UNIVERSITY OF OTTAWA

OTTAWA-CARLETON INSTITUTE FOR MECHANICAL AND AEROSPACE ENGINEERING

Online Model-Free Distributed Reinforcement Learning Approach for Networked Systems of Self-organizing Agents

Author:
Yiqing Chen

Supervisor:
Davide Spinello
Co-supervisor:
Mohammed Abouheaf

A thesis submitted in partial fulfillment of the requirements for the Master of Applied Science degree
in
Mechanical Engineering

uOttawa

Online Model-Free Distributed Reinforcement Learning Approach for Networked Systems of Self-organizing Agents

Yiqing Chen

Abstract

Control of large groups of robotic agents is driven by applications including military, aeronautics and astronautics, transportation network, and environmental monitoring. Cooperative control of networked multi-agent systems aims at driving the behavior of the group via feedback control inputs that encode the groups' dynamics based on information sharing, with inter-agent communications that can be time varying and be spatially non-uniform. Notably, local interaction rules can induce coordinated behaviour, provided suitable network topologies.

Distributed learning paradigms are often necessary for this class of systems to be able to operate autonomously and robustly, without the need of external units providing centralized information. Compared with model-based protocols that can be computationally prohibitive due to their mathematical complexity and requirements in terms of feedback information, we present an online model-free algorithm for some nonlinear tracking problems with unknown system dynamics. This method prescribes the actuation forces of agents to follow the time-varying trajectory of a moving target. The tracking problem is addressed by an online value iteration process which requires measurements collected along the trajectories. A set of simulations are conducted to illustrate that the presented algorithm is well-functioning in various reference-tracking scenarios.

Acknowledgments

Fisrt of all, I have to give thanks to my thesis supervisor Dr. Davide Spinello and my co-supervisor Dr. Mohammed Abouheaf for their patient guidance, suggestions and support throughout my graduate studies. I am thankful for their patience, extensive knowledge and constructive comments on my research. Their expertise was extremely valuable in creating the research topic and methodology especially. Additionally, they gave me enough freedom to conduct my research with diligence and a careful attention to detail has helped me become a better independent researcher. Without the precious support and feedback of my supervisors, it would not be possible to conduct this research. I am quite proud to be one of their students at University of Ottawa.

Meanwhile, I want to express my gratefulness to my colleagues and friends for their friendship, care and support. Special thanks go to my colleagues, Sarmad Tanveer, Payam Parvizi, Nathaniel Mailhot, and Ruikun Zhou who help me with some research problems. In addition, although COVID-19 pandemic has slowed down my research work, my friends Yixuan Wei, Qichuan Shi, Shang Gao, Mengzhen Yuan and Churan Liang still encouraged me to spend such difficult time. In the end, I want to thank all my family members, my dad Deping Chen, my mom Nana Song, and my sister Yanni Chen for their strong support, love and caring, who motivate me to continually pursue my goal.

Abbreviations

- **ANNs** Artificial Neural Networks
- **MDP** Markov Decision Process
- **MDPs** Markov Decision Processes
- **MAS** Multi-agent System
- **MASs** Multi-agent systems
- **PI** Policy Iteration
- **PID** Proportional-Integral-Derivative
- **PD** Proportional-Derivative
- **RL** Reinforcement Learning
- **TD** Temporal Difference
- **VI** Value Iteration
- **WSN** Wireless Sensor Network
- **WSNs** Wireless Sensor Networks

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Flocking is the collective movement of numerous self-propelled agents exhibited by a lot of animals such as fish, bees and fungi [1] [2]. In addition, flocking is regarded as an emergent behavior, induced by simple rules followed by all agents, which do not need any centralized information. Ref. [2] presents three fundamental principles for featuring the flocking behaviors, alignment, cohesion and separation, specifically.

Autonomous tracking is a class of control problems that requires the system to follow a desired reference or behavior without explicit human guidance [3]. Ideally, the tracking process aims at driving the tracking errors to zero, while the system performance is improved. Cooperative control algorithms for multiple agents in general require the full knowledge of the system's dynamics or model, and often they require the knowledge of the environment [1]. This may be prohibitive or impossible, without the prior knowledge of the system dynamics in the model-based implementation, and therefore a model-free approach may be desirable if a model-based approach is not possible. As a model-free approach, machine learning uses neural networks to estimate unknown quantities necessary to implement control algorithms.

Reinforcement learning (RL) is one of the most popular machine learning techniques, where an agent is able to learn to make decisions in a completely unknown environment to accomplish its goal [4]. There is no explicit supervisor to guide the agent's learning process. The system can estimate the state of the navigation environment and take the appropriate actions depending on a reward returned by the navigation environment. Artificial neural networks (ANNs) can be a class of machine learning tools, which, in the context of autonomous navigation, use a set of input-output tuples to train themselves to learn tracking behaviors [5]. Hence,

neural networks are applied to approximate optimal actions and then solve control problems with unknown dynamics.

1.1 Motivation

Autonomous tracking is quite challenging for networked multi-agent systems, since many problem-solving algorithms are either offline or rely on an accurate model [6] [7]. When it comes to designing controllers for underactuated high order systems, the complexity of the technique escalates [8]. Hence, more and more researchers are developing online data-driven techniques for such complex systems [7]. Unlike the tracking problems for a single agent, a large number of agents lead to high dimensional problems. For networked multi-agent systems, it can be infeasible to implement the model-based control that aims at driving the behavior of the group via feedback control inputs that encode the group's dynamics based on information sharing and local interation rules on the topology of the communication graph. Moreover, with the growing complexity of real-world multi-agent systems, integrating multi-function autonomous vehicles, mobile sensor networks, and other devices [7] [8], it is more difficult to formulate accurate models for such systems.

To overcome the above problems typically characterizing such complex models, the development of machine learning techniques that are less dependent on the prior knowledge of the plant dynamics has become necessary in several contexts [8]. Machine learning techniques, such as reinforcement learning, have been increasingly applied in the area of control systems during the last decades. Various machine learning techniques can be scalable to autonomous motion and intelligent decision making for some demanding or complicated control problems.

1.2 Thesis Objectives and Contributions

The main goal of the thesis is the formulation of a motion control algorithm for a system of mobile agents with multi-objective tasks such as tracking and formation control while navigating a specific environment, with the aim of using a model-free approach to obtain adaptive tracking gains that rely only on data locally collected along the trajectories. This is a first step towards a fully model-free formulation that would include the estimation of the environment and of other quantities nec-

essary to implement coordination algorithms based on local interactions.

Consistently, the main contribution is the formulation and implementation of a motion control and coordination algorithm for multi-agent systems (MASs) with a model-free component, using a reinforcement learning framework for online, model-free adaptive gains determination, in this case regarding trajectory tracking. The extension of this work would include the model-free formulation of additional optimization tasks, including the navigation environment and the local interaction rules involved in coordination of certain classes of MASs. The approximators are based on actor-critic neural network iterators that estimate optimal control policies and evaluate their performance in the sense of Bellman's optimality.

1.3 Thesis Outline

The remaining of the thesis can be structured below:

- **Chapter 2** presents the theoretical background on multi-agent systems, and provides the literature review of optimal control and machine learning methods.
- **Chapter 3** formulates the optimal tracking problems together with the model-free temporal difference solutions, and introduces our main contribution, where a trajectory tracking control in a model-free fashion is designed to drive agents, based on reinforcement learning. The rest of the chapter presents the setup and numerical simulation results of several scenarios.
- **Chapter 4** summarizes the conclusions and suggests a few future feasible study directions.

Chapter 2

Background and Literature Review

Section 2.1 presents a literature review of sensor networks, where some challenges and issues are summarily discussed. Based on graph theory, section 2.2 provides a basic theoretical background on sensor networks. Flocking control problems are discussed in section 2.3. Reference-tracking regulation and nonlinear optimization problems are formulated in section 2.4. Section 2.5 summarily presents a literature review about some categories of machine learning algorithms and their applications. Reinforcement learning control methods for nonlinear systems used throughout this thesis are illustrated in section 2.6.

2.1 Background of Sensor Networks

A wireless sensor network (WSN) comprises numerous sensing agents, where each agent (or sensor node) can function measuring and calibration, and sensing agents generate a wireless communication network (example in Figure 2.1) [9] [10]. Practical applications of WSNs involve, but are not limited to, comfort control in buildings [11], environment monitoring [12] [13], traffic control [14] [15], automated manufacturing [16], and surveillance systems [17]. When dealing with the dynamic environment, stationary WSNs may be often deficient and a mobile sensing technique is popular because of outstanding performance with respect to its high-resolution detection capability and adaptability [18] [19].

Within a sensor network, mobility can boost its sensing coverage both in time and space and robustness against the dynamically changing environment [18]. Each mobile sensor is resource-constrained among WSNs; it motions with a confined communication range, limited computational power and low memory. These

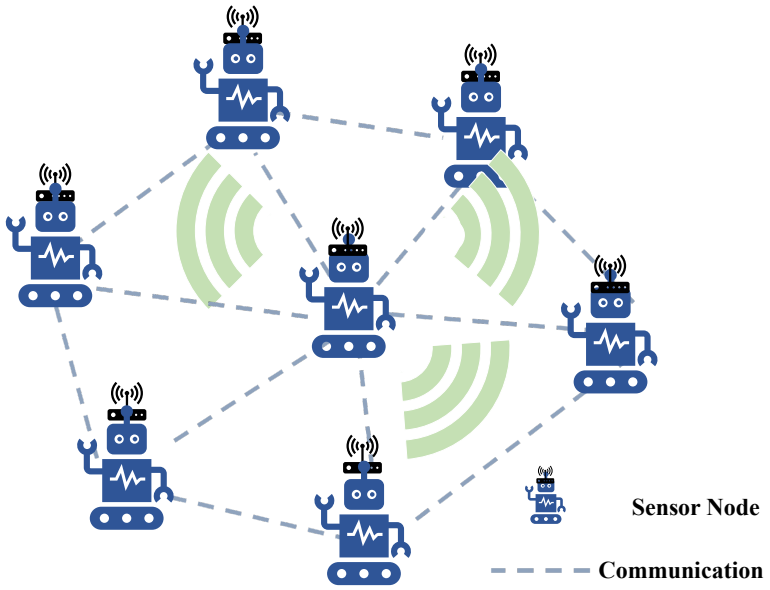

Figure 2.1: An Example of a Wireless Sensor Network [19]

mobile sensing agents generate a WSN in order to communicate with their neighbors. Each sensor has fixed abilities, but as a group, they are able to operate multiple missions at a level compatible to several high end mobile sensor. To achive multiple assignments for example, surveillance [20], exploration [21] and environmental monitoring [12] [13], distributed control of mobile sensors has attracted considerable attention.

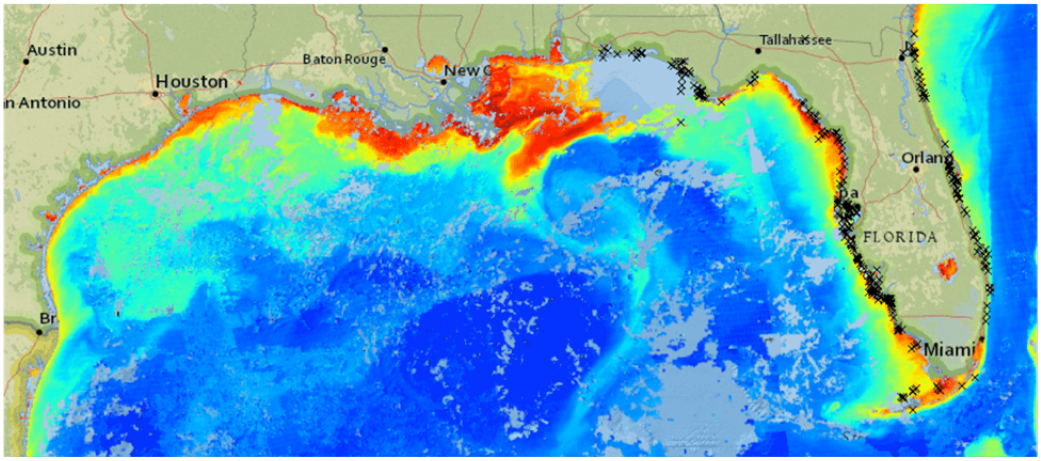

Figure 2.2: Estimated Environment of Chlorophyll Measured by Observation System in Florida, USA [22]

Environmental monitoring has recently attracted a lot of attention of control researchers and environmental experts [23], for demanding problems in distributed

control of mobile sensors. It is owing to many practical applications, as for example tracking pollutants in a distributed field. One application can be tracing harmful algal blooms in a pond [23]. In some cases, snappily propagating noxious algal blooms in ponds or in seas may result in a lot of cyanotoxins. Exposure to water poisoned with algal cyanotoxins brings on chronic health effects or serious acute respirtory syndrome to human beings and side effects to marine life. The content of chlorophyll can be regarded as a measure nearly affiliated to cyanobacterial and algal biomass in Figure 2.2. The other practical application is to detect hazardous chemicals [12] [24]. Multiple mobile sensing agents are deployed in a particular area. The chemicals odor plume or fire is traced by robots. Robots then monitor and predict the odor source location, via the information of the movement of odor molecules. It is effective and applicable to lower the risk to human or animal lives.

In [2], Olfati-Saber proposed embracive analyses of flocking algorithms. The flocking algorithms were initially developed as the simulations of the motions of flocking agent in computer graphics, in which each intelligent agent follows multiple rather simple local rules (illustrated in Ref. [25]). An agent in a flock coordinates with the motions of its neighbors and tries to gather whilst avoiding collisions. Generally, the swarming behaviors are regarded as the outcomes of natural optimization in Ref. [21]. A flocking algorithm can be applied to move mobile sensor networks [26] [27].

2.2 Graph Representation of Sensor Networks

We will illustrate a graph representation for sensor networks in this section. The flock-like behaviors are modeled based on algebraic graph theory.

2.2.1 Basics of Graph Theory

A graph $\mathcal{G}$, represented in Figure 2.4 is a pair $(\mathcal{V}, \mathcal{E})$ that is comprised of a number of vertices $\mathcal{V} = \{1, 2, ..., N\}$ and edges $\mathcal{E} \subseteq \{(i, j) : j \in \mathcal{V}, \quad j \neq i\}$ (without any self-loops). The graph $\mathcal{G}$ is undirected when $(i, j) \in \mathcal{E} \iff (j, i) \in \mathcal{E}$, where all edges are bidirectional. A graph (defined in Ref. [28]) is connected if and only if for each pair of vertices i and j in graph $\mathcal{G}$, and there should be a path where i and j are end vertices.

The adjacency matrix $\boldsymbol{A} = [a_{ij}]$ of a graph is a matrix without zero elements

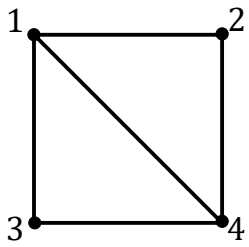

Figure 2.3: A Simple Graph

that satisfy the inequation $[a_{ij}] \neq 0 \iff (i,j) \in \mathcal{E}$. Typically, a_{ij} can be defined as

$$a_{ij} = \begin{cases} 1, \text{ if } ij \text{ is an edge} \\ 0, \text{ otherwise.} \end{cases} \tag{2.1}$$

For example, following is the adjacency matrix for the undirected graph in Figure 2.3:

$$\boldsymbol{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array} \tag{2.2}$$

where the labels $1, 2, 3, 4$ are for vertices in the graph.

A graph can be called weighted if and only if the elements of its adjacency matrix are not only $0-1$ elements [28]. Hence, weighted graphs, associated to position-dependent adjacency elements, are mainly used in our work [2]. As for an undirected graph $\mathcal{G}$, the adjacency matrix $\boldsymbol{A}$ is symmetric (i.e. $\boldsymbol{A}^T = \boldsymbol{A}$). The neighbors of node i can be given as

$$\mathcal{N}_i = \{j \in \mathcal{V} : [a_{ij}] \neq 0\} = \{j \in \mathcal{V} : (i,j) \in \mathcal{E}\} \tag{2.3}$$

For instance, the simple wireless sensor network in Figure 2.1 can be represented by the graph of Figure 2.4.

2.2.2 Dynamic Graph

Let $\boldsymbol{q}_i \in \mathbb{R}^m$ indicate the position of node i for all $i \in \mathcal{V}$. We denote all nodes in a graph as

$$\boldsymbol{q} = [\boldsymbol{q}_1^T, \boldsymbol{q}_2^T, \boldsymbol{q}_3^T, ..., \boldsymbol{q}_N^T]^T \tag{2.4}$$

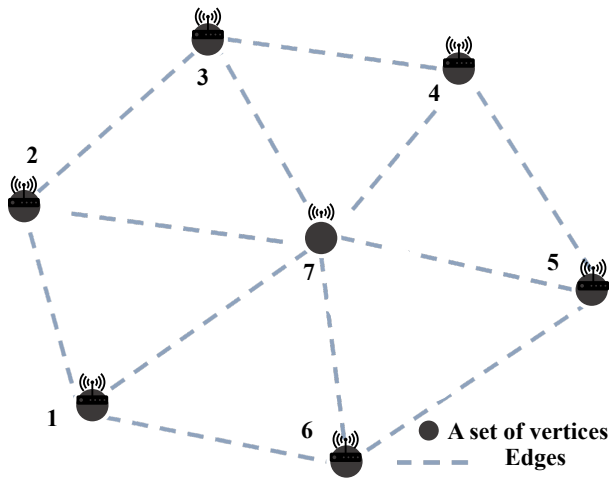

Figure 2.4: Graph Representation

where $\boldsymbol{q}$ is the global configuration and all dynamic nodes in a group are with differential equations

$$\begin{aligned} \dot{\boldsymbol{q}}_i &= \boldsymbol{v}_i \\ \dot{\boldsymbol{v}}_i &= \boldsymbol{u}_i \end{aligned} \tag{2.5}$$

where $\boldsymbol{q}_i, \boldsymbol{v}_i, \boldsymbol{u}_i \in \mathbb{R}^m, \forall i \in \mathcal{V}$. $\boldsymbol{q}_i$ is the position of agent i, $\boldsymbol{v}_i$ is the velocity, and $\boldsymbol{u}_i$ is the control input signal. Note that the pair $(\ \boldsymbol{q}_i(\ ), \boldsymbol{v}_i(\ )\ )$ is the state trajectory of all dynamic nodes in a group. The centroid of the group can be given by

$$\boldsymbol{q}_c = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{q}_i, \quad \boldsymbol{v}_c = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{v}_i \tag{2.6}$$

with average poition $\boldsymbol{q}_c$, and average velocity $\boldsymbol{v}_c$.

Therefore, we can consider each vertex in the graph as a dynamic particle, because of its simple mathematical representation of mobility. This is one of the most simple forms of the so-called dynamic graph [2] [28].

2.2.3 Communication Graph

Suppose that $r > 0$ denotes the interaction or communication range between two sensing agents. An open ball with radius r, indicates the set of the neighbors of agent i which can be determined as

$$\mathcal{N}_i = \{j \in \mathcal{V} : \|\boldsymbol{q}_j - \boldsymbol{q}_i\| < r\} \tag{2.7}$$

where $\| \cdot \|$ is the Euclidean norm in $\mathbb{R}^m$. Given a communication range $r > 0$, the undirected communication graph $\mathcal{G}(\boldsymbol{q}) = (\mathcal{V}, \mathcal{E}(\boldsymbol{q}))$ [29] is defined by the set of edges $\mathcal{E}(\boldsymbol{q})$ and $\mathcal{V}$.

The mathematic topology of a wireless sensor network (WSN) with a limited communication range is an undirected communication graph [29]. Each sensing node can be defined as a vertex, and an edge denotes a transmission communication between a pair of mobile sensors. The neighborhood set of sensing node i can be represented in (2.7).

Therefore, WSNs can be treated via graph-theoretic representation. As illustrated in Figure 2.5, i^{th} agent gathers the transmitted measurements from the other four neighbors in a limited communication range r. In this thesis, we assume the agents in the sensor network to be homogeneous [30]. That is, each agent has the same limited communication range.

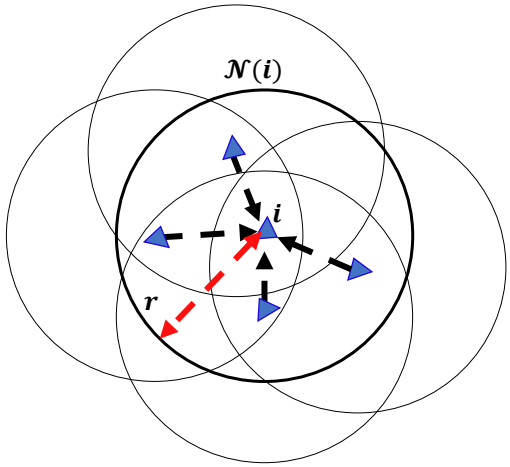

Figure 2.5: Model of a Wireless Sensor Network [21]

2.3 Flocking Control

In order to operate a certain task, mobile nodes (or agents in a group) can be driven by a flocking algorithm [2]. Additionally, the biologically-inspired flocking control [2] [31] supports to make and keep the communication graph connected. The control is combined by three behaviors [2]: cohesion, separation and alignment.

2.3.1 Preliminaries

σ-norm

The $\| \cdot \|_\sigma$ or σ-norm [2] is a mapping $\mathbb{R}^m \mapsto \mathbb{R}_{\geq 0}$, and unlike the classical norms $\| \cdot \|$, it is differentiable everywhere

$$\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_\sigma = \frac{1}{\epsilon}[\sqrt{1 + \epsilon\|\boldsymbol{q}_j - \boldsymbol{q}_i\|^2} - 1] \tag{2.8}$$

where ϵ is a positive real number.

Bump Function

As a scalar function $\rho_h(\cdot)$, a bump function smoothly changes from 0 to 1. Herein, bump functions are used for flocking algorithms [2] with smooth adjacency matrices. Here, we used the following bump function presented in Ref. [2]:

$$\rho_h(z) = \begin{cases} 1, & z \in [0, h) \\ \frac{1}{2}\left[1 + \cos\left(\pi \frac{(z-h)}{(1-h)}\right)\right], & z \in [h, 1] \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

where $h \in (0, 1)$. Obviously, $\rho_h(z)$ is a C^1-smooth function with the property that $|\rho_h'(z)|$ can be uniformly bounded in z and $\rho_h'(z) = 0$ throughout the interval $[1, \infty)$ [2]. We will construct a smooth adjacency matrix via the definition of this bump function.

