Proceedings of the 2019 IEEE/ASME
International Conference on Advanced Intelligent Mechatronics
Hong Kong, China, July 8-12, 2019

# Learning Decentralized Control Policies for Multi-Robot Formation

Chao Jiang, Zhuo Chen, and Yi Guo

*Abstract*— **Decentralized formation control has been extensively studied using model-based methods, which rely on model accuracy and communication reliability. Motivated by recent advances in deep learning techniques whereby an intelligent agent is trained to compute its actions directly from high-dimensional raw sensory inputs using end-to-end decision-making policies, we consider the problem of learning decentralized control policies for multi-robot formation. A deep neural network is designed to model the control policy that maps the robot's local observations to control commands. We propose to use a centralized training framework based on supervised learning for control policy learning. The learned policy is then deployed on each robot in a decentralized manner for online formation control. Our proposed approach is verified and evaluated in experiments using a robotic simulator. Simulation results demonstrate satisfactory performance of formation control. Compared with existing methods for formation control, the proposed approach does not need inter-robot communication, and avoids hand-engineering the components of perception and control separately.**

## I. Introduction

Multi-robot formation control has been an active research area due to its significance in practical applications such as exploration of unknown environments, cooperative security patrols, and autonomous transportation [1]. Conventional model-based methods explicitly model the kinematics (and/or dynamics) of the robots and the sensing and/or communication graphs to generate decentralized control input to each robot [2], [3]. Model-based methods generate analytic control inputs and can be implemented efficiently in real time for decentralized formation control. However, model-based methods highly depend on model accuracy, and are vulnerable to model uncertainties and external disturbances. Also, many model-based methods rely on communication to exchange neighboring state information, thus put stress on communication reliability and security. Motivated by recent advances in deep learning methods with enhanced capability in perception representation [4], [5], we propose a different approach, the learning-based approach, for decentralized formation control. We consider the problem of learning robot control policies to achieve multi-robot formation from the robot's local observation without inter-robot communication. Particularly, we address the problem: 1) given expert demonstration data of decentralized formation control, whether we can learn robot control policies; and 2) whether we can use learned policies to control the robots achieving formation using raw sensor inputs in a decentralized manner.

*This work was partially supported by the US National Science Foundation under Grants CMMI-1527016, CMMI-1825709, and IIS-1838799.

The authors are with the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ, USA 07030
Emails:{cjiang6,zchen39,yi.guo}@stevens.edu

### A. Related Work

Decentralized formation control of multi-agent systems was initially inspired from the natural phenomena of bird flocking and fish schooling, where a group of agents follow the same direction and speed while maintaining certain geometric shape [1]–[3]. Existing model-based methods of multi-agent formation control can be classified into *position-, displacement-, and distance-based* controls, where the desired formation is defined by the *absolute position* of each agent with respect to a global reference frame, the desired *displacements* with respect to a global reference frame, and the desired *inter-agent distances*, respectively [3]. There is a tradeoff between the sensing capability and interaction topology in the above categorization, and distance-based control requires less sensing capability but more interactions among agents. Recently, researchers proposed vision-based control schemes to alleviate the requirement of inter-robot communication [6]–[9]. For example, in [9], the dynamics between the leader and follower robots is modeled using the distance, orientation and bearing angle, and a dynamic feedback controller was designed using an observer that estimates the neighbor's speed. These model-based formation control methods either require the availability of full or partial state measurement, or rely on deliberate perception module that explicitly returns measurements (such as bearing and distances) from robot sensors through traditional sensor fusion techniques.

In the last few years, a new research direction has been actively pursued, which utilizes the enormous expressive capability of deep neural nets to directly process high-dimensional raw sensing data for self-driving cars [10], [11]. Also, the human-level intelligence for robot control has been proposed with the aim to develop robot decision-making policies that generate actions directly from raw observation, so as to mimic the functioning and learning process of human brains [4]. Recent deep learning methods have achieved great success in learning representation from raw sensing data using deep neural networks (DNN), and thus provide an end-to-end framework for learning robot control policies that combine multiple decoupled modules from robot perception to action.

DNNs and various learning algorithms have been developed for learning control policies in a vast range of robotic problems. To briefly mention some important work in this new direction, Giusti et al. [12] trained a DNN model to predict the direction of forest trails from the images captured by a quadrotor's onboard camera. The predicted trail direction was used for reactive control of quadrotor navigation. The model was trained via supervised learning with real-world dataset and generated accurate prediction comparable to human performance. In [11], Rausch et al. designed and trained an end-to-end controller that calculates steering commands of autonomous vehicles given the image observation from a front-facing camera. In [13], Zhang et al. proposed a learning approach based on model predictive control and guided policy search to learn deep control policies for autonomous aerial vehicles. Long et al. [14] studied the supervised learning of the DNN policy for multi-robot collision avoidance based on a centralized training framework. Foerster et al. [15] proposed end-to-end frameworks for learning inter-robot communication protocols to achieve collective task objectives. The aforementioned work demonstrates promising results of the end-to-end policy learning from robots' raw observation in different robotic applications.

### B. Contribution and Organization

In this paper, we focus on the problem of learning decentralized formation control policies for multi-robot systems, which can directly operate on individual robots' local perception without inter-robot communication. To this end, we designed a DNN model that maps the robot's observation via onboard LIDAR sensor to robot motor control. The model was trained in a centralized manner using supervised learning with expert demonstration data generated by a model-based controller. The trained model is then deployed on each robot as a decentralized controller that only relies on local observation to achieve formation. The proposed approach is verified and evaluated throughout extensive set of experiments in V-REP simulator. Robot simulations show that our approach achieves over $90\%$ success rate, and the performance (such as formation errors) is satisfactory.

