Adaptive Fuzzy Reinforcement Learning for Flock Motion Control

by

Shuzheng Qu

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Examining Committee

The following served on the Examining Committee for this thesis.

Carleton Member: Hicham Chaoui
Associate Professor, Department of Electronics
Carleton Univeristy

Internal Member(s): Davide Spinello
Associate Professor, Department of Mechanical Engineering
University of Ottawa

Supervisor(s): Wail Gueaieb
Professor, School of Electrical Engineering & Computer Science
University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

The flock-guidance problem enjoys a challenging structure where multiple optimization objectives are solved simultaneously. This usually necessitates different control approaches to tackle various objectives, such as guidance, collision avoidance, and cohesion. The guidance schemes, in particular, have long suffered from complex tracking-error dynamics. Furthermore, techniques that are based on linear feedback or output feedback strategies obtained at equilibrium conditions either may not hold or degrade when applied to uncertain dynamic environments. Relying on potential functions, embedded within pre-tuned fuzzy inference architectures, lacks robustness under dynamic disturbances.

This thesis introduces two adaptive distributed approaches for the autonomous control of multi-agent systems. The first proposed technique has its structure based on an online fuzzy reinforcement learning Value Iteration scheme which is precise and flexible. This distributed adaptive control system simultaneously targets a number of flocking objectives; namely: 1) tracking the leader, 2) keeping a safe distance from the neighboring agents, and 3) reaching a velocity consensus among the agents. In addition to its resilience in the face of dynamic disturbances, the algorithm does not require more than the agent's position as a feedback signal. The effectiveness of the proposed method is validated with two simulation scenarios and benchmarked against a similar technique from the literature.

The second technique is in the form of an online fuzzy recursive least squares-based Policy Iteration control scheme, which employs a recursive least squares algorithm to estimate the weights in the leader tracking subsystem, as a substitute for the original reinforcement learning actor-critic scheme adopted in the first technique. The recursive least squares algorithm demonstrates a faster approximation weight convergence. The time-invariant communication graph utilized in the fuzzy reinforcement learning method is also improved with time-varying graphs, which can smoothly guide the agents to reach a speed consensus. The fuzzy recursive least squares-based technique is simulated with a few scenarios and benchmarked against the fuzzy reinforcement learning method. The scenarios are simulated in CoppeliaSim for a better visualization and more realistic results.

Acknowledgements

I owe my deepest appreciation to my thesis supervisor Dr. Wail Gueaieb and my research mentor Dr. Mohammed Abouheaf for their patient guidance, suggestions and support throughout my graduate studies. This thesis would not have been possible without their advice and encouragement.

Meanwhile, I want to express my gratefulness to my colleagues and friends for their friendship, care and support.

Finally, I must express my very profound gratitude to my family, especially my parents, whose love, guidance, and support are with me in whatever I pursue. They have always been the ultimate models to me.

Table of Contents

List of Tables

List of Figures

Abbreviations

ADP Approximate Dynamic Programming 12, 20, 21, 27, 28

FIS Fuzzy Inference System 2, 17, 24, 25, 34, 38, 74, 75

LMS Least Mean Squares 52

LTSM Long Short-term Memory 15

RL Reinforcement Learning 2–4, 6, 8, 11–15, 17, 19–23, 25, 27, 28, 34–36, 38, 51, 74, 75

RLS Recursive Least Squares 2–4, 19, 20, 26–28, 51–53, 60, 62, 74, 75

TD Temporal Difference 23

TS Takagi-Sugeno 6, 16, 18, 20, 25, 38–40

Chapter 1

Introduction

1.1 Overview

Flocking is the form of collective behaviour of multiple agents interacting with each other to achieve a common group objective. The idea of flocking comes from the groups of natural living beings such as flocks of birds, schools of fish, herds of wildebeest, etc. This collective behaviour of animals has attracted many researchers to study the fundamentals behind it and inspired people to put into use the strategy when controlling a large number of agents to cooperate together without centralized control.

The research of flocking was started by Craig Reynolds in 1986. He initiated the study of flocking by inventing a simulation model of biological flocking, which he called Boids [82]. The model simulates the behaviour of birds. The name "boid" corresponds to a shortened version of "bird-oid object", which refers to a bird-like object [12]. There are three heuristic rules introduced by Reynolds in his simulation. They are known as (i) separation: steer to avoid collision with nearby flock-mates; (ii) alignment: attempt to match velocity with nearby flock-mates; and (iii) cohesion: attempt to stay close to nearby flock-mates.

Flocking control in robotics represents the synchronization in multi-agent systems with neighbour-based rules and variable coupling topology through short-range information exchanging [107]. The control strategies have been deeply developed and widely used in applications because of their certain advantages, such as fast response, high adaptability to the environment, strong stability and connectivity of multiple robots, and ease of specifying the flocking behavior. The proposed flocking control technologies of nowadays have shown great application prospects within the fields including industrial and agricultural production, exploration using autonomous robots (land, ocean, space, nuclear),

transportation, and national defense industries [26]. These areas involve applications such as capturing evader targets, surveillance zone forming, moving large objects in fixed formations, autonomous fleets control, exploration and rescue of unknown environments, etc.

Reinforcement Learning (RL) in flocking control has drawn lots of researchers' and engineers' attention and has been investigated extensively because of its algorithm structure's simplicity and the generality of the settings. As a machine learning method, it does not require the knowledge of the surrounding environment but learns and improves its policy or strategy by interacting with it. This property becomes a significant advantage when it is applied to a multi-agent control system, which makes it highly adaptive to dynamic variants.

As another effective system control technique - fuzzy logic, can be understood as an inference process of dealing with imprecise, vague, or uncertain system information. Different from traditional control methods which are point-to-point control, fuzzy logic control is a range-to-point or range-to-range control technique [11]. The utilization of fuzzy rules and membership functions makes fuzzy control systems extremely flexible, easy to improve system accuracy and allow for the integration of other control schemes.

The Recursive Least Squares (RLS) algorithm is widely utilized as an adaptive parameter estimator which takes new data into account to improve previous estimations of the parameters in a linear or linearized aspects to model the observed system. RLS algorithms' high adaptbility and stability can provide practical approaches to performing weight estimation in a Policy Iteration mechanism of a Markov decision process.

In this thesis, we design and apply two distributed adaptive control methods to a multi-agent system to achieve a set of objectives in a flocking control problem. The RL technique is integrated with a Fuzzy Inference System (FIS) to provide a highly adaptive online learning algorithm based on the Value Iteration mechanism, which is referred to herein as Fuzzy-RL approach. The effectiveness of the proposed method is validated with two simulation scenarios and benchmarked against a similar technique from the literature. Another approach based on the Policy Iteration mechanism is also developed by integrating an RLS algorithm with the same FIS. This method as an extension to the first one is simulated in another robotic simulator with different scenarios to give a better visualization of its strategies when applied in a close to real-world environment.