2.3.2 Separation Behavior

We use a collective potential function similar to Ref. [2] to enforce the separation flocking rule, given by

$$U(\boldsymbol{q}) = \frac{1}{2}\sum_i \sum_{j \neq i} \psi_\alpha(\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_\sigma) \tag{2.10}$$

where $\boldsymbol{q}$ can be the global configuration and $\psi_\alpha(z)$ can be the function of a smooth pairwise repulsive/attractive potential [2]. It makes the collective potential a smooth function. That is, for example, even when two agents coincide (i.e. $\boldsymbol{q}_i = \boldsymbol{q}_j$), there is no singular configurations for a collective potential with the property that it is

differentiable everywhere. $\psi_\alpha(z)$ is given by

$$\psi_\alpha(z) = \int_{d_\alpha}^{z} \phi_\alpha(s)\, ds \tag{2.11}$$

with a minimal point at d_α and a finite cut-off when $r_\alpha = \|r\|_\sigma$. $\phi_\alpha()$ is a gradient-based term (also named as react force) given by

$$\phi_\alpha(z) = \rho_h(z/r_\alpha)\phi(z - d_\alpha) \tag{2.12}$$

$$\phi(z - d_\alpha) = \frac{1}{2}[(a+b)\sigma_1(z - d_\alpha + c) + (a-b)] \tag{2.13}$$

where $z = \|\boldsymbol{q}_j - \boldsymbol{q}_i\|_\sigma$, $r_\alpha = \|r\|_\sigma$ and $\sigma_1(\tilde{z}) = \tilde{z}/\sqrt{1+\tilde{z}^2}$. $\phi(\boldsymbol{.})$ is an uneven sigmoid function with factors $c = |a-b|/\sqrt{4ab}, 0 < a \leq b$ which guarantees $\phi(0) = 0$ and $\phi_\alpha(z) = 0$, $\forall z \geq r_\alpha$.

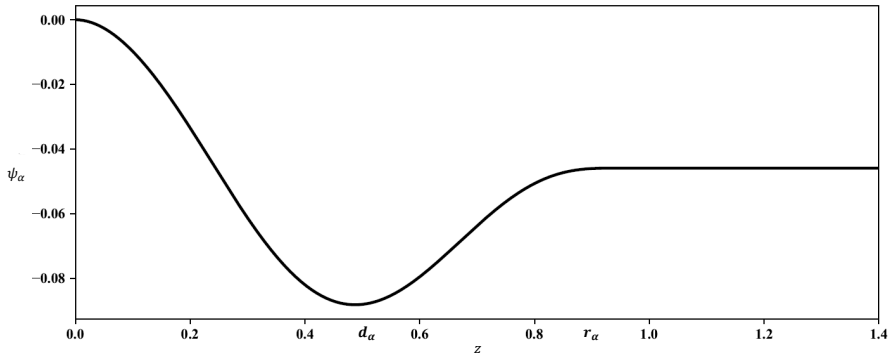

Figure 2.6: Example of a Collective Potential

A collective potential, of which a simple example is shown in Figure 2.6 (here parameters $d_\alpha = 0.488, z_\alpha = 0.936$), generates a reaction force which is attracting if a pair of two agents are apart but repelling if the distance between the two agents are quite small. There is an equilibrium point when $z = d_\alpha$, which can be chosen to make the reaction force first become zero. For $z > 0$, the potential shapes the reaction force to reach zero smoothly when z becomes d_α which is faintly smaller than r_α. Therefore, we generally configure parameters as $0 < d_\alpha < r_\alpha$, which yields the gradient of the collective potential to be a zero vector when the communication to agent i is not connected from its neighboring agent j. In this case, despite the limited communication, a continuously differentiable reaction force can be constructed.

Gradient-based Component

We introduced the separation model (2.12) proposed in Ref. [2], to construct the mobile sensing behaviors. Taking partial derivative of $U(\boldsymbol{q})$ (2.10) associated with $\boldsymbol{q}_i$, we obtain

$$\nabla U(\boldsymbol{q}) = -\sum_{j\in\mathcal{N}_i} \phi_\alpha(\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_\sigma)\boldsymbol{n}_{ij} \tag{2.14}$$

where $\boldsymbol{q}$ is the global configuration and $\boldsymbol{n_{ij}}$ is a vector, given by

$$\boldsymbol{n}_{ij} = \frac{\boldsymbol{q}_j - \boldsymbol{q}_i}{\sqrt{1+\epsilon\|\boldsymbol{q}_j - \boldsymbol{q}_i\|^2}} \tag{2.15}$$

where $j \in \mathcal{N}_i$ and $\epsilon \in \mathbb{R}_{>0}$. Furthermore, the separation term of agent i is defined as

$$\begin{aligned} \boldsymbol{u}_i^s &= -\nabla U(\boldsymbol{q}) \\ &= \sum_{j\in\mathcal{N}_i} \phi_\alpha(\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_\sigma)\boldsymbol{n_{ij}}. \end{aligned} \tag{2.16}$$

2.3.3 Alignment Behavior

Graph Laplacian

For a graph $\mathcal{G}$, its adjacency matrix is defined as $\boldsymbol{A} = [a_{ij}]$. The degree matrix of the graph $\mathcal{G}$ can be a diagonal matrix $\boldsymbol{D^A}$, of which diagonal entries are row sums of matrix $\boldsymbol{A}$, i.e., $\boldsymbol{D^A} = \text{diag}(\sum_{j=1}^N a_{ij})$. The graph Laplacian $\boldsymbol{L} = [\ell_{ij}] \in \mathbb{R}^{N\times N}$ can be defined as

$$\boldsymbol{L} = \boldsymbol{D^A} - \boldsymbol{A} \tag{2.17}$$

where the graph Laplacian $\boldsymbol{L}$ must have a right eigenvector of $\underline{1} = [\underbrace{1,1,...,1}_{N}]^T$, which refers to the N-vector of ones associated with eigenvalue $\lambda_1(\boldsymbol{L}) = 0$ (proof is given in Ref. [32]).

In Ref. [32], $\lambda_2(\boldsymbol{L})$ is associated with algebraic connectivity for a graph. A graph can be described as a connected graph, when $\lambda_2(\boldsymbol{L}) > 0$ [28]. It is necessary to study the connectivity of a communication graph, by using the graph Laplacian. Particularly, an m-dimensional graph Laplacian can be given by

$$\widehat{\boldsymbol{L}} = \boldsymbol{L} \otimes \boldsymbol{I}_m \tag{2.18}$$

where the identity matrix $\boldsymbol{I}_m \in \mathbb{R}^{m\times m}$ and $\otimes$ denotes the Kronecker product (detailedly defined in [28]). The quadratic velocity disagreement function [2] $\Psi_{\mathcal{G}} : \mathbb{R}^{mN} \mapsto \mathbb{R}_{\geq 0}$ is defined as

$$\begin{aligned}\Psi_{\mathcal{G}}(\boldsymbol{v}) &= \frac{1}{2}\,\boldsymbol{v}^T\widehat{\boldsymbol{L}}\,\boldsymbol{v} \\ &= \frac{1}{4}\sum_{(i,j)\in\mathcal{E}} a_{ij}\,\|\boldsymbol{v}_j-\boldsymbol{v}_i\|^2\end{aligned} \tag{2.19}$$

where $\boldsymbol{v} = [\boldsymbol{v}_1^T, \boldsymbol{v}_2^T, ..., \boldsymbol{v}_N^T]^T \in \mathbb{R}^{mN}$ and $\boldsymbol{v}_i \in \mathbb{R}^m$ stands for the velocity of agent i. Therefore, differentiating $\Psi_{\mathcal{G}}(\boldsymbol{v})$ in terms of $\boldsymbol{v}$, we get

$$\nabla\Psi_{\mathcal{G}}(\boldsymbol{v}) = \widehat{\boldsymbol{L}}\,\boldsymbol{v} \tag{2.20}$$

which is the so-called consensus term [21] [33].

Velocity Matching

A smooth spatially weighted adjacency matrix can be written as [2]:

$$\begin{aligned}&\boldsymbol{A}(\boldsymbol{q}) = [\,a_{ij}(\boldsymbol{q})\,] \\ &a_{ij}(\boldsymbol{q}) = \rho_h(\|\boldsymbol{q}_j-\boldsymbol{q}_i\|_\sigma/r_\alpha) \in [0,1], \quad j\neq i\end{aligned} \tag{2.21}$$

with $r_\alpha = \|r\|_\sigma$. $\boldsymbol{q} = [\boldsymbol{q}_1^T, \boldsymbol{q}_2^T, ..., \boldsymbol{q}_N^T]^T$ is the global position and $\boldsymbol{q}_j$ is a neighbour of $\boldsymbol{q}_i$ in the graph, and ρ_h is the bump function. It can be seen that $\|a_{ij}\| = 0$, when the associated distance between agent i and $j \in \mathcal{N}_i$ is bigger than r_α.

Hence, the alignment control $\boldsymbol{u}_i^a$, which is accountable for regulating the velocity of each agent (index i) to the weighted average of those of its neighbouring agents (proof in Ref. [2]), is defined as

$$\begin{aligned}\boldsymbol{u}_i^a &= -\nabla\Psi_{\mathcal{G}}(\boldsymbol{v}) = -\sum_{j\in\mathcal{N}_i}\widehat{\boldsymbol{L}}_{ij}\,\boldsymbol{v}_j \\ &= \sum_{j\in\mathcal{N}_i} a_{ij}(\boldsymbol{q})(\boldsymbol{v}_j-\boldsymbol{v}_i)\end{aligned} \tag{2.22}$$

where $\nabla\Psi_{\mathcal{G}}$ (2.20) is consensus term.

2.3.4 Cohesion Behavior

Flocking centralized information is used to achieve cohesion behavior [34] [35]. Each agent in a flocking system is driven via a consensus method to predict the flocking centroid and attempts to stay closer to its neighbours via local communication. A virtual leader is introduced to facilitate and guide agents to predict the centroid faster, eliminating fragmentation phenomena (proof in Ref. [2]).

Inspired by classical Proportional-Derivative (PD) controller, the cohesion term (also named as the tracking term in Ref. [31]) is given by

$$\begin{aligned} \boldsymbol{u}_i^c &= f^c(\boldsymbol{q}_\ell, \boldsymbol{v}_\ell, \boldsymbol{q}_i, \boldsymbol{v}_i) \\ &= -k_p(\boldsymbol{q}_i - \boldsymbol{q}_\ell) - k_d(\boldsymbol{v}_i - \boldsymbol{v}_\ell) \end{aligned} \tag{2.23}$$

where $\boldsymbol{q}_\ell \in \mathbb{R}^m$ refers to the position and $\boldsymbol{v}_\ell \in \mathbb{R}^m$ refers to the velocity of the virtual leading agent (or leader). Weights $k_p,\ k_d \in \mathbb{R}_{>0}$ are positive feedback gains.

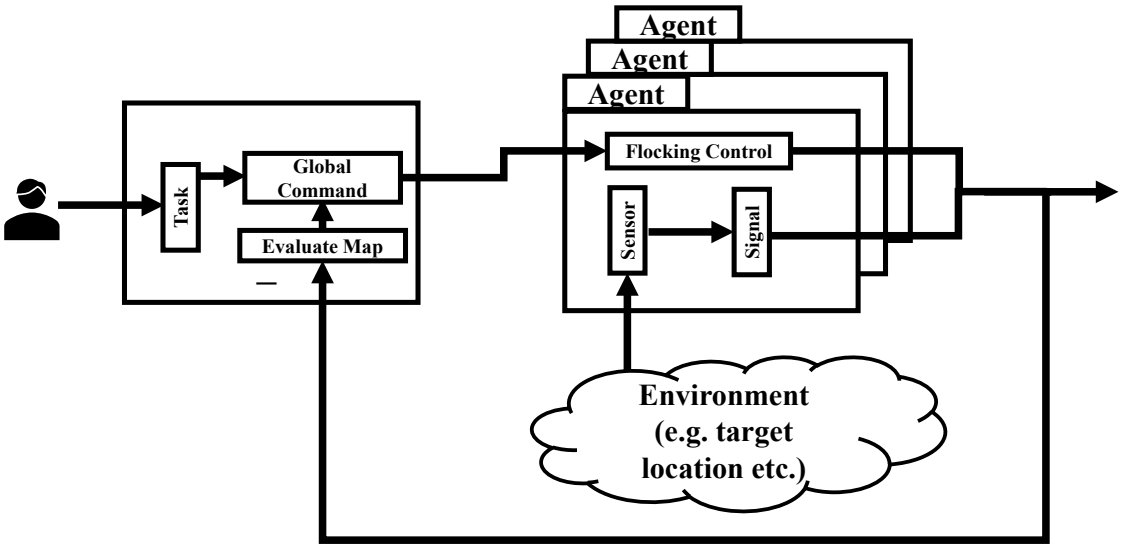

Figure 2.7: Representation of Tracking Control [12]

Also, $\boldsymbol{q}_\ell, \boldsymbol{v}_\ell$ can be states of a target when we consider the tracking problems [2] [31]. This is why the cohesion term is also called as the tracking term. In Figure 2.7, each sensing agent is driven to achieve the task assigned by engineers. The environment can stand for information from a monitored surveillance area, and for example agents locate odor source [12] [24] [36], the chemical particle [27] [37], the oil spills [13], etc. In engineering context, the virtual leader (or the moving target)

can indicate a reference trajectory assigned to every agent. Therefore, the main objective is to build tracking controllers that guide all agents to track the time-varying reference trajectory via local information exchange. In our work, we assume that all agents have full knowledge of states $\boldsymbol{q}_\ell, \boldsymbol{v}_\ell$ of a target, when tracking it.

2.3.5 Combined Behavior

In this section, we describe a distributed combined control [2], which drives agents to accomplish certain missions cooperatively. The dynamics of agent i in (2.5), where $\boldsymbol{u}_i$ is the input of agent i, is given by

$$\boldsymbol{u}_i = k_s \boldsymbol{u}_i^s + k_a \boldsymbol{u}_i^a + k_c \boldsymbol{u}_i^c \tag{2.24}$$

where $k_s, k_a, k_c \in \mathbb{R}_{>0}$ denote the weight parameters for all behavior-based terms. Inspired by separation, alignment, cohesion behavior [2], $\boldsymbol{u}_i^s$ (2.16), $\boldsymbol{u}_i^a$, (2.22), $\boldsymbol{u}_i^c$ (2.23) are gradient-based, consensus, and tracking term, respectively. These terms are combined to drive each agent to flock together toward a target location. Hence, $\boldsymbol{u}_i$ is specifically written as

$$\begin{aligned} \boldsymbol{u}_i = & \overbrace{k_s \sum_{j \in \mathcal{N}_i} \phi_\alpha(\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_\sigma) \boldsymbol{n}_{ij}}^{\text{gradient-based term}} + \overbrace{k_a \sum_{j \in \mathcal{N}_i} a_{ij}(\boldsymbol{q})(\boldsymbol{v}_j - \boldsymbol{v}_i)}^{\text{consensus term}} \\ & + \underbrace{k_c(-k_p(\boldsymbol{q}_i - \boldsymbol{q}_\ell) - k_d(\boldsymbol{v}_i - \boldsymbol{v}_\ell))}_{\text{tracking term}} \end{aligned} \tag{2.25}$$

Shown in Figure 2.8, agents are driven to track the trajectory of a target, to synchronize their speeds to a common value, and at the same time, to avoid collisions with their neighbors.

2.3.6 Limitations

There are, nevertheless, some disadvantages for practical application. The dynamics of each agent is often complicated and impossible to be modelled accurately in real-world scenarios. For example, nonlinear mobile robots have been used in a variety of industrial applications. Abouheaf and Gueaieb [31] introduced a method for a number of nonholonomic systems via input-output feedback linearization. An extended Takagi-Sugeno-Kang fuzzy method is presented in Ref. [31] to achieve a local control objective (collision avoidance). Jadaliha et

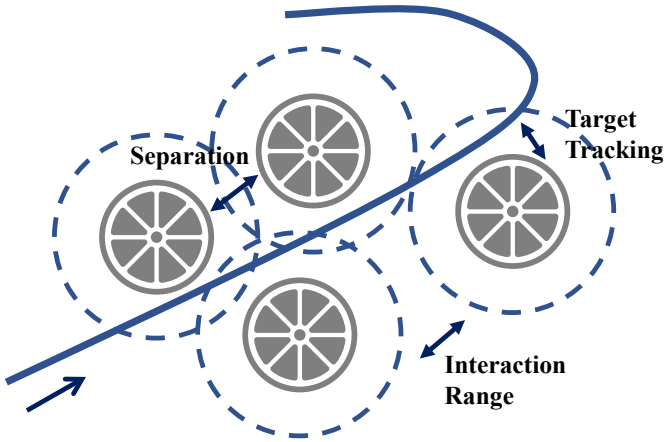

Figure 2.8: Combined Control Behavior [31]

al. [37] proposed a distributed control law to a group of nonholonomic differentially driven vehicles. The control in Ref. [37] is constructed by the gradient ascent depending on the recursively estimated fields as well as the swarming effort. In addition, most tracking control methods often require the knowledge of reference trajectories [31]. However in practical applications, the dynamical models of reference trajectories are often unknown or uncertain [38]. Accordingly, it could be computationally prohibitive to introduce high uncertainties in the implementation.

Hence, a model-free approach could be advisable when a model-based approach is hard for implementations. This approach aims at obtaining adaptive tracking gains which rely only on measurements locally collected along the trajectories.

2.4 Nonlinear Control

2.4.1 Formulation of Tracking Problem

In order to unify and simplify the notations of the states, i.e. $\boldsymbol{q}_i, \boldsymbol{q}_\ell$, $\boldsymbol{v}_i, \boldsymbol{v}_\ell$, we use the new notations $\boldsymbol{x}_i, \boldsymbol{x}_\ell$ to describe the systems' states. The control system is discretized into a discrete-time system so that it can be integrated into a finite difference adaptive control structure. Consider the communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

with N agents, each agent with local dynamics defined as

$$\boldsymbol{x}_i(k+1) = \boldsymbol{f}_i(\boldsymbol{x}_i(k), \boldsymbol{u}_i(k)) \tag{2.26}$$

where $\boldsymbol{x}_i(k) = \left[\boldsymbol{q}_i^T(k), \boldsymbol{v}_i^T(k)\right]^T \in \mathbb{R}^n$ refers to the state trajectory of agent i, and $\boldsymbol{u}_i(k) \in \mathbb{R}^m$ is regarded as the tracking control signal for agent $i \in \mathcal{V} = \{1, 2, ..., N\}$. k is the discrete-time index and $\boldsymbol{f}_i : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$, $\forall i$.

Suppose that $\boldsymbol{x}_\ell(k) = \left[\boldsymbol{q}_\ell^T(k), \boldsymbol{v}_\ell^T(k)\right]^T \in \mathbb{R}^n$ is the reference trajectory. For the tracking problem, the control objective is to build the control $\boldsymbol{u}_i(k)$ for the system (2.26). The dynamics of the reference system [3] is defined by

$$\boldsymbol{x}_\ell(k+1) = \boldsymbol{f}_\ell(\boldsymbol{x}_\ell(k)) \tag{2.27}$$

Regarding the local tracking error as $\boldsymbol{e}_i(k) = \boldsymbol{x}_i(k) - \boldsymbol{x}_\ell(k)$, then the equation follows from (2.26) and (2.27) that

$$\boldsymbol{e}_i(k+1) = \boldsymbol{f}_i(\boldsymbol{e}_i(k) + \boldsymbol{x}_\ell(k), \boldsymbol{u}_i(k)) - \boldsymbol{f}_\ell(\boldsymbol{x}_\ell(k)) \tag{2.28}$$

The tracking goal is given by

$$\lim_{k\to\infty} \|\boldsymbol{e}_i(k)\| = 0,\ i \in \{1, 2, ..., N\} \tag{2.29}$$

which means that each state synchronizes to the reference state. And this behavior is the so-called tracking behavior [39] or cohesion behavior [2] in flocking control.

2.4.2 PID Control

As one of the most common control algorithms, Proportional-Integral-Derivative (PID) controller [40] [41] has been broadly used in control engineering. PID controllers are popular because of their functional simplicity as well as their robust performance in a wide range of operating conditions, allowing control engineers to apply them in a forthright way [41]. As its name indicates, PID control method comprises three fundamental factors: proportional, integral and derivative, which can be tuned to reach optimal response. The classical PID control theory and the performances of tuning a closed-loop control system are reviewed in [40]. The three basic coefficients are the control gains:

- k_p: proportional;

- k_i: integral;
- k_d: derivative.

The discretized PID controller can be constucted at discrete time k:

$$\begin{aligned} \boldsymbol{u}_i(k) = \boldsymbol{u}_i^{\text{PID}}(k) &= g_i(\underbrace{\boldsymbol{e}_i(0), \boldsymbol{e}_i(1), ..., \boldsymbol{e}_i(k)}_{k+1}) \\ &= k_p\, \boldsymbol{e}_i(k) + k_i \sum_{\kappa=0}^{k} \boldsymbol{e}_i(\kappa) + k_d\, (\boldsymbol{e}_i(k) - \boldsymbol{e}_i(k-1)) \end{aligned} \tag{2.30}$$

where $\boldsymbol{e}_i(k)$ is the local tracking error vector (2.28), and the three gains $k_p, k_i, k_d \in \mathbb{R}_{\leq 0}$ (named as pinning gains [42]), and $g_i : \mathbb{R}^n \mapsto \mathbb{R}^m$. The control diagram is shown in Figure 2.9 based on the PID ontroller.