The contributions of our work are twofold: 1) A novel DNN-based approach is proposed to learn decentralized control policies from robots' local observations rendered in raw sensing data. The method avoids the need of inter-robot communication, and relieves the efforts of hand-engineering the components of perception and control separately that is commonly used in model-based formation control methods. 2) The technical challenge of neural network architecture design for policy representation is addressed, and the input design is specified for formation control. To the best of our knowledge, this is the first paper that addresses decentralized formation control for multi-robot systems using deep learning methods.

The rest of the paper is organized as follows: Section II formulates the problem of learning decentralized formation control policy for multi-robot systems. Section III provides the design of the proposed control policy learning approach. We demonstrate the simulation results and evaluation of our approach in Section IV. We conclude our work and discuss future work in Section V.

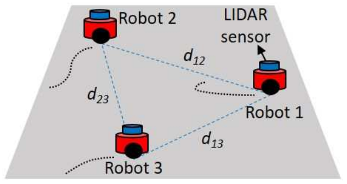


Fig. 1: The schematic diagram of the formation control scenario.

## II. Problem Statement

In this paper, we consider the formation control problem with three mobile robots that are equipped with LIDAR sensors as shown in Fig. 1. Each robot has an on-board self-localization system that can measure its current orientation, $\phi_i(t)$. The desired formation is defined as the desired orientation, $\phi^*$, desired speed (which is a scalar), $v^*$, and the desired relative distance between any two robots, $d_{ij}$ for $i, j = 1, 2, 3$ and $i \neq j$. For simplicity, we assume $d_{ij} = d^*$ that is a positive constant. The three robot team achieves formation if the robots follow the desired orientation $\phi^*$ and the desired speed $v^*$, and keep the distance $d^*$ from each other.

The decentralized control policy can be regarded as a generalization of a decentralized control law. In our case, the decentralized control policy maps the input of the robot's local sensor observation (i.e., occupancy map from the robot onboard LIDAR sensor) to the output of the robot's motor control.

We address the problem of learning a decentralized control policy and using it for online formation control. For this purpose, we consider two phases in the paper, the training phase and the testing phase as shown in Fig. 2. In the training phase, the objective is to learn an end-to-end control policy:

$$\hat{\boldsymbol{u}}_i = \boldsymbol{F}(\boldsymbol{y}_i, \Delta\phi_i, d^*), \tag{1}$$

where the input of the control policy consists of the occupancy map $\boldsymbol{y}_i$ and two auxiliary inputs including the difference between the robot's current orientation and the desired orientation, $\Delta\phi_i = \phi_i - \phi^*$, and the desired formation distance, $d^*$; and the control policy outputs the robot motor control $\hat{\boldsymbol{u}}_i = [\hat{u}_{il}, \hat{u}_{ir}] \in \mathbb{R}^2$ with $\hat{u}_{il}$ and $\hat{u}_{ir}$ being the left and right motor control, respectively. A DNN model is designed as the parameterized function representation of the control policy $\boldsymbol{F}(\cdot)$ defined in (1).

During the training phase, expert demonstration is needed to train the DNN, which can be obtained either by a human driver that demonstrates the appropriate steering actions given a certain vehicle state, or by model-based methods to generate analytic control laws with full state measurement. In this paper, we use the model-based method to generate expert demonstration. Specifically, given the robot states obtained by the measurement system, the model-based controller generates the robot control, $\boldsymbol{u}_i$, as expert demonstration. Meanwhile, the corresponding local observation, $\boldsymbol{y}_i$, from the robot's onboard LIDAR sensor and the auxiliary inputs,

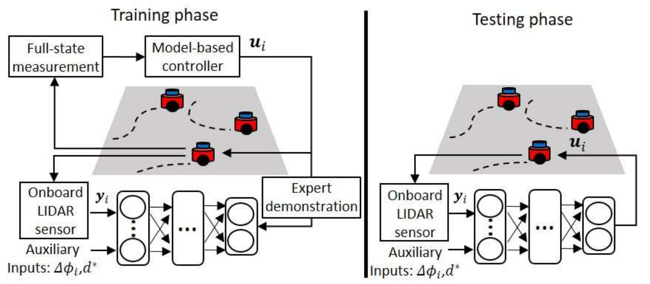


Fig. 2: The overview of our proposed policy learning scheme.

$\Delta\phi_i$ and $d^*$, are recorded. The expert control $\boldsymbol{u}_i$, LIDAR observation $\boldsymbol{y}_i$ and the auxiliary data $\Delta\phi_i$, $d^*$ are then fed to the DNN for training.

In the testing phase, the trained DNN policy is executed on each robot $i$ in a decentralized manner to compute robot control command through a feedforward pass of DNN based on current robot LIDAR observation and auxiliary inputs. The robot motor control law $\boldsymbol{u}_i = K\hat{\boldsymbol{u}}_i$, where $K$ is a constant gain that adjusts the robot's speed to the desired speed $v^*$. Note that the common speed of the robot team is not an input to the DNN in the training phase, and can be adjusted directly during the testing phase through linear scaling.

In the next section, we present our DNN architecture that represents the control policy, and a supervised learning method to train the policy with expert demonstration data that collectively represent a latent observation-to-control mapping.

