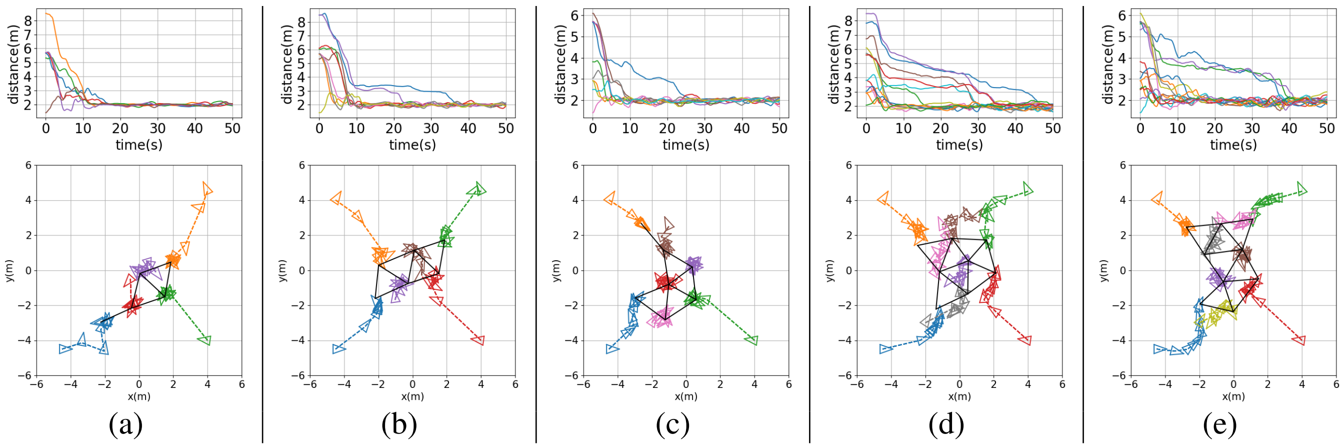


Figure 4. Testing results for: (a) five-robot team ($N = 5$), (b) six-robot team ($N = 6$), (c) a seven-robot team ($N = 7$), (d) eight-robot team ($N = 8$), and (e) nine-robot team ($N = 9$). Top row: Time histories of inter-robot distance, $d_{ij}(t), i = 1, \ldots, N, j \in \mathcal{N}_i$; Bottom row: Robot trajectories with robot positions (denoted by colored small triangles) sampled every 10 s, and the black solid lines indicate the final formation achieved.

Table 2. Success rates over 100 runs.

Number of robots:	4	5	6	7	8	9
Success rate % (5% tolerance):	100	100	98	90	78	64
Success rate % (10% tolerance):	100	100	100	100	100	100

To further evaluate performance, we tested 100 initial conditions for each of the multi-robot teams with sizes from 4 to 9. We show in Fig. 5(a) the box plot of the group formation error $\overline{\mathcal{E}}_g$ as defined in Section 4.3. We can see that the median formation errors are between 2% to 6% for robot team sizes from 4 to 9. Fig. 5(b) shows the box plot of the convergence time $T_{converge}$. we can see that the median convergence time is around 15 s. From Fig. 5, we can see that while the DNN model was trained using a five-robot team, the learned controller is scalable to other sizes of multi-robot team and the performance are satisfactory under the metrics of group formation error and convergence time.