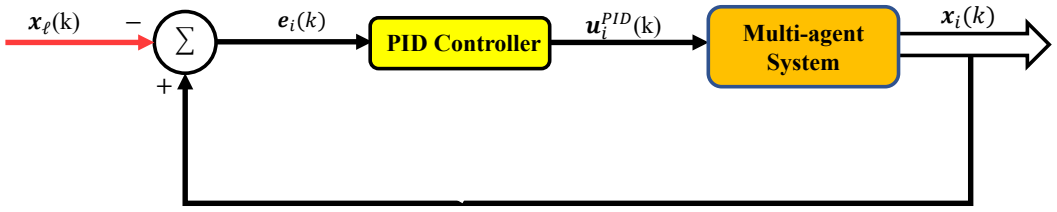

Figure 2.9: PID Control

While traditional PID control is broadly used in control problems, its performance may not always be acceptable as the result of nonlinear and time-varying effects for industrial applications. Firstly, since the sizes of the real-world dynamical systems become increasingly large, most real-world systems are too complex to obtain or identify precise mathematical models [43] [44]. Also, once the number of agents (N) is extremely big, the control process can be effected by the curse of dimensionality [4]. Secondly, the tracking error serise $\boldsymbol{e}_i(0), \boldsymbol{e}_i(1), ..., \boldsymbol{e}_i(k)$ is difficult to be stored instantly. Hence, a good storage capacity for data and information should be required [41]. Thirdly, the weights are the conventional PID coefficients, which are often fixed gains and tuned traditionally [45]. That is, PID manual tuning is conventionally based on past engineering experience, and even for an experienced control engineer, adjusting nonlinear or unknown systems is extremely hard. Hence, some approaches have been introduced to make gains adjusted adaptively [6] [45].

Therefore, machine learning techniques are widely used in multi-agent tracking control, such as fuzzy-logic [46], Bayesian approach [47], reinforcement learning [48] [49], etc. One of the general forms [49] for machine learning control can be defined as

$$\boldsymbol{u}_i(k) = \breve{g}_i\left(\boldsymbol{E}_i(k)\right) \tag{2.31}$$

where $\boldsymbol{u}_i$ is the machine learning controller that will be discussed in the following sections, and $\boldsymbol{E}_i(k)$ is described as

$$\boldsymbol{E}_i(k) = [\underbrace{\boldsymbol{e}_i{}^T(k), \boldsymbol{e}_i{}^T(k-1), ..., \boldsymbol{e}_i{}^T(k+1-K)}_{K}]^T \tag{2.32}$$

where K is the number of terms stored in the cumulative error vector [6] [45] and $\boldsymbol{E_i}(k) \in \mathbb{R}^{nK}$ for agent i [6].

2.4.3 Optimal Control

Optimal control is one of the mathematical disciplines with a number of applications in mathematics, engineering and operations research [50] [51]. It aims at deriving a control by optimizing cost functions defined by users which capture desired design objectives, over a period of time. As a development of the calculus of variations, optimal control is comprised of a set of mathematic procedures to find control policies [50]. The method is greatly owing to the study of Bellman [52], after Edward's contributions in calculus of variations [51].

For optimal tracking problems, it is necessary to construct a controller, which drive agents to track the desired reference trajectories and to achieve the optimal performance [6] [53]. Extensive work exists for solving the optimal tracking problem relied on the system's model [43]. The state errors and the expected control are applied to obtain the performance index, and then the optimal tracking control problem can be reformulated as the optimal regulation control problem of the state errors associated to the performance index [43] [44].

However for several real-world scenarios, there is often no prior or sufficiently accurate information about the dynamics [54] [55]. With a totally unknown model, the control problem for nonlinear dynamical systems was solved in [6] using input-output information. In these cases, a class of model-free algorithms were presented in [6] [45] [54].

The performance index in [50] with respect to the long-run cost can be given by

$$\mathrm{J}_i = \sum_{k=0}^{\infty} R_i(\boldsymbol{E}_i(k), \boldsymbol{u}_i(k)) \tag{2.33}$$

where $R_i(\boldsymbol{.})$ is the quadratic objective cost function [50]

$$R_i(\boldsymbol{E}_i(k), \boldsymbol{u}_i(k)) = \frac{1}{2}(\boldsymbol{E}_i^T(k)\ \boldsymbol{S}_i\ \boldsymbol{E}_i(k) + \boldsymbol{u}_i^T(k)\ \boldsymbol{Z}_i\ \boldsymbol{u}_i(k)) \tag{2.34}$$

where $\boldsymbol{S}_i > 0 \in \mathbb{R}^{nK \times nK}$ and $\boldsymbol{Z}_i > 0 \in \mathbb{R}^{m \times m}$. $\boldsymbol{E}_i(k)$ is the cumulative error vector defined in (2.32). Further, the optimal tracking problem for the system (2.26) can be reformulated into an optimal regulation problem [50], which is, to figure out the control

$$\boldsymbol{u}_i^*(k) = \arg\min_{\boldsymbol{u}_i} \mathrm{J}_i \tag{2.35}$$

associated to the performance index (2.33).

The term model-free means that the dynamics of the system (2.26) and the reference system (2.27) are completely unknown, i.e., $\boldsymbol{f}_i(\boldsymbol{x}_i(k), \boldsymbol{u}_i(k))$ and $\boldsymbol{f}_\ell(\boldsymbol{x}_\ell(k))$ are unknown. Consequently, the tracking error dynamics (2.28) is unknown. In the thesis, each agent tracks the reference trajectory using the measurable state information rather than the accurate dynamics in (2.28). Machine learning methods to solve the optimal problems are going to be reviewed in the following section.

2.5 Machine Learning Methods

2.5.1 Reinforcement Learning

As an important topic in machine learning, reinforcement learning (RL) can be applicable to solving optimal control problems. RL is one of the three fundamental machine learning algorithms (The book [49] classified the machine learning algorithms as: RL, supervised learning and unsupervised learning), where the agent (controller) interacts with the environment (controlled plant) and changes its actions (control signals) according to stimuli obtained in response to its actions in order to minimize the cumulative reward [49]. RL is also called action-based learning and it tends to emulate human behavior [4] [50]. The advantage of using RL in control problems, is that an agent is able to self-learn to adapt to the time-varying environment, and be trained online, synchronously improving system performance

[56].

Therefore, RL has long been applied on games and simulated environments [49] [50] and during the last two decades it has been explored to multi-agent systems in real world applications, such as area coverage [57], obstacle avoidance [58] [59], etc. The RL control diagram for multi-agent is illustrated in Figure 2.10.

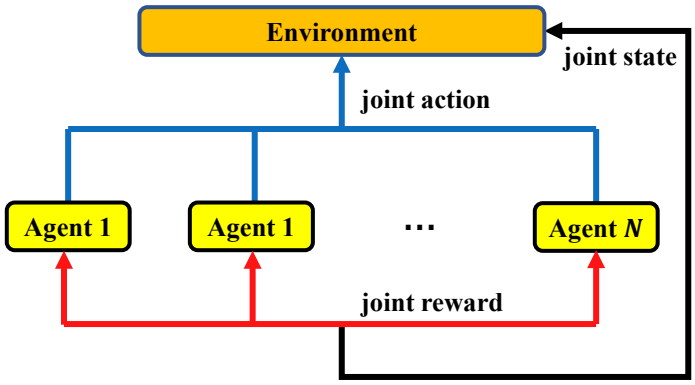

Figure 2.10: Multi-agent Reinforcement Learning [49]

An RL control diagram is shown in Figure 2.11, where the controller, based on reinforcement and state feedback associated to its previous action, estimates the control which will bring about a better performance [49]. The reinforcement signal is generally defined as a performance evaluation function, associated to the state plus the control.

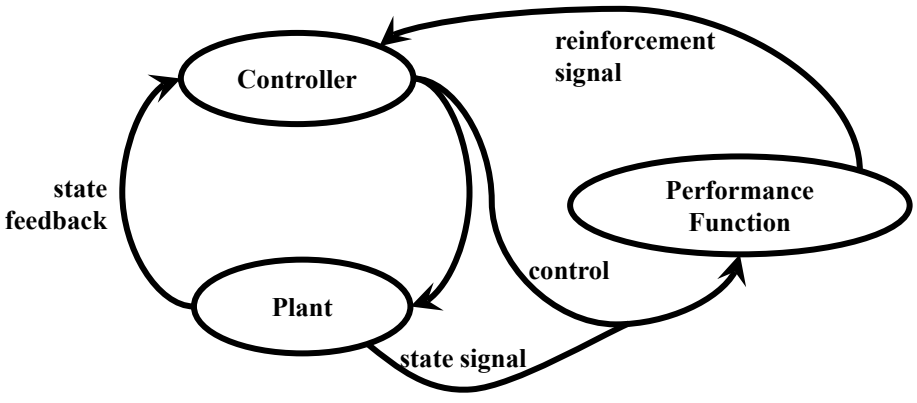

Figure 2.11: Reinforcement Learning Controller [49]

Elements of Reinforcement Learning

There are some fundamental elements [8] of the reinforcement learning (RL) strategy: a reward, a policy and a state (s) from the environment. At each time step $t = 0, 1, 2, ...$, the agent chooses a state $s_t \in S$ as an input, which stands for the state of the environment at t, and S stands for the set of all states. Depending on the current state, the agent does an action $a_t \in A(s_t)$, in which $A(s_t)$ refers to the set of all possible actions from state s_t. At the next time step $t + 1$, the state transition is from s_t to s_{t+1} after taking action a_t. The transition corresponds to a immediate reward $r_{t+1} \in R$, depending on how "good" or "bad" is selecting action a_t at state s_t. This framework was first presented in [60] and then extended to the research by Barto [61]. Comprehensive treatment of RL fundamentals are provided by Sutton and Barto [8].

Figure 2.12 shows that an agent (of multi-agent systems) interacts with its environment by taking an action and this action can be associated to a reward (negative or positive RL signal [8] [50]) as mentioned above. As a control engineering term [50], the reward implies a control cost increase. In this thesis, we consider the reward as a negative signal.

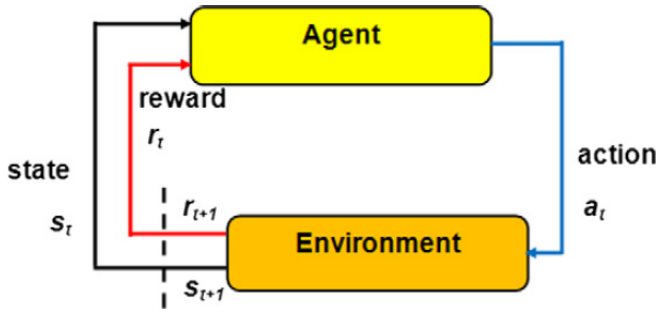

Figure 2.12: Reinforcement Learning [49]

The RL agent interaction is featured by the action, state and reward signal [8]:

- The state signal is regarded as the state of the environment.
- The RL agent affects the environment, throughout the action signal.
- The reward signal indicates the negative feedback regarding to the action [8] [49].

2.5.2 Cumulative Reward

Reward (or reward signal) is one of the main elements of the reinforcement learning (RL) algorithms. In RL, the overall purpose is to obtain a policy (mapping from every state to every possible action) which yields the minimum long-term accumulated reward [4]. To influence the agent's behavior, a reward mechanism is adopted. The reward function applied is often custom designed based on the problem in hand. It is the objective of the agent to learn from the environment's behavior by estimating and minimizing the long-term total expected reward [49].

Herein, we suppose that an RL task is with an associated terminal state. Then, the cumulative reward R_t (named as the return in Ref. [49]) is defined as

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_{t+T+1} = \sum_{k=0}^{T} r_{t+k+1} \tag{2.36}$$

where T is the terminal time step. Such a cumulative reward definition is applicable to a class of RL tasks with a terminal state [49]. That is, the agent-environment interaction terminates when it reaches the specific state. In such cases, the experiments of the interaction can be broken down to episodes, where each episode [49] refers to a succession of the interactions between the environment and the agent, which starts at $t = 0$ and ends when the environment achieves the terminal state. Nevertheless, terminal states may not exist in some situations. In such scenarios, the agent-environment interaction continues indefinitely $T = \infty$. When $T = \infty$, the collective reward can be also named as the performance index in Ref. [50].

Besides, a discounted term can be used to determine how much the RL agent is concerned about the rewards in the distant future, compared with those in the immediate future. An example of a discounted cumulative reward function is given by

$$R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + ... + \gamma^T r_{t+T+1} = \sum_{k=0}^{T} \gamma^k\, r_{t+k+1} \tag{2.37}$$

where the parameter γ is named as the discount rate, $0 \leq \gamma \leq 1$. The discount rate is a weighing factor that affects the current estimation of the future rewards [4]. If $\gamma \to 0$, the agent will be thoroughly myopic, which only learn about the actions that lead to the immediate reward. On the other hand, when $\gamma \to 1$, the agent will assess the performance of its actions, depending on the sum total of all future rewards. In addition, since $\gamma < 1$, the reward sequence $\{r_{t+1}, r_{t+2}, ..., r_{T+k+1}\}$ and the sum of future rewards, remain bounded as $T \to \infty$ [4].

2.5.3 Markov Decision Processes

Generally, the feedback from the environment at time step $t+1$ to the action a_t chosen at time step t, relies on the past information of the agent's states and actions [4]. Such dynamics can be described by the conditional probability

$$\Pr\{s_{t+1} = s',\ r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\} \tag{2.38}$$

for every possible state $s' \in S$, reward r, and all available values of early states, rewards and actions, Pr stands for the probability. A Markov process is a special case of such systems. A process satisfies the Markov property whenever its status at time step $t+1$ only relies on that at time step t [4]. Based on this, an environment has the Markov property when the probability of its state s_{t+1} at time $t+1$ only depends on its state s_t and the action a_t. Hence, the probability is given by

$$\Pr\{s_{t+1} = s',\ r_{t+1} = r \mid s_t,\ a_t\} \tag{2.39}$$

for all $s' \in S$, r, $s_t \in S$ and $a_t \in A(s_t)$. Any reinforcement learning (RL) task satisfying the Markov property is named as a Markov decision process (MDP) [49]. Moreover, if there are only a finite number of state and action spaces in those tasks, then those tasks can be called finite Markov decision processes (MDPs) [49].

Given an action a_t and a state s_t at time t, the probability for each available or possible next state s' is called a transition probability [49] and is given by

$$\mathrm{P}^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s,\ a_t = a\} \tag{2.40}$$

Likewise, the expected value of the next reward associated with action a at state s and transiting to state s' can be defined by

$$R^a_{ss'} = \mathrm{E}\{r_{t+1} \mid s_t = s,\ a_t = a,\ s_{t+1} = s'\} \tag{2.41}$$

This reward signal informs the agent how "bad" or "good" it is to take action a at state s instantly [49]. The main goal of the agent is to minimize the total long-term reward, not the immediate reward r_{t+1} at any specific time step. Accordingly, a value function is based on each state-action pair which indicates the long-lasting evaluation of any state [62]. Because the value function reflects the long-lasting performance of the state-action pairs (i.e. particular policies), the agent gives it special consideration [62]. Over the learning process, the agent receives the imme-

diate reward signal r_{t+1} and then uses it to calculate the value function for each state s [4] [49]. The book [49] defines the state-value function $V^{\pi}(s)$ (the value of a state over a policy π) as below

$$\begin{aligned} V^{\pi}(s) &= \mathrm{E}_{\pi}\{R_t \mid s_t = s\} \\ &= \mathrm{E}_{\pi}\{\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \mid s_t = s\} \\ &= \mathrm{E}_{\pi}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+2} \mid s_t = s\} \\ &= \sum_{a} \pi(s,a) \sum_{s'} \mathrm{P}^{a}_{ss'}[R^{a}_{ss'} + \gamma \, \mathrm{E}_{\pi}\{\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+2} \mid s_{t+1} = s'\}] \\ &= \sum_{a} \pi(s,a) \sum_{s'} \mathrm{P}^{a}_{ss'}[R^{a}_{ss'} + \gamma \, V^{\pi}(s')] \end{aligned} \tag{2.42}$$

where $\pi(s,a)$ refers to the probability of choosing action a in state s according to policy π, and $\mathrm{E}_{\pi}\{\}$ stands for the expected value of a certain variable, given the policy π followed by the agent.

Likewise, the action-value function $Q^{\pi}(s,a)$ is defined in Ref. [49], which refers to the expected reward of choosing action a in state s with policy π

$$\begin{aligned} Q^{\pi}(s,a) &= \mathrm{E}_{\pi}\{R_t \mid s_t = s, \; a_t = a\} \\ &= \mathrm{E}_{\pi}\{\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \mid s_t = s, \; a_t = a\} \\ &= \mathrm{E}_{\pi}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+2} \mid s_t = s, \; a_t = a\} \\ &= \sum_{a} \pi(s,a) \sum_{s'} \mathrm{P}^{a}_{ss'}[R^{a}_{ss'} + \gamma \, Q^{\pi}(s', \, a')] \end{aligned} \tag{2.43}$$

Equations (2.42) and (2.43) stand for the Bellman equations for the state-value equation V^{π} and the action-value equation Q^{π} under policy π, respectively [49]. These equations balance all the possible circumstances by their probability of occurring [62]. They also state that the value of the starting state is equivalent to the discounted value associated to the expected next state and the long-run reward.

The main objective of RL tasks, is to find a policy which provides the smallest long-term reward [4]. To put it differently, they are required to obtain the policy which has a smaller or equivalent expected cumulative reward than other policies for all states. Value functions validate the efficiency of all possible policies [49].

A better policy associates a smaller value function. A policy π is better than or equivalent to a policy π' when the expected cumulative reward brought by π is smaller than or equivalent to that brought by π' for all states [62]. That is to say, policy π is better or as good as π' when

$$V^{\pi}(s) \leq V^{\pi'}(s) \quad \forall s \in S \tag{2.44}$$

A policy that minimizes the long-term reward is called an optimal policy π^*. The state-value function yielding the minimum reward is defined as the optimal state-value function V^* [49], and can be calculated from

$$V^*(s) = \min_{\pi} V^{\pi}(s) \tag{2.45}$$

Likewise, the optimal action-value function [49] $Q^*(s,a)$ is expressed as

$$Q^*(s,a) = \min_{\pi} Q^{\pi}(s,a) \tag{2.46}$$

The optimal state-value function and the optimal action-value function must satisfy the Bellman equations (2.42), and (2.43), which can be rewritten as

$$V^*(s) = \min_{a} \sum_{s'} \mathrm{P}^{a}_{ss'}[R^{a}_{ss'} + \gamma\, V^*(s')] \tag{2.47}$$

$$Q^*(s,a) = \sum_{s'} \mathrm{P}^{a}_{ss'}[R^{a}_{ss'} + \gamma \min_{a'} Q^*(s',a')] \tag{2.48}$$

Equations (2.47) and (2.48) are called the Bellman optimality equations [4] for the state-value function $V(s)$ and the action-value function $Q(s,a)$, respectively. Some dynamic programming methods to solve the above Bellman equations is going to be reviewed in the following section.

2.5.4 Dynamic Programming

Dynamic programming, as one of the key methods in reinforcement learning (RL), is a class of algorithms that are applied to obtain the optimal policy when given a full dynamical model for the environment as a Markov decision process (MDP) [4] [50]. As mentioned above, the main idea of dynamic programming, and of RL generally, is to utilize value functions and to structure the search for the best policy. In this section, we will review the dynamic programming algorithms that function

in calculating the optimal policies and the value functions. The book [4] classifies dynamic programming into the value iteration (VI) and policy iteration (PI) based on how the optimal policies are obtained, in the RL context.

Policy Iteration

The existence of an optimal policy is guaranteed since the number of policies is finite [50], regarding a finite Markov decision process problem. Given a policy π, we can evaluate whether it is the best policy or not by trying an action $a \neq \pi(s)$ at state s and subsequently following the existing policy π [62]. If this new policy π' with action a is better than to the initial policy π, then the value function $V^{\pi'}$ must be smaller than or equal to V^{π} (i.e. $V^{\pi'} \leq V^{\pi}$). Hence, policy π' becomes a more optimal policy for the environment. To take all possible policies into consideration, we can operate this process repeatedly to find the optimal policy π^*. This iterative or recursive way is the so-called policy iteration (PI) [50]. Suppose that for each state, we have a series of approximate value functions V_0, V_1, V_2, ..., where the initial approximation V_0 can be selected randomly, then over the iteration process, the successive approximations can be obtained using Bellman equation (2.42)

$$\begin{aligned} V_{k+1}(s) &= \mathrm{E}_{\pi}\{r_{t+1} + \gamma\, V_k(s_{t+1}) \mid s_t = s\} \\ &= \sum_{a} \pi(s,a) \sum_{s'} \mathrm{P}^{a}_{ss'}[R^{a}_{ss'} + \gamma\, V_k(s')] \end{aligned} \tag{2.49}$$

In a PI strategy, there are two main steps (summarized in Ref. [50]): policy evaluation and improvement. The first step is to evaluate the current policy through the state-value function (2.49), and the second step is used to improve the current policy by considering a new greedy policy π', defined as

$$\pi'(s) \leftarrow \arg\min_{a} \sum_{s'} \mathrm{P}^{a}_{ss'}[R^{a}_{ss'} + \gamma\, V^{\pi}(s')] \tag{2.50}$$

In the end, the outcome converges to the optimal policy [4] [50].

Value Iteration

As a simpler dynamic programming method, value iteration (VI) solves the Bellman equation (2.47) iteratively [4]. Similar to policy iteration (PI) methods, it requires comprehensive information of the state-transition probabilities as well as the rewards for each transition [62]. In this case, the updating rule [4] can be for-

mulated as

$$V_{k+1}(s) = \min_a \sum_{s'} \mathrm{P}^a_{ss'}[R^a_{ss'} + \gamma\, V_k(s')] \tag{2.51}$$

for all $s \in S$, where γ is the discount rate, $0 \leq \gamma \leq 1$. After converging to a value function [4], the policy is given by

$$\pi(s) \leftarrow \arg\min_a \sum_{s'} \mathrm{P}^a_{ss'}[R^a_{ss'} + \gamma\, V(s')] \tag{2.52}$$

for all $s \in S$, where $\arg\min_a$ denotes the action at state s that is associated with the minimum state-value function [4].

2.5.5 Temporal Difference Learning

The temporal difference (TD) learning methods are the extension of dynamic programming methods, which can be model-free and easily applicable online with step-by-step computation [4]. Similar to dynamic programming, TD learning methods update the estimations after interactions between the environment and the agent without waiting for a final result (i.e., waiting till the next step $t+1$) [50]. TD learning methods can learn the optimal policy straightway from their own experience. Unlike dynamic programming techniques, they do not need a full model for the environment [4] [44].

One of the simplest TD learning methods, known as TD(0) [4], uses the immediate interaction information, the estimated value $V(s_{t+1})$ and the immediate reward r_{t+1}, to generate a target at time step $t+1$ in the form of

$$\text{target} = r_{t+1} + \gamma\, V(s_{t+1}) \tag{2.53}$$

where γ is the discount rate and the term target indicates the desirable motion direction. The difference between the target and the previous estimate $V(s_t)$, named as the TD error, can be applied to update the previous estimate, and is expressed as

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t)] \tag{2.54}$$

where $0 \leq \alpha \leq 1$ is the learning rate. This rate determines how much the agent learns from the newly discovered information [4] (i.e., estimated value $V(s_{t+1})$ and the immediate reward r_{t+1}). If $\alpha = 0$, the agent does not learn anything, but for the case where $\alpha = 1$, the agent only learns from the newly discovered information.