## III. Proposed Approach

The overall diagram of the learning-based formation control is illustrated in Fig. 3. During online testing, each robot acquires the occupancy map $\boldsymbol{y}_i$ from its onboard LIDAR observation and the robot's current orientation $\phi_i$ accessed via robot self-localization or compass. The occupancy map $\boldsymbol{y}_i$, the orientation difference $\Delta\phi_i$ and the desired formation distance $d^*$ are passed to the DNN policy to compute the robot control $\hat{\boldsymbol{u}}_i$. Note that the learned DNN policy drives the robot to converge to the desired speed $v^*$. The speed of the robots can be adjusted by scaling the policy output $\hat{\boldsymbol{u}}_i$ with a gain $K$.

### A. Model Architecture

The capability of DNNs to model complex and nonlinear functions lends themselves well to the control policy representation. We propose a DNN as shown in the dotted-line box of Fig. 3, which is composed of a convolutional neural network (CNN) and a fully-connected (FC) network to approximate the control policy function defined in (1). The parameterized representation of the control policy using the DNN is denoted as $\hat{\boldsymbol{u}}_i = \boldsymbol{F}(\boldsymbol{y}_i, \Delta\phi_i, d^*; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the parameters of the DNN model.

*1) Model Input:* The input of the control policy model is the three-tuple $(\boldsymbol{y}_i, \Delta\phi_i, d^*)$ including the grayscale occupancy map $\boldsymbol{y}_i$ with both width and height size of 50 and channel size of 1 (i.e. $50\times50\times1$), the difference between the desired orientation and the current robot orientation, $\Delta\phi_i$, and the desired formation distance $d^*$.

*2) Model Output:* The output of the control policy model is a 2-dimensional vector of continuous control command of the robot's motor control $\hat{\boldsymbol{u}}_i$.

*3) CNN:* The CNN functions as feature representation that extracts discriminate features capturing the information of other robots from LIDAR observation. The CNN has three convolutional layers. The first convolutional layer generates a number of feature maps by a convolution operation on the image input, and the last two convolutional layers generate several feature maps given the output from the previous layer, respectively. Let $\boldsymbol{z}^j_{n,\nu}$ denote the value of the $j$th feature map at coordinate $(n, \nu)$ generated by each convolutional layer, then $\boldsymbol{z}^j_{n,\nu}$ is given by

$$\boldsymbol{z}^j_{n,v} = \sigma\left(\boldsymbol{b}_j + \sum_{l=1}^{L}\sum_{m=1}^{M} \boldsymbol{W}^j_{l,m} * \boldsymbol{o}_{n+l,v+m}\right), \tag{2}$$

where $\boldsymbol{W}^j_{l,m}$ represent the value of the $L \times M$ weight matrix $\boldsymbol{W}^j$ at coordinate $(l, m)$; $\boldsymbol{b}_j$ represents the bias; $\boldsymbol{o}_{n+l,v+m}$ denotes the value of the image input or the feature map of the previous layer at the coordinates $(n + l, v + m)$. The results of the convolution operation at each convolutional layer is activated by the rectified linear unit (ReLU), denoted as $\sigma(\cdot)$, which is given by

$$\sigma(z) = \max(0, z) \tag{3}$$

The output of the last convolution layer is then flatten into a vector as the input of the fully-connected network. The details of arithmetic operations of CNN can be found in [16].

*4) FC Network:* The 2-layer fully-connected network is used to uniformly approximate the continuous function that fuses the extracted features from the CNN, the orientation difference $\Delta\phi_i$, the desired formation distance $d^*$, and outputs the robot control command $\hat{\boldsymbol{u}}_i$. Each layer of the FC network consists of a number of hidden units, the output of which is given by

$$\boldsymbol{h} = g\left(\boldsymbol{W_h} * \boldsymbol{d} + \boldsymbol{b}_h\right) \tag{4}$$

where $\boldsymbol{W}_h$ and $\boldsymbol{b}_h$ represent the weights and bias, respectively; $\boldsymbol{d}$ denotes the vector of output from the previous layer. $g(\cdot)$ denotes the activation operation. The output of the first and second layer are followed by ReLU and linear activation, respectively.

### B. Training Phase

The training process is essentially to optimize the DNN parameters $\boldsymbol{\theta}$ such that given the three-tuple $(\boldsymbol{y}_i, \Delta\phi_i, d^*)$, the error between the DNN output, $\hat{\boldsymbol{u}}_i = \boldsymbol{F}(\boldsymbol{y}_i, \Delta\phi_i, d^*; \boldsymbol{\theta})$, and the expert control $\boldsymbol{u}_i$ provided by expert demonstration is minimized.

We define the loss function for each iteration of the mini-batch learning as the Euclidean loss:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_B}\sum_{j=1}^{N_B} \|\boldsymbol{F}(\boldsymbol{y}_j, \Delta\phi_j, d^*; \boldsymbol{\theta}) - \boldsymbol{u}_j\|^2 \tag{5}$$