1.2 Motivation and Objectives

Conventional centralized control techniques have been challenged by the increasing complexity of cooperative robotic systems. Such techniques lack autonomy when applied in a complex multi-agent system. Techniques that are based on linear feedback or output feedback strategies obtained at equilibrium conditions either may not hold or degrade when applied to uncertain dynamic environments. Relying on potential functions, embedded within pre-tuned fuzzy inference architectures, lacks robustness under dynamic disturbances. For instance, a fleet of self-driving surface vehicles, unmanned aircraft, and inventory or item distributing robots all demand the control scheme of collective behaviours under complex environments. Distributed control mechanisms have been investigated and widely adopted in this domain [30,90]. Due to the simplicity of its system structure and the ability of managing competing control objectives, distributed control provides an improved way of solving such cooperative control problems by reducing the inter-dependency among the agents. For example, a typical multi-agent control system usually needs to achieve three objectives: navigation, separation, and cohesion. A distributed control structure cannot only compromise the possibly conflicting goals to provide an optimal solution but should also reduce the heavy information dependence for each agent than the traditional methods. The multi-agent control systems constructed from this type of technique often involve feedback strategies and communication graphs [2, 29].

The most direct way of a fuzzy logic controller's design is to determine the membership functions and fuzzy rules by examining the relevant control system which is operated by a human, or another controller that has already been employed on the same or a similar system. The reason behind this is the lack of systematic procedures for designing fuzzy logic control systems. The proposed fuzzy logic systems will need to be improved or adjusted on the membership functions and fuzzy rules if relevant testings fail, and this may lead to a long tuning process. The focus of fuzzy logic development turns into the direction of providing the control system's ability to learn by itself, which greatly increases the adaptability of the system. Fuzzy systems expanded in the way of combining with neural networks such as RL can obtain the capability of self-learning by making their parameter-tuning process automated [78].

The main objective of the work is to design an online adaptive distributed technique for the autonomous control of flock systems to meet a set of goals. Two proposed techniques with adaptive and flexible structures are based on online Fuzzy-RL and Fuzzy-RLS schemes, which simultaneously target the set of goals; namely, following a leader, collision avoidance, and reaching a flock velocity consensus. A communication layer designated by a graph topology is used to guide the flock of agents to reach a consensus on a common flock

velocity. In addition to its resilience in the face of dynamic disturbances, the algorithm does not require more than the agent position as a feedback signal.

1.3 Contribution

This work contributes a distributed online adaptive Fuzzy-RL technique and an extended Fuzzy-RLS technique to guide a flock of agents without a prior knowledge about the environment, the flock dynamics, or the desired mission trajectory. It does so while compromising conflicting individual and collective objectives. The algorithms take advantage of a novel flexible collision avoidance mechanism that does not employ complex potential functions. This avoids the dependence on pre-calculated guidance and collision-avoidance control laws, which are known to be inefficient in dynamically varying ill-defined environments.

1.4 Thesis Organization

The rest of the paper is organized as follows:

- Chapter 2 reviews the relevant literature of multi-agent control techniques.
- Chapter 3 provides background information about some of the computational tools adopted in the thesis, such as RL, neural networks, fuzzy logic, RLS algorithm, and communication graph.
- Chapter 4 formulates the flocking control problem and presents a high-level description of the proposed control structures.
- Chapter 5 introduces a Fuzzy-RL control system and explains the details of the proposed control approach with respect to each of the objectives. It also demonstrates the simulation results with two different simulation scenarios along with their analysis. The results are also benchmarked against another technique from the literature.
- Chapter 6 details the second proposed method, a Fuzzy-RLS control system, and explains the details of the extended control approach with respect to the previous method. Simulations are conducted in a realistic robotic simulator, CoppeliaSim, while the results are analyzed and compared with those of the first method presented in the previous chapter.

- Chapter 7 lists a few concluding remarks and suggests possible future research directions.

1.5 Scholarly Outcome

The research presented in the thesis has led to the following publications:

- [6]: Mohammed Abouheaf, Shuzheng Qu, Wail Gueaieb, Rami Abielmona, and Moufid Harb. Responding to illegal activities along the canadian coastlines using reinforcement learning. *IEEE Instrumentation and Measurement Magazine*, 24(2):118–126, April 2021

- [78]: Shuzheng Qu, Mohammed Abouheaf, Wail Gueaieb, and Davide Spinello. An adaptive fuzzy reinforcement learning cooperative approach for the autonomous control of flock systems. In *IEEE International Conference on Robotics and Automation (ICRA 2021)*, pages 1–7, Xi'an, China, May 2021. (Accepted: Mar. 2021)

- [79]: Shuzheng Qu, Mohammed Abouheaf, Wail Gueaieb, and Davide Spinello. A policy iteration approach for flock motion control. In *IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pages 1–7, October 2021. (Accepted: Sep. 2021)

Chapter 2

Literature Review

This chapter introduces a literature review about numerous approaches and applications that are relevant to the techniques proposed in this thesis including flocking motion control, RL control, and fuzzy logic control. Some brief discussion about the techniques' common shortage is also provided.

In flocking motion control, a multi-agent system used for controlling warehouse robots to cooperative with each other to fetch and delivery packages is introduced. Three typical flock motion control objectives proposed by Reynolds in 1986 is discussed with examples. Two flock motion control systems designed for nonholonomic autonomous vehicles and unmanned aerial vehicles are studied in details. As a widely adopted flock motion scheme, distributed control approaches are applied in many applications and research problems. Section 2.1.1 provides a few examples that utilize the distributed scheme to achieve Reynolds flock control objectives. Section 2.1.2 gives a literature study on the leader-follower structure which is commonly used in flock motion control approaches.

A literature review on RL is given in Section 2.2. This section firstly gives a general overview on RL approaches then talks about its related applications including a robot navigation system for path planning. As classic approaches in RL, value and policy iterations are discussed respectively with adaptive control problems. Section 2.2.3 introduces an actor-critic neural network structure in RL, and two approaches using different approximation neural networks to implement the actor-critic structure are reviewed.

The last section focuses on fuzzy logic control approaches, especially for multi-agent systems. Takagi-Sugeno (TS) rule based fuzzy control systems are presented with two examples in this section. Fuzzy logic control is widely adopted in multi-agent systems for various purposes including navigation or path planning, target tracking and collision

avoidance. Section 2.3.1 reviews a few multi-agent control algorithms that utilize fuzzy logic systems to achieve the collision/obstacle avoidance objective.

2.1 Flock Motion Control

The study of flock motion control in multi-agent systems pertains to the behavior analysis of a large number of agents, such as flocks of flying birds and fish schools. Flocking motion control can be manifested in many multi-agent applications where conflicting objectives may coexist, such as fleets of self-driving vehicles, unmanned aircraft, and large warehouse robots [80, 105]. A multi-agent system named "Kiva" is demonstrated in [105]. The system provides cooperative control approach to warehouse robots which are independent and computational units that can receive requests and perform optimal allocation actions individually. An innovative concept proposed and already practiced in the system is the consideration of receiving stations as inventory station agents. The mobile package-fetching-delivering robots cooperate together with the inventory station agents to form a multi-agent cooperative system. The flock motion control of the Kiva system consists task allocation, path planning, and motion planning. However, the control system is only designed for the specific warehouse robots, which is not model-free [105]. The autonomy of such interacting systems relies on how well the pre-tuned control strategies can cope with the unmodeled dynamics or even perform under unexpected operational scenarios.