Figure 2.13 shows the backup rule in a diagram form for TD(0). TD learning methods are called bootstrapping methods [49] because they update the state node s depending on one sample transition to the instantly following state s'.

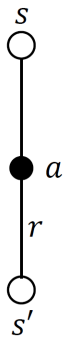

Figure 2.13: Backup diagram for TD(0) [49]

There are two classes of TD learning methods [49] [63]:

- On-policy TD learning: The policy that is being estimated is used to make decisions during the learning process [49].
- Off-policy TD learning: Off-policy methods do not have an expected policy to be followed [63]. Their estimations are updated independently of the policy. Q-learning [64], as one of the most popular off-policy TD learning methods, will be detailedly described in the following section.

Q-learning

Q-learning [63] [64], as one of the most useful temporal difference (TD) learning methods, with great attention from the machine learning communities. The "Q" in Q-learning stands for "quality", objective of which is to determine how useful a given action is in obtaining several future rewards. Thus, the Q-learning agent estimates the Bellman equation (2.48) (also named as Q-value [4]) recursively to get an optimal policy. The simplest Q-learning is the one-step Q-learning [4], where the target is defined as

$$\text{target} = r_{t+1} + \gamma \min_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{2.55}$$

where $0 \leq \gamma \leq 1$ is the discount rate, and $0 \leq \alpha \leq 1$ is the learning rate.

Q-learning [62] also utilizes the immediate new experience from the interaction as a target. Hence, the TD error is the difference between the current Q-value and

the target. The update rule for Q-learning is expressed by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1-\alpha) + \alpha \left[r_{t+1} + \gamma \min_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \right] \tag{2.56}$$

where r_{t+1} refers to the environment-returned reward of action a under state s.

Similar to the forementioned backup diagram of the TD learning, the backup diagram of Q-learning is shown in Figure 2.14. Since update rule (2.56) is updating a state-action pair, the top node and the bottom node are regarded as an action node [4] [49]. Besides, the backup also defines the action associated with the minimum Q-value over all possible actions under the next state [4].

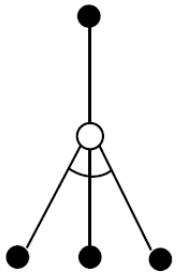

Figure 2.14: Backup Diagram for Q-learning [49]

Q-table, shown in Figure 2.15 is one of the simplest data structures used to minimize the Q-value function $Q(...)$ [49]. Q-table is a lookup table that includes Q-values. The input of the table should be the agent's state-action pair and the output is a Q-value associated to the pair [65] [66]. A selection of a control input is decided by Q-values [65]. Subsequently, the Q-value corresponding to the selected control input is updated based on a reward that is relative to the agent's state-action pair. The optimal Q-table can be found by iteratively updating Q-values illustrated in Figure 2.16. Basically, Q-table guides an agent to the best action at each state [49].

Nevertheless, conventional Q-table takes much time to reach the optimal Q-values [65]. Hence, some appropriate methods [67] [68] can be considered to approximate the Q-value.

2.5.6 Neural Network Approximation

When considering the tracking problems in our work, the conventional Q-table methods can be applied to analytically solve Bellman equation (2.48) and deter-

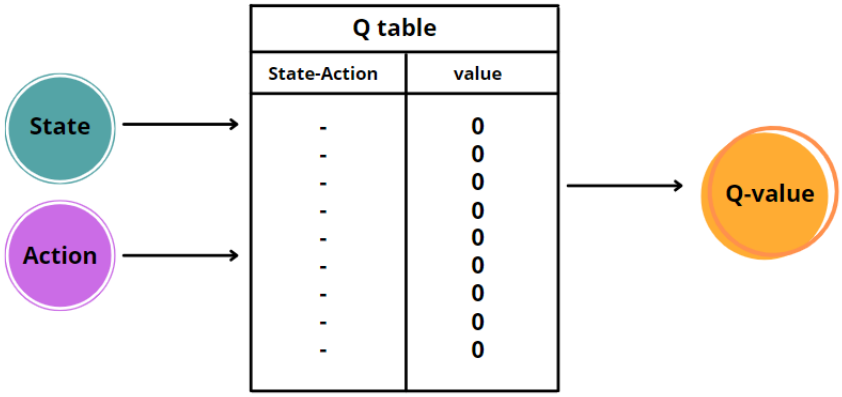

Figure 2.15: Q-table [49]

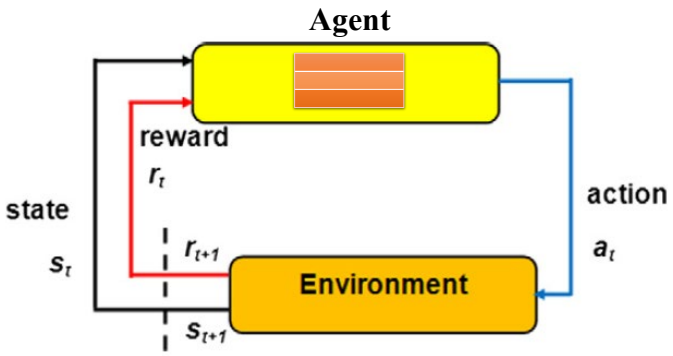

Figure 2.16: Q-table Process [49]

mine optimality in an offline fashion [69]. The offline methods are solved backward in time, with the full knowledge of the system's dynamics. However, for the offline methods, it is infeasible to address the tracking problems with no prior or sufficiently accurate information about the system's dynamics. Therefore, the online methods can be used to surmount the deficiency. The online methods move forward in time, and recover optimality by using neural network approximators and iterators to estimate quantities in the future.

Inspired by the human brain (Figure 2.17), artificial neural networks (ANNs) comprise numerous layers, associated with interconnected processing elements [5]. ANNs are widely used as excellent function approximators to estimate reinforcement learning (RL) solutions [68] [69]. Neural networks make approximations of the value update as well as policy evaluation steps of RL. The neural network solution uses two different approximators which learn the solution to the

Bellman equations. While one of the two neural networks predicts the optimal control strategy, the other neural network estimates the temporal difference (TD) solution.

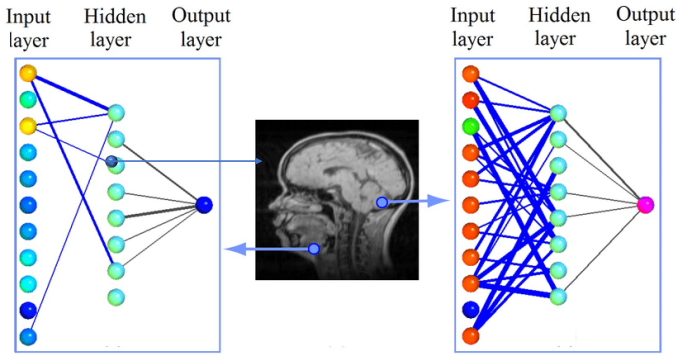

Figure 2.17: Brain-inspired Neural Networks [5]

The adaptive critics approaches [8] [70], also named as actor-critic learning approaches, are applied to find neural network solutions for optimal control problems. The approaches implement two-step RL processes via different neural network approximation methods [68] [71]. The solutions of the Bellman equations are estimated using feedforward neural structures defined by the critic structures [8]. However, the optimal control strategies are estimated by another feedforward neural network, which is named as the actor structures [8].

The tuning processes for the weights of adaptive critics, are coupled interactive, which means the actor weights are regulated while the critic weights are tuned regarding to evaluations of the environment [8] [72]. An adaptive critics structure is illustrated in Figure 2.18. The action-value function update step (2.56) is rewritten as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \, \epsilon_{\text{TD}} \tag{2.57}$$

where $0 \leq \alpha \leq 1$ is the learning rate and ϵ_{TD} is given by

$$\epsilon_{\text{TD}} = r_{t+1} + \gamma \min_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{2.58}$$

where γ is the discount factor and r_{t+1} is the immediate reward.

In Figure 2.18, the critic neural network tries to predict the optimal action-value function for the state s_t over the action a_t. The actor neural network controller is responsible for obtaining the action a_t with minimum action-value. Then the state s_t immediately transits to s_{t+1} after taking the action a_t. Execute the adaptation

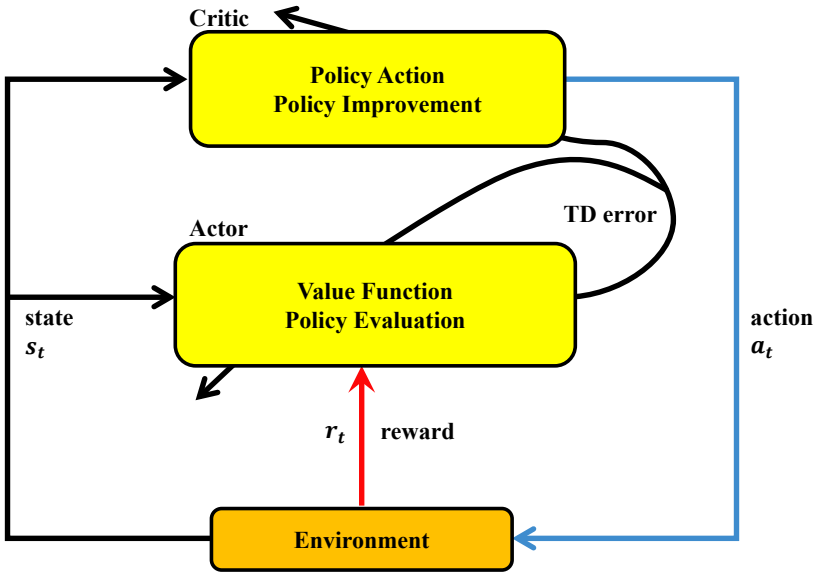

Figure 2.18: Adaptive Critics Structure [8]

process in Figure 2.18 iteratively, until the final optimal decision is found. In general, the control strategy and value function can be improved through iteration and converge to the optimal action-value function with optimal control policy, via adaptive critics approaches [8].

Adaptive critics for an optimal tracking problem, associated with nonlinear cost function, is presented in [67] [73]. A Proportional-Derivative (PD) controller is together with an RL method to control the movement of a two-link flexible manipulator to follow a reference trajectory in [74]. A trajectory-tracking controller based on adaptive critics is proposed for a fully self-governed underwater robot in [69]. The nonlinearities can be compensated over the learning process. The adaptive critics are widely used in single-agent systems [69] [74] and multi-agent systems [73] [75], where each agent has its own adaptive critics structure. In a distributed fashion, these structures solve the graphical games, associated with neighbor information [76] [77]. In this thesis, the adaptive critics method will be applied to deal with a class of multi-agent problems in graph frameworks [43] [78] [79].

2.6 Reinforcement Learning Controller

Reinforcement learning (RL) controller tries to minimize its future rewards [80]. It can be equivalently described as the minimization of a long-term cost in a control

engineering context [50]. In this way, RL method works as an adaptive algorithm that converges online to an optimal control policy for a thoroughly unknown system [4] [50]. It solves the Bellman equation online with measurements along the system trajectories, without any knowledge of the dynamic model [80] [81].

Recall that the objective cost $R_i(.)$ (2.34) is used to calculate the performance index J_i (2.33). Similarly, we define the Bellman temporal difference equation [50] using the Q-learning method:

$$Q_i(\boldsymbol{E}_i(k), \boldsymbol{u}_i(k)) = R_i(\boldsymbol{E}_i(k), \boldsymbol{u}_i(k)) + Q_i(\boldsymbol{E}_i(k+1), \boldsymbol{u}_i(k+1)) \tag{2.59}$$

where $Q_i(\boldsymbol{E}_i(k), \boldsymbol{u}_i(k))$ refers to the action-value with the current input pair at k and $Q_i(\boldsymbol{E}_i(k+1), \boldsymbol{u}_i(k+1))$ stands for the action-value with the future input pair, cumulative error $\boldsymbol{E}_i(k+1)$ and control $\boldsymbol{u}_i(k+1)$ at time $k+1$. Similar to Q-learning [6], the optimal control policy $\boldsymbol{u}_i^*(k)$ is defined by

$$\boldsymbol{u}_i^*(k) = \arg\min_{\boldsymbol{u}_i(k)} Q_i(\boldsymbol{E}_i(k), \boldsymbol{u}_i(k)) \tag{2.60}$$

Substituting (2.60) in (2.59) gives the Bellman optimality equation [50]:

$$Q_i^*(\boldsymbol{E}_i(k), \boldsymbol{u}_i^*(k)) = R_i(\boldsymbol{E}_i(k), \boldsymbol{u}_i^*(k)) + Q_i^*(\boldsymbol{E}_i(k+1), \boldsymbol{u}_i^*(k+1)) \tag{2.61}$$

where $Q_i^*(\boldsymbol{E}_i(k), \boldsymbol{u}_i^*(k))$ refers to the optimal action-value at time k, and $Q_i^*(\boldsymbol{E}_i(k+1), \boldsymbol{u}_i^*(k+1))$ stands for the future optimal action-value at time $k+1$ over the optimal control policy $\boldsymbol{u}_i^*(k+1)$.

2.6.1 Adaptive Critics Implementation

We adopt adaptive critics in the form of artificial neural networks (ANNs) to obtain optimal control policies. The best strategy is approximated using an actor network, when the action-value of this strategy is approximated by means of a critic network [8] [68] [69]. The solving value function $Q_i(...)$ is approximated by the following structure [6] [7]:

$$\hat{Q}_i(\boldsymbol{E}_i(k), \hat{\boldsymbol{u}}_i(k)) = \frac{1}{2}[\boldsymbol{E}_i^T(k)\ \hat{\boldsymbol{u}}_i^T(k)]\ \boldsymbol{W}_i \begin{bmatrix} \boldsymbol{E}_i(k) \\ \hat{\boldsymbol{u}}_i(k) \end{bmatrix} \tag{2.62}$$

where $\boldsymbol{W}_i$ refer to the critic weights for the approximation of the solving value function:

$$\boldsymbol{W}_i = \begin{bmatrix} \boldsymbol{W}_{\boldsymbol{E}_i \boldsymbol{E}_i} & \boldsymbol{W}_{\boldsymbol{E}_i \hat{\boldsymbol{u}}_i} \\ \boldsymbol{W}_{\hat{\boldsymbol{u}}_i \boldsymbol{E}_i} & \boldsymbol{W}_{\hat{\boldsymbol{u}}_i \hat{\boldsymbol{u}}_i} \end{bmatrix} > 0 \in \mathbb{R}^{(nK+m)\times(nK+m)} \tag{2.63}$$

$\boldsymbol{E}_i(k) \in \mathbb{R}^{nK}$ is the cumulative error vector for agent i at time k and $\hat{\boldsymbol{u}}_i(k) \in \mathbb{R}^m$ stands for the approximation of the optimal control policy given by

$$\hat{\boldsymbol{u}}_i(k) = \boldsymbol{\Omega}_i \boldsymbol{E}_i(k) \tag{2.64}$$

where $\boldsymbol{\Omega}_i \in \mathbb{R}^{m\times nK}$ are the associated weights. The target, which is used to update the critic weights, is defined as

$$\tilde{Q}_i = R_i(\boldsymbol{E}_i(k), \hat{\boldsymbol{u}}_i(k)) + \hat{Q}_i(\boldsymbol{E}_i(k+1), \hat{\boldsymbol{u}}_i(k+1)) \tag{2.65}$$

where $R_i(\boldsymbol{.})$ is the objective cost (2.34). Similarly, the target of the approximated optimal policy $\hat{\boldsymbol{u}}_i(k)$, which is applied to update the actor weights [50], is defined as

$$\tilde{\boldsymbol{u}}_i = -\left[\left(\boldsymbol{W}_{\hat{\boldsymbol{u}}_i \hat{\boldsymbol{u}}_i}\right)^{-1} \boldsymbol{W}_{\hat{\boldsymbol{u}}_i \boldsymbol{E}_i} \right] \boldsymbol{E}_i(k) \tag{2.66}$$

In Figure 2.19, the adaptive critics approximate the control $\hat{\boldsymbol{u}}_i$ to drive agents ($\forall i$) to track the state of the reference system $\boldsymbol{x}_\ell$.

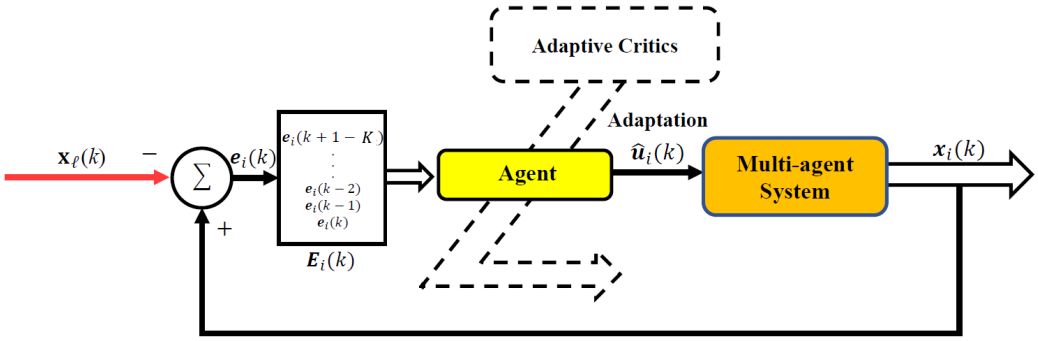

Figure 2.19: Adaptive Critics Diagram

2.6.2 Update Rules for Adaptive Critics

Adaptive critics provide a method to construct a controller that can adapt online to the unknown system dynamics, depending on minimization of the objective cost [8] [68]. A gradient descent approach [82], in this thesis, is used as the update law

for the neural network approximation. In Figure 2.20, the update process of the adaptive critics is illustrated.

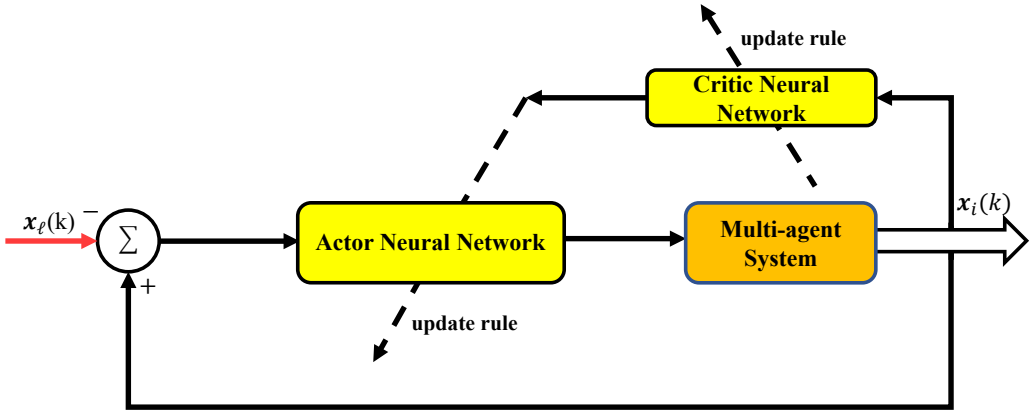

Figure 2.20: Update Process for Adaptive Critics

Update Rule for Critic Neural Network

The approximation error δ_i^{critic} of the solving value function can be computed by

$$\delta_i^{\text{critic}} = \hat{Q}_i(\boldsymbol{E}_i(k), \hat{\boldsymbol{u}}_i(k)) - \tilde{Q}_i \tag{2.67}$$

where $\hat{Q}_i$ is the approximated value (2.62) and $\tilde{Q}_i$ is the target values (2.65). Define the least-squares error of the critic network $\varepsilon_i^{\text{critic}}$ as

$$\varepsilon_i^{\text{critic}} = \frac{1}{2}(\delta_i^{\text{critic}})^2 = \frac{1}{2}\left(\hat{Q}_i(\boldsymbol{E}_i(k), \hat{\boldsymbol{u}}_i(k)) - \tilde{Q}_i\right)^2 \tag{2.68}$$

Differentiating $\varepsilon_i^{\text{critic}}$ in terms of $\boldsymbol{W}_i$, we get

$$\begin{aligned} \frac{\partial \varepsilon_i^{\text{critic}}}{\partial \boldsymbol{W}_i} &= \frac{\partial \varepsilon_i^{\text{critic}}}{\partial \delta_i^{\text{critic}}} \frac{\partial \delta_i^{\text{critic}}}{\partial \boldsymbol{W}_i} \\ &= \left(\hat{Q}_i(\boldsymbol{E}_i(k), \hat{\boldsymbol{u}}_i(k)) - \tilde{Q}_i\right) \begin{bmatrix} \boldsymbol{E}_i(k) \\ \hat{\boldsymbol{u}}_i(k) \end{bmatrix} \begin{bmatrix} \boldsymbol{E}_i^T(k) & \hat{\boldsymbol{u}}_i^T(k) \end{bmatrix} \end{aligned} \tag{2.69}$$

It gives the update rule for the critic weights

$$\begin{aligned} \boldsymbol{W}_i^{[r+1]} &= \boldsymbol{W}_i^{[r]} - \eta_c \left(\frac{\partial \varepsilon_i^{\text{critic}}}{\partial \boldsymbol{W}_i} \right)^{[r]} \\ &= \boldsymbol{W}_i^{[r]} - \eta_c \left(\left(\hat{Q}_i(\boldsymbol{E}_i(k), \hat{\boldsymbol{u}}_i(k)) - \tilde{Q}_i \right) \begin{bmatrix} \boldsymbol{E}_i(k) \\ \hat{\boldsymbol{u}}_i(k) \end{bmatrix} \begin{bmatrix} \boldsymbol{E}_i^T(k) & \hat{\boldsymbol{u}}_i^T(k) \end{bmatrix} \right)^{[r]} \end{aligned} \tag{2.70}$$

where $0 < \eta_c < 1$ is the learning rate of a critic neural network and r refers to the index of the iteration.

The update process of the critic neural network for agent i in a multi-agent system is shown in Figure 2.21. The input, consisting of cumulative error vector $\boldsymbol{E}_i$ and control $\hat{\boldsymbol{u}}_i$, is fed into the critic neural network. After the critic neural network outputs the approximation of the value function $\hat{Q}_i(...)$, the weights $\boldsymbol{W}_i$ will be updated immediately by the rule (2.70).