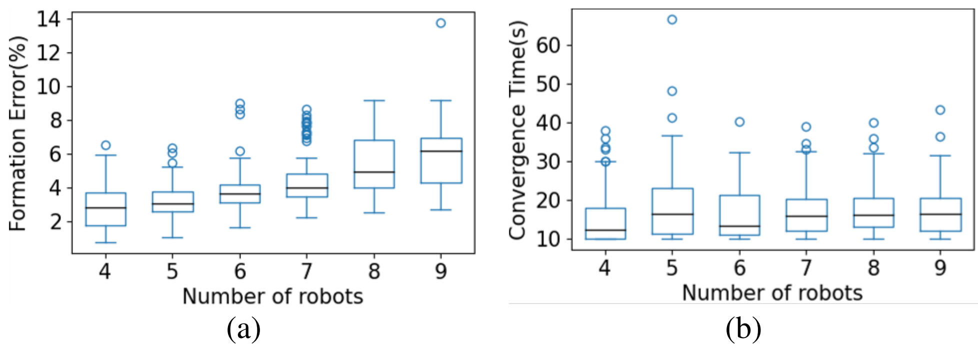


Figure 5. Box plot for 100 initial conditions of multi-robot testing. The central mark in each box is the median, the edges of the boxes are the 25th and 75th percentiles, the whiskers extend to the maximum/minimum, and the circles represent outliers. (a) Group formation error for multi-robot teams with sizes from 4 to 9; (b) Convergence time for multi-robot teams with sizes from 4 to 9.

To further compare the performance between the expert control and our trained DNN models, we show the box plot of 100 initial conditions for the 5-robot case in Fig. 6. We can see that the expert control achieved 1.76% in median formation error and 12.9 s in median convergence time, while our DNN model achieved 3.12% and 18.2 s, respectively. We can see that the expert policy outperforms the end-to-end policy slightly. This is expected given that the expert policy, as defined in (7) and (8), uses the global position of the robots, which is assumed to be observable perfectly. The end-to-end policy, on the other hand, uses LiDAR observations as input which is noisy. It should be noted that the goal of the proposed method was not to outperform the expert controller given ideal state measurements. The main advantages of our method over the expert controller are that 1) the end-to-end computational model by our method mitigates accumulation of error and latency in traditional pipeline of computational modules used by the expert control method; 2) our method does not need a localization system to obtain global robot positions, reducing overall system complexity.

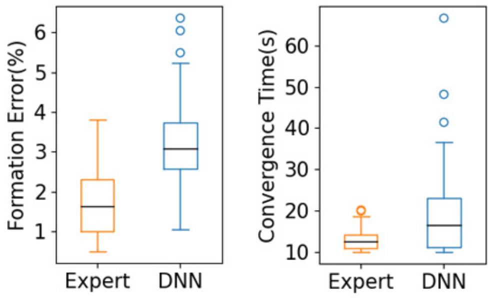


Figure 6. Comparison between the expert control and our trained DNN model for the 5-robot case with 100 initial conditions.

4.6 Other Formation Shapes

The triangulation formation control we designed can be extended to other formation shapes, such as line formation and circle formation. For such cases, we used additional landmarks (i.e., stationary robots positioned at pre-selected reference positions for other robots to achieve formation objectives) and modified the expert controller to achieve the desired formations.

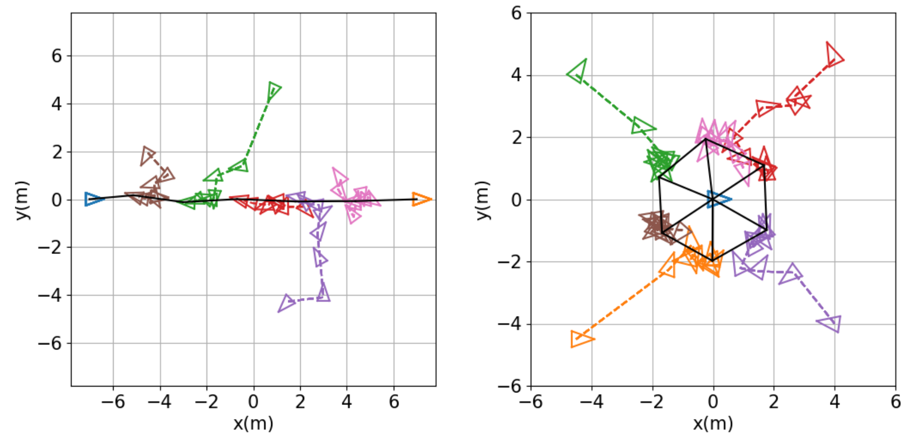
Line formation: The objective of the line formation is for the robots to position themselves at an equal distance from one another in a line between two landmarks . We simulated a 7-robot team with 2 stationary robots serving as landmarks that were 14 meters apart at each end of the desired line. The other 5 robots were controlled by the same expert controller (7) and (8) that we used for the triangulation formation. We ran the same training algorithm (i.e., Algorithm 1) with the same hyperparameters as before. Fig. 7(a) shows the testing result that the robots achieved the desired line formation.