As initiated by Craig Reynolds in 1986, the concept of flocking motion study consists three heuristic rules is introduced in his simulation [82]. They are known as (i) separation: steer to avoid collision with nearby flock-mates; (ii) alignment: attempt to match velocity with nearby flock-mates; and (iii) cohesion: attempt to stay close to nearby flock-mates. In the context of the current manuscript, the common objectives of the flocking motion control scheme can be summarized as the guidance of agents towards a common goal, preventing collision among the flock by imposing a minimum safety distance between the agents, and the cohesion of the flock members towards a common speed [29]. In [2], a flock motion control system is introduced for nonholonomic autonomous vehicles. The system classifies flocking control objectives into two categories: team and local. The local control objective is to remain a minimum separation distance between any vehicle and its neighbor, which coincides with the separation objective introduced by Reynolds. This objective is achieved using a collision avoidance scheme that involves an extended fuzzy system. The other two objectives are classified as team objectives: each vehicle is requested to align its speed with other flock members to reach a consensus; then each vehicle should also keep itself along the desired virtual trajectory. A navigational control system implemented from smooth

state feedback control laws is used to achieve the trajectory tracking team objective; a synchronization control system that is implemented from a smooth position dependent adjacency matrix and a communication graph structure is used for the last objective which is reaching speed consensus as a flock [2]. The control strategies are adapted online in real-time without prior knowledge of the agent dynamics.

The control of flock motion can be further extended with multi-objectives achievement where each agent decides its action based on a compromise between a set of competing, and possibly conflicting objectives. The multiple objectives proposed by such flocking motion problems usually contain the three heuristic rules as mentioned before. A flocking motion control system introduced in 2018 is designed for unmanned aerial vehicles swarm [91]. The work extends a classic Reynolds flock motion problem into a multi-objective compromising control study. In addition to the three basic Reynolds objectives: separation, alignment, and cohesion, the unmanned aerial vehicles control system is also able to perform target seeking and obstacle avoidance. The additional two objectives are considered as path planning rules of the system due to the absence of a leader in the flock. The proposed control system adopts modular Q-Learning technique in RL to treat each of the five objectives as a distinct module, where each module is characterized by a single Q-table. The optimal policy selects the action by searching for the largest weighted sum of Q-values across all Q-tables [91]. During the training of the autonomous aerial vehicles, Q-tables are updated using a Q-learning technique, where each of the vehicle is evaluated in the system with its current state, and an action will be selected by taking the consideration of possible states followed and optimal rewards obtaining in the future. Target seeking is trained separately with the other four objectives. During the training of the adaptive system, only one individual agent of the flock is selected to perform the target seeking task but all the other objectives are trained on each of the agents. Since the flock motion should have the characteristic of cohesion, the individual agent that is trained with target seeking scenarios will be seen as the leader of the flock and guides the rest of the agents to the planned targets.

2.1.1 Distributed Control Scheme

Since the early works in [71, 82], researchers have been reflecting back these ideas of flock motion to develop distributed control approaches for cooperative systems. They have been investigated and widely adopted in the domain of flock motion control nowadays [30, 90]. Distributed control technique owns its popularity because of its simplicity in a multi-agent system design and effectiveness on objectives control. They provide an alternative solution based on relaxing the inter-dependency requirements among the agents, such as the communication range of each agent, the amount of information required about its

surrounding agents and the environment, etc. A key aspect about any distributed controller is its ability to optimize a set of simultaneously conflicting individual, local, and team objectives, like navigation, separation, and cohesion. The underlying control methods are often based on theories of feedback mechanisms and communication graphs [54, 65].

A distributed flock control scheme is proposed for control and analysis of multiple autonomous agents with point mass dynamics and communication graph theory [108]. The work tries to achieve the three Reynolds flock motion objectives where two cooperative control laws are designed to focus on the control of different centers of the flock. The mass dynamics is utilized to estimate the mass center of a group of flock, and the geometric flock center based on the positions of each agent is calculated. The mass and geometric centers can help with the control laws achieve flock formation including cohesiveness and velocity consensus. A communication graph is designed for the inner connection between the agents so that the dynamic information can be exchanged. Collision avoidance is realized by applying a smooth attractive/repulsive potential using fuzzy logic strategies. Each control subsystem in the proposed work generates its corresponding control signals with respect to each of the objectives. A combined final control strategy will then be applied on each agent for an optimal control purpose by compromising all the competing objectives in a distributed control fashion. Additions of the sub-control signals is processed as a simple way of aggregating all the strategies, which is widely adopted for distributed flock motion control in multi-agent systems.

In [33], distributed control approach has been utilized with a neighbor-based tracking protocol to solve a leader-follower tracking problem under noisy environments. Many of the flock motion control systems do not take into account the noise effects when agents communicate with each other through graph networks. However, in real life the information exchange inside a multi-agent system is often subject to different kinds of constraints. The distributed control approach proposed in [33] adopts a closed loop tracking control system with a directed graph topology, where some added noise is taken as part of the neighbours' position/coordination information shared with the agent. The distributed control system shows stable and convergent performance in the simulation.

Time-varying feedback control approach and consensus theory are common techniques applied in distributed control systems to guide the motion of a flock on target tracking and obstacle avoidance [13, 54]. The authors in [13] introduce a novel estimation on the coordination or relative position of a flock center by using graph topology. The estimation uses consensus concept that makes the followers reach an agreement regarding the flocking center position [13]. The approximation of the flock center is used to provide guidance to the agents so that they can tend towards the dynamic center during all the time and keep tracking the desired path. A few agents selected as the leaders are set to be independent

from this cohesion control strategy. Distributed control scheme is also employed to achieve the Reynolds flock motion objectives in this approach. An artificial potential force function is used for the design of control law of collision avoidance. Fuzzy logic based approach is used for alignment control of the agents. Each flock motion control objective has its corresponding control signal, and the aggregation of them forms the final control strategy. Multi-agent systems benefit from distributed control strategies due to their high robustness, flexibility and independence. Such systems can apply less operations to individual agents to achieve flock control objectives efficiently comparing to other control approaches.

2.1.2 Leader-Follower Structure

The leader-follower structure has been broadly employed in flock motion control because of its simplicity and controllability when applied in distributed control systems [32, 42, 77]. A flock motion control scheme is designed in 2015 for multiple quadrotors using a leader-follower structure [32]. They define the flock motion problem in a leader-follower manner as: in a multi-agent system, all the followers should follow the leader and all of the agents can track a reference signal such as a planned path, which is received by the leader. In addition, the agents should also maintain some safety distances between each other to avoid collisions during and after the flock formation. The lead-follower is stated to be an important structure since it can enhance the performance by extending working areas easily when the unmanned agents are used for the purposes of searching, surveillance, inspection, and exploration etc. [32]. The work also demonstrates a comparison between weighted and unweighted relative positions and velocities as the system inputs from a quadrotor's neighbors. The weighted information is used in the control system to perform repulsion or attraction between the agents to avoid collision. The simulation result shows an improved performance when using weighted neighbors' relative information for the control system since the intensity of control strategies can be adjusted based on the distances and velocities.

In a leader-follower structure, a leader is usually independent from the rest of flock and controlled directly by a command generator. It guides the rest of the flock following a desired trajectory or reaching a destination by providing coordination and dynamic information to the system. Therefore, a leader may not only be considered as a moving target but also a desired mission trajectory or a virtual time-dependent signal [78]. The leader-follower hierarchical structure in distributed control systems gives more flexibility when applied in flock motion control problems [42, 77]. From this architecture, the followers are able to approach to a desired target collectively and track even more complex trajectories of the leader with the same dynamical objectives [42]. Other approaches that does not

utilize the leader-follower architecture may still be capable with distributed control strategies. A proportional-derivative controller is used to monitor the trajectory of an agent in a flock motion system in [38], where the agents target not only the center of the flock but also their own trajectories and need to remain in a certain shape from the beginning.