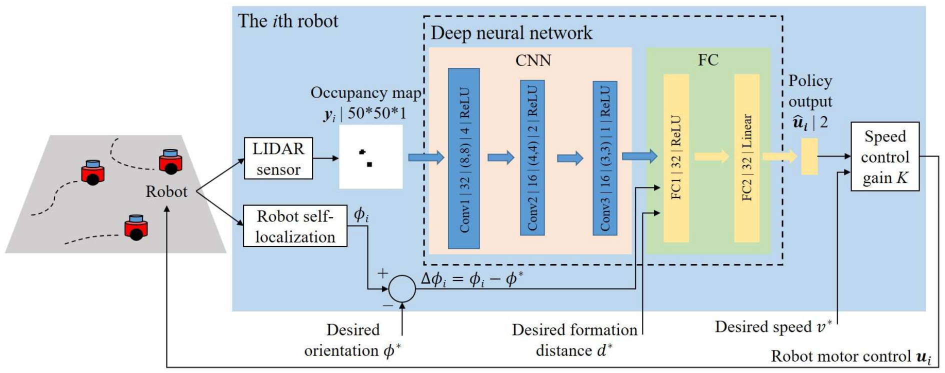


Fig. 3: The overall diagram of the DNN-based formation control policy.

```
Algorithm 1: Training of DNN
Input  : Occupancy map image $\boldsymbol{y}$, orientation difference
         $\Delta\phi$, desired formation distance $d^*$, and expert
         control $\boldsymbol{u}$;
Output: DNN parameters $\boldsymbol{\theta}$;
1  Initialize: DNN parameters $\boldsymbol{\theta}_0$;
2  for epoch $t \leftarrow 1$ to $T$ do
3      Initialize: first moment vector $\boldsymbol{m}_0$;
4      Initialize: second moment vector $\boldsymbol{v}_0$;
5      Initialize: learning rate $\eta_0$;
6      for batch $k \leftarrow 1$ to $B$ do
7          for sample $l \leftarrow 1$ to $N_B$ do
8              Calculate predicted control
               $\hat{\boldsymbol{u}}_l = \boldsymbol{F}(\boldsymbol{y}_l, \Delta\phi_l, d^*; \boldsymbol{\theta}_{k-1})$;
9              Calculate loss $\mathcal{L}_l(\boldsymbol{\theta}_{k-1}) = \|\hat{\boldsymbol{u}}_l - \boldsymbol{u}_l\|^2$;
10         end
11         $\mathcal{L}(\boldsymbol{\theta}_{k-1}) = \sum_{l=1}^{N_B} \mathcal{L}_l(\boldsymbol{\theta}_{k-1})/N_B$;
12         Calculate gradient $\boldsymbol{g}_k \leftarrow \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_{k-1})$;
13         Update first moment
           $\boldsymbol{m}_k \leftarrow \beta_1 \cdot \boldsymbol{m}_{k-1} + (1-\beta_1) \cdot \boldsymbol{g}_k$;
14         Update second moment
           $\boldsymbol{v}_k \leftarrow \beta_2 \cdot \boldsymbol{v}_{k-1} + (1-\beta_2) \cdot \boldsymbol{g}_k^2$;
15         Update learning rate
           $\eta_k \leftarrow \eta_0 \cdot \sqrt{1-\beta_2^k}/(1-\beta_1^k)$;
16         Update DNN parameters
           $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} - \eta_k \cdot \boldsymbol{m}_k/(\sqrt{\boldsymbol{v}_k} + \hat{\epsilon})$;
17     end
18 end
```

where $N_B$ is the mini-batch size.

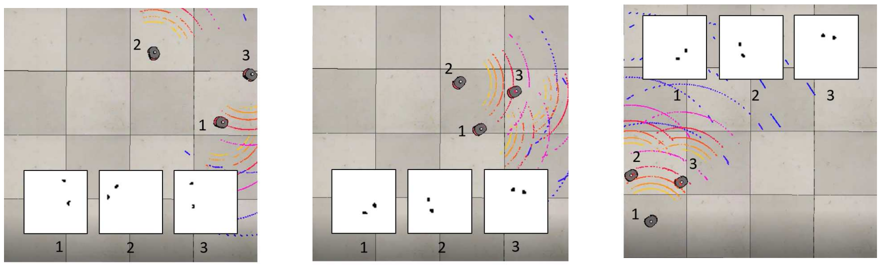
We use a gradient-descent based training algorithm with Adam optimizer [17] to learn the optimal parameters of the DNN. Adam optimizer uses adaptive learning rates for each parameters in the DNN model. The training algorithm is summarized in Algorithm 1. The algorithm is fed with the four-tuples of occupancy map image $\boldsymbol{y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3]$, the orientation difference $\Delta\phi = [\Delta\phi_1, \Delta\phi_2, \Delta\phi_3]$, the desired formation distance $d^*$, and the expert control $\boldsymbol{u}$ as input. The output is the optimal DNN parameters $\boldsymbol{\theta}$. Note that the training is conducted in a centralized manner, and we do not distinguish each individual robot in the training phase.