Circle formation: The objective of the circle formation is for M robots to position themselves into an M-sided regular polygonal formation with a landmark at the center. We simulated using $M = 6$ robots with one additional robot stayed stationary at the desired center position. We modified the expert controller (7) to the following:

$$\begin{aligned}\mathbf{v}_i = &- K_c \sum_{j \in \mathcal{N}_i} \frac{\|\boldsymbol{p}_i - \boldsymbol{p}_j\| - d^*}{\|\boldsymbol{p}_i - \boldsymbol{p}_j\|} \cdot (\boldsymbol{p}_i - \boldsymbol{p}_j) \\ &- K_l \frac{\|\boldsymbol{p}_i - \boldsymbol{p}_l\| - d^*}{\|\boldsymbol{p}_i - \boldsymbol{p}_l\|} \cdot (\boldsymbol{p}_i - \boldsymbol{p}_l),\end{aligned} \tag{9}$$

where $d^* = 2$ is the radius of the circle, $p_l = (0, 0)$ is the position of the center robot, and the control gains were set to $K_l = 10$, $K_c = 1$. We ran the same training algorithm (i.e., Algorithm 1) with the modified expert control above with the same hyperparameters as before. Fig. 7(b) shows the testing result that 6 robots achieved the desired circle formation.

Comparison with other works: To empirically compare our method with other learning based method on formation control, we used the publicly available implementation of the work (Agarwal et al., 2020) to evaluate performances of both line formation and circle formation for 100 trials. In the case of line formation, the mean of the group formation error, $\overline{\mathcal{E}}_g/d^*$ (as defined in Section 4.3), obtained in Agarwal et al. (2020) was 7.4$\%$ with a standard deviation (SD) 2.9$\%$; and using our method, the mean was 1.8$\%$ with SD 1.1$\%$. In the case of circle formation, the formation error obtained in Agarwal et al. (2020) was 4.2$\%$ with SD 1.2$\%$, and ours was 3.4$\%$ with SD 1.4$\%$. Note that our simulation was implemented with realistic robot models in a robot simulator, while Agarwal et al. (2020) used a point mass robot model. Thus our method achieves comparable or better formation error with more complicated robot models.



(a) (b)

Figure 7. Other formation shapes: (a) Line formation; (b) Circle formation. Robot positions (denoted by colored small triangles) are sampled every 10 s. The black solid lines indicate the final formation achieved.

5 DISCUSSION

To further compare our proposed method with recent work on learning-based multi-robot control, we summarize in Table 3 the main differences. Existing methods can be categorized into reinforcement learning (RL) (such as Agarwal et al. (2020); Li et al. (2022); Yan et al. (2022); Blumenkamp et al. (2022)) and learning from demonstration (LfD) (such as Li et al. (2020); Tolstaya et al. (2020a) and our work), depending on the training paradigm. The RL method does not require an expert policy, but its trial-and-error nature could make the training intractable for multi-robot systems. The intractability issue exacerbates when a realistic environment is considered, where the dimensionalities of the robot state and observation spaces increase dramatically. As shown in the table, the RL-based methods (Li et al., 2022; Yan et al., 2022; Blumenkamp et al., 2022) were only validated for up to 5 robots when a realistic robot model was considered. It's noteworthy to mention that Li et al. (2022) incorporated LfD into RL to mitigate the training intractability issue. Agarwal et al. (2020) used up to 10 robots for validation, however, the robot model was simplified as point mass. In contrast, our method employs a LfD paradigm which exploits expert demonstrations to guide the control policy search, thus considerably reduces the policy search space. Our method was validated for up to 9 robots with a realistic robot model and high-dimensional observation space. Indeed, formation control of multi-robot systems has been well studied in the control regime using analytical model-based methods (Guo, 2017; Cortés and Egerstedt, 2017), and the dynamic model-based expert controller used in this paper is mathematically provably correct and guarantees formation convergence of multi-robot systems (Mesbahi and Egerstedt, 2010).

