Deep Learning Based Formation Control for the Multi-Agent Coordination

Qishuai Liu, Emmanuel Moulay, Patrick Coirault, Qing Hui

Deep Learning Based Formation Control for the Multi-Agent Coordination

1st Qishuai Liu
Department of Electrical and Computer Engineering
University of Nebraska-Lincoln
Lincoln, USA
qishuai.liu@huskers.unl.edu

2nd Emmanuel Moulay
XLIM (UMR CNRS 7252)
University of Poitiers
Poitiers, France
emmanuel.moulay@univ-poitiers.fr

3rd Patrick Coirault
LIAS (EA 6315)
University of Poitiers
Poitiers, France
patrick.coirault@univ-poitiers.fr

4th Qing Hui
Department of Electrical and Computer Engineering
University of Nebraska-Lincoln
Lincoln, USA
qing.hui@unl.edu

***Abstract*—Formation control of multi-agent systems has been an important task in the fields of automatic control and robotics. The aim of this paper is to develop a deep learning based formation control strategy for the multi-agent systems by using the backpropagation algorithm. Specifically, the deep learning network can be treated as the feedback controller, thus the multi-agent system can use the network output as its input to achieve the formation control. The algorithm has been tested on a multirobot system to verify the effectiveness of the proposed method.**

I. Introduction

The cooperative control of multi-agent systems (MASs) has drawn a great attention in recent years in different areas [1] including reconnaissance, surveillance, and security [2], [3]. In such applications, maintaining the network topology and connectivity of agents are important for some tasks including target localization, oceanic search, and undersea oil pipeline maintenance [4]–[6].

Formation control is one of the most interesting research topics among cooperative control of MASs [7]. In many applications, a group of autonomous robots are required to follow the specified trajectory and maintain the network topology at the same time because of the system cost, redundancy, and structure flexibility of the system [8]–[10]. In formation control, for a group of coordinated agents, different control topologies can be adopted according to different tasks. Some of them may need a leader for other agents so that they can follow the leader in a specified way. Those robots carry onboard sensing equipment to maintain the relative topologies between them. Generally speaking, the robots have the limited communication ability, thus they cannot use all the global information of the other robots. Given this situation, the design of the controller for each robot should not be centralized but be based on the local information.

In order to design the distributed controller for the multi-agent formation, we need to consider some issues such as the stability of the controller, controllability of different formation patterns, and safety and uncertainties in the formation control. The traditional control approaches to solve these issues are to develop the leader-follower strategy [11], virtual structure approach, and behavior-based method. The formation control is classified as regulation control and tracking control in some situations. Generally, this may need to consider the system dynamic and design the controller carefully. In some cases, this is hard to implementation. In order to overcome this drawback, some researchers develop artificial neural network based controllers [12]. Deep learning (DL), which is based on the nonlinear continuous functions [13], [14], is part of machine learning [15] and is dedicated to the modeling of nonlinear behaviors by a learning process involving deep neural networks (DNN) [16]. In the past several years, the deep learning method is applied in many aspects such as pattern recognition [17], speech recognition [18], and computer vision [19].

One of the machine learning based methods is the adaptive dynamic programming, which has been used as the model-based algorithm to design the controller for dynamical systems [20]. This method uses a cost function to find the feedback controller named policy by solving the Bellman equation of the value function [20]. DNN approximations of value function and policy are used in ADP to stabilize discrete-time systems. The policy can be obtained from the value function [21].

Another machine learning based method is reinforcement learning, which is used to develop the model-free algorithms of controllers for dynamical systems [22]. The main aim of this method is to learn from experiments on what to do in different situations so that it can maximize a reward function [23]. By using this method, the designed controller is a model-free strategy that can be obtained by solving the Bellman equation of the value function. This method can be used for both stochastic processes using MDP and deterministic processes. By incorporating the DNN into the reinforcement learning,

978-1-7281-0084- 5/19/$31.00

a new method named deep reinforcement learning (DRL) has been proposed for the control of dynamical systems [24], [25]. The value function and the policy are approximated by DNN in the DRL approach.