Before the training process, the parameters of the DNN are randomly initialized as $\boldsymbol{\theta}_0$. The training process consists of $T$ epochs. In each epoch $t \in [1, T]$, the initial learning rate $\eta_0$, the initial estimates of the first moment $\boldsymbol{m}_0$ and the second moment $\boldsymbol{v}_0$ are selected. Each epoch comprises $B$ iterations of parameters updating, where $B$ is the number of mini-batches. In each iteration $k \in [1, B]$, one mini-batch containing $N_B$ sampled data are used to update the DNN parameters. Specifically, given each sampled data $l \in [1, N_B]$, we calculate the predicted control $\hat{\boldsymbol{u}}_l$ through a feedforward pass in line 8. The loss $\mathcal{L}_l(\boldsymbol{\theta}_{k-1})$ associated with the $l$th sample is calculated as the squared error between the predicted control and the expert control, as shown in line 9. Then, we take an average of the loss over the $N_B$ samples in line 11. The gradient of the averaged loss, $\boldsymbol{g}_k$, is calculated in line 12. Then, the estimates of the first moment $\boldsymbol{m}_k$ and the second moment $\boldsymbol{v}_k$ are updated in line 13 and 14, where $\beta_1$ and $\beta_2 \in [0, 1)$ are the exponential decay rates for the moment estimates, and $\boldsymbol{g}_k^2$ denotes the element-wise square. Then, the learning rate $\eta_k$ is updated in line 15, where $\beta_1^k$ and $\beta_2^k$ denote $\beta_1$ and $\beta_2$ to the power $k$, respectively. The parameters of DNN is updated via gradient descent rule shown in line 16 toward the minimization of the loss function, where $\hat{\epsilon}$ is a small constant for numerical stability. After $T$ epochs of training process, the DNN parameters $\boldsymbol{\theta}$ converge to optimum in the sense that the loss function (5) is minimized. The DNN model with the learned parameters is then used by each robot in online testing phase.

## C. Testing Phase

In the online testing phase, the trained model is deployed on each robot in a decentralized manner. That is, the DNN model is executed on each robot to compute its motor control command given its own observation via onboard sensor as input.

The online formation control algorithm applied to each robot is summarized in Algorithm 2. The algorithm takes as input the occupancy map image $\boldsymbol{y}_i(t)$, the robot orientation $\phi_i$, the desired robot orientation $\phi^*$, and the desired formation distance $d^*$. The algorithm outputs the robot motor control. At each time step $t \in [1, T]$, each robot obtain an



(a) t=0s (b) t=5s (c) t=15s

Fig. 4: Snapshots of online formation control experiment in V-REP simulator at (1) t=0s; (b) t=5s; and (c) t=15s. The squared images show the occupancy map generated from the observation of the corresponding robot, where the black dots indicate the surrounding robots of each robot. The colored arcs visualize the LIDAR scanning.

**Algorithm 2:** Online Formation Control

```
Input  : Occupancy map image $y_i(t)$, robot orientation
         $\phi_i(t)$, desired robot orientation $\phi^*$, and desired
         formation distance $d^*$;
Output: Robot motor control $u_i(t)$;
1  Load the DNN model with parameters $\theta$ trained using
   Algorithm 1;
2  Initialize robot position $p_i(0)$ and orientation $\phi_i(0)$;
3  for Time step $t \leftarrow 1$ to $T$ do
4      Obtain an image of occupancy map $y_i(t)$ via
       LIDAR observation;
5      Obtain robot orientation $\phi_i(t)$ via self-localization
       system;
6      Calculate orientation difference
       $\Delta\phi_i(t) = \phi_i(t) - \phi^*$;
7      Calculate DNN policy output $\hat{u}_i(t)$ through a
       feedforward pass;
8      Calculate robot motor control $u_i(t) = K\hat{u}_i(t)$;
9      Output $u_i(t)$;
10 end
```

image of occupancy map $\boldsymbol{y}_i(t)$ via its current LIDAR observation and the current orientation $\phi_i(t)$. It then calculates the difference between the its current and desired orientation, $\Delta\phi_i(t)$. Then the three-tuple $(\boldsymbol{y}_i(t), \Delta\phi_i(t), d^*)$ is passed to the DNN model as input, which computes the policy output $\hat{\boldsymbol{u}}_i(t)$. Then, the policy output is multiplied with the control gain $K$ to generate the motor control command $\boldsymbol{u}_i$, where $K$ can be chosen as $K = v^*/v_{train}$ with $v_{train}$ being the common robot speed used in the training phase.

## IV. Experiment Results

In this section, we present the results of the proposed policy learning approach and verify the effectiveness of the learned policy in simulation experiments of online formation control with different desired formation distances and velocities.

### A. Experiment Setup

*1) Robot Simulation:* The robot simulation is performed in the robot simulator V-REP [18]. V-REP is a virtual robot experimentation platform which provides realistic robot dynamics and 3D rendering. Fig. 4 shows the snapshots of a simulation experiment in V-REP. We choose the differential drive mobile robot, Pioneer P3-DX, as the robot platform. The robot is equipped with a Velodyne VPL16 LIDAR sensor to perform laser scan measurement. The data of laser scan measurement in a square region of $5 \times 5$ m$^2$ centered at the sensor position are represented by a $50 \times 50$ gray-scale image of 2D occupancy map. Thus, the spatial resolution of the occupancy map is $0.1$ m/pixel. Note that, given the same laser scan range, increasing the spatial resolution of the occupancy map results in a larger size of the image and more computational cost. The tradeoff between the image size and the spatial resolution of the occupancy map can be balanced considering practical requirements of applications. The robot control algorithms are implemented in a client program written in Python, which communicates with the V-REP simulator via the provided remote API. Thus, the V-REP simulator sends simulation data to the client program for robot control calculation, and computes the robot dynamics given the control commands received from the client program. The simulation frequency is set to 20 Hz.

*2) DNN Implementation:* The first convolutional layer (Conv1) has 32 filters with filter size 8×8 and stride 4. The output of the Conv1 layer is 32 feature maps of dimension 12×12. The second convolutional layer (Conv2) has 16 filters with size 4×4 and stride 2. The output of the Conv2 layer is 16 feature maps of dimension 5×5. The final convolutional layer (Conv3) has 16 filters with size 3×3 and stride 1. The output of the Conv3 layer is 16 feature maps of dimension 3×3. All three convolutional layers are activated by ReLU. The feature maps of the last convolutional layer are flattened into a 144-dimensional vector which is fed to the FC network. The dimension of each hidden layer of the FC network is 32, and the dimension of the output layer is 2. The first and second layer are followed by ReLU and linear activation, respectively.