The scalability of a control policy was evaluated by testing it with different numbers of robots than that in training. Among the RL-based works, Yan et al. (2022) and Li et al. (2022) did not demonstrate the scalability of their methods. Agarwal et al. (2020) used a GNN architecture, but the zero-shot generalizability (i.e., a policy trained with a fixed number of robots is directly tested with a different number of robots) is low as the success rate of the learned policy decreases when the number of robots in testing differs from that in training. However, they showed that the scalability can be improved when curriculum learning is exploited. Li et al. (2020), Tolstaya et al. (2020a), and our work adopted GNN architectures with LfD training paradigm and demonstrated a high level of scalability. Another advantage of our approach is that it does not need localization to obtain global positions of the robots to compute control actions as required in other works (Agarwal et al., 2020; Li et al., 2022; Tolstaya et al., 2020a; Blumenkamp et al., 2022).

Table 3. Summary of works on learning-based multi-robot control

Reference	Tasks	Method/ Architecture	Robot Model	Policy Input	# of Robots Trained (Tested)	Scalability	Need Localization
Agarwal et al. (2020)	Coverage, line, formation	RL/GNN	Point mass	Absolute pose	5 (2-10)	Yes	Yes
Li et al. (2022)	Path planning	RL+LfD/ CNN+FC	Holonomic	LiDAR, velocity, position	3-5 (3-5)	No	Yes
Yan et al. (2022)	Formation + path planning	RL/RNN	Ackermann-steering	Distance, angle	3-5 (3-5)	No	No
Blumenkamp et al. (2022)	Path planning	RL/GNN	Holonomic	Absolute pose	5 (5)	–	Yes
Li et al. (2020)	Path planning	LfD/ CNN+GNN	Point mass	Binary map	4-12 (4-14)	Yes	No
Tolstaya et al. (2020a)	Flocking	LfD/GNN	Point mass	Absolute pose	100 (50-150)	Yes	Yes
This Paper	Triangulation formation	LfD/ CNN+GNN	Nonholonomic	LiDAR	5 (3-9)	Yes	No

"–" represents the case where result was not presented.

6 CONCLUSION

In this paper, we have presented a novel end-to-end decentralized multi-robot control for triangulation formation. Utilizing GNN's capability to model inter-robot communication, we designed GNN-based algorithms for learning scalable control policies. Experimental validation was performed in the robot simulator, CoppeliaSim, which has showed satisfactory performance for varying size of multi-robot teams. Future work includes implementation on real robot platforms and testings.

CONFLICT OF INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

AUTHOR CONTRIBUTIONS

All authors contributed to the technical approach. Chao Jiang led the paper writing. Xinchi Huang led coding and debugging. Yi Guo advised the team.

FUNDING

Xinchi Huang and Yi Guo were partially supported by the US National Science Foundation under Grants CMMI-1825709 and IIS-1838799.

ACKNOWLEDGMENTS

The authors would like to thank Suhaas Yerapathi, a former graduate student, for his work coding an early version of the algorithm.

SUPPLEMENTAL DATA

A movie file is submitted together with this paper.