The main drawback of these two methods is that the Bellman equation is hard to solve. Thus, the neural network can be used as a controller to ensure the control of dynamical systems [26]. In our paper, we extend the neural network to use the DNN as the controller for discrete-time dynamical systems and use it for the formation control. This method, first developed in [27] for shallow neural networks and in [28] for DNN under a restrictive assumption on the system, is based on the ability of a DNN for self-tuning its weights by using a backpropagation algorithm. Thus, it can avoid the solving the Bellman equation in ADP or DRL. Also, this method can be used if the vector field of the discrete-time nonlinear system is differentiable. Thus if the discrete-time nonlinear system is unknown, it is still possible to use numerical differentiation methods for backpropagation.

This paper is organized as follows: Section II gives some preliminaries and Section III introduces the deep learning based controller design. In Section IV, we introduce how to transform the formation control problem into the feedback control so that we can use the proposed algorithm to solve it. We also have done the experiment to verify the proposed algorithm in Section V. Finally, we give a conclusion in Section VI.

II. Preliminary

The notation used in here is fairly standard. Specifically, $\mathbb{R}$ denotes the set of real numbers, $\mathbb{N}$ denotes the set of nonnegative integers, $\mathbb{R}^n$ denotes the set of n-dimensional real column vectors, $\mathbb{R}^{n\times m}$ denotes the set of n-by-m matrices, $(\cdot)^T$ denotes transpose, and $(\cdot)^{-1}$ denotes inverse, respectively. $\|\cdot\|$ denotes the Euclidean norm. $\mathrm{diag}(x)$ denotes a square diagonal matrix with the elements of vector x on the main diagonal. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a dynamic directed graph with the set of vertices $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ representing the set of edges. A graph with the property that $(v_i, v_j) \in \mathcal{E}$ implies $(v_j, v_i) \in \mathcal{E}$ is said to be undirected. The adjacent matrix $A \in \mathbb{R}^{N\times N}$ associated with the directed graph $\mathcal{G}$ is defined by nonnegative adjacency elements a_{ij} as $A(t) = [a_{ij}(t)]$. We assume $(v_i, v_j) \in \mathcal{E}(t)$ if and only if $a_{ij} = 1$, $(v_i, v_j) \notin \mathcal{E}(t)$ if and only if $a_{ij} = 0$, and $a_{ii} = 1$ for all $i \in \mathcal{N}$, where $\mathcal{N} = \{1, 2, \ldots, N\}$ denotes the node index of $\mathcal{G}(t)$. The set of neighbors of the node v_i is denoted by $\mathcal{N}^i(t) = \{v_j \in \mathcal{V} : (v_i, v_j) \in \mathcal{E}(t), j = 1, 2, \ldots, |\mathcal{N}|, j \neq i\}$, where $|\mathcal{N}|$ denotes the cardinality of $\mathcal{N}$. The degree matrix of a dynamic graph $\mathcal{G}(t)$ is defined by $\delta(t) = [d_{ij}(t)]$, where $i, j \in \{1, 2, \ldots, |\mathcal{N}|\}$ and

$$d_{ij}(t) = \begin{cases} \sum_{j=1}^{|\mathcal{N}|} a_{ij}(t), & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

Moreover, we can define Laplacian matrix of the dynamic graph $\mathcal{G}$ as $L(t) = \delta(t) - A(t)$. If the graph is called strongly connected if there exists a path from any node to any other node in the dynamic graph.

Next, let $\mathcal{K} \subset \mathbb{R}^n$ be a compact set such that $0 \in \mathcal{K}$ and called the learning state space. Consider the following system:

$$x(k+1) = f(x(k), u(k)),\ k \in \mathbb{N},\ x(0) = x_0 \in \mathbb{R}^n \quad (1)$$

where $x(k) = (x_1(k), \ldots, x_n(k)) \in \mathcal{K}$ is the state vector, $u(k) = (u_1(k), \ldots, u_m(k)) \in \mathbb{R}^m$ is the control input with $m \leq n$, and $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ a continuous function such that $f(0, 0) = 0$ and $f = (f_1, \ldots, f_n)$ with $f_i : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$. The objective is to ensure that the solutions of system (1) converge to the origin by using a DNN as a controller. To this end, we consider the extended system

$$x(k-s+1) = f(x(k-s), u(k-s)),\ k \in \mathbb{N},\ 0 \leq s \leq r \quad (2)$$

where $(x(k-r+1), \ldots, x(k+1)) \in \mathcal{K}^{r+1}$ is the state vector with $x(k-s+1) = (x_1(k-s+1), \ldots, x_n(k-s+1)) \in \mathbb{K}$, $(u(k-r), \ldots, u(k)) \in \mathbb{R}^{m(r+1)}$ the control input with $m \leq n$ and $u(k-s) = (u_1(k-s), \ldots, u_m(k-s)) \in \mathbb{R}^m$, and $r \geq 0$ such that the following assumption holds:

Assumption 1. Let

$$E(k) = \frac{1}{2}\|x(k+1)\|^2 \quad (3)$$

if

$$\lim_{k\to\infty} \left(\frac{\partial E(k)}{\partial u_1(k-r)}, \ldots, \frac{\partial E(k)}{\partial u_m(k-r)}, \ldots, \frac{\partial E(k)}{\partial u_1(k)}, \ldots, \frac{\partial E(k)}{\partial u_m(k)}\right) = 0$$

then we have $\lim_{k\to\infty} x(k) = 0$.

Next, let us recall the definition of a trajectory of the system (1) given in [29, p. 13].

Definition 1. Given an initial value $x_0 \in \mathcal{K}$ and a control sequence $u(k)$, we define a trajectory $x_u(k, x_0)$ of the system (1) by $x_u(0, x_0) = x_0$ and

$$x_u(k+1, x_0) = f(x_u(k, x_0), u(k)),\ k \in \mathbb{N}$$

The classical feedback control aims at finding $u(x(k))$ to stabilize the system (1), as shown by the Lyapunov based methods in [30], the model predictive control in [31] or the neural-network-based ADP algorithm in [32], following the framework illustrated in Figure 1.

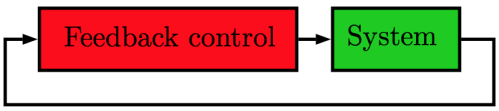

Fig. 1. Classical feedback control.

III. Method

In this section, we will introduce the backpropagation algorithms used in the DNN. The DL feedback control aims at finding a feedback control $U : \mathcal{K} \to \mathbb{R}^m$ as a DNN to ensure the convergence of the state of the system (1) toward zero by using the backpropagation algorithm and without using classical feedback control theory or the Bellman equation.

Let us denote $U(k-s) = (U_1(k-s), \dots, U_m(k-s)) \in \mathbb{R}^m$ the output of the DNN with the input $x(k-s) \in \mathcal{K}$ where $0 \leq s \leq r$. It means that the DNN as the $r+1$ inputs $x(k-s) \in \mathbb{R}^n$ and the $r+1$ outputs $U(k-s) \in \mathbb{R}^m$ where $0 \leq s \leq r$. We are facing problems for backpropagation due to the fact that we have no target for the output $U(k-s)$ of the DL feedback control and moreover $U(k-s) \in \mathbb{R}^m$ whereas $x(k+1-s) \in \mathbb{R}^n$ with $m \leq n$. The main goal is to make $x(k)$ converge to zero, so we consider the backpropagation error (3) for the input $x(k-s)$ of the DNN. The tuning of the weights $w_{ij}(k)$ between neurons i and j for the input $x(k-s)$ of the DNN is done by using the backpropagation algorithm

$$\begin{aligned} w_{ij}(k-s+1) &= w_{ij}(k-s) - \lambda \delta w_{ij}(k-s) \\ &= w_{ij}(k-s) - \lambda \frac{\partial E(k)}{\partial w_{ij}(k-s)} \end{aligned} \tag{4}$$

with $\lambda > 0$ being the learning rate and the chain rule:

$$\frac{\partial E(k)}{\partial w_{ij}(k-s)} = \frac{\partial E(k)}{\partial \sigma_j(k-s)} \frac{\partial \sigma_j(k-s)}{\partial z_j(k-s)} \frac{\partial z_j(k-s)}{\partial w_{ij}(k-s)}$$

where $k \in \mathbb{N}$, $0 \leq s \leq r$, and

$$\sigma_j(k-s) = \varphi_j(z_j(k-s)) = \varphi_j\left(\sum_l w_{lj}\sigma_l(k-s)\right)$$

in which $k \in \mathbb{N}$ and $0 \leq s \leq r$. The only difference with usual backpropagation is the first term $\frac{\partial E(k)}{\partial U_j(k-s)}$ due to the fact that the backpropagation error $E(k)$ does not depend directly on $U_j(k-s)$. Consider the following assumption for f:

Assumption 2. The function $(x,u) \to f(x,u)$ is differentiable on $\mathcal{K} \times \mathbb{R}^m$.