*3) Computer Configuration:* The simulation experiments are conducted on a computer with an Intel®XEON$^{\text{TM}}$E3-1535M (3.1 GHz × 8) CPU and an Nvidia Quadro®M2200 GPU. The robot simulator V-REP and the model-based controller program for data collection run on the CPU, and our

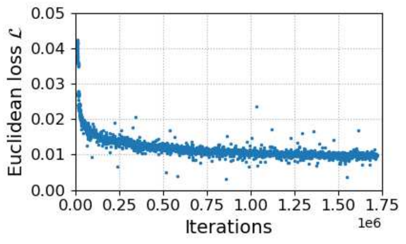


Fig. 5: Euclidean loss over training iterations.

Tensorflow programs implementing the training and testing of the proposed DNN run on a GPU for faster computation.

## B. *Training Experiments*

*1) Data Collection:* The training data were collected in the robot simulator V-REP, where a model-based controller was implemented to control each robot to provide expert demonstration data. We use the distance-based formation controller proposed in [19] to generate the motor control for the robots. We create a simulation scenario with three mobile robots. The robots' initial positions $\boldsymbol{p}_i(0)$ are randomly selected from a circular region with a radius of 3 m, and the initial orientations $\phi_i(0)$ are randomly selected from $[0, 2\pi)$. For each simulation run during data collection, a constant desired orientation $\phi^*$ is randomly selected from $[0, 2\pi)$, and the desired speed $v^*$ is set to $0.7$ m/s. The desired formation distance $d^*$ is randomly selected from the set $\{1, 1.5, 2\}$ m. The duration of each simulation run is 12 s. At each time step $t$, the control output $\boldsymbol{u}_i(t)$ computed by the model-based controller, the corresponding occupancy map $\boldsymbol{y}_i(t)$ from LIDAR observation, the orientation difference $\Delta\phi_i(t)$ of each robot $i$, $i = 1, 2, 3$, and the desired formation distance $d^*$ was recorded and stored as one sample of training data. The entire training data set includes $110160$ samples, which were grouped into $3442$ mini-batches with the batch size $N_B = 32$. The sampled data collected by all three robots are used to train the control policy in a centralized training framework.

*2) Training Results:* With the collected training data, we then train the DNN model using Algorithm 1 to find an appropriate control policy. The initial learning rate is set as $\eta_0 = 0.0001$; the exponential decay rates for the moment estimates are initialized as $\beta_1 = 0.9$ and $\beta_2 = 0.999$, respectively. The constant $\hat{\epsilon}$ is set to $10^{-8}$. The training epoch is set as $T = 500$. The evolution of loss over training iterations is shown in Fig. 5. One can see that the loss of the Adam optimizer converges around $0.009$ after $1.2 \times 10^6$ iterations of training. The training results demonstrate that our proposed learning scheme is converging and minimizes difference between the predicted control output and the expert control. Next, we verify and evaluate the learned policy in online testing experiments.

## C. *Testing Experiments*

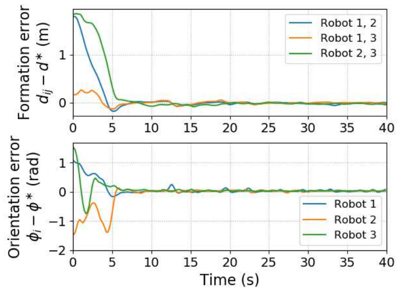
After sufficient training, the parameters of the DNN model remain unchanged, and the DNN model is deployed on each robot for online formation control. As shown in Fig. 2, the learned policy model calculates the control command from the robot's own observation via the onboard LIDAR sensor and the auxiliary inputs. We evaluate the learned robot control policy in the cases of constant and time-varying desired velocity, respectively. In all the online testing experiments, the magnitude of the desired velocity is set to $v^* = 0.7$ m/s, thus the speed control gain $K = 1$ as the speed after convergence given by the trained control policy is $0.7$ m/s.

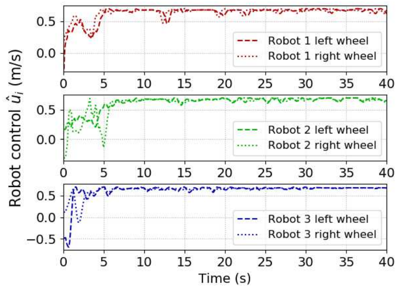
*1) Case 1 with Constant Desired Velocity:* We first show a testing case with a constant desired velocity in Fig. 6. The desired orientation is $\phi^* = 3.76$ rad, and the desired formation distance is 2 m. One can see from Fig. 6a that the formation error between the robots and orientation error of each robot decrease and converge around zero after about 6 s. Fig. 6b shows the control commands of the robots' left and right wheel speed. It can be seen that the control of the robot wheel speed converges around the desired speed $0.7$ m/s of the robot after 6 s. Fig. 6c shows the trajectories of the robots, where the triangles and the dotted lines represent the position of the robots and the formation at every 3 seconds, respectively. We can see that the group of robots achieves the desired formation and moves at the desired velocity.