REFERENCES

Agarwal, A., Kumar, S., Sycara, K., and Lewis, M. (2020). Learning transferable cooperative behavior in multi-agent teams. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 1741–1743

Bechlioulis, C. P., Giagkas, F., Karras, G. C., and Kyriakopoulos, K. J. (2019). Robust formation control for multiple underwater vehicles. *Frontiers in Robotics and AI* 6, 90

Blumenkamp, J., Morad, S., Gielis, J., Li, Q., and Prorok, A. (2022). A framework for real-world multi-robot systems running decentralized gnn-based policies. In *IEEE International Conference on Robotics and Automation*. 8772–8778

Chen, Z., Jiang, C., and Guo, Y. (2019). Distance-based formation control of a three-robot system. In *Chinese Control and Decision Conference*. 5501–5507

Cortés, J. and Egerstedt, M. (2017). Coordinated control of multi-robot systems: A survey. *SICE Journal of Control, Measurement, and System Integration* 10, 495–503

Devo, A., Mezzetti, G., Costante, G., Fravolini, M. L., and Valigi, P. (2020). Towards generalization in target-driven visual navigation by using deep reinforcement learning. *IEEE Transactions on Robotics* 36, 1546–1561

Falanga, D., Kim, S., and Scaramuzza, D. (2019). How fast is too fast? the role of perception latency in high-speed sense and avoid. *IEEE Robotics and Automation Letters* 4, 1884–1891

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 32

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H., Kohli, P., et al. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*. 1146–1155

Foerster, J. N., Assael, Y. M., De Freitas, N., and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*. 2137–2145

Gama, F., Isufi, E., Leus, G., and Ribeiro, A. (2020). Graphs, convolutions, and neural networks: From graph filters to graph neural networks. *IEEE Signal Processing Magazine* 37, 128–138

Gama, F., Marques, A. G., Leus, G., and Ribeiro, A. (2018). Convolutional neural network architectures for signals supported on graphs. *IEEE Transactions on Signal Processing* 67, 1034–1049

Gronauer, S. and Diepold, K. (2021). Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review* , 1–49

Guo, Y. (2017). *Distributed Cooperative Control: Emerging Applications* (Wiley)

Gupta, J. K., Egorov, M., and Kochenderfer, M. (2017). Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems* (Springer), 66–83

Jiang, C., Chen, Z., and Guo, Y. (2019). Learning decentralized control policies for multi-robot formation. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. 758–765

Jiang, C. and Guo, Y. (2020). Multi-robot guided policy search for learning decentralized swarm control. *IEEE Control Systems Letters* 5, 743–748

Jiang, J. and Lu, Z. (2018). Learning attentional communication for multi-agent cooperation. In *Proceedings of the International Conference on Neural Information Processing Systems*. 7265–7275

Kabore, K. M. and Güler, S. (2021). Distributed formation control of drones with onboard perception. *IEEE/ASME Transactions on Mechatronics* 27, 3121–3131

Kahn, G., Villaflor, A., Ding, B., Abbeel, P., and Levine, S. (2018). Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *IEEE International Conference on Robotics and Automation*. 5129–5136

Li, M., Jie, Y., Kong, Y., and Cheng, H. (2022). Decentralized global connectivity maintenance for multi-robot navigation: A reinforcement learning approach. In *IEEE International Conference on Robotics and Automation*. 8801–8807

Li, Q., Gama, F., Ribeiro, A., and Prorok, A. (2020). Graph neural networks for decentralized multi-robot path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 11785–11792

Loquercio, A., Kaufmann, E., Ranftl, R., Müller, M., Koltun, V., and Scaramuzza, D. (2021). Learning high-speed flight in the wild. *Science Robotics* 6, eabg5810

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the International Conference on Neural Information Processing Systems*. 6382–6393

Mesbahi, M. and Egerstedt, M. (2010). *Graph Theoretic Methods in Multiagent Networks* (Princeton University Press)

Mustapha, A., Mohamed, L., and Ali, K. (2020). An overview of gradient descent algorithm optimization in machine learning: Application in the ophthalmology field. In *Smart Applications and Data Analysis*, eds. M. Hamlich, L. Bellatreche, A. Mondal, and C. Ordonez (Cham: Springer International Publishing), 349–359

Panagou, D., Stipanović, D. M., and Voulgaris, P. G. (2015). Dynamic coverage control in unicycle multi-robot networks under anisotropic sensing. *Frontiers in Robotics and AI* 2, 3