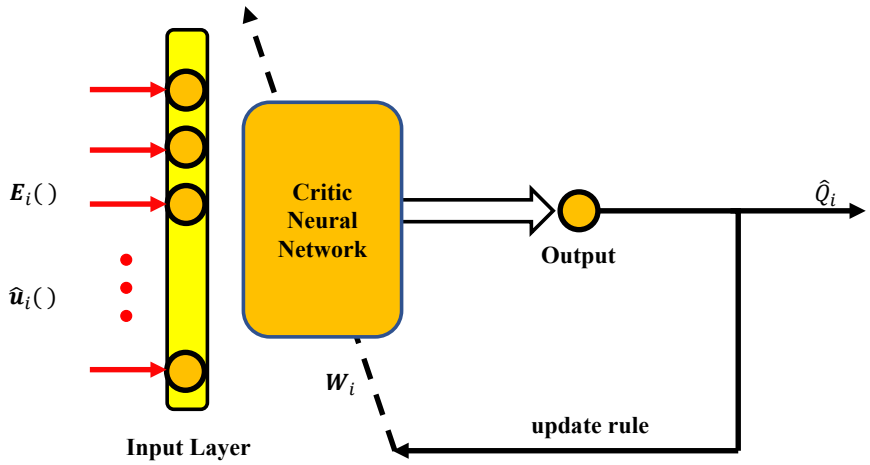

Figure 2.21: Update Process for Critic Neural Network

Update Rule for Actor Neural Network

Similarly, the approximation error vector δ_i^{actor} of the control policy is given by

$$\boldsymbol{\delta}_i^{\text{actor}} = \hat{\boldsymbol{u}}_i(k) - \tilde{\boldsymbol{u}}_i \tag{2.71}$$

where $\tilde{\boldsymbol{u}}_i$ is the target of the approximated policy (2.66). Define the least-squares error of actor network $\varepsilon_i^{\text{actor}}$ as

$$\varepsilon_i^{\text{actor}} = \frac{1}{2}\,\delta_i^{\text{actor}\,T}\delta_i^{\text{actor}} = \frac{1}{2}\,(\hat{\boldsymbol{u}}_i(k) - \tilde{\boldsymbol{u}}_i)^T(\hat{\boldsymbol{u}}_i(k) - \tilde{\boldsymbol{u}}_i) \tag{2.72}$$

Taking partial derivative of $\varepsilon_i^{\text{actor}}$ with respect to $\boldsymbol{\Omega}_i$, we get

$$\begin{aligned}\frac{\partial \varepsilon_i^{\text{actor}}}{\partial \boldsymbol{\Omega}_i} &= \frac{\partial \varepsilon_i^{\text{actor}}}{\partial \boldsymbol{\delta}_i^{\text{actor}}}\frac{\partial \boldsymbol{\delta}_i^{\text{actor}}}{\partial \boldsymbol{\Omega}_i} \\ &= (\hat{\boldsymbol{u}}_i(k) - \tilde{\boldsymbol{u}}_i)\boldsymbol{E}_i^T(k)\end{aligned} \tag{2.73}$$

In a similar fashion, the update law for the actor weights $\boldsymbol{\Omega}_i$ is

$$\begin{aligned}\boldsymbol{\Omega}_i^{[r+1]} &= \boldsymbol{\Omega}_i^{[r]} - \eta_a \left(\frac{\partial \varepsilon_i^{\text{actor}}}{\partial \boldsymbol{\Omega}_i}\right)^{[r]} \\ &= \boldsymbol{\Omega}_i^{[r]} - \eta_a \left((\hat{\boldsymbol{u}}_i(k) - \tilde{\boldsymbol{u}}_i)\boldsymbol{E}_i^T(k)\right)^{[r]}\end{aligned} \tag{2.74}$$

where $0 < \eta_a < 1$ is an actor neural network learning rate and r refers to the index of the iteration.

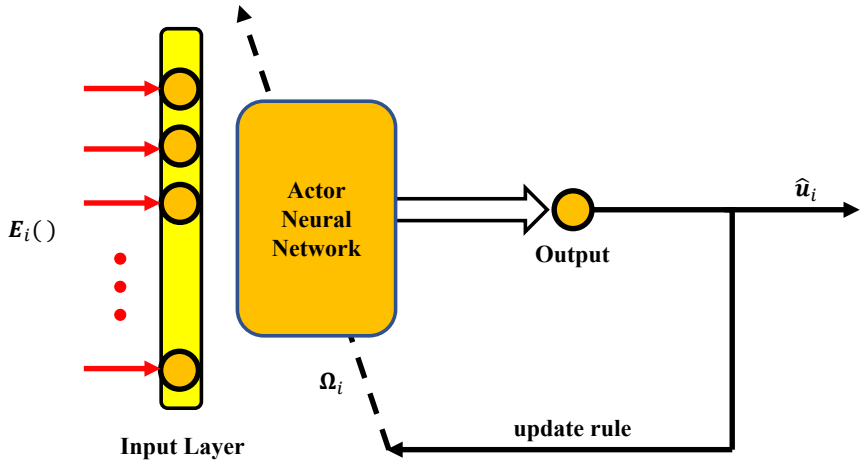

Figure 2.22: Update Process for Actor Neural Network

Figure 2.22 illustrates a update process of the actor neural network for agent i in a multi-agent system. The control policy $\hat{\boldsymbol{u}}_i$ is the output of the actor neural network, and the weights $\boldsymbol{\Omega}_i$ are autonomously tuned via the associated update rule in (2.74). In conclusion, the critic and actor weights are regulated online throughout a VI, shown in Algorithm 1 where the superscript $[r]$ is the iteration index.

Algorithm 1 Adaptive Critics Implementation

Initialization:

Weights $\boldsymbol{W}_i^{[0]}$ and $\boldsymbol{\Omega}_i^{[0]}$
Initial cumulative number K
Recent error record $\boldsymbol{e}_i{}^T(0), \boldsymbol{e}_i{}^T(-1), ..., \boldsymbol{e}_i{}^T(1-K)$
Maximum number of learning iterations N_T

1: **for** $r = 0$ to N_T **do**
2: Obtain $\boldsymbol{E}_i^{[r]}(k)$ and calculate approximation $\hat{\boldsymbol{u}}_i^{[r]}(k)$.
3: Compute the objective cost $R_i(\boldsymbol{E}_i^{[r]}(k), \hat{\boldsymbol{u}}_i^{[r]}(k))$ and update $\boldsymbol{E}_i^{[r]}(k+1)$.
4: Find the approximatie policy $\hat{\boldsymbol{u}}_i^{[r]}(k+1)$ and calculate the approximatie value $\hat{Q}_i^{[r]}(\boldsymbol{E}_i^{[r]}(k+1), \hat{\boldsymbol{u}}_i^{[r]}(k+1))$.
5: Calculate the targets $\tilde{Q}_i^{[r]}$ and $\tilde{\boldsymbol{u}}_i^{[r]}$.
6: Apply (2.70) to update the critic weights $\boldsymbol{W}_i^{[r+1]}$.
7: Apply (2.74) to update the actor weights $\boldsymbol{\Omega}_i^{[r+1]}$.
8: Terminate upon convergence of $\|\hat{Q}_i^{[r+1]}(...) - \hat{Q}_i^{[r]}(...)\|$.
9: **end for**

2.6.3 Adaptive Flocking Control

Recall that the flocking control drives a class of agents with dynamics (2.26) to track a leader system in (2.27) cooperatively, while at the same time, agents are also required to follow some other local interaction rules [21] [33]. Adaptive critics offer an online learning method for this class of systems to be able to operate adaptively and autonomously [6] [7]. Hence, an adaptive flocking control can be given by

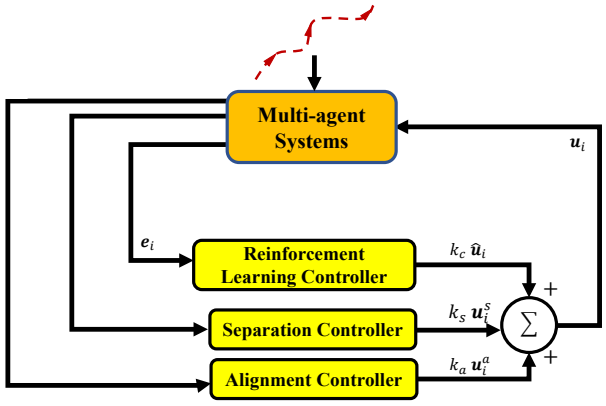

Figure 2.23: Adaptive Flocking Control

$$\boldsymbol{u}_i = k_s \boldsymbol{u}_i^s + k_a \boldsymbol{u}_i^a + k_c \hat{\boldsymbol{u}}_i \tag{2.75}$$

where $k_s, k_a, k_c \in \mathbb{R}_{>0}$. $\boldsymbol{u}_i^s$ and $\boldsymbol{u}_i^a$ are for the separation and velocity consensus, respectively. The last term is the tracking control via adaptive critics. Therefore, $\boldsymbol{u}_i$ drives agents to cope with a class of nonlinear flocking problems in Figure 2.23.

2.7 Summary

This chapter summarizes an overview of the literature on machine learning algorithms and their applications. After discussing the reference-tracking regulation problem, we presented an adaptive control structure to tackle this problem. In the following chapter, we will describe more details of the implementation of the presented control structure using online measurements.

Chapter 3

Optimal Control Formulation

Section 3.1 formulates the online tracking problem for multi-agent systems and presents a combined flocking controller. Section 3.2 provides two case studies of the reference-tracking problems. Several concluding remarks are made in section 3.3.

3.1 Problem Formulation

Recall the discrete dynamics of the multi-agent system in (2.26). We assume that agents operate in a two-dimensional planar area, where the state $\boldsymbol{x}_i(k)$ is measured online and given by

$$\begin{aligned} \boldsymbol{x}_i(k) &= [\, \boldsymbol{q}_i^T(k) \quad \boldsymbol{v}_i^T(k)]^T \\ &= [\, q_{i\{x\}}(k) \;\; q_{i\{y\}}(k) \;\; v_{i\{x\}}(k) \;\; v_{i\{y\}}(k) \,]^T \end{aligned} \tag{3.1}$$

where $q_{i\{x\}}(k)$, $q_{i\{y\}}(k)$ stand for the cartesian positions and $v_{i\{x\}}(k)$, $v_{i\{y\}}(k)$ refer to the cartesian velocities in the coordinates x, y for agent $i \in \{1, 2, ..., N\}$ at discrete time k. We suppose that every agent is able to sense the same state information of the target, and the state information, with respect to the target's dynamics (2.27), is given by

$$\begin{aligned} \boldsymbol{x}_\ell(k) &= [\boldsymbol{q}_\ell^T(k) \quad \boldsymbol{v}_\ell^T(k)]^T \\ &= [\, q_{\ell\{x\}}(k) \;\; q_{\ell\{y\}}(k) \;\; v_{\ell\{x\}}(k) \;\; v_{\ell\{y\}}(k) \,]^T \end{aligned} \tag{3.2}$$

where $q_{\ell\{x\}}(k)$, $q_{\ell\{y\}}(k)$ refer to the positions and $v_{\ell\{x\}}(k)$, $v_{\ell\{y\}}(k)$ refer to the velocities in x, y directions for the moving target at discrete time k. Note that we assume each agent in the group can measure the states of its neighbors only in a

limited communication range, but all agents can measure the direct states of the target. This is a strong assumption to be able to isolate the implementation of the model-free tracking algorithm from the consensus part of the controller. In the literature on consensus applied to motion control of multi-agent systems, tracking and alignment are treated simultaneously, with agents reaching consensus about the estimate of the leader's trajectory through the same underlying communication network topology [33] [34]. Moreover, this assumption for the isolation is required for behaviour-based flocking control that naturally integrates various competing flocking behaviours (i.e., tracking, consensus and separation) [21] [33]. The online tracking error (shown in Figure 3.1) can be expressed by

$$\boldsymbol{e}_i(k) = [e_i^{\Lambda}(k)] = \begin{bmatrix} e_i^1(k) \\ e_i^2(k) \\ e_i^3(k) \\ e_i^4(k) \end{bmatrix} = \begin{bmatrix} q_{i\{x\}}(k) - q_{\ell\{x\}}(k) \\ q_{i\{y\}}(k) - q_{\ell\{y\}}(k) \\ v_{i\{x\}}(k) - v_{\ell\{x\}}(k) \\ v_{i\{y\}}(k) - v_{\ell\{y\}}(k) \end{bmatrix} \tag{3.3}$$

where $\Lambda \in \{1, 2, 3, 4\}$. In order to control this system, the cumulative error vector

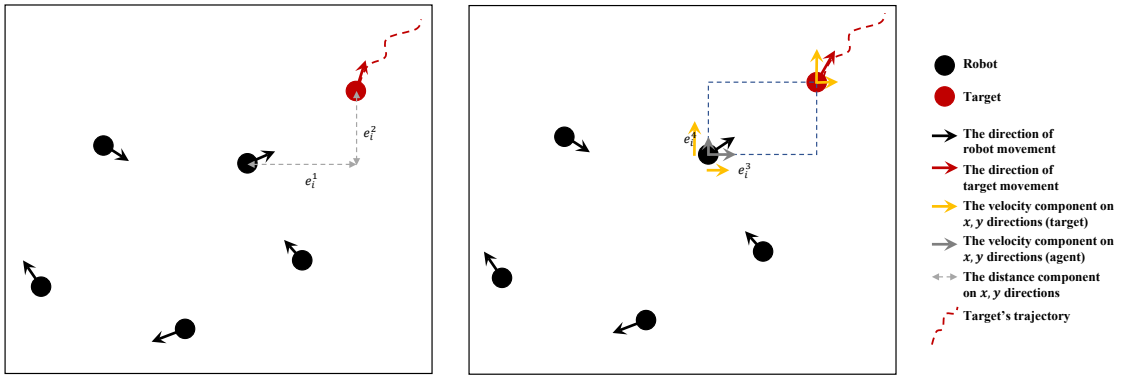

Figure 3.1: Online Measurements of Tracking Error

can be reformulated as:

$$\boldsymbol{E}_i(k) = [E_i^{\Lambda}(k)] = \begin{bmatrix} e_i^1(k), ..., e_i^1(k+1-K) \\ e_i^2(k), ..., e_i^2(k+1-K) \\ e_i^3(k), ..., e_i^3(k+1-K) \\ e_i^4(k), ..., e_i^4(k+1-K) \end{bmatrix} \tag{3.4}$$

where K is the number of terms stored. Noticeably, the model-free learning solutions do not need any information about the system dynamics (i.e., f_ℓ, f_i), where they act as black-box methods [7]. In addition, the methods are implemented on-

line, where only the real-time measurements are collected. Motivated by machine learning literature (reviewed in Chapter 2), an adaptive flocking control integrates adaptive critics (Algorithm 1) and classical flocking control (2.75) to drive agents to learn autonomously to track a moving target:

$$\begin{aligned} \boldsymbol{u}_i(k) &= k_s\, \boldsymbol{u}_i^s(k) + k_a\, \boldsymbol{u}_i^a(k) + k_c\, \hat{\boldsymbol{u}}_i(k) \\ &= -k_s\, \nabla U(\boldsymbol{q}(k)) - k_a\, \nabla \Psi_{\mathcal{G}}(\boldsymbol{v}(k)) + k_c\, \hat{\boldsymbol{u}}_i(k) \end{aligned} \tag{3.5}$$

where $k_s, k_a, k_c \in \mathbb{R}_{>0}$ and $\nabla U(\boldsymbol{.})$, $\nabla \Psi_{\mathcal{G}}(\boldsymbol{.})$ are defined in (2.16) and (2.22), respectively. $\boldsymbol{q}(k)$, $\boldsymbol{v}(k)$ are the global states:

$$\begin{aligned} \boldsymbol{q}(k) &= [\boldsymbol{q}_1^T(k), \boldsymbol{q}_2^T(k), ..., \boldsymbol{q}_N^T(k)]^T \\ \boldsymbol{v}(k) &= [\boldsymbol{v}_1^T(k), \boldsymbol{v}_2^T(k), ..., \boldsymbol{v}_N^T(k)]^T \end{aligned} \tag{3.6}$$

Inspired by the classical Proportional-Derivative flocking control of Olfati-Saber in Ref. [2], the associated postion control terms and velocity control terms can be summed to build the coupled control $\hat{\boldsymbol{u}}_i(k)$, which is equivalently a logic combination of scalar signals $\hat{u}_i^1(k)$, $\hat{u}_i^2(k)$, $\hat{u}_i^3(k)$ and $\hat{u}_i^4(k)$:

$$\begin{aligned} \hat{\boldsymbol{u}}_i(k) &= [\hat{u}_{i\{x\}}(k) \;\; \hat{u}_{i\{y\}}(k)]^T \\ &= [\hat{u}_i^1(k) + \hat{u}_i^3(k) \;\;\; \hat{u}_i^2(k) + \hat{u}_i^4(k)]^T \end{aligned} \tag{3.7}$$

where $\hat{u}_i^1(k), \hat{u}_i^2(k)$ correct the position deviations in x, y directions and $\hat{u}_i^3(k), \hat{u}_i^4(k)$ correct the velocity deviations in x, y directions. To simplify and unify the notations, we define $\hat{u}_i^\Lambda(k)$, $\Lambda \in \{1, 2, 3, 4\}$ (i.e., one for each of the four adaptive learning loops) that can be given by

$$\hat{u}_i^\Lambda(k) = \boldsymbol{\Omega}_i^\Lambda \boldsymbol{E}_i^\Lambda(k) \tag{3.8}$$

where the superscript Λ stands for a control loop of each state, and $\boldsymbol{\Omega}_i^\Lambda$ are the control gains that are determinated by the actor neural networks, which will be updated via the rules in (2.74). Similarly, the weights of the critic neural networks $\boldsymbol{W}_i^\Lambda$ are used to estimate the solving value:

$$\hat{Q}_i^\Lambda(\boldsymbol{E}_i^\Lambda(k), \hat{u}_i^\Lambda(k)) = \frac{1}{2} \begin{bmatrix} \boldsymbol{E}_i^\Lambda(k) \\ \hat{u}_i^\Lambda(k) \end{bmatrix}^T \boldsymbol{W}_i^\Lambda \begin{bmatrix} \boldsymbol{E}_i^\Lambda(k) \\ \hat{u}_i^\Lambda(k) \end{bmatrix} \tag{3.9}$$

Algorithm 2 Adaptive Flocking Control

Initialization:

Weights $\left(\boldsymbol{W}_i^{\Lambda}\right)^{[0]}$ and $\left(\boldsymbol{\Omega}_i^{\Lambda}\right)^{[0]}, \forall \Lambda \in \{1,2,3,4\}$

Global initial states $\boldsymbol{q}(0)$ and $\boldsymbol{v}(0)$

Cumulative error $\left(\boldsymbol{E}_i^{\Lambda}(0)\right)^{[0]}$

Parameters k_s, k_a, k_c

Maximum number of learning iterations N_T

for $r = 0$ to N_T **do**
  for $\Lambda = 1$ to 4 **do**
    Obtain $\left(\boldsymbol{E}_i^{\Lambda}(k)\right)^{[r]}$
    Scalar $\left(\hat{u}_i^{\Lambda}(k)\right)^{[r]} = \left(\boldsymbol{\Omega}_i^{\Lambda}\right)^{[r]} \left(\boldsymbol{E}_i^{\Lambda}(k)\right)^{[r]}$
  end for
  $\left(\hat{\boldsymbol{u}}_i(k)\right)^{[r]} = [\left(\hat{u}_i^1(k)\right)^{[r]} + \left(\hat{u}_i^3(k)\right)^{[r]}, \left(\hat{u}_i^2(k)\right)^{[r]} + \left(\hat{u}_i^4(k)\right)^{[r]}]^T$
  $\left(\boldsymbol{u}_i(k)\right)^{[r]} = -k_s \nabla U(\boldsymbol{q}(k)) - k_a \nabla \Psi_{\mathcal{G}}(\boldsymbol{v}(k)) + k_c \left(\hat{\boldsymbol{u}}_i(k)\right)^{[r]}$
  Drive agents via $\left(\boldsymbol{u}_i(k)\right)^{[r]}$
  for $\Lambda = 1$ to 4 **do**
    Obtain $\left(\hat{Q}_i^{\Lambda}\right)^{[r]}(...)$
    Calculate the targets $\left(\tilde{Q}_i^{\Lambda}\right)^{[r]}$ and $\left(\tilde{u}_i^{\Lambda}\right)^{[r]}$
    $\left(\boldsymbol{W}_i^{\Lambda}\right)^{[r]} \rightarrow \left(\boldsymbol{W}_i^{\Lambda}\right)^{[r+1]}$
    $\left(\boldsymbol{\Omega}_i^{\Lambda}\right)^{[r]} \rightarrow \left(\boldsymbol{\Omega}_i^{\Lambda}\right)^{[r+1]}$
  end for
end for

where the weights $\boldsymbol{W}_i^{\Lambda}$ can be updated via the rules in (2.70). The adaptive control process is detailed out in Algorithm 2, where the notation r refers to iteration index.

In the next section, a set of computer simulations will be conducted to illustrate the efficiency of the presented algorithm in react to various reference-tracking scenarios.

3.2 Simulation Results and Analysis

We set up the parameters for simulations in section 3.2.1. Section 3.2.2 shows the scenario for tracking a moving target with a constant velocity. Section 3.2.3 presents another scenario for tracking a reference trajectory with a time-varying velocity.

3.2.1 Simulation Setup

The simulations are performed in Python 3, with parameters in Table 3.1. Proper initialization requires engineers' prior experience. We tested a lot of possible values of the parameters by guessing, experience, or the analysis of previously evaluated results in Ref. [2] [6] [31].