2.1.3 Other Techniques

Many other techniques have also been applied in flock motion control applications. Sliding-mode control is a common approach that is used to handle the coordination or trajectory tracking tasks within multi-agent systems [17, 21, 31, 33]. The control technique proposed in [31] adopts a sliding-mode technique to provide robustness to the flock motion control strategy with respect to the environment uncertainties and varying missions in trajectory tracking. A Lyapunov candidate function and a potential function usually need to be selected and designed when constructing a sliding-mode controller. In [31], a potential function is designed to satisfy collision avoidance scenarios between the agents based on their relative velocities and positions. The potential function consists two terms where the first term is an attractive potential function, and the second is a repulsive potential function. Another adaptive control mechanism is used to trace the peaks of unknown fields for a multi-agent system under uncertain environments in [37]. A server-based approach is employed to control a group of embedded control systems in [68]. A common shortcoming showed in many of the flock motion control techniques is their dependence on the prior knowledge of a system model that is free of unstructured uncertainties.

2.2 Reinforcement Learning

As a significant area in machine learning, RL can be generally summarized as a mechanism which can help an agent reach its goal by interacting with the unknown environment in various decision-making problems. The environment determines consequences for taken actions and can be either dynamic or static [88]. The agent learns the pattern through feedbacks (rewards) from the environment after an action is taken followed by a state change. When a training is finished, the agent should be able to obtain the optimal strategy which is also considered as the optimal policy. RL techniques offer flexible adaptations and model-free process for multi-agent systems when interacting within complex environments [15, 16]. They have been applied in various robotic applications, such as autonomous helicopters [1], multi-robot predator avoidance [41], guidance for biped humanoid robots [56], multi-robot collision avoidance [57], and efficient driving systems [75]. RL-based trajectory-tracking or

path-following control mechanisms for unmanned surface and aerial vehicles are presented in [96,104]. A path planning approach using RL control for a crowd aware robot navigation system is demonstrated in [18]. The problem is defined as a navigation task where a ground robot is required to reach a target point through a crowd of humans. This task is considered as a sequential decision making problem which can be definitely studied through a RL framework. The state of a robot in the proposed RL system is modeled as a vector that contains the robot dynamic information at each time step, it includes the robot's relative distance to the target area, velocities in each dimension, and its radius [18]. A reward mechanism is designed to be used in the Bellman equation to update the value functions. The mechanism gives the highest reward to the robot when it reaches to the target point while penalizes the robot when it stays too close to a human. The results simulated in different scenarios show that the proposed RL system can successfully anticipate human dynamics and navigate robots in crowds.

2.2.1 Value Iteration

RL control systems are usually implemented using a Value Iteration or Policy Iteration algorithm. Value Iteration algorithms look for the optimal value function in RL problems by iteratively updating the estimated value function of the system online. Many applications employ Value Iteration algorithms to obtain near optimal control for either discrete or continuous-time nonlinear systems under Approximate Dynamic Programming (ADP) or RL environments, which can be processed without the exact knowledge of system dynamics [14, 70, 100]. Rottmann and Burgard introduced an online RL Value Iteration control system that studies the unknown system dynamics and value function based on Gaussian approximation process models [85]. This learning approach extends the Value Iteration algorithm which has its value function in the RL system designed from the Bellman equation. The value function reveals the expected long-term reward when an agent is at one state and it only depends on the following state's value with a given policy [85]. The actions are taken so that they can result in the highest state value from the given policy. Then the value function of the system is updated iteratively from a Gaussian process approximation. An interesting simulation is demonstrated in the paper with the proposed control approach. The goal of the simulation is to control a real miniature blimp reaching a desired height and make the blimp remain at this height. The state space of the system is designed with two components: the distance difference between the desired and actual height of the blimp, and the vertical velocity of the blimp. Penalties will be applied whenever there exists a height difference. The result shows that the blimp is successfully maintained at the goal altitude by taking a learning time around 40s. This RL Value Iteration algorithm approach

by Rottmann and Burgard demonstrates an adaptive and stable control performance of the system.

A common approach adopted to perform the RL Value Iteration approximation of the value function is neural networks [49, 59]. Neural networks can be used to approximate the optimal value function at each time step and the process stops when a convergence of the approximation is reached. Another RL Value Iteration algorithm is proposed in [100] to solve optimal control problems for discrete-time nonlinear systems. Multi-layer back propagation neural networks are utilized in this paper to approximate the system value function and compute the control strategy. The control approach is proven to have convergent characteristics with the derived value function by presenting a simulation of a discretized pendulum system. The result shows that the system states can iteratively converge to the optimal value and the control laws are admissible at the end [100]. The state value function derived for this Value Iteration approach is based on an performance index function which is required to be minimized by the optimal control scheme.

2.2.2 Policy Iteration

In a Policy Iteration process, the algorithm starts with an arbitrary policy then iteratively evaluate and improve the policy until convergence. Multi-agent applications widely employ RL Policy Iteration control systems since such control approaches can not only stabilize the nonlinear dynamic systems, but also achieve the optimal consensus online [50, 76]. A modified version of the classic RL Policy Iteration approach for path planning including obstacle avoidance is demonstrated in [7]. The system value function and policy are redefined by taking the consideration of environment configuration such as obstacles. The modified algorithm involves creations of new subspaces of actions that are actually available to be taken in each state and not colliding with the environment, which can improve the effectiveness of the approach and save computation time [7]. Two algorithms including a conventional Policy Iteration approach and a modified version proposed in the paper are simulated in a same path planning problem, where the proposed approach shows improved results by using much less computation time [7].

Similar to Value Iteration approaches, neural networks such as actor-critic schemes can be used as the estimators for policy evaluation processes as well [20]. A Strong Emergent Policy approximation approach is introduced to approximate optimal decentralized policies for cooperative multi-agent systems with a distributed variant of policy iteration [73]. In the paper, they use neural networks to approximate and update the value and policy functions. The Policy Iteration algorithm of the system consists two alternating parts

which are policy evaluation and improvement. A state value function is used to evaluate the policy at each time step and the policy will be improved through a mechanism by selecting the action that can maximize the value function. The process stops when the policy converges and the optimal policy should be reached by then. A classic multi-agent problem: Pursuit and Evasion is simulated with the proposed Strong Emergent Policy approximation to examine the system's learning ability [73]. In a Pursuit and Evasion problem, several agents selected as the evaders move randomly on a 2D plane and need to be captured by the pursuers. The pursuers are controlled by the proposed system and learn to capture all the evaders using following the rule: two pursuers have to be in the same cell together with an evader to proceed the capture action. The Policy Iteration algorithm shows effective results where the agents capture all the evaders and converge to an optimal policy in less than 100 episodes. An increased number of pursuers can even help the system converge to the optimal policy with a higher total reward in about only 20 episodes [73].