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (JMLR Workshop and Conference Proceedings), 627–635

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks* 20, 61–80

Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots* 8, 345–383

Sukhbaatar, S., Fergus, R., et al. (2016). Learning multiagent communication with backpropagation. *Advances in Neural Information Processing Systems* 29, 2244–2252

Tolstaya, E., Gama, F., Paulos, J., Pappas, G., Kumar, V., and Ribeiro, A. (2020a). Learning decentralized controllers for robot swarms with graph neural networks. In *Conference on Robot Learning*. 671–682

Tolstaya, E., Paulos, J., Kumar, V., and Ribeiro, A. (2020b). Multi-robot coverage and exploration using spatial graph neural networks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 8944–8950

Wang, Y., Yue, Y., Shan, M., He, L., and Wang, D. (2021). Formation reconstruction and trajectory replanning for multi-uav patrol. *IEEE/ASME Transactions on Mechatronics* 26, 719–729

Wang, Z. and Gombolay, M. (2020). Learning scheduling policies for multi-robot coordination with graph attention networks. *IEEE Robotics and Automation Letters* 5, 4509–4516

Yan, Y., Li, X., Qiu, X., Qiu, J., Wang, J., Wang, Y., et al. (2022). Relative distributed formation and obstacle avoidance with multi-agent reinforcement learning. In *IEEE International Conference on Robotics and Automation*. 1661–1667

Zhang, Z. and Scaramuzza, D. (2018). Perception-aware receding horizon navigation for MAVs. In *IEEE International Conference on Robotics and Automation*. 2534–2541

Zhou, S., Phielipp, M. J., Sefair, J. A., Walker, S. I., and Amor, H. B. (2019). Clone swarms: Learning to predict and control multi-robot systems by imitation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 4092–4099

FIGURE CAPTIONS

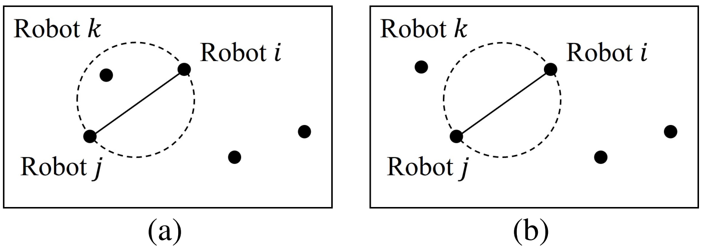


Figure 1. Gabriel graph: (a) robot i and j are not neighbors as robot k exists in the circle whose diameter is defined by the distance between robots i and j; (b) robot i and j are valid neighbors as there are no other robots in the circle.

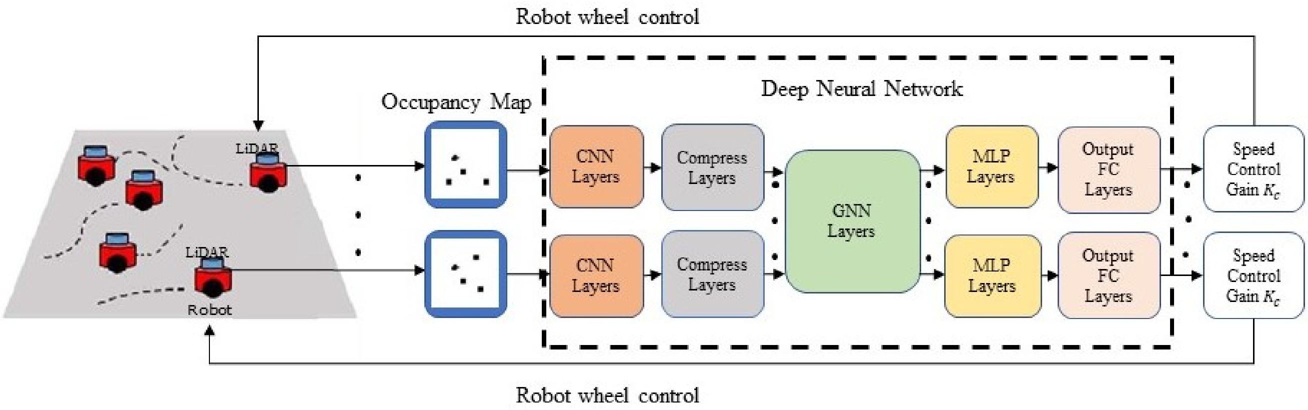


Figure 2. The overall diagram of the end-to-end GNN-based decentralized formation control.