By using the system (2) and Assumption 2, we can obtain the following derivative for the backpropagation error (3) with respect to the output $U_j(k-s)$, $k \in \mathbb{N}$ and $0 \leq s \leq r$

$$\frac{\partial E(k)}{\partial U_j(k-s)} = \sum_{i=1}^{n} x_i(k+1) \frac{\partial f_i(x(k), U(k))}{\partial U_j(k-s)}. \tag{5}$$

In order for the backpropagation algorithm (3)–(5) to converge, we must add the following assumption:

Assumption 3. The function $(x,u) \to f_i(x,u)^2$ is convex on $\mathcal{K} \times \mathbb{R}^m$.

By using Assumption 3 and the fact that

$$E(k) = \frac{1}{2} \sum_{i=1}^{n} f_i(x(k), U(k))^2$$

we can deduce that $U_j(k-s) \mapsto E(k)$ is convex for all $1 \leq j \leq m$ and $0 \leq s \leq r$. So the backpropagation algorithm converges and

$$\lim_{k\to\infty} \left(\frac{\partial E(k)}{\partial U_1(k-r)}, \dots, \frac{\partial E(k)}{\partial U_m(k-r)}, \dots, \frac{\partial E(k)}{\partial U_1(k)}, \dots, \frac{\partial E(k)}{\partial U_m(k)}\right) = 0.$$

Finally, Assumption 1 implies that $\lim_{k\to\infty} x(k) = 0$. This can explain why the backpropagation algorithm is applied on the extended system (2) and not on the system (1). Thus, we have the following result:

Theorem 1. Under Assumptions 1–3, the solutions of the closed-loop system (1) with the DL feedback control given by the backpropagation algorithm (3)–(5) on the system (2) converge to the origin.

The DL feedback control approach is illustrated in Figure 2.

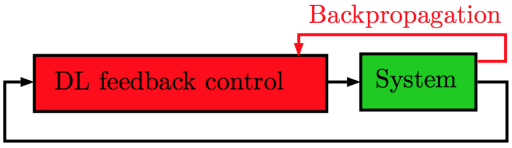

Fig. 2. DL feedback control.

Remark 1. If the derivatives $\frac{\partial f_i(x(k),U(k))}{\partial U_j(k-s)}$ are not available due to the fact that f is unknown, then we can use numerical differentiation methods for obtaining an approximate value of these derivatives [33]. Moreover, if $r = 0$, then Assumption 2 is replaced by the one that $u_j \to f_i(x,u)^2$ is convex for all $1 \leq i \leq n$ and $1 \leq j \leq m$.

If the system (1) is affine, i.e.,

$$x(k+1) = \phi(x(k)) + \sum_{j=1}^{m} g_j(x(k)) u_j(k) \tag{6}$$

with $\phi : \mathbb{R}^n \to \mathbb{R}^n$ being continuous such that $g_j(x) = (g_{1j}(x), \dots, g_{nj}(x))$, then we have

$$\frac{\partial E(k)}{\partial U_j(k)} = \sum_{i=1}^{n} x_i(k+1) g_{ij}(x(k)) \tag{7}$$

for all $1 \leq j \leq m$. Note that affine systems are very common in applications [34]. Finally, consider the very special case of linear systems of the form:

$$x(k+1) = Ax(k) + Bu$$

with $A \in \mathbb{R}^{n\times n}$ and $B = [b_{ij}] \in \mathbb{R}^{n\times m}$, then we have

$$\frac{\partial E(k)}{\partial U_j(k)} = \sum_{i=1}^{n} b_{ij} x_i(k+1)$$

for all $1 \leq j \leq m$.

These results ensure that the backpropagation algorithms converge for the system (1) and the fundamental approximation theorem ensures that a DNN can approximate any continuous function on a compact set $\mathcal{K}$. Now we can present how to implement the DL control algorithm.