Least-squares as another estimation technique has also been applied for policy approximations in [48, 106]. The least squares policy iteration concept for RL control problems is firstly proposed by Lagoudakis and Parr in 2003 [45]. The algorithm is mainly based on another related technique: Least squares Policy Iteration Q-learning, which approximates the state-action value function using an off-policy strategy, and the algorithm is adopted for the implementation of the least squares policy iteration approach [45]. The approach provides another way of learning and converging to the optimal policy other than using a neural network. An improved incremented least squares policy iteration algorithm is also proposed in [48]. The algorithm extends the previous least squares policy iteration algorithm with temporal-difference strategy, which still utilizes the off-policy scheme to learn and approximate the state-action value function. The off-policy learning scheme permits action selection and policy improvement without a dynamic model [48].

2.2.3 Actor-Critic Structure

To help approximate the unknown value function or policy and assess the quality of the action taken, actor-critic neural networks are usually utilized for this purpose in RL [94]. The actor approximates the optimal strategy-to-follow while the critic reflects the value of being at a certain state while taking a particular action. The actor-critic architecture has been adopted for cooperative control problems in [3]. An explicit supervisor is added to an actor-critic RL unsupervised structure in [84], which makes the system supervised to improve the effectiveness of learning. This method is simulated as a manipulator controller for robot arms and also tested on real robots. An approach that combines fuzzy logic with

actor-critic RL system is proposed in [98]. The combined structure covers a common actor-critic system's shortage of limited ability in performing large-scale or multi-dimension computations. Actor-critic RL structure can also be used as an approach for nonlinear dynamic systems tracking control [101]. It attracts many researchers' attention in this area due to the structure's capability in performing object tracking with control optimization simultaneously.

Multi-agent systems commonly utilize the actor-critic structures because they can not only consider the optimal actions of the agents but also improves the policies that need complex multi-agent cooperation [36,58]. A multi-agent RL system that employs the actor-critic neural network is able to approximate the optimal policies with inner communications between the agents [72]. The proposed control system implements the neural network based actor-critic framework by using recurrent Long Short-term Memory (LTSM) layers. The inner communication is realized among the agents through the LTSM neural network which is designed to be bidirectional to reduce the sequence dependency in both the actor and critic networks. During the training phase of the proposed system, the actor network generates the best strategy-to-follow at each time step for every agent from a sequence of observations, and the critic network evaluates the policies by computing estimations on the current state's value. The actor-critic RL system is tested with a multi-agent cooperative navigation problem, where the agents are required to cooperative with each other to reach a set of landmarks through the established communication scheme. Experiment results show converged performance of the system after numbers of episode training [72]. Another RL approach adopts classic feed-forward neural networks for the implementation of the actor-critic structure in a multi-agent system [40]. In this approach, the neural networks' learning weights in each layer are updated during the training phase and stored in a matrix format. Similar to the other actor-critic structures, the actor neural network iteratively provides improved control strategy based on an objective function which is predicted by the critic neural network from a reached state and control action [40]. A consensus of the agents can be easily achieved from the proposed multi-agent RL system.

Despite RL scheme's high potential in learning how to interact with ill-defined environments, they suffer from a major shortcoming stemming from the discrete (non-smooth) nature of their action space, which may lead to a "jerky" behavior when they are applied to control a real-world system. However, this is possible to solve by combining them with a fuzzy logic engine, taking advantage of its nonlinear approximation ability, to get a fuzzy RL algorithm with a continuous output space.

2.3 Fuzzy Logic Control

Fuzzy logic has been a widespread technique which is commonly applied in control systems when dealing with imprecise, vague, or uncertain dynamics. It is featured by the ability of controlling complex systems without requiring a mathematical model of the system's dynamics, which is achieved by collecting meta-knowledge of the system's functionality in terms of linguistic relations [110]. Fuzzy control schemes have been applied in many robotic control systems because of their control abilities on leading to a balanced performance for the conflict resolution of multiple optimization criteria, such as navigation for autonomous wheeled robot [52] and autonomous vehicle guidance [23]. The wheeled robot is considered as an unstructured dynamic system with intrinsic nonlinearity and unmodelled disturbance in [52], where a fuzzy logic control system is appropriate and practical to satisfy the requirements including trajectory tracking, path following and stabilisation. The fuzzy control structure is designed to decrease the complexity of the system by separating the entire structure into a few low-dimensionality fuzzy systems and combining them as distributed control, where the manual construction of each rule base becomes easy [52].

As a significant part of the knowledge base of a fuzzy inference system, fuzzy rules represent the professional control or modeling experience in making meaningful connection between the input variables of the fuzzy system to the output variables. As an example of TS rule based fuzzy control system, a state feedback controller for non-linear systems is implemented with a fuzzy logic system which consists of a group of linear fuzzy local models in [87]. The fuzzy system is constructed through TS rules to realize the multiple linear local fuzzy models. The consequent designed in TS rules are in the format of a system of functions. Therefore, both linear and nonlinear consequent can be expressed in a TS fuzzy system. An integration mechanism is then designed to give the overall output. A classic nonlinear problem of inverted pendulum is simulated with the proposed approach. The result shows the controller can effectively stabilize the system to reach an equilibrium in a short time. Another TS proposed fuzzy system is demonstrated in [113]. The system is designed for motion control on autonomous underwater robots. There are five linguistic variables created for the TS fuzzy rules: negative large, negative small, zero, positive small, and positive large. Corresponding membership functions are designed in triangular form. The fuzzy controller provides a stable and fast response in the simulation where the underwater robot needs to submerge and remain at a certain depth. Both of the proposed fuzzy systems need to be provided with expert experience when constructing reliable fuzzy sets, which reveals the shortcoming of adaptivity in many fuzzy control systems.

Multi-agent systems adopt fuzzy logic control for various purposes, such as navigating the agents for path following or target tracking. A general fuzzy logic adjustable autonomy

model is proposed to manage and control a set of agents to make qualitative decisions on movement monitoring in a complex environment [63]. Communication graph topology is often used to cooperate with fuzzy logic when solving multi-agent consensus problems. As demonstrated in [22], a distributed adaptive output feedback control framework based on both fuzzy logic and directed communication topology is designed for the consensus of a multi-agent system. The proposed fuzzy logic adaptive control system can approximate unknown nonlinear functions to match with the agent's linear system by taking neighbors information as the input. Schwartz et el. have introduced a Fuzzy Actor Critic Learning technique for tackling illegal, unreported and unregulated fishing vessels in 2017 [8]. The approach is based on an evader-pursuer scenario, which is one or more pursuers are assigned with the mission of catching an evader efficiently. A FIS plays a significant role in the proposed approach to navigate the agents to the evader as pursuing actions. The FIS consists of three main components: singleton fuzzification block, fuzzy inference engine, and average-center defuzzification block. An actor-critic structure combined with the FIS provides extended adaptability for the system by improving the FIS's learning weights iteratively, where the FIS corresponds as the actor part and the critic network approximates the value function to make improvement on the policy/weights [8]. The learning weights are applied as the output parameter during the defuzzification process. The proposed Fuzzy Actor Critic Learning approach is simulated in a multi-agent framework where an evader is selected and also applied with the learning process to avoid capture. The algorithm terminates when the evader is caught by a pursuer or the process reaches the maximum iterations. Three inputs are fed into the FIS system to generate the control output: distance between a pair of agents (evader and pursuer), and two headings of the vessels. To simplify the computation process and make the problem less challenging, a set of triangular fuzzy set is utilized and the speeds of the pursuers are set higher than the evader. The result shows that the proposed Fuzzy Actor Critic Learning system can successfully navigate the pursuers to catch the evader in different scenarios after numbers of learning episodes.