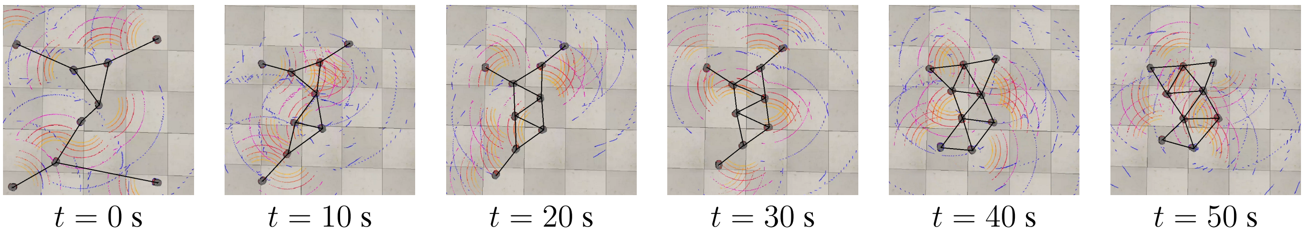


Figure 3. Snapshots of online formation control of 9 robots in CoppeliaSim simulator. The colored arcs visualize the LIDAR scanning.

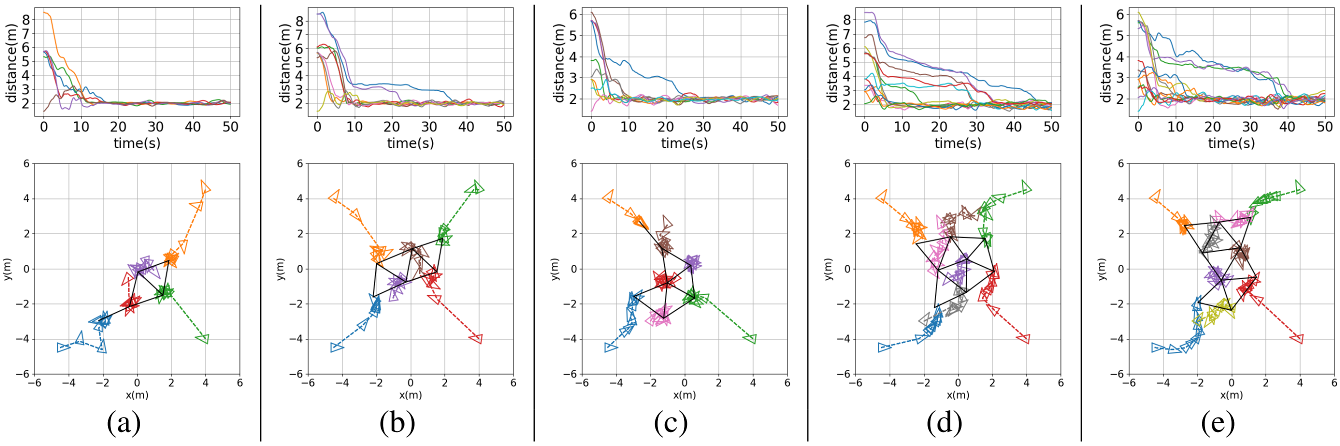
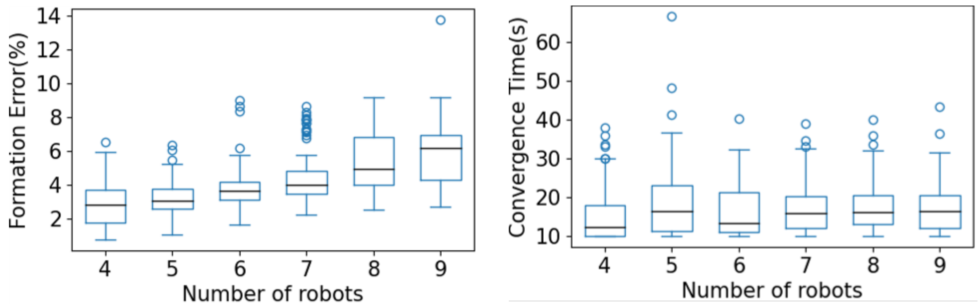