Algorithm 1 explains how the backpropagation algorithm presented in Section III is used for building a DNN called DL feedback control. It stops when the draw of new random states in $\mathcal{K}$ does not make evolve the weights of the DNN. The learning process on the system (2) allows to find a DL feedback control for the system (1). If Algorithm 1 uses only the states $\{x(k) \in \mathcal{K} : k \in \mathbb{N}\}$ of a solution of the system (1) without using random states, it will learn badly with too few states in $\mathcal{K}$ and therefore too little information. It is possible

Algorithm 1: DL feedback control algorithm on $\mathcal{K}$

Input: $x(k-r) \in \mathcal{K}$ and $\epsilon > 0$

Output: DNN U on $\mathcal{K}$ for the system (1)

1 Apply the backpropagation algorithm (3)–(5) to the system (2) with the input $x(k-s)$ of the DNN where $0 \leq s \leq r$.
2 **if** $\|\delta w(k-s)\|^2 = \sum_{i,j}(k-s)^2 \geq \epsilon$ **then**
3 $w_{ij}(k-s) \leftarrow w_{ij}(k-s+1)$
4 **if** $E(k) < E(k-1)$ **then**
5 $x(k-s) \leftarrow x(k-s+1)$ for all $0 \leq s \leq r$
6 **if** $x(k-s+1) \in \mathcal{K}$ *for all* $0 \leq s \leq r$ **then**
7 go to 1
8 **else**
9 Choose randomly $x \in \mathcal{K}$
10 $x(k-r) \leftarrow x$
11 go to 1
12 **end**
13 **else**
14 go to 1
15 **end**
16 **else**
17 Choose randomly $x \in \mathcal{K}$
18 $x(k-r) \leftarrow x$
19 go to 1
20 **end**

to improve the convergence speed of the DL feedback control by replacing Step 4 of Algorithm 1 with $E(k) < \lambda E(k-1)$ where $0 < \lambda < 1$.

Remark 2. We use all the states $(x(k), \ldots, x(k-r))$ in the input of the DNN for tuning its weights contrary to what is done in [27], [28] where only both states $(x(k), x(k-r))$ are used under the restrictive assumption that the system (2) is uniquely invertible. Moreover an algorithm, namely Algorithm 1, is provided explaining how tuning the weights of the DNN which is not the case in [27], [28].

IV. Formation Control of Multi-Agent Systems

In this part, we will present the formation control problem of multi-agent systems and show how to use Algorithm 1 to solve it. The formation control problem we consider here is based on the consensus protocol, which means each agent will have a copy of its own understanding consensus state x_{ci}. However, the real state of each agent will differ from its understanding consensus state that the formation topology can be maintained. Here we consider the real state of the agent as $x_i(t)$ and each agent has its own understanding consensus state x_{ci}. The relationship between x_{ci} and x_i can be written as:

$$x_i(t) = x_{ci}(t) + p_i \tag{8}$$

where p_i is the vector denoting the real state of each agent $x_i(t)$ deviating its understanding consensus state x_{ci}. Moreover, the dynamic of the understanding consensus state can be written as follows:

$$x_{ci}(t+1) = x_{ci}(t) + u_i(t) \tag{9}$$

for each agent.

The goal of the consensus protocol is to reach

$$\lim_{t\to\infty} x_{c1}(t) = \cdots = \lim_{t\to\infty} x_{cN}(t) \tag{10}$$

for all N agents.

Next, we need to define the error dynamic for each agent i as:

$$\begin{aligned} e_i(t) &= x_{ci}(t) - 1/N \sum_{j=1}^{N} x_{cj}(t) \\ &= 1/N\Big(\sum_{j=1}^{N}(x_{ci}(t) - x_{cj}(t))\Big) \end{aligned} \tag{11}$$