A lot of the proposed fuzzy logic systems suffer from reduced adaptability due to fixed fuzzy rule antecedents and consequents values, which would require manual tuning procedures for better accuracy when the operating environment changes. Fuzzy systems can be augmented with RL or neural networks to increase their adaptability by performing self-constructing/optimizing [81, 112], or automating their parameter-tuning process [78]. In a mobile robot navigation system, the fuzzy controller is designed as an integration of a fuzzy inference system and an RL algorithm [24]. The consequents of the fuzzy control system are updated through the RL algorithm to improve the system's adaptability. As another example, a neuro-fuzzy algorithm is designed to control a robotic manipulator by employing neural network in [25]. The neuro-fuzzy control structure is realized as a

self-learning system by automatically deleting or generating fuzzy rules using a predefined error reduction threshold. Once a fuzzy rule has its error reduction calculated below the threshold, it will be deleted from the system [25].

2.3.1 Collision Avoidance

Fuzzy logic based models have made great contributions to the development of autonomy multi-agent system control [22, 27, 63]. Fuzzy logic inference systems are widely used as the control technique for separation or collision avoidance in flocking control problems [29, 86, 99]. The fuzzy inference systems in such problems are usually designed from a repulsive potential function that takes the desired separation distance and relative distance between two agents as the input. The fuzzy control system in this case can be considered as a realization of the potential function. This approach can also be modified to avoid unknown obstacles on paths in a more challenged flocking control problem [109]. Another fuzzy logic based extended TS inference system is introduced to achieve the collision avoidance objective in a multi-agent system in [35].

A multi-agent fuzzy control approach utilizes the concept of fuzzy neighborhood with a grading scheme is introduced in [55]. In the paper, each agent is considered with the same grading intensity scheme which is characterized by a set of numbers in a certain range. The intensities between pairs of agents are based on their relative distances. The proposed algorithm focuses more on the collision avoidance by setting a larger intensity number when two agents get too close to each other, and a small intensity value is assigned when the pair of agents are far from each other. Since a zero intensity indicates the system will not generate control signal to adjust the distance between two agents, a repulse-attraction scheme is realized by using this scheme to perform flock motion control, where a fuzzy logic system is utilized to implement the system [55]. There are eight fuzzy rules designed for the system with corresponding intensity and control force as the output, and a relative distance is used as the input. The simulation is done on a 2D plane with a set of agents. The result shows the proposed fuzzy control approach allows the agents maintain a stable state and move in a simultaneous velocity.

Another fuzzy logic-based dynamic obstacle avoidance system is proposed in [44]. The algorithm navigates a mobile robot from a starting point to a target location without colliding with obstacles, which includes the collision with other mobile robots by considering them as dynamic obstacles. The fuzzy system consists of four principal components that are: fuzzification interface, knowledge base, decision-making logic, and defuzzification interface. As with other demonstrated fuzzy logic systems, the fuzzification interface maps

the input variables into linguistic variables to be further used by the fuzzy inference engine. The defuzzification interface transforms a fuzzy output into a crisp value. The knowledge base contains fuzzy rules and membership functions and the decision-making logic generates the fuzzy output [44]. One sensor value is fed into the proposed fuzzy system as the input and three outputs including linear-distance, velocity and turn-angle are generated for the mobile agent to avoid collisions. The experiment results show a working simulation scenario where the agents reach the target location without colliding with the obstacles on the path.

2.4 Summary

This chapter highlights the literature review on common approaches of flocking motion control, RL adaptive control systems, and fuzzy logic control. They are reviewed with related techniques that are conducted in this thesis. Relevant applications are also enumerated with special focus on the control of multi-agent systems. The shortcomings of some of the techniques are briefly discussed and possible ways of improvements are suggested. The literature review provides useful information for the understanding and designing of adaptive Fuzzy-RL and Fuzzy-RLS algorithms later in the thesis.

Chapter 3

Background

This chapter gives the background introduction to the techniques on which this research is founded, including approximate dynamic programming, RL, fuzzy logic, RLS and communication graph. As a typical structure used in RL, the actor-critic structure is explained in detail. Fuzzy membership functions and fuzzy rules that are used to construct the TS fuzzy inference system are introduced. Another approximation approach by using RLS algorithm is demonstrated with its fundamental concepts. Finally, a communication graph topology that is usually employed in multi-agent systems is explained.

3.1 Approximate Dynamic Programming

Due to the fact that there is no general method to extract an exact closed-form solution for any arbitrary nonlinear differential equation, approximation is one of the best strategies to solve such problems [102]. As an approximation methodology, ADP has become a powerful tool for certain types of multistage stochastic and dynamic problems in the area of operations research [74]. It has shown solid theoretical results to optimal control and related applications [103]. As a modeling framework, it is based on the Markov Decision Processes that provides several strategies for solving problems in discrete, stochastic, and sequential environments [53]. ADP offers good flexibility which enables the combination of simulation strengths with the intelligence of optimization [74]. Many control problems nowadays require the solution system to optimize over time, that is, current decisions made by the system's control unit will have effects on the system's future performance. These simulation problems always require their solution models to include the calculation of decision future impacts. ADP offers a powerful framework that determines decision impact

estimation on the future [74]. The estimation is then used to generate better decisions. ADP in this case can be seen as a form of "optimizing simulator" [74].

The discrete-time multistage stochastic control processes to be solved by an ADP model consist of a state space set S. During each time step t, the system enters into a particular state $S_t \in S$, from this state the system will take a decision x_t from the feasible decision set X_t. The decision made by the system then results in costs, which typically are produced from $C_t(S_t, x_t)$. Subsequently, the system arrives into a new state which is conditionally independent of all previous states and decisions, and this is with a probability of $P(S_{t+1}|S_t, x_t)$ [61]. Thus, the decision in such a process determines three objects: the immediate cost value, the environment within which future decisions are applied, and the influences on the future costs [61]. The ultimate objective is to find the optimal policy π that can be recognized as a decision generator or function $X^{\pi}_{S_t}$. It returns a corresponding decision x_t with the input of any possible states $S_t \in S$. The typical function used by ADP models in finding the optimal policy is Bellman's equation.

ADP can be applied to large-scale processes by solving Bellman's equations forward in time and this is done in an iterative manner, which enables the system to generate sample paths through the states [61]. Furthermore, ADP uses approximate cost functions that are updated through the iterations, which enables the approximation of future decisions' impacts.

3.2 Brief Introduction to Reinforcement Learning

RL is a machine learning technique that gives agents the ability to learn and optimize their strategies to solve decision-making problems upon interacting with their unknown dynamic environments [94]. This is done by tracking the benefit of being at a certain state of the environment while following some action in order to maximize a reward criterion to eventually reach a target state, where the actions are approximated through strategies. This enables the system to transit from one state to another that represents a better situation along the path of finding an optimal solution for the optimization problem. Therefore, it is important to notice that the actions chosen may affect not only the immediate reward but also the learning agent's next state and the subsequent rewards [8].