Figure 4. Testing results for: (a) five-robot team ($N = 5$), (b) six-robot team ($N = 6$), (c) seven-robot team ($N = 7$), (d) eight-robot team ($N = 8$), and (e) nine-robot team ($N = 9$). Top: Time histories of inter-robot distance, $d_{ij}(t), i = 1, \ldots, N, j \in \mathcal{N}_i$; Bottom: Robot trajectories with robot positions (denoted by colored small triangles) drawn every 10 s, and the black solid lines indicate the final formation achieved.



(a) (b)

Figure 5. Box plot for 100 initial conditions of multi-robot testing. The central mark in each box is the median, the edges of the boxes are the 25th and 75th percentiles, the whiskers extend to the maximum/minimum, and the circles represent outliers. (a) Group formation error for multi-robot teams with sizes from 4 to 9; (b) Convergence time for multi-robot teams with sizes from 4 to 9.

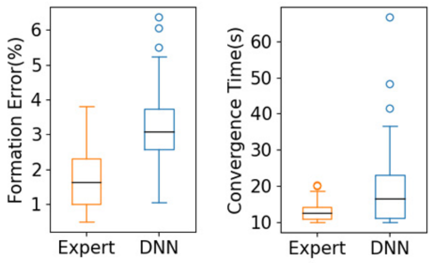
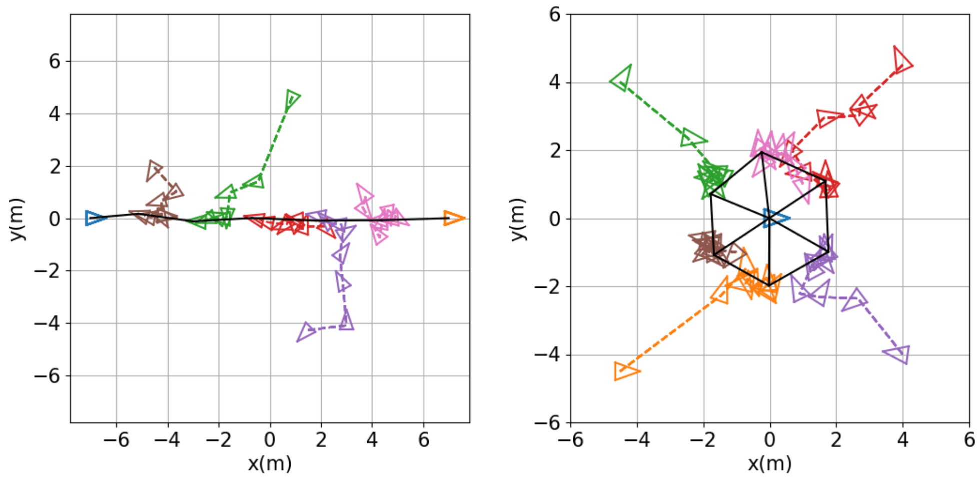


Figure 6. Comparison between the expert control and our trained DNN model for the 5-robot case with 100 initial conditions.



(a) (b)

Figure 7. Other Formation Shapes: (a) Line formation; (b) Circle formation. Robot positions (denoted by colored small triangles) are drawn every 10 s. The black solid lines indicate the final formation achieved.