Table 3.1: Parameters for Simulations

Parameters	Values	Equation
N	5	(2.4)
K	3	(3.4)
a	5.0	(2.13)
b	5.0	(2.13)
h	0.2	(2.9)
r	0.75	(2.7)
η_a	0.5	(2.74)
η_c	0.001	(2.70)

The cumulative number K of stored errors is set as 3, which leads to $\boldsymbol{E}_i^{\Lambda}(k) = [e_i^{\Lambda}(k)\ e_i^{\Lambda}(k-1)\ e_i^{\Lambda}(k-2)]^T$, $\Lambda \in \{1,2,3,4\}$ (i.e., one for each of the four adaptive learning control loops). Same as the dynamics used in [26] [27], discretizing the system function with the sampling period 0.01 s leads to

$$\begin{aligned} q_{i\{x\}}(k+1) &= q_{i\{x\}}(k) + 0.01\ v_{i\{x\}}(k) \\ q_{i\{y\}}(k+1) &= q_{i\{y\}}(k) + 0.01\ v_{i\{y\}}(k) \\ v_{i\{x\}}(k+1) &= v_{i\{x\}}(k) + 0.01\ u_{i\{x\}}(k) \\ v_{i\{y\}}(k+1) &= v_{i\{y\}}(k) + 0.01\ u_{i\{y\}}(k) \end{aligned} \tag{3.10}$$

where $u_{i\{x\}}(k)$, $u_{i\{y\}}(k)$ are the control signals (i.e. accelerations) in x, y directions. And the control signals drive agents to flock together and track the reference trajectory cooperatively.

The actor and critic learning rates can be fixed as $\eta_a = 0.5$ and $\eta_c = 0.001$, respectively. The critic learning rate η_c is set as a notably small parameter to compensate for the relative big initial tracking errors [7] [83]. Inspired by Ref. [7], $\boldsymbol{S}_i$, $\boldsymbol{Z}_i$ are set as

$$\boldsymbol{S}_i = \boldsymbol{I}_{3\times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{Z}_i = 1 \tag{3.11}$$

According to the initializations for the associated weights of adaptive critics in Ref. [7], the initial positive-definite critic weights $\boldsymbol{W}_i^{\Lambda}$ for the control loops Λ are initialized to

$$\boldsymbol{W}_i^{\Lambda} = \begin{bmatrix} 0.09 & 0.03 & -0.06 & 0.04 \\ 0.04 & 0.11 & -0.03 & 0.01 \\ -0.06 & -0.03 & 0.13 & 0.014 \\ 0.04 & 0.01 & 0.01 & 0.045 \end{bmatrix} \tag{3.12}$$

for all $i \in \{1,2,3,4,5\}$ and similarly the initial actor weights $\boldsymbol{\Omega}_i^{\Lambda}$ [7] for the control loops Λ are given by

$$\begin{aligned} \boldsymbol{\Omega}_i^1 &= \begin{bmatrix} 0.833 & 1.134 & 0.95 \end{bmatrix} \\ \boldsymbol{\Omega}_i^2 &= \begin{bmatrix} 0.833 & 1.134 & 0.635 \end{bmatrix} \\ \boldsymbol{\Omega}_i^3 &= \begin{bmatrix} 1.33 & 1.34 & 1.35 \end{bmatrix} \\ \boldsymbol{\Omega}_i^4 &= \begin{bmatrix} 1.33 & 1.34 & 1.35 \end{bmatrix} \end{aligned} \tag{3.13}$$

for all $i \in \{1,2,3,4,5\}$.

In addition, five agents ($N = 5$) are placed at the initial positions:

$$\begin{aligned} \boldsymbol{q}_1(0) &= \begin{bmatrix} -1.0 & 0.0 \end{bmatrix}^T [m] \\ \boldsymbol{q}_2(0) &= \begin{bmatrix} 0.5 & -0.5 \end{bmatrix}^T [m] \\ \boldsymbol{q}_3(0) &= \begin{bmatrix} -0.707 & -0.707 \end{bmatrix}^T [m] \\ \boldsymbol{q}_4(0) &= \begin{bmatrix} 0.0 & 1.0 \end{bmatrix}^T [m] \\ \boldsymbol{q}_5(0) &= \begin{bmatrix} 1.0 & 1.2 \end{bmatrix}^T [m] \end{aligned} \tag{3.14}$$

with the same initial velocities $[1.0 \;\; 1.0]^T [m/s]$. Note that the initialization of the state values can be set randomly, but for the case when the distances between each agent and its neighbors are too small, we must set the critic learning rate η_c to a relatively large parameter in order to compensate for the excessively small initial tracking errors.

Two scenarios will be considered:

- Agents are required to track a moving target with a constant velocity;
- Agents are required to track a moving target with a time-varying velocity, instantaneously following the local interaction rules.

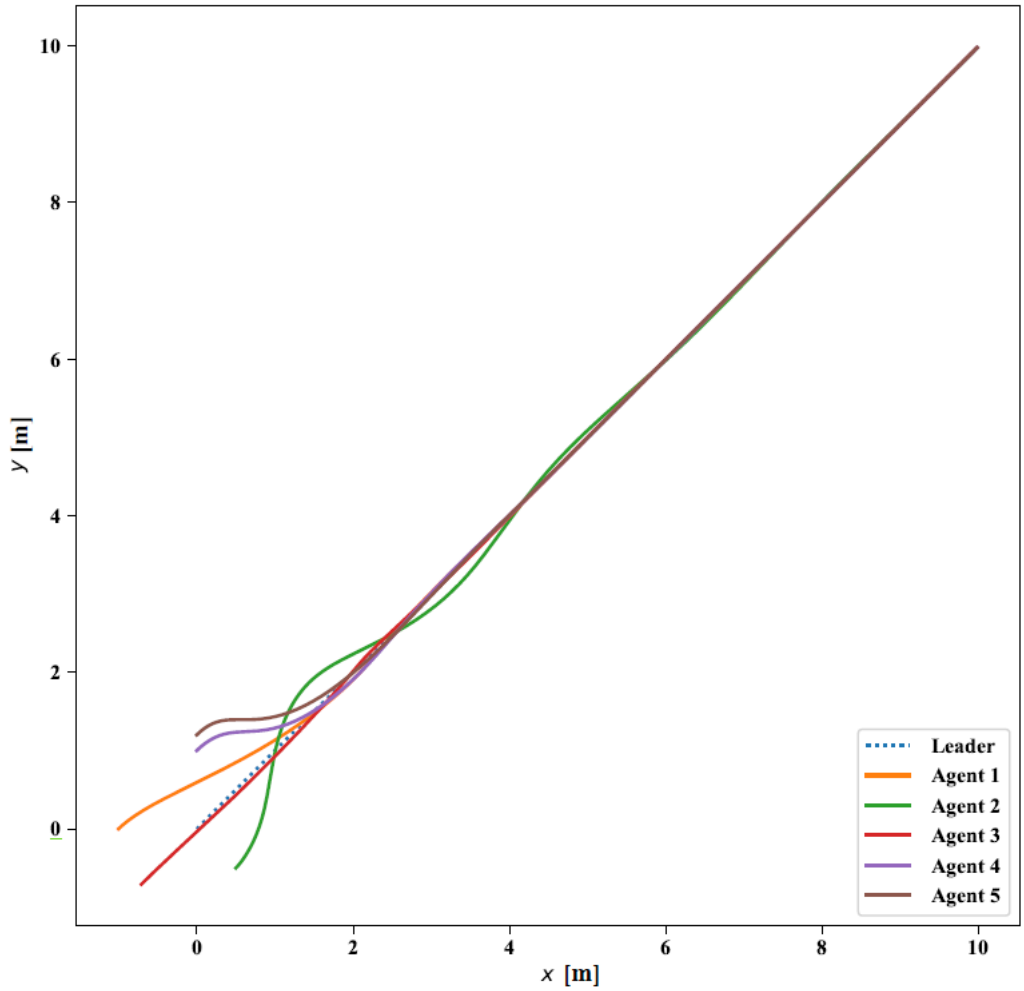

Figure 3.2: Tracking of Constant Velocity Target (With Only Tracking Component)

3.2.2 Reference Trajectory with a Constant Velocity

In this scenario, the multi-agent system is required to track a linear trajectory starting at an initial position $\boldsymbol{q}_\ell(0) = [0 \;\; 0]^T[m]$:

$$\begin{aligned} q_{\ell\{x\}}(k) &= 0.01\, k \cdot v_{\ell\{x\}}(k) \\ q_{\ell\{y\}}(k) &= 0.01\, k \cdot v_{\ell\{y\}}(k) \end{aligned} \tag{3.15}$$

with a constant velocity $v_{\ell\{x\}}(k) = v_{\ell\{y\}}(k) = 1\ [m/s]$. This scenario will be simulated to analyze the tracking performance of adaptive critics, where only the tracking component in (3.5) is active. Figure 3.2 shows that the agents track the desired trajectory. In addition, we extract the trajectories into x and y directions in Figure 3.3 with iteration step k from 0 to 1000.

Tracking errors e_i^Λ, $\Lambda \in \{1,2,3,4\}$ can be used to analyze the adaptive con-

troller. The tracking errors consist of position tracking errors e_i^1, e_i^2 and velocity tracking errors e_i^3, e_i^4, which is controlled via the control loops. In Figure 3.4, the errors e_i^1, e_i^2 between target's postion and each agent's, are shown in (a) and (b), where (a) is in x direction and (b) is in y direction. Secondly, the velocity tracking errors e_i^3, e_i^4 in the associated directions, are shown in (c) and (d) respectively. Consequently, the controller stabilizes the tracking error.

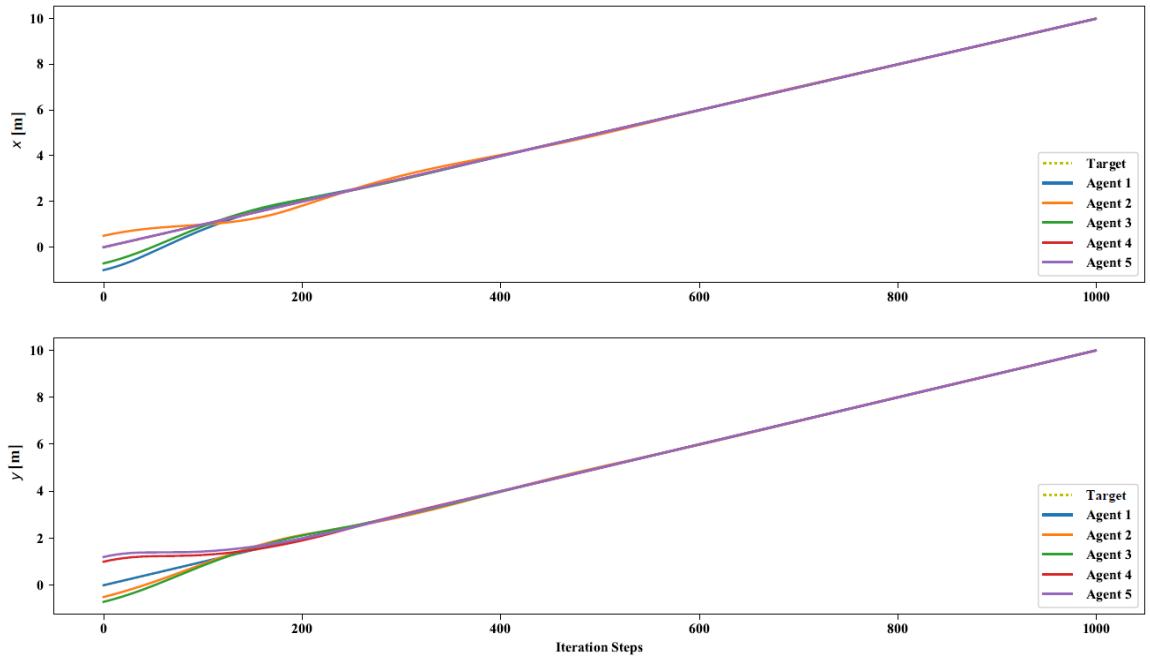

Figure 3.3: Tracking of Constant Velocity Target in x, y Directions

In order to illustrate how adaptive critics work to achieve the control objectives, we take an agent as an example. It is clear that critic weights converge in approximately 700 iterations quickly, shown in figure 3.6. Otherwise, actor weights are adaptively updated to associated control gains in Figure 3.5.

Table 3.2: Control Gains for Agent 2

$\mathbf{\Omega}_2^\Lambda$	Control Gains
$\mathbf{\Omega}_2^1$	$\begin{bmatrix} 0.7989 & 1.0787 & 0.9151 \end{bmatrix}$
$\mathbf{\Omega}_2^2$	$\begin{bmatrix} 0.8089 & 1.1158 & 0.6105 \end{bmatrix}$
$\mathbf{\Omega}_2^3$	$\begin{bmatrix} -0.3689 & -0.1775 & -0.2138 \end{bmatrix}$
$\mathbf{\Omega}_2^4$	$\begin{bmatrix} -0.3908 & 0.5345 & -0.2674 \end{bmatrix}$

Therefore, the agents' states can synchronize to the leader's. In Figure 3.7, the states $q_{2\{x\}}, v_{2\{x\}}$ are controlled to track the desired trajectory. Similarly in Fig-

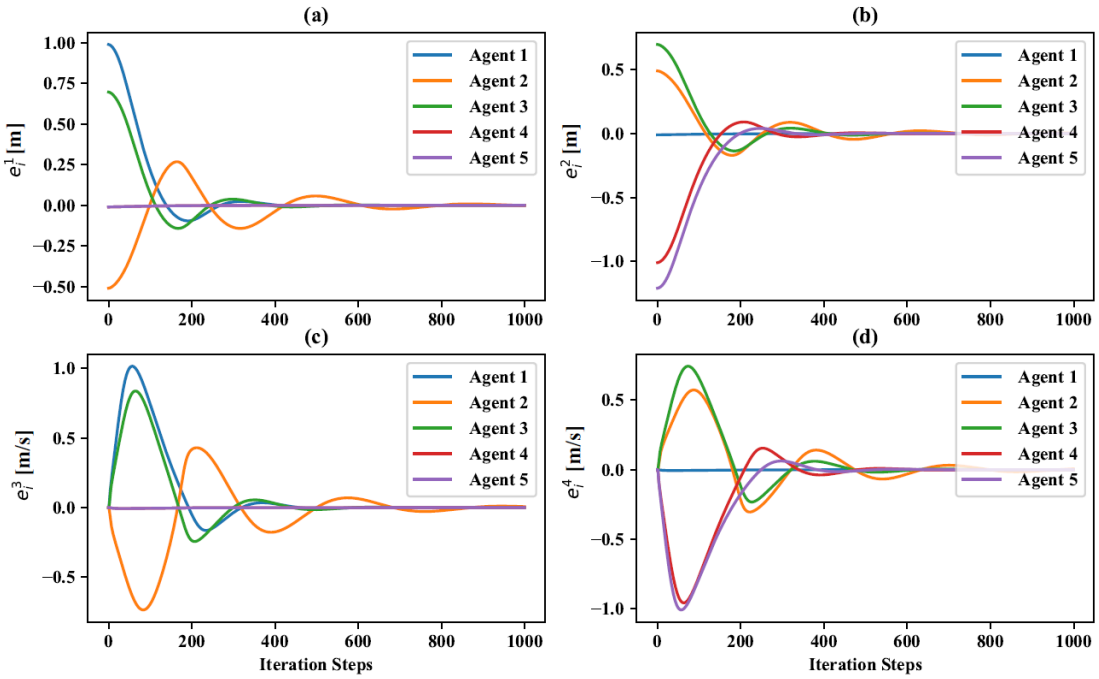

Figure 3.4: Controller's Performance in x and y Directions

ure 3.8, $q_{2\{y\}}, v_{2\{y\}}$ are controlled over the duration. The final control gains for agent 2 given by the proposed algorithm are listed in Table 3.2, which generate the control signals to drive the agent in x and y directions.

This scenario can be applicable for mobile robotic systems with physical constraints. For example, sea vessels [7] often have limited capabilities for their velocities (e.g. surge velocity). A vessel will follow a constant velocity when the leader's velocity exceeds the maximum setting of the vessel.

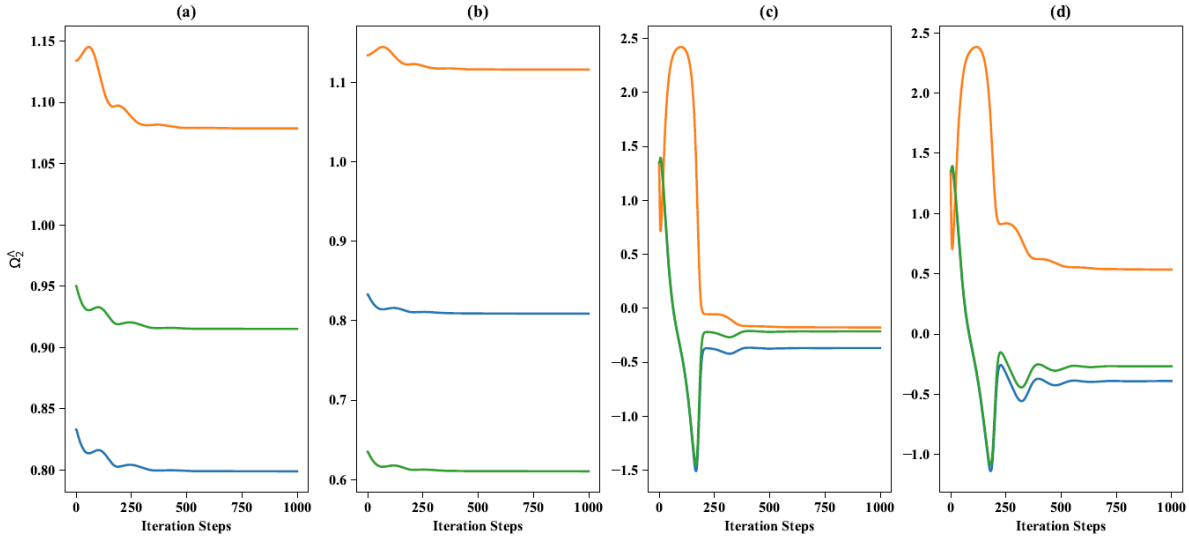

Figure 3.5: Adaptation of Actor Weights (a) $\Lambda = 1$; (b) $\Lambda = 2$; (c) $\Lambda = 3$; (d) $\Lambda = 4$.

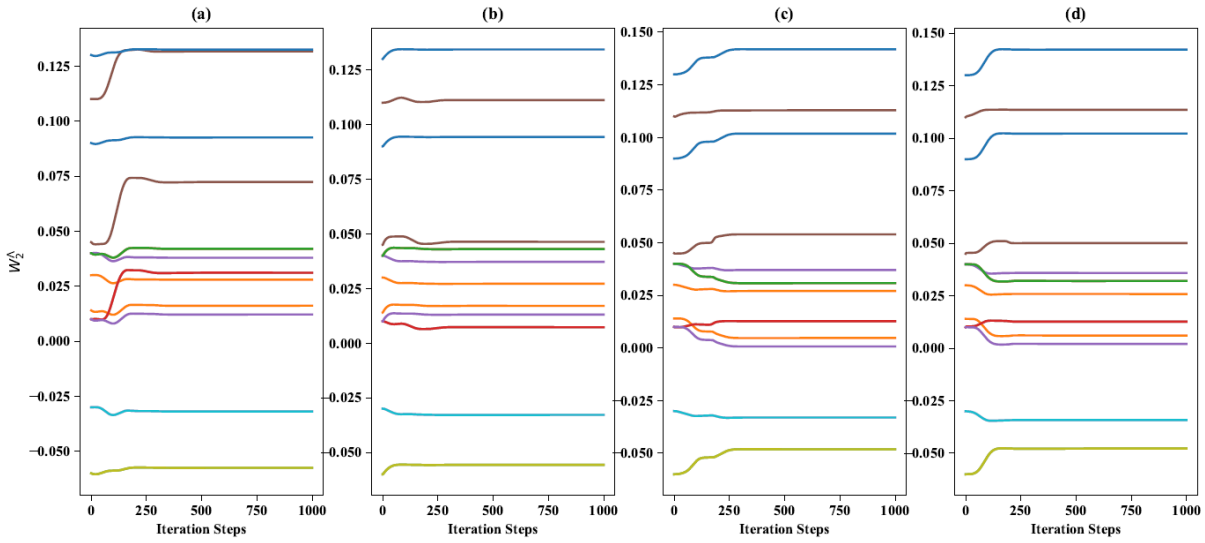

Figure 3.6: Adaptation of Critic Weights (a) $\Lambda = 1$; (b) $\Lambda = 2$; (c) $\Lambda = 3$; (d) $\Lambda = 4$.