A RL agent has the ability of taking an action with the intention of either exploring or exploiting its current situation. Exploration is the generation of random actions to explore the potential higher rewards from the environment while exploitation refers to approximating the optimal action based on the current experience. A high exploration

rate may cause the agent to earn a low cumulative reward at the end while acquiring a broad knowledge about the environment. Yet, a high exploitation rate can lead the agent to the situation of experience shortage. The RL methodology imitates the analytical solution environment of the optimization problems where the optimal strategy and its output action value are interrelated [47].

Figure 3.1 demonstrates the communications between the learning agent and the environment. Assuming that this is processed in a discrete-time fashion, then at each time step t, the learning agent takes a representative state S_t of the environment and a reward R_t as inputs. The agent takes an action x_t that is considered to yield the highest reward. The learning agents typically start without any prior knowledge about the system. They need to discover for themselves which actions yield the highest reward by exploiting them [8]. At the next time step, the agent receives a reward R_{t+1} from the environment and then transits into another state S_{t+1}.

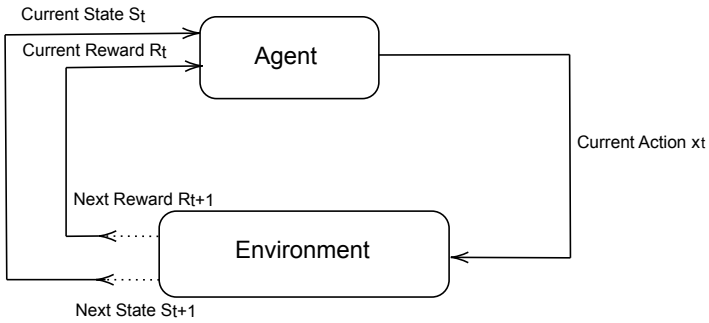


Figure 3.1: Reinforcement Learning evolution

In comparison to Supervised Learning, where the agent learns from examples provided by an expert, the only way RL agents can learn is to interact with their environment [93]. Even though Supervised Learning requires complete learning examples to reach a satisfactory performance, it is typically impractical to get a collection of examples of all behaviours that are both desirable and representative of all possible situations [93]. This reveals the advantage of RL techniques when facing the challenge of training data shortage or absence. RL techniques not only offer flexible adaptations while interacting within complex environments, but also offer a balance between exploration and exploitation [15, 16]. The techniques are usually implemented using either a Value Iteration or a Policy Iteration algorithm. The difference between them depends on how the strategy is evaluated and then updated accordingly [4].

3.3 Critic-Actor Structure

Typical machine learning problems usually have the main objective as finding the correct approximation of a set of inputs to the final output. In other words, the learning system is required to determine a map that is linking the inputs to the corresponding output at each time step. With the types of continuous and discrete function mapping, machine learning problems can be distinguished into regression or classification. Neural network model is a powerful approximation tool when used for solving machine learning problems. They are built up of multiple neurons with many connections between them. Since the approximation problems usually are not linearly separable in the input signals, part of the neuron connection forms an intermediate layer between input and output units, also named as a hidden layer [43].

To help approximate the unknown value function and to assess the quality of the action taken in RL, actor-critic neural networks are usually utilized [94]. The adaptive actor-critic structures are employed as approximation tools in RL systems. The actor is a neural network that approximates the optimal strategy-to follow while the critic is also a neural network that estimates the value (i.e., reward) of being at a certain state when taking a particular action.

A standard critic-actor structure is shown in Figure 3.2. Taking the current agent's state as inputs, the actor approximates an optimal action from its experience or knowledge of the environment which it acquired so far. Then, the critic part of the system estimates the future system's performance by evaluating the value function of each learning agent with respect to their objectives [8]. As a consequence of its action in the next processing step, the learning agent receives the corresponding reward and transits to the next state. Usually, the critic part of a RL system stands as a state-value function. After each policy approximation, the critic evaluates the new state with the reward to determine the difference between the actual and the expected situation [47]. By observing the temporal reward, the value function of the current state, and the next state into the system, we can evaluate the Temporal Difference (TD) error. The TD error is also considered as the prediction error based on which the actor and critic systems are tuned [8].

3.4 Fuzzy Logic

Fuzzy logic is widely employed when dealing with control problems that have imprecise, vague, or uncertain system information. It can provide not only a way of including vague

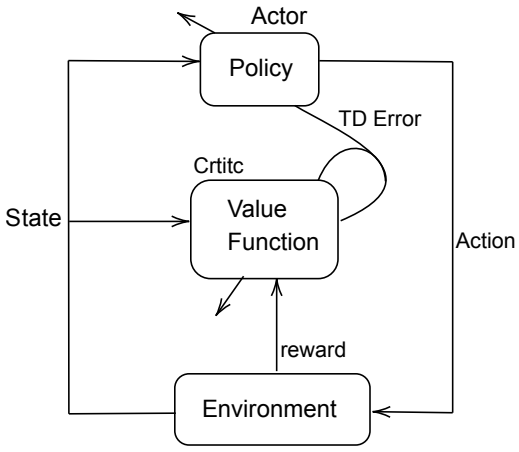


Figure 3.2: Critic-Actor Structure

human assessments in computing problems but also a balanced solution for conflict resolution of multiple criteria [89]. The fundamental problem of automatic control is the computation of precise response of the system under a given set of conditions. Unlike the conventional adaptive control techniques that are based on explicit plant dynamics, such as the differential equations of the system, fuzzy logic only requires the practical knowledge of an experienced operator [92].

Fuzzy logic owes its popularity to its ability to incorporate human-like expertise in controlling complex systems without the need for a prior definition of the system's dynamic model. It does so by defining the acquired meta-knowledge about the system's functionality in terms of linguistic relations [28]. Control systems based on fuzzy logic are applied in a wide range of applications and they appear to be the most pervasive in consumer electronics [19].

3.4.1 Fuzzy Inference System

The fuzzy membership functions and fuzzy rules form a core knowledge base in a fuzzy system. The knowledge base is utilized for every processing component inside a fuzzy system, including fuzzification, decision-making, and defuzzification. All these components form the complete structure of a FIS, which is also known as a fuzzy rule-based system or fuzzy models [60].

A schematic diagram of a FIS is demonstrated in Figure 3.3. A FIS is composed of

four conventional units as discussed: fuzzification, knowledge base, decision-making, and defuzzification. The process of a FIS is as follow: first, the scalar inputs are fed into the fuzzification interface, which converts them into matching degrees within their linguistic values; then a decision-making unit performs the inference operations on the fuzzy rules; lastly, the defuzzification unit processes the fuzzy results and transforms them into a crisp real-valued output. All the operations require interactions between the units and the knowledge base.