*2) Cases 2 and 3 with Time-Varying Desired Velocity:* In these cases, the desired orientation $\phi^*(t)$ changes as a function of time. Note that, in the training phase, only constant desired orientations $\phi^*$ are used to train the DNN, and the time-varying desired velocities are not in the training set. In this subsection, we show that we can achieve formation with time-varying desired velocities. This is due to the reason that $\Delta\phi$ (i.e., the difference between the robot's current orientation and the desired orientation) is chosen as the input to the DNN that learns the mapping from $\Delta\phi$ after training.

For Case 2, we set the desired orientation as $\phi(t) = \omega \cdot t$ with the angular velocity $\omega = 0.1$ rad/s, and the desired formation distance is 1 m. This setup requires the robot team to keep a desired formation and meanwhile moves in a circular trajectory. The testing results of this case are shown in Fig. 7. One can see from Fig. 7a that the formation and orientation errors decrease and converge around zero after 20 s. Fig. 7b shows the control output of the robots' left and right wheel speed. It can be seen that the speed control of the robots converges around $0.7$ m/s after 20 s. Fig. 7c shows the trajectories of the robots, where the triangles and the dotted lines represent the position of the robots and the formation at every 3 seconds, respectively. We can see that the group of robots achieves the desired formation and moves at the desired velocity.

For Case 3, we set the desired orientation as $\phi(t) = \sin \omega t$ with the angular velocity $\omega = 0.2$ rad/s, and the desired formation distance is selected as $1.5$ m. This means the robot team is required to keep a desired formation and meanwhile moves in a sinusoidal-like trajectory where the desired orientation is changing sinusoidally. From Fig. 8a one can see that the formation and orientation errors decrease and converge around zero after $15$ s. Fig. 8b shows the control command of robot wheel speed which converge around $0.7$


(a)


(b)

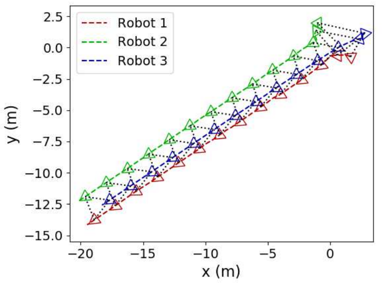
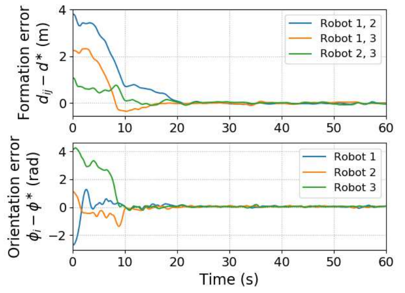
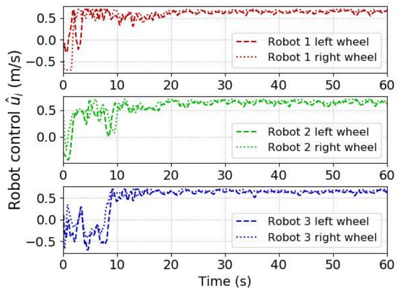

(c)

Fig. 6: Case 1 with constant desired orientation $\phi^* = 3.76$ rad and desired formation distance $d^* = 2$ m: (a) formation and orientation error; (b) robot control; (c) robot trajectories.


(a)


(b)

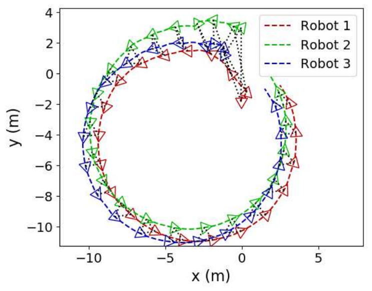
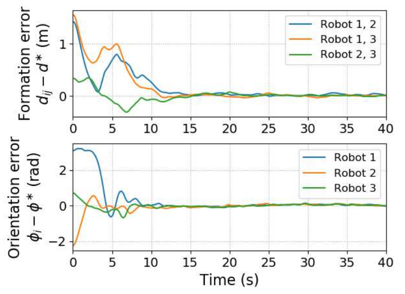
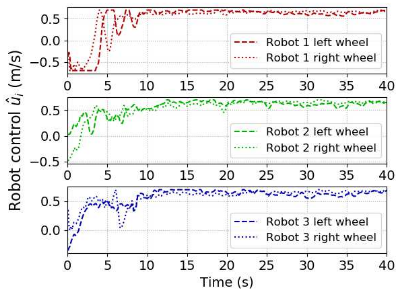

(c)

Fig. 7: Case 2 with time-varying desired orientation $\phi^*(t) = 0.1t$ rad and desired formation distance $d^* = 1$ m: (a) formation and orientation error; (b) robot control; (c) robot trajectories.


(a)


(b)

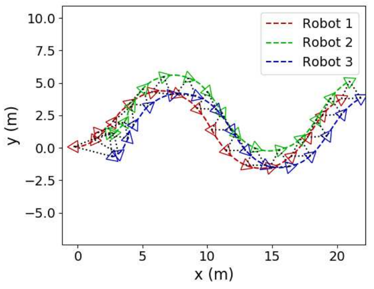

(c)

Fig. 8: Case 3 with time-varying desired orientation $\phi^*(t) = \sin(0.2t)$ rad and desired formation distance $d^* = 1.5$ m: (a) formation and orientation error; (b) robot control; (c) robot trajectories.

m/s after 15 s. The trajectories of the robots are shown in Fig. 8c.

The computational cost of calculating the robot control by the DNN model given an occupancy map in online testing is 2.1 ms on average over 100 runs, which satisfies real-time control requirement.