Then, we have

$$\begin{aligned} e_i(t+1) &= x_{ci}(t+1) - 1/N \sum_{j=1}^{N} x_{cj}(t+1) \\ &= 1/N\Big(\sum_{j=1}^{N}(x_{ci}(t) - x_{cj}(t))\Big) \\ &= 1/N \sum_{j=1}^{N}\Big(x_{ci}(t) - x_{cj}(t)\Big) \\ &\quad + 1/N \sum_{j=1}^{N}\Big(u_i(t) - u_j(t)\Big) \end{aligned} \tag{12}$$

which means:

$$e_i(t+1) = e_i(t) + 1/N \sum_{j=1}^{N}\big(u_i(t) - u_j(t)\big) \tag{13}$$

Equation (13) can be written as:

$$e(t+1) = e(t) + 1/N(I_N - \mathbf{1}_N\mathbf{1}_N^T)u(t) \tag{14}$$

where $e(t) = (e_1(t), \ldots, e_N(t))^T$, $u(t) = (u_1(t), \ldots, u_N(t))^T$, $\mathbf{1}_N$ is a column vector witch has N elements with each element equals 1. The system error (14) can be seen as the specific form of the dynamical system (2) with the function $f = e(t) + 1/N(I_N - \mathbf{1}_N\mathbf{1}_N^T)u(t)$. Thus, our goal has become to design a DNN controller based on Algorithm 1 to force the error state $e(t)$ to converge to 0.

V. Experiment

In this part, we will implement an experiment to show the effectiveness of the Algorithm 1 on our multirobot platform. We first use the Gazebo simulation environment to train the DNN controller to obtain the network model. After that we could use the trained network in the real robot to show the effectiveness of Algorithm 1. The multirobot system used here is 4 Pioneer 3dx robots. Each robot is with a Nvidia TX1 so that they can communicate with each other by the Robot Operating System (ROS). The DNN we used here has 6 hidden

layers and each layer is composed of 100 neurons. We choose the ReLU function as the activation function. After training the network, the robot system can achieve the formation control by walking along the path generated by the algorithm.

Let $(r_{xi}, r_{yi}), \theta_i$ denote the Cartesian position and orientation of the ith robot, respectively. Thus, we can denote the state of the robot $x_i = [r_{xi}, r_{yi}, \theta_i]^T$. The linear and angular speed of the robot can be denoted by (v_i, w_i). Thus, we have the kinematic equation of the ith robot as follows:

$$\dot{r}_{xi} = v_i \cos(\theta_i),\ \dot{r}_{yi} = v_i \sin(\theta_i),\ \dot{\theta}_i = w_i \qquad (15)$$

Then, by linearizing (15) around a fixed point off the center of the wheel axis $(\bar{x}_i, \bar{y}_i)$ of the robot, where $\bar{x}_i = r_{xi} + d_i \cos(\theta_i)$, $\bar{y}_i = r_{yi} + d_i \sin(\theta_i)$, and $d = 0.15$m, we have the equation

$$\begin{bmatrix} v_i \\ w_i \end{bmatrix} = \begin{bmatrix} \cos(\theta_i) & \sin(\theta_i) \\ -(1/d)\sin(\theta_i) & (1/d)\cos(\theta_i) \end{bmatrix} \begin{bmatrix} a_{xi} \\ a_{yi} \end{bmatrix}.$$

This equation is a simple kinematic one that can transfer the action of each robot to the velocity command.

Next, we can define the understanding consensus state for each robot as $x_{ci} = [x^r_{ci}, y^r_{ci}, \theta^r_{ci}]^T$. The $[x^r_{ci}, y^r_{ci}]$ and θ^r_{ci} denote the reference position and orientation of the understanding formation center of the robots team for each robot i, respectively. Since the formation center of the robots team is changing dynamically, each robot would maintain a local variable x_{ci}, which is the sensing value of the state s^r by each robot i.

Moreover, there is a relationship (8) that we should clarify between the real state x_i and the understanding consensus state x_{ci}:

$$\begin{bmatrix} \bar{x}_i \\ \bar{y}_i \end{bmatrix} = \begin{bmatrix} x^r_{ci} \\ y^r_{ci} \end{bmatrix} + \begin{bmatrix} \cos(\theta^r_{ci}) & -\sin(\theta^r_{ci}) \\ \sin(\theta^r_{ci}) & \cos(\theta^r_{ci}) \end{bmatrix} \begin{bmatrix} \tilde{x}_{if} \\ \tilde{y}_{if} \end{bmatrix}, \qquad (16)$$