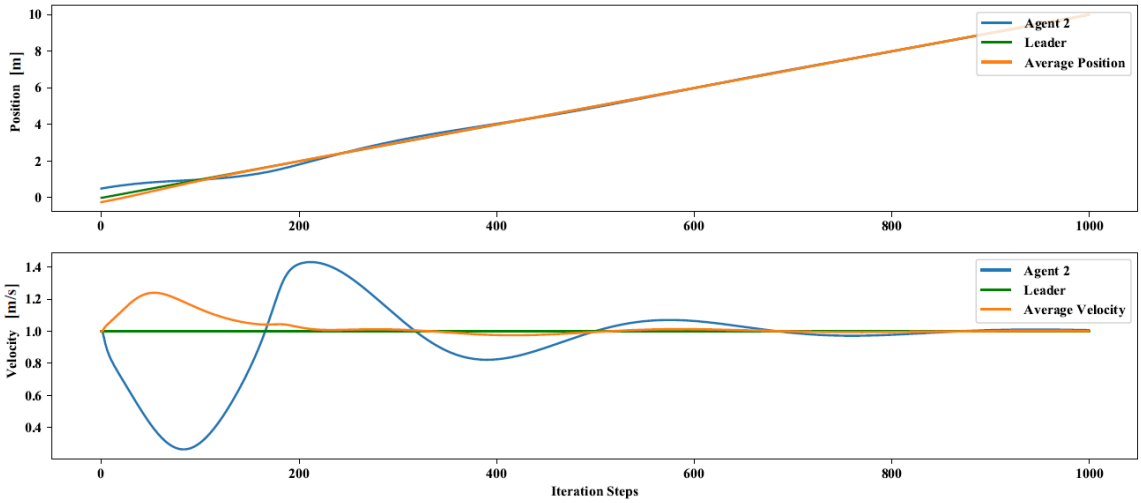

Figure 3.7: Changing States in x Direction

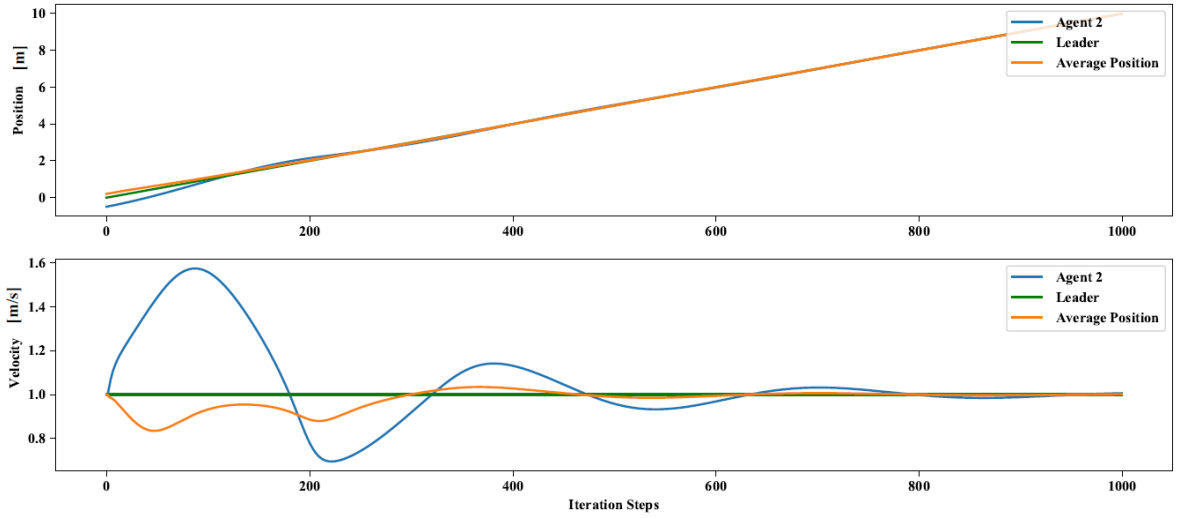

Figure 3.8: Changing States in y Direction

3.2.3 Reference Trajectory with a Time-varying Velocity

In this set of simulations, the multi-agent system is set to track a sinusoidal trajectory defined by

$$\begin{aligned} q_{\ell\{x\}}(k) &= 0.01\, k \cdot v_{\ell\{x\}}(k) \\ q_{\ell\{y\}}(k) &= 0.4 \sin(0.01 \cdot 0.3\, k) \end{aligned} \tag{3.16}$$

starting form $q_{\ell\{x\}}(0) = q_{\ell\{y\}}(0) = 0\ [m]$ and velocity measurements are represented by

$$\begin{aligned} v_{\ell\{x\}}(k) &= 1 \\ v_{\ell\{y\}}(k) &= 0.12 \cos(0.01 \cdot 0.3\, k) \end{aligned} \tag{3.17}$$

In Figure 3.9, the agents are driven to track the desired trajectory with some local interaction rules (i.e. separation, alignment and consensus), with iteration step k from 0 to 10000. We extract the trajectories into two directions, and the extraction of x direction is shown in Figure 3.10 during the duration of simulation.

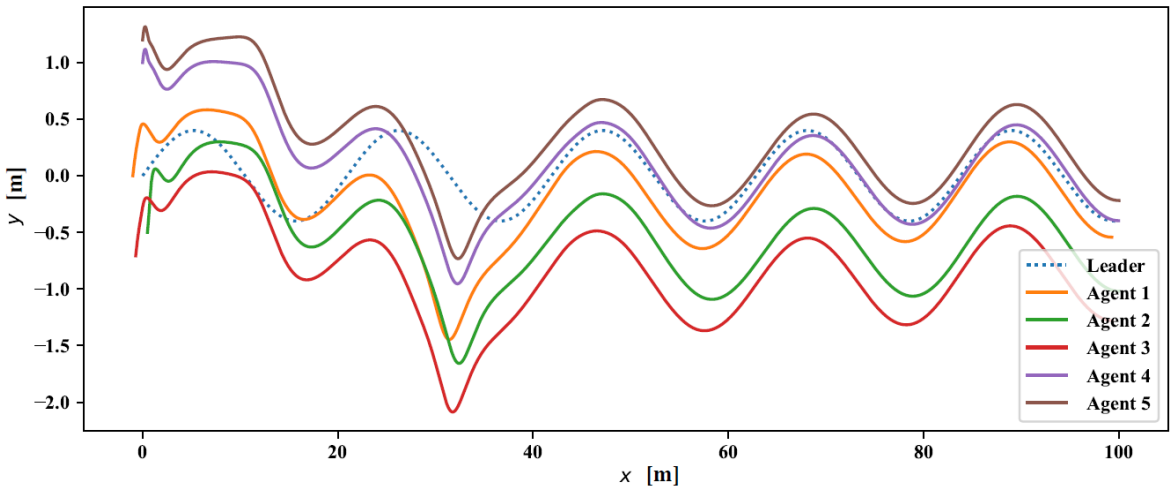

Figure 3.9: Tracking of Time-varying Velocity Target

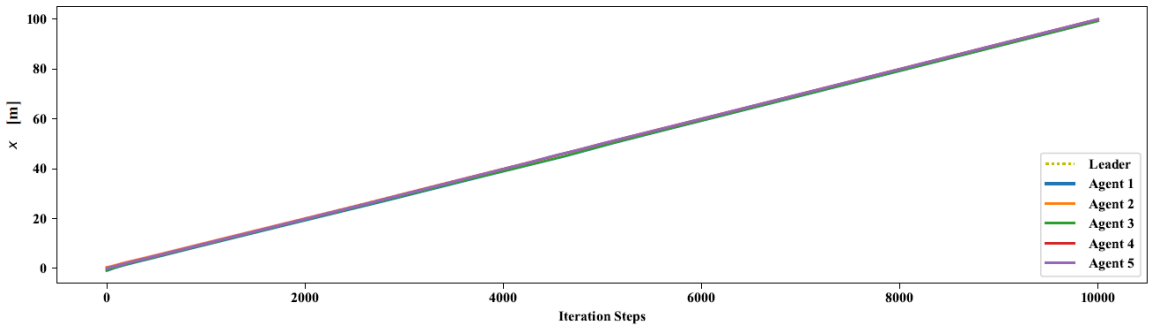

Figure 3.10: Tracking of Changing Velocity Target in x Direction

The adaptive learning process enables the agents to track the time-varying trajectory, and the tracking errors e_i^{Λ} are regulated as illustrated in Figure 3.11. The figures show the capability of the process to adjust to the high maneuverability associated with the reference trajectory.

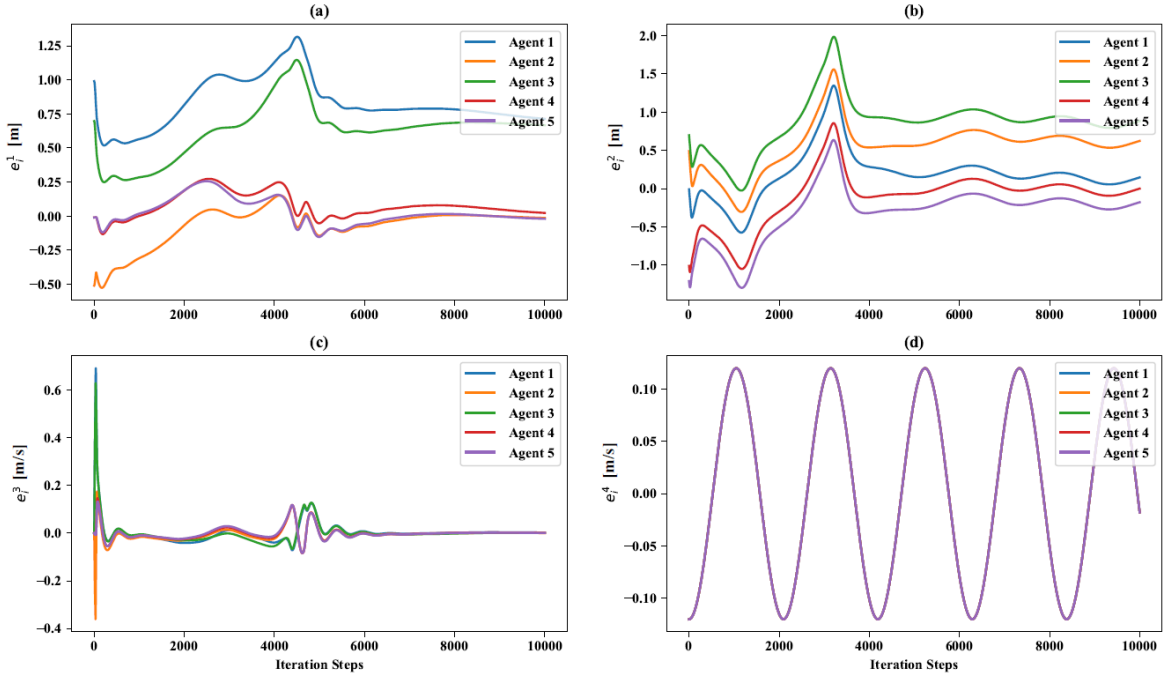

Figure 3.11: Controller's Performance in x and y Directions

Hence, each agent adaptively follows the moving target with time-varying velocity, and tries to maintain a certain formation via locally interacting with its neighbours. In addition, we use the centroid of the group to evaluate the per-

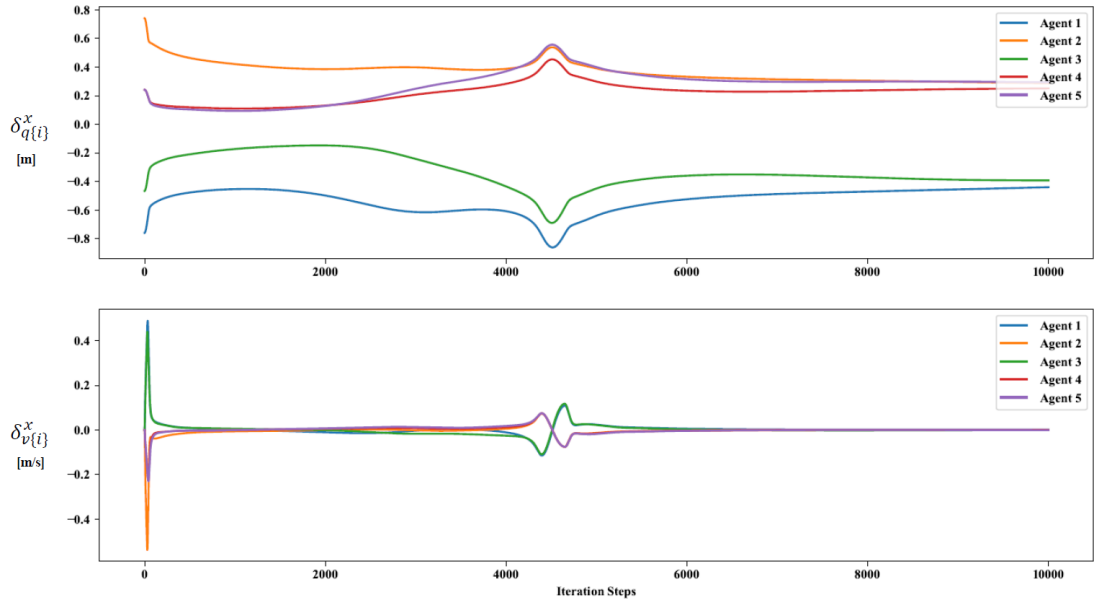

Figure 3.12: Consensus Performance in x Direction

formance of the consensus behaviors in terms of the agent i:

$$\begin{aligned} \boldsymbol{\delta}_{\boldsymbol{q}\{i\}} &= [\delta^x_{\boldsymbol{q}\{i\}} \;\; \delta^y_{\boldsymbol{q}\{i\}}]^T = \boldsymbol{q}_c - \boldsymbol{q}_i \\ \boldsymbol{\delta}_{\boldsymbol{v}\{i\}} &= [\delta^x_{\boldsymbol{v}\{i\}} \;\; \delta^y_{\boldsymbol{v}\{i\}}]^T = \boldsymbol{v}_c - \boldsymbol{v}_i \end{aligned} \tag{3.18}$$

for all $i \in \{1, 2, 3, 4, 5\}$, where $\boldsymbol{q}_c, \boldsymbol{v}_c$ are the centralized information defined in (2.6). In Figure 3.12 and Figure 3.13, the consensus performances are shown.

Figure 3.14 shows the update of the critic weights for agent 2. It is observed that the critic weights take more time to converge compared to the earlier scenario, which is due to the complexity of the independent trajectory of the leader. In similar fashion, the changing of the adaptive weights of the actor networks is shown in Figure 3.15, as imposed by the value iteration structure.

In Figure 3.16, the position of agent 2 in x direction is controlled to track the desired trajectory. After a slight fluctuation period, the velocity of agent 2 converges to the steady-state value. Similarly in Figure 3.17, the associated states in y direction are controlled.

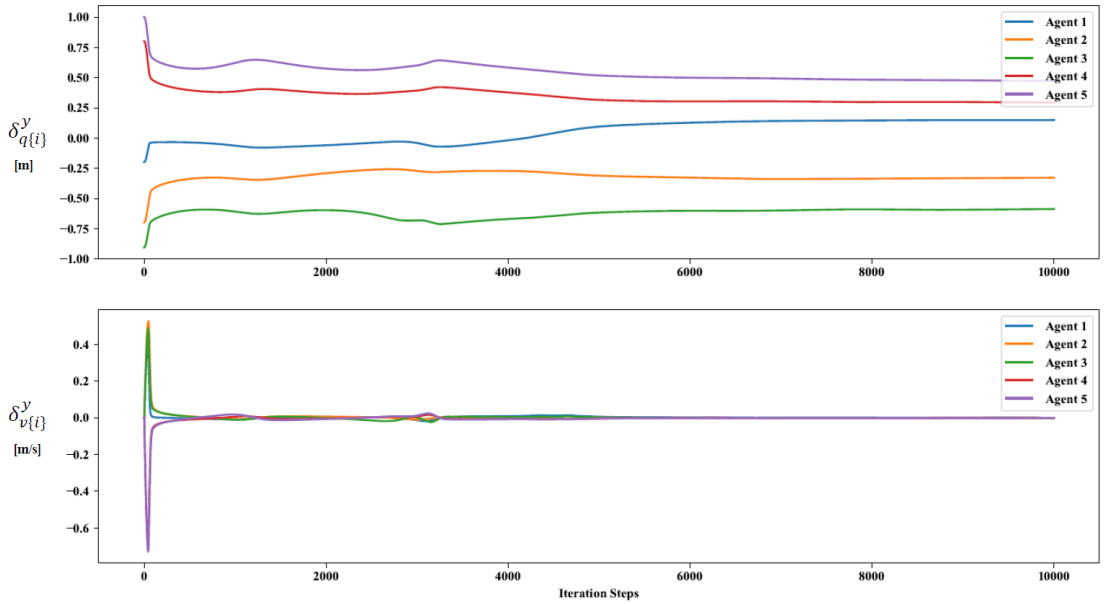

Figure 3.13: Consensus Performance in y Direction

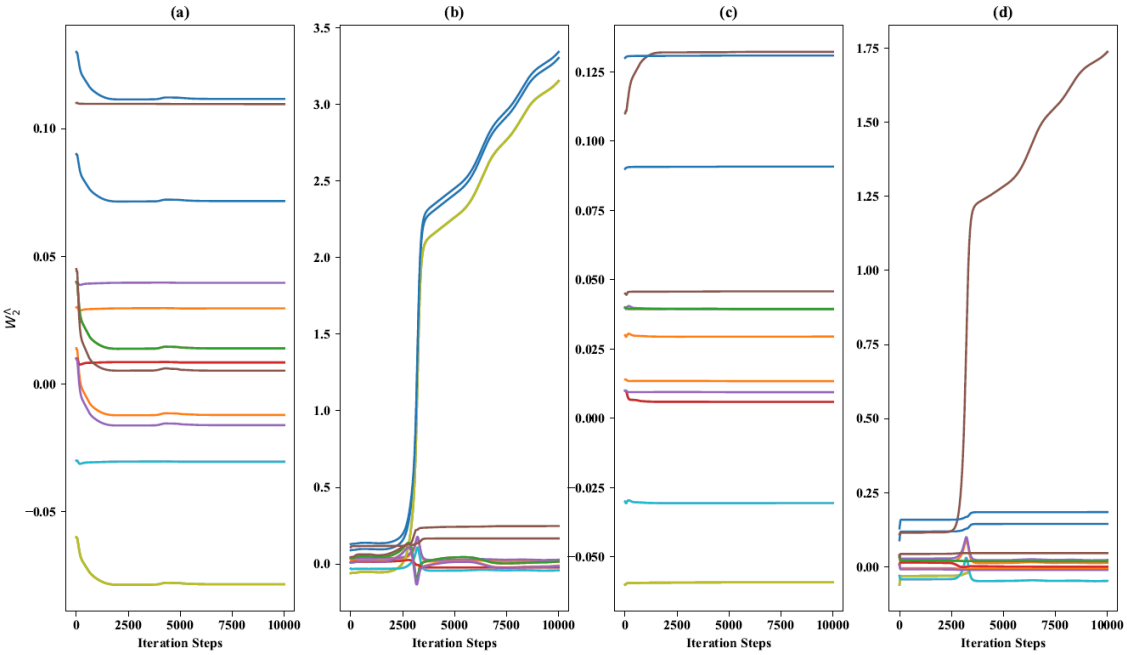

Figure 3.14: Adaptation of Critic Weights (a) $\Lambda = 1$; (b) $\Lambda = 2$; (c) $\Lambda = 3$; (d) $\Lambda = 4$.

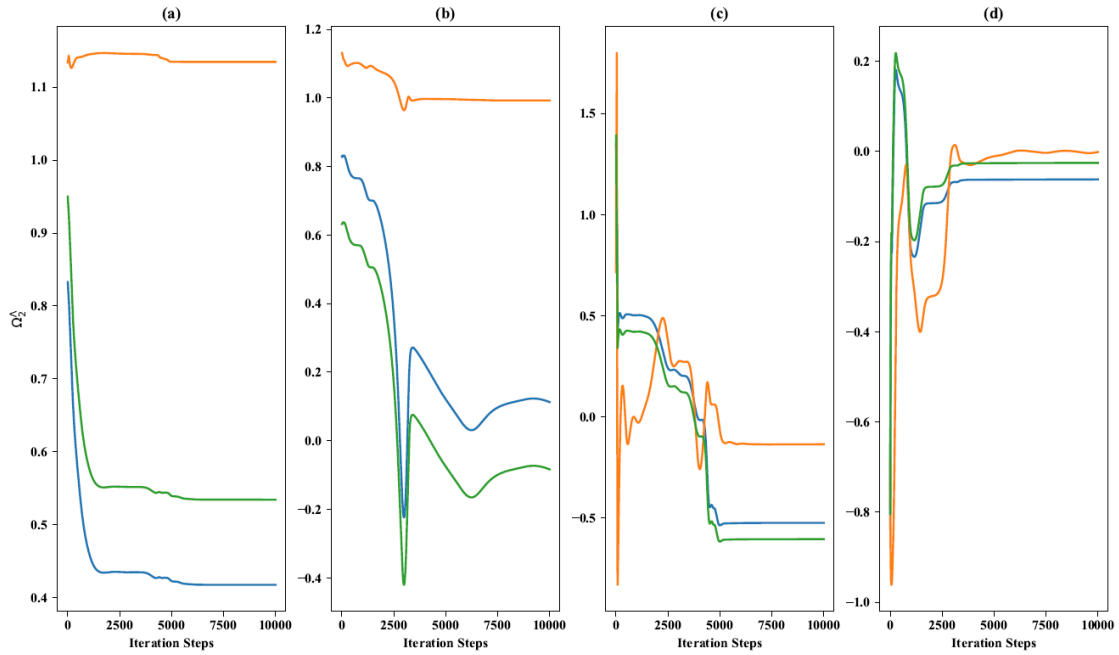

Figure 3.15: Adaptation of Actor Weights (a) $\Lambda = 1$; (b) $\Lambda = 2$; (c) $\Lambda = 3$; (d) $\Lambda = 4$.

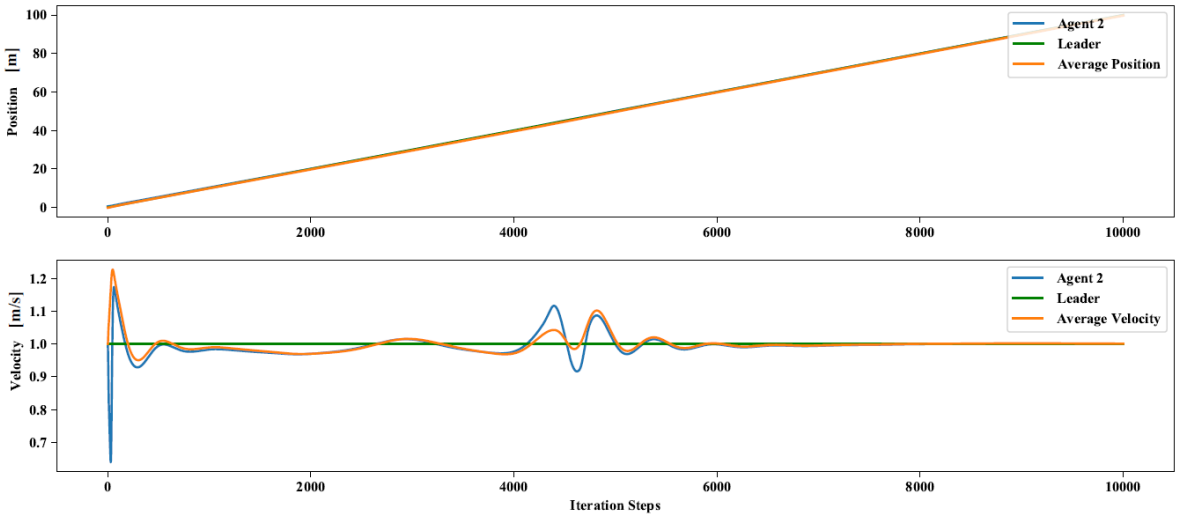

Figure 3.16: Changing States in x Direction

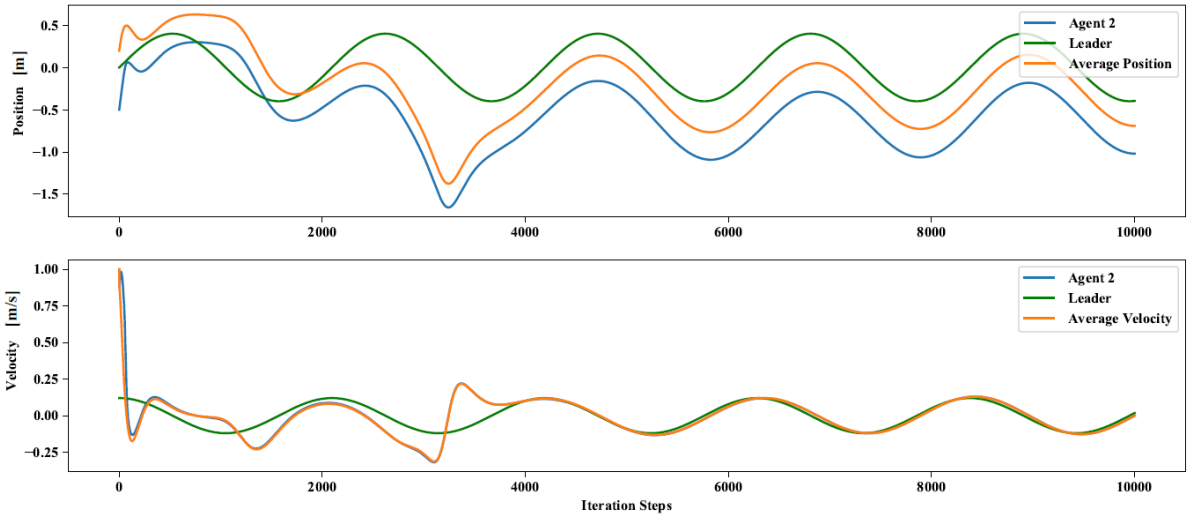

Figure 3.17: Changing States in y Direction

Herein, we gave the comparison of the control signals of the presented method corresponding to a Proportional-Derivative (PD) tracking control used in Ref. [2], in order to show the performance of the presented method with respect to decreasing the control effort of the system. In Figure 3.18, the consensus term $\boldsymbol{u}_i^a(k)$ in (3.5) can be represented into x and y directions, i.e. the extracted scalars $u_{i\{x\}}^a$ and $u_{i\{y\}}^a$ for agent i (e.g. the subscript $\{x\}$ means the associated term in x direction). Compared with the case with the constant weights in Ref. [2] with the same initial conditions and parameters (Note that the word constant means that the associated weights are unchanged for the adaptive critics during the simulation), the proposed law with the adaptive weights make $u_{2\{x\}}^a$ and $u_{2\{y\}}^a$ converge at roughly 8000 iterations.