*3) Statistical Results:* We conducted extensive simulation experiments to verify our proposed approach for learning formation control policies, and present the statistical results of 100 simulation runs. The simulation time is set to 60 s. We consider a simulation run successful if the formation error between any robot $i$ and $j$, $i \neq j$, converges in the sense that the temporal average of the relative formation error $|d_{ij} - d^*|/d^*$ over the most recent 20 s is smaller than 5% or 10%. The statistic results of our approach are summarized in Table I which shows our approach achieves 90% and 96% success rate under 5% and 10% error metrics, respectively. To further evaluate the performance of our approach, we show the statistical results of the convergence time, the relative formation error and the orientation error over all successful runs under 10% error metrics in Fig. 9. One can see that the median convergence time of our approach is about 10 s. The median relative formation error is 2.5% and the median orientation error is 0.031 rad. Compared with existing model-based formation control methods with relative formation errors of 1% − 10% in general ([3]), the performance of our learning-based method is satisfactory.

The formation errors in online testing experiments are

TABLE I: Statistical results over 100 runs.

| | 5% error tolerance | 10% error tolerance |
|---|---|---|
| Convergence percentage (%) | 90 | 96 |

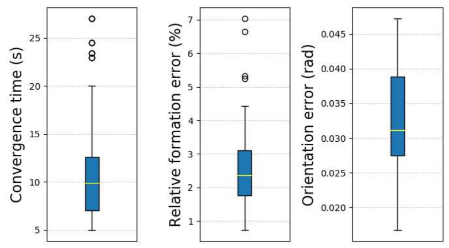


Fig. 9: Box plot of convergence time, relative formation error and orientation error after convergence over successful runs. The central mark in each box is the median, the edges of the boxes are the 25th and 75th percentiles, the whiskers extend to the maximum/minimum, and the circles represent outliers.

mainly the result of the quantization errors introduced when converting the LIDAR sensory data to the occupancy map. In this paper, the spatial resolution of the occupancy maps is $0.1$ m/pixel, i.e., any LIDAR scan data within $0.1$ m cannot be reflected by the image pixels. Reducing the quantization errors can be realized by increasing the size of the occupancy map images to increase the spatial resolution of the images. However, larger size of images requires more computational power to process the image input and compute the robot control commands in real time. The tradeoff between the image size and the spatial resolution of the occupancy map can be balanced according to practical requirements of applications.

## V. Conclusion and Future Work

In this paper, we proposed a novel approach for learning decentralized formation control of multiple robots. The control policy was modeled using a DNN that directly maps the robot's own observation to control actions. The model was trained with a centralized learning framework based on supervised learning. The trained model was deployed on each robot as a decentralized controller which only used the robot's local observation and no inter-robot communication was performed. Extensive simulated experiments were conducted in a robot simulator, and the results verified the effectiveness of the proposed learning approach for robot formation control.

As our first attempt solving a decentralized formation control problem using deep learning method, we only considered the three robot formation scenario with equal desired distance between any two robots. This is due to the reason that the robot has no additional sensors or communication equipment to distinguish its neighboring robots. Advanced deep learning methods such as semantic labeling [20] may be developed so that the robot can assign IDs to its neighboring robot for different desired relative distance requirement. We plan to study learning control policy from raw vision perception with semantic labels in the future. Also, extending the current work to general distributed learning of multi-robot systems is in the scope of our future research.

## References

[1] Y. Guo, *Distributed Cooperative Control: Emerging Applications*. John Wiley & Sons, 2017.

[2] Z. Qu, *Cooperative control of dynamical systems: applications to autonomous vehicles*. Springer Science & Business Media, 2009.

[3] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[5] C. Amato, G. Konidaris, A. Anders, G. Cruz, J. P. How, and L. P. Kaelbling, "Policy search for multi-robot coordination under uncertainty," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1760–1778, 2016.

[6] R. Vidal, O. Shakernia, and S. Sastry, "Following the flock [formation control]," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 14–20, 2004.

[7] H. Wang, D. Guo, X. Liang, W. Chen, G. Hu, and K. K. Leang, "Adaptive vision-based leader–follower formation control of mobile robots," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 4, pp. 2893–2902, 2017.

[8] X. Liang, H. Wang, Y.-H. Liu, W. Chen, and T. Liu, "Formation control of nonholonomic mobile robots without position and velocity measurements," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 434–446, 2018.

[9] T. Gustavi and X. Hu, "Observer-based leader-following formation control using onboard sensor information," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1457–1462, 2008.

[10] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1073–1087, 2017.

[11] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick, "Learning a deep neural net policy for end-to-end control of autonomous vehicles," in *American Control Conference*, pp. 4914–4919, 2017.

[12] A. Giusti, J. Guzzi, D. C. Ciresan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots.," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.

[13] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search," in *IEEE International Conference on Robotics and Automation*, pp. 528–535, 2016.

[14] P. Long, W. Liu, and J. Pan, "Deep-learned collision avoidance policy for distributed multiagent navigation," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 656–663, 2017.

[15] 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*, pp. 2137–2145, 2016.

[16] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[18] E. Rohmer, S. P. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1321–1326, 2013.

[19] D. V. Dimarogonas and K. H. Johansson, "On the stability of distance-based formation control," in *IEEE Conference on Decision and Control*, pp. 1200–1205, 2008.

[20] J. Dequaire, P. Ondrúška, D. Rao, D. Wang, and I. Posner, "Deep tracking in the wild: End-to-end tracking using recurrent neural networks," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 492–512, 2018.