where $[\tilde{x}_{if}, \tilde{y}_{if}]^T$ represents the desired deviation vector of the ith robot relative to the geometric center of the formation. The initial positions for the robots are $x1 = 0.935$m, $y1 = 0.935$m, $x2 = 2.205$m, $y2 = 0$m, $x3 = 1.46$m, $y3 = 1.46$m, and $x4 = -0.81$m, $y4 = 2.32$m. Also, we add a virtual leader for the robots team and its initial position is $x0 = 0$m and $y0 = 0$m. Its velocity is $v_{0x} = 0.01$m/s and $v_{0y} = 0.01$m/s. Also, $\tilde{x}_{1f} = 1.5$m, $\tilde{y}_{1f} = 1.5$m, $\tilde{x}_{2f} = -1.5$m, $\tilde{y}_{2f} = 1.5$m, $\tilde{x}_{3f} = 1.5$m, $\tilde{y}_{3f} = -1.5$m, and $\tilde{x}_{4f} = -1.5$m, $\tilde{y}_{4f} = -1.5$m. Figures 3 and 4 show the trajectory of each robot moving in x and y direction, respectively. Moreover, Figures 5 and 6 show the velocity of each robot in x and y direction, respectively.

Next, we will perform the experiment on the real robot by using the trained DNN before. Figure 7 shows the robots are in the initial position, Figure 8 shows that the robots are running, and Figure 9 shows that they have achieved the formation finally.

VI. Conclusion

In this paper, we consider the formation control problem by using a deep learning based feedback control method. The backpropagation algorithm is used to train the DNN to obtain the controller. In order to implement this method, we transfer the formation control problem into a general problem (2) that can be solved by Algorithm 1. Finally, we verified the effectiveness of the algorithm on our multirobot platform.

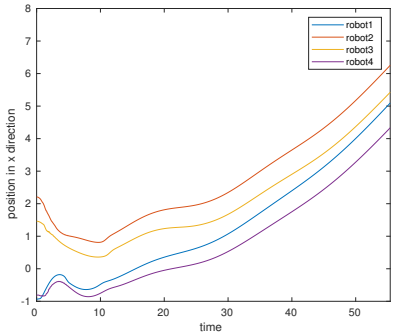

Fig. 3. The x direction trajectory of each robot.

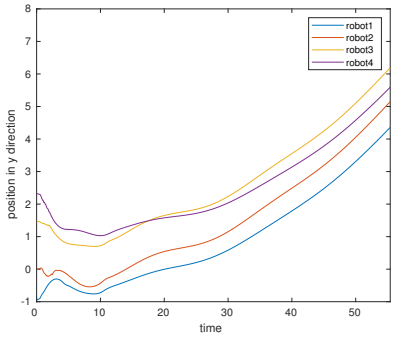

Fig. 4. The y direction trajectory of each robot.

References

[1] X. Ge, A. Han, D. Ding, X. Zhang, and B. Ning. A survey on recent advances in distributed sampled-data cooperative control of multi-agent systems. *Neurocomputing*, 275:1684–1701, 2018.

[2] M. Fields, E. Haas, S. Hill, C. Stachowiak, and L. Barnes. Effective robot team control methodologies for battlefield applications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5862–5867, 2009.

[3] J. Ding, J. Gillula, H. Huang, M. Vitus, W. Zhang, and C. Tomlin. Hybrid systems in robotics. *IEEE Robotics & Automation Magazine*, 18(3):33–43, 2011.

[4] H. Rezaee and F. Abdollahi. Pursuit formation of double-integrator dynamics using consensus control approach. *IEEE Transactions on Industrial Electronics*, 62(7):4249–4256, 2015.

[5] Z. Lin, L. Wang, Z. Han, and M. Fu. Distributed formation control of multi-agent systems using complex laplacian. *IEEE Transactions on Automatic Control*, 59(7):1765–1777, 2014.

[6] H. Li, P. Xie, and W. Yan. Receding horizon formation tracking control of constrained underactuated autonomous underwater vehicles. *IEEE Transactions on Industrial Electronics*, 64(6):5004–5013, 2017.

[7] Y.-Q. Chen and Z. Wang. Formation control: a review and a new consideration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3181–3186, 2005.

[8] D. Stilwell and B. Bishop. Platoons of underwater vehicles. *IEEE Control Systems Magazine*, 20(6):45–52, 2000.