The control effort is defined in Ref. [84], as an integral equation with the norm of the control signal. Herein, the definition in Ref. [84] can be mathematically applied to the consensus term $\boldsymbol{u}_i^a(k)$ in (3.5), and is given by

$$O_i(\boldsymbol{u}_i^a(k)) = \sum_{k=0}^{N_T} \|\boldsymbol{u}_i^a(k)\| \tag{3.19}$$

where the maximum number of learning iterations $N_T = 10000$. Hence, $O_2 = 6985.32$ can be easily obtained for the case with the constant weight. But for the case with the adaptive weights, the agent can be driven with a lower effort $O_2 = 3953.78$. In practical engineering, low control effort is highly desirable because this may result in long working-life for actuators.

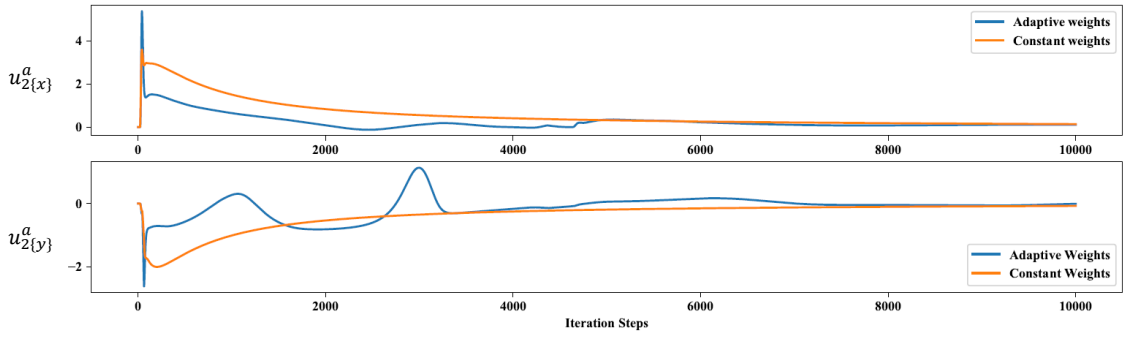

Figure 3.18: Consensus Control Comparison

3.3 Summary

This chapter provides the implementation of the adaptive critics. First of all, we tackle the online tracking problems involving unknown systems' dynamics, using a reinforcement learning framework. This method does not require both all followers' and target's dynamics. Instead, it utilizes online measurements of tracking errors to approximate control strategies, based on neural networks. Additionally, the consensus behaviour is achieved based on graph theory, while the separation behaviour is accomplished through smooth potential functions. Finally, a series of simulations demonstrate that the proposed controller performs effectively in various reference tracking scenarios.

Chapter 4

Summary and Future Work

Technological developments in fields such as autonomous technology, wireless communication, and sensors, allow networked multi-agent systems (MASs) to cooperatively accomplish various missions in a distributed manner [34] [35]. Wireless sensor network is an important application of MASs, where a group of sensing agents can track the reference trajectories cooperatively.

This thesis presents an online reinforcement learning mechanism for tracking with MASs. The solution can be obtained over a value iteration process associated to the Bellman equation [7] [49]. This method does not rely on any conventional error dynamics that generally lead to difficult-to-implement control strategies [48]. It uses online measurements of the system trajectories to make optimal control decisions. Furthermore, it is not necessary to find the solution on identifying the full dynamics of systems, when model-free control methods are used. The adaptive critics are then applied to estimate the optimal strategies and the associated values.

This work is based on several idealizing assumptions:

- Local interactions within agents in the sensor networks are characterized without any communication delay;
- Signals measured by all sensing agents are always accurate and instantaneous;
- All mobile sensing agents are purely kinematic and fully actuated.

Nevertheless, the practical implementation of the presented approach can be much more challenging. Firstly, one of the main hurdles is communication delay. Communication delay is inherent in a mobile sensor network, when sensing

agents constantly motion and synchronously execute various tasks via sharing local information throughout the entire network. Secondly, this research work could also be extended to the three-dimensional (3D) case, which is not considered in this thesis. Thirdly, state estimators based on noisy measurements can be embedded in mobile agents to investigate the coupling with the proposed control law. In addition, there is also the issue of perfect knowledge of the target's trajectory.

In view of big challenges happening in real-world applications, ample opportunities exist to further extend and enhance the proposed approach. Numerous avenues can be taken to further study the current work.

Bibliography

[1] Daniel Grünbaum. "Schooling as a strategy for taxis in a noisy environment". In: *Evolutionary Ecology* 12.5 (1998). Publisher: Springer, pp. 503–522.

[2] Reza Olfati-Saber. "Flocking for multi-agent dynamic systems: Algorithms and theory". In: *IEEE Transactions on automatic control* 51.3 (2006). Publisher: IEEE, pp. 401–420.

[3] Frank Lewis. *Applied optimal control & estimation: Digital design and implementation*. Facsimile edition. Englewood Cliffs, N.J: Prentice Hall, Feb. 1, 1992.

[4] Vincent François-Lavet et al. "An introduction to deep reinforcement learning". In: *arXiv preprint arXiv:1811.12560* (2018).

[5] Anil K. Jain, Jianchang Mao, and K. Moidin Mohiuddin. "Artificial neural networks: A tutorial". In: *Computer* 29.3 (1996). Publisher: IEEE, pp. 31–44.

[6] Mohammed Abouheaf, Wail Gueaieb, and Frank Lewis. "Online model-free reinforcement learning for the automatic control of a flexible wing aircraft". In: *IET Control Theory & Applications* 14.1 (2020). Publisher: Wiley Online Library, pp. 73–84.

[7] Mohammed Abouheaf et al. "Trajectory tracking of underactuated sea vessels with uncertain dynamics: An integral reinforcement learning approach". In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2020, pp. 1866–1871.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] Malik Tubaishat and Sanjay Madria. "Sensor networks: an overview". In: *IEEE potentials* 22.2 (2003). Publisher: IEEE, pp. 20–23.

[10] Deborah Estrin et al. "Connecting the physical world with pervasive networks". In: *IEEE pervasive computing* 1.1 (2002). Publisher: IEEE, pp. 59–69.

[11] Won-Suk Jang and William M. Healy. "Wireless sensor network performance metrics for building applications". In: *Energy and Buildings* 42.6 (2010). Publisher: Elsevier, pp. 862–868.

[12] Lino Marques, André Martins, and Anibal T. de Almeida. "Environmental monitoring with mobile robots". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 3624–3629.

[13] Antonio Guerrero-González et al. "A multirobot platform based on autonomous surface and underwater vehicles with bio-inspired neurocontrollers for long-term oil spills monitoring". In: *Autonomous Robots* 40.7 (Oct. 1, 2016), pp. 1321–1342.

[14] Yucheng Huang et al. "A prototype IOT based wireless sensor network for traffic information monitoring". In: *International journal of pavement research and technology* 11.2 (2018). Publisher: Elsevier, pp. 146–152.

[15] Jiayi Lu et al. "Artificial agent: The fusion of artificial intelligence and a mobile agent for energy-efficient traffic control in wireless sensor networks". In: *Future Generation Computer Systems* 95 (2019). Publisher: Elsevier, pp. 45–51.

[16] Craig Schlenoff et al. "A literature review of sensor ontologies for manufacturing applications". In: *2013 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*. IEEE, 2013, pp. 96–101.

[17] Songhwai Oh et al. "Tracking and coordination of multiple agents using sensor networks: System design, algorithms and experiments". In: *Proceedings of the IEEE* 95.1 (2007). Publisher: IEEE, pp. 234–254.

[18] Amarjeet Singh et al. "Mobile robot sensing for environmental applications". In: *Field and service robotics*. Springer, 2008, pp. 125–135.

[19] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. "Wireless sensor network survey". In: *Computer networks* 52.12 (2008). Publisher: Elsevier, pp. 2292–2330.

[20] Ljiljana eri, Maja tula, and Darko Stipaniev. "Engineering of holonic multi agent intelligent forest fire monitoring system". In: *AI Communications* 26.3 (2013). Publisher: IOS Press, pp. 303–316.

[21] Jongeun Choi, Joonho Lee, and Songhwai Oh. "Biologically-inspired navigation strategies for swarm intelligence using spatial Gaussian processes". In: *IFAC Proceedings Volumes* 41.2 (2008). Publisher: Elsevier, pp. 593–598.

[22] Olivia Salmerón-García et al. "Regionalization of the Gulf of Mexico from space-time chlorophyll-a concentration variability". In: *Ocean Dynamics* 61.4 (Apr. 1, 2011), pp. 439–448.

[23] Petter Ogren, Edward Fiorelli, and Naomi Ehrich Leonard. "Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment". In: *IEEE Transactions on Automatic control* 49.8 (2004). Publisher: IEEE, pp. 1292–1302.

[24] Abhinav Sinha et al. "Consensus-based odor source localization by multi-agent systems". In: *IEEE transactions on cybernetics* 49.12 (2018). Publisher: IEEE, pp. 4450–4459.

[25] Craig W. Reynolds. "Flocks, herds and schools: A distributed behavioral model". In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 1987, pp. 25–34.

[26] Jongeun Choi, Joonho Lee, and Songhwai Oh. "Swarm intelligence for achieving the global maximum using spatio-temporal Gaussian processes". In: *2008 American Control Conference*. IEEE, 2008, pp. 135–140.

[27] Jongeun Choi, Songhwai Oh, and Roberto Horowitz. "Cooperatively learning mobile agents for gradient climbing". In: *2007 46th IEEE Conference on Decision and Control*. IEEE, 2007, pp. 3139–3144.

[28] Miroslav Fiedler. "Algebraic connectivity of graphs". In: *Czechoslovak mathematical journal* 23.2 (1973). Publisher: Institute of Mathematics, Academy of Sciences of the Czech Republic, pp. 298–305.

[29] Ian F. Akyildiz et al. "A survey on sensor networks". In: *IEEE Communications magazine* 40.8 (2002). Publisher: IEEE, pp. 102–114.

[30] Lynne E. Parker. *Heterogeneous multi-robot cooperation*. Section: Technical Reports. Feb. 1, 1994.

[31] Mohammed Abouheaf and Wail Gueaieb. "Flocking motion control for a system of nonholonomic vehicles". In: *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE, 2017, pp. 32–37.

[32] Norman Biggs, Norman Linstead Biggs, and Biggs Norman. *Algebraic graph theory*. 67. Cambridge university press, 1993.

[33] Zachary Young and Hung Manh La. "Consensus, cooperative learning, and flocking for multiagent predator avoidance". In: *International Journal of Advanced Robotic Systems* 17.5 (2020). Publisher: SAGE Publications Sage UK: London, England, pp. 1–19.

[34] Chandreyee Bhowmick et al. "Flocking control of multi-agent system with leader-follower architecture using consensus based estimated flocking center". In: *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2016, pp. 166–171.

[35] Zongyao Wang and Dongbing Gu. "Distributed cohesion control for leader-follower flocking". In: *2007 IEEE International Fuzzy Systems Conference*. IEEE, 2007, pp. 1–6.

[36] Qiang Lu and Qing-Long Han. "Cooperative control of a multi-robot system for odor source localization". In: *IECON 2011-37th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2011, pp. 2359–2364.

[37] Mahdi Jadaliha, Joonho Lee, and Jongeun Choi. "Adaptive control of multiagent systems for finding peaks of uncertain static fields". In: *Journal of Dynamic Systems, Measurement, and Control* 134.5 (2012). Publisher: American Society of Mechanical Engineers Digital Collection, pp. 1–8.

[38] Mohammed I. Abouheaf et al. "Multi-agent discrete-time graphical games and reinforcement learning solutions". In: *Automatica* 50.12 (2014). Publisher: Elsevier, pp. 3038–3053.

[39] Fei Chen and Wei Ren. *Distributed average tracking in multi-agent systems*. Springer International Publishing, 2020.

[40] Karl Johan Astrom and Tore Hagglund. *PID controllers theory design and tuning*. Instrumentation Systems &; 2nd UK ed. edition (Jan. 1 1995), 1995.

[41] Katsuhiko Ogata. *Modern control engineering*. Aug. 25, 2009.

[42] Mohammed I. Abouheaf, Frank L. Lewis, and Magdi S. Mahmoud. "Model-free adaptive learning solutions for discrete-time dynamic graphical games". In: *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 3578–3583.

[43] Draguna Vrabie et al. "Adaptive optimal control for continuous-time linear systems based on policy iteration". In: *Automatica* 45.2 (2009). Publisher: Elsevier, pp. 477–484.

[44] Mohammed Abouheaf and Wail Gueaieb. "Model-free adaptive control approach using integral reinforcement learning". In: *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*. IEEE, 2019, pp. 1–7.

[45] Ning Wang et al. "Model-free optimized tracking control heuristic". In: *Robotics* 9.3 (2020). Publisher: Multidisciplinary Digital Publishing Institute, pp. 1–25.

[46] Lucas Wan and Ya-Jun Pan. "Improving performance for multi-agent systems using fuzzy-logic tuning and mixed feedback controller". In: *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society. Oct. 2020, pp. 267–272.

[47] Yunfei Xu et al. "Sequential Bayesian prediction and adaptive sampling algorithms for mobile sensor networks". In: *IEEE Transactions on Automatic Control* 57.8 (2011). Publisher: IEEE, pp. 2078–2084.

[48] Pengming Zhu et al. "Multi-robot flocking control based on deep reinforcement learning". In: *IEEE Access* 8 (2020). Publisher: IEEE, pp. 150397–150406.

[49] Richard S. Sutton and Andrew G. Barto. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998.

[50] Frank L. Lewis, Draguna Vrabie, and Vassilis L. Syrmos. *Optimal control*. John Wiley & Sons, 2012.

[51] A.E. Bryson. "Optimal control-1950 to 1985". In: *IEEE Control Systems Magazine* 16.3 (June 1996). Conference Name: IEEE Control Systems Magazine, pp. 26–33.

[52] Richard E. Bellman and Stuart E Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 2015.

[53] Mohammed Abouheaf and Wail Gueaieb. "Multi-agent reinforcement learning approach based on reduced value function approximations". In: *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE, 2017, pp. 111–116.

[54] Biao Luo et al. "Model-free optimal tracking control via critic-only Q-learning". In: *IEEE transactions on neural networks and learning systems* 27.10 (2016). Publisher: IEEE, pp. 2134–2144.

[55] Boyoon Jung. "Cooperative target tracking using mobile robots". PhD thesis. United States – California: University of Southern California.

[56] Sandip Sen and Gerhard Weiss. "Learning in multiagent systems". In: *Multiagent systems: A modern approach to distributed artificial intelligence* (1999). Publisher: MIT Press Cambridge, MA, pp. 259–298.

[57] Adekunle A. Adepegba, Suruz Miah, and Davide Spinello. "Multi-agent area coverage control using reinforcement learning". In: *The Twenty-Ninth International Flairs Conference*. 2016, pp. 368–373.

[58] William Donald Smart. *Making reinforcement learning work on real robots*. Brown University Providence, 2002.

[59] Jeff Michels, Ashutosh Saxena, and Andrew Y. Ng. "High speed obstacle avoidance using monocular vision and reinforcement learning". In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 593–600.

[60] Richard Bellman. "Dynamic programming". In: *Science* 153.3731 (1966). Publisher: American Association for the Advancement of Science, pp. 34–37.

[61] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE transactions on systems, man, and cybernetics* 5 (1983). Publisher: IEEE, pp. 834–846.

[62] Ning Wang. "Model-Free Optimized Tracking Control Heuristic". Thesis. University of Ottawa, Sept. 2, 2020.

[63] Mohit Sewak. "Temporal difference learning, SARSA, and Q-learning". In: *Deep Reinforcement Learning*. Springer, 2019, pp. 51–63.

[64] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989). Publisher: King's College, Cambridge United Kingdom.

[65] Y. Hirashima et al. "Q-learning algorithm using an adaptive-sized Q-table". In: *Proceedings of the 38th IEEE Conference on Decision and Control*. Proceedings of the 38th IEEE Conference on Decision and Control. Vol. 2. Dec. 1999, pp. 1599–1604.

[66] Chathurangi Shyalika. *A beginners guide to Q-Learning*. Medium. Nov. 16, 2019.

[67] Bahare Kiumarsi et al. "Optimal and autonomous control using reinforcement learning: A survey". In: *IEEE transactions on neural networks and learning systems* 29.6 (2017). Publisher: IEEE, pp. 2042–2062.

[68] Paul J. Werbos. "Neural networks for control and system identification". In: *Proceedings of the 28th IEEE Conference on Decision and Control,* IEEE, 1989, pp. 260–265.

[69] Rongxin Cui et al. "Adaptive neural network control of AUVs with control input nonlinearities using reinforcement learning". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.6 (2017). Publisher: IEEE, pp. 1019–1029.

[70] Bernard Widrow, Narendra K. Gupta, and Sidhartha Maitra. "Punish/reward: Learning with a critic in adaptive threshold systems". In: *IEEE Transactions on Systems, Man, and Cybernetics* 5 (1973). Publisher: IEEE, pp. 455–465.

[71] Dimitri P. Bertsekas and John N. Tsitsiklis. "Neuro-dynamic programming: an overview". In: *Proceedings of 1995 34th IEEE conference on decision and control*. Vol. 1. IEEE, 1995, pp. 560–564.

[72] Paul Werbos. "Approximate dynamic programming for realtime control and neural modelling". In: *Handbook of intelligent control: neural, fuzzy and adaptive approaches* (1992). Publisher: Van Nostrand, pp. 493–525.

[73] Mohammed I. Abouheaf, Frank L. Lewis, and Magdi S. Mahmoud. "Differential graphical games: Policy iteration solutions and coupled Riccati formulation". In: *2014 European Control Conference (ECC)*. IEEE, 2014, pp. 1594–1599.

[74] Santanu Kumar Pradhan and Bidyadhar Subudhi. "Real-time adaptive control of a flexible manipulator using reinforcement learning". In: *IEEE Transactions on Automation Science and Engineering* 9.2 (2012). Publisher: IEEE, pp. 237–249.

[75] Mohammed I. Abouheaf and Magdi Sadek Mahmoud. "Policy iteration and coupled riccati solutions for dynamic graphical games". In: *International Journal of Digital Signals and Smart Systems* 1.2 (2017). Publisher: Inderscience Publishers (IEL), pp. 143–162.

[76] Mohammed I. Abouheaf and Frank L. Lewis. "Approximate dynamic programming solutions of multi-agent graphical games using actor-critic network structures". In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2013, pp. 1–8.

[77] Mohammed I. Abouheaf and Frank L. Lewis. "Dynamic graphical games: online adaptive learning solutions using approximate dynamic programming". In: *Frontiers of Intelligent Control and Information Processing*. World Scientific, 2015, pp. 1–48.

[78] Asma Al-Tamimi, Frank L. Lewis, and Murad Abu-Khalaf. "Discrete-time nonlinear HJB solution using approximate dynamic programming: Convergence proof". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.4 (2008). Publisher: IEEE, pp. 943–949.

[79] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A comprehensive survey of multiagent reinforcement learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008). Publisher: IEEE, pp. 156–172.

[80] Said G. Khan et al. "Reinforcement learning and optimal adaptive control: An overview and implementation examples". In: *Annual reviews in control* 36.1 (2012). Publisher: Elsevier, pp. 42–59.

[81] Dimitri P. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA, 2019.

[82] Nikhil Ketkar. "Stochastic gradient descent". In: *Deep learning with Python: A hands-on introduction*. Ed. by Nikhil Ketkar. Berkeley, CA: Apress, 2017, pp. 113–132.

[83] Mohammed Abouheaf, Nathaniel Mailhot, and Wail Gueaieb. "An online reinforcement learning wing-tracking mechanism for flexible wing aircraft". In: *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*. 2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE). June 2019, pp. 1–7.

[84] Michal Bartys and Bartlomiej Hryniewicki. "The trade-off between the controller effort and control quality on example of an electro-pneumatic final control element". In: *Actuators*. Vol. 8. Issue: 1. Multidisciplinary Digital Publishing Institute, 2019, pp. 1–21.