[9] D. Scharf, F. Hadaegh, and S. Ploen. A survey of spacecraft formation flying guidance and control (part ii): Control. In *IEEE American Control Conference*, Boston, USA, 2004.

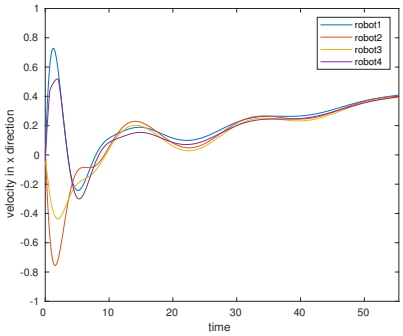

Fig. 5. The velocity of x direction trajectory of each robot.

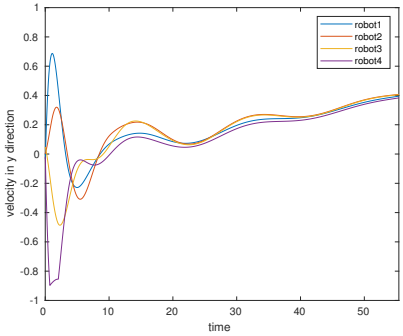

Fig. 6. The velocity of y direction trajectory of each robot.

Fig. 7. The initial position for all robots.

Fig. 8. The robots are running.

Fig. 9. The final position for all robots.

[10] A. Robertson, T. Corazzini, and J. P. How. Formation sensing and control technologies for a separated spacecraft interferometer. In *IEEE American Control Conference*, pages 1574–1579, Philadelphia, USA, 1998.

[11] N. Cowan, O. Shakerina, R. Vidal, and S. Sastry. Vision-based follow-the-leader. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1796–1801, 2003.

[12] Z. Cai. *Intelligent Control: Principles, Techniques and Applications*, volume 7. World Scientific, 1997.

[13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[14] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[15] S. Ao, B. Rieger B, and M. Amouzegar. *Machine Learning and Systems Engineering*, volume 68. Springer Science & Business Media, 2010.

[16] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[18] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[19] J. Janai, F. Güney, A. Behl, and A. Geiger. Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *arXiv preprint arXiv:1704.05519*, 2017.

[20] D. Liu, Q. Wei, D. Wang, X. Yang, and H. Li. *Adaptive Dynamic Programming with Applications in Optimal Control*. Springer, 2017.

[21] H. Jiang, H. Zhang, K. Zhang, and X. Cui. Data-driven adaptive dynamic programming schemes for non-zero-sum games of unknown discrete-time nonlinear systems. *Neurocomputing*, 275:649–658, 2018.

[22] F. Fourati, M. Chtourou, and M. Kamoun. Stabilization of unknown nonlinear systems using neural networks. *Applied Soft Computing*, 8:1121–1130, 2008.

[23] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[24] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[25] R. Hafner and M. Riedmiller. Reinforcement learning in feedback control. *Machine Learning*, 84(1-2):137–169, 2011.

[26] Demetri Psaltis, Athanasios Sideris, and Alan A Yamamura. A multilayered neural network controller. *IEEE Control Systems Magazine*, 8(2):17–21, 1988.

[27] W. Li and J-J. Slotine. Neural network control of unknown nonlinear systems. In *IEEE American Control Conference*, pages 1136–1141, 1989.

[28] F. Fourati, M. Chtourou, and M. Kamoun. Stabilization of unknown nonlinear systems using neural networks. *Applied Soft Computing*, 8(2):1121–1130, 2008.

[29] L. Grüne and J. Pannek. *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer, 2011.

[30] W. Lin. Feedback stabilization of general nonlinear control systems: A passive system approach. *Systems & Control Letters*, 25(1):41–52, 1995.

[31] F. Allgöwer and A. Zheng. *Nonlinear Model Predictive Control*. Birkhäuser, 2012.

[32] J. Sarangapani. *Neural Network Control of Nonlinear Discrete-Time Systems*. CRC Press, 2006.

[33] J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2016.

[34] Y. Hasegawa. *Control Problems of Discrete-Time Dynamical Systems*, volume 19 of *Studies in Systems, Decision and Control*. Springer, 2015.