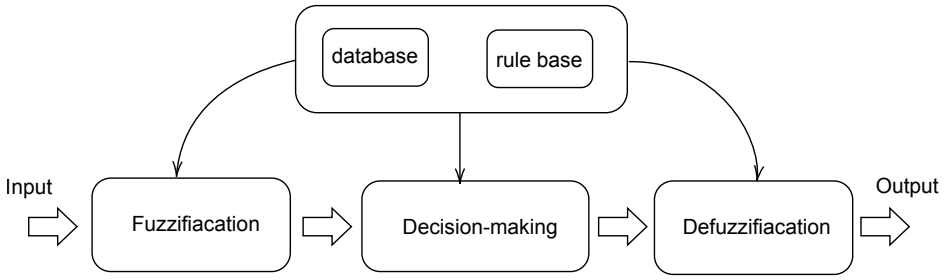


Figure 3.3: Fuzzy Inference System

3.4.2 Fuzzy Rules and Membership Functions

Fuzzy rules play a critical role in a fuzzy control system. In this thesis, the technique of TS fuzzy rules is applied into an online combined Fuzzy-RL system. Fuzzy systems built with TS rules are used as powerful practical control systems for modeling and control of complex systems nowadays [9]. A TS fuzzy system is featured by presenting the local dynamics of each fuzzy implication as a linear system model [60]. TS fuzzy rules consist of fuzzy antecedents and functional consequents. The rules are in the form of:

$$l_i : \text{if } x_1 \text{ is } A_{i1},\ x_2 \text{ is } A_{i2},\ \ldots,\ \text{and } x_n \text{ is } A_{in},\ \text{then } o_i\ =\ f(x_1, x_2, \ldots, x_n)$$

where l_i is the i^{th} fuzzy rule; $x_1,\ x_2,\ \ldots,\ x_n$ are the inputs to the system; A_{ij} denotes the antecedent fuzzy sets of the i^{th} rule and j^{th} input; o_i, as a function of the inputs, represents the output of the i^{th} linear subsystem [9].

The antecedent fuzzy sets are also considered as fuzzy membership functions. Gaussian distribution functions with different mean and variance values are typical fuzzy membership functions. Triangular functions are also a common type of fuzzy membership functions. Projecting a certain input onto its corresponding membership function can be confined as

a normalized value in the range of 0 to 1 [64]. A value of 0.7, for example, means that the linguistic input is approximately 70% of the maximum value it can have [64].

3.5 Recursive Least Squares

RLS developed from the least-squares regression algorithm is commonly used as the adaptive coefficients estimator which tries to minimize a linear least squares loss function associated with the system. Let us consider a linear regression model where the coefficients are unknown and need to be estimated from a given set of points or data. The loss function of a linear system with d parameters estimation is in the following form

$$L(\hat{\boldsymbol{\theta}}, k) = \frac{1}{2} \sum_{i=1}^{k} \lambda \left(y(i) - \boldsymbol{\varphi}^T(i) \hat{\boldsymbol{\theta}} \right)^2 \tag{3.1}$$

where $y(i)$ is the i^{th} observed value of the system, $\boldsymbol{\varphi}(i) \in \mathbb{R}^{d \times 1}$ is the vector of calculated values from measured signals and given variables, $\hat{\boldsymbol{\theta}} \in \mathbb{R}^{d \times 1}$ is the vector form of the corresponding estimated parameters, and λ is the forgetting factor. Then the RLS algorithm is used to perform the estimation that minimizes the loss function value presented in Equation (3.1) [95]. The following recursive form is given for real-time parameter estimation, which continuously performs the recursion when new data feeds in:

$$\hat{\boldsymbol{\theta}}(t) = \hat{\boldsymbol{\theta}}(t-1) + \boldsymbol{K}(t) \left(y(t) - \boldsymbol{\varphi}^T(t) \hat{\boldsymbol{\theta}}(t-1) \right),$$

where

$$\boldsymbol{K}(t) = \boldsymbol{P}(t-1)\boldsymbol{\varphi}(t) \left(\lambda \boldsymbol{I} + \boldsymbol{\varphi}^T(t) \boldsymbol{P}(t-1) \boldsymbol{\varphi}(t) \right)^{-1}$$

and

$$\boldsymbol{P}(t) = \left(\boldsymbol{I} - \boldsymbol{K}(t) \boldsymbol{\varphi}^T(t) \right) \boldsymbol{P}(t-1)/\lambda,$$

t is the recursion index, $\boldsymbol{P}(t)$ is usually defined as the covariance matrix, and $\boldsymbol{K}(t)$ is the update gain. The algorithm can be generally considered as a gradient-based method where the gradient is derived from the squared loss function [46]. The performance of the RLS algorithm has a strong relation with the forgetting factor λ. This tuning parameter operates to shift weights from the old data to the new data, which leads to a compromise between the estimation accuracy, misadjustment, and stability [69].

3.6 Communication Graphs

Communication graphs can provide natural abstractions for how information is shared between agents in a network and are widely used to quantify or simplify the many moving parts of dynamic systems [62]. The graph-based communication network contains a high-level description of the network topology in terms of objects referred to as vertices and edges [62]. The graph communication model is needed in the multi-agent system so that the set of agents can follow protocols to reach consensus. A communication graph can be defined as follow: a set of n agents connected by a directed/undirected communication graph (digraph) denoted by $G = (V, A, E)$. The directed graph has a set of agents (vertices) V of cardinality $|V| = n$, an adjacency matrix A, and a set of edges E. The edges represent the established connections between the agents. The adjacency matrix A indicates the connection degrees of all edges.

Consensus algorithms are commonly used in graph communication-based applications. They can help to control a set of agents reaching identical values, such as aligning their speeds or headings. The existing consensus algorithms are generally divided into two categories: consensus without a leader and consensus with a leader. Consensus with a leader is also considered as leader-follower consensus or distributed tracking [51].

Let $x_i \in \mathbb{R}$ be a value passed on by agent i to other agents in the effort to reach a consensus. Two agents i and j are said to be in agreement if and only if $x_i = x_j$ [34]. The set of agents of a communication graph is in consensus if and only if $x_i = x_j$, $\forall i, j \in 1, ..., n$, where $i \neq j$ [34]. A first order consensus protocol follows

$$u_i = \sum_{j=n}^{n} a_{ij}(x_j - x_i), \tag{3.2}$$

where the value $u_i \in \mathbb{R}$ is the control input applied to the agent i, and $a_{ij} \in A$ denotes the connection weights between i and j.

3.7 Summary

In this chapter, a background introduction is given to the techniques on which this research is founded, including ADP, RL, fuzzy logic, RLS algorithm and the communication graph.

ADP provides the solutions to discrete-time multistage stochastic control problems and RL as a machine learning mechanism can be used to implement different ADP solutions. It

enables the agents to study and optimize their strategies to solve decision-making problems within an unknown dynamic environment. Actor-critic neural networks are often utilized to approximate the unknown value function and assess the quality of the action taken in RL. They can be considered as approximation tools in RL systems. RLS provides an alternative way of parameter estimation, which can be used to realize the Policy Iteration mechanism in an ADP framework. Communication graphs are commonly employed in multi-agent systems for the information sharing purpose. It plays a significant role when guiding a flock of agents to reach certain types of consensus.

Chapter 4

Problem Formulation and Control Structure

This chapter formally defines the flocking control problem and specifies its objectives. The dynamic equations of a moving agent are introduced and explained in Section 4.1. The flocking control objectives that are also considered as constraints are explained in detail and demonstrated with mathematical equations. The control decision taken by each agent consists of competing signals that are corresponding to each of the constraints. Each control signal is introduced with its responsibility of control in Section 4.2. Each agent makes its overall control decision from the control signals which are based on its current situation to meet all the specified objectives.

4.1 The Flocking Control Problem

Flocking control problems can differ from their specified objectives among various situations. In this thesis, three typical objectives that are commonly addressed in flocking control are discussed and achieved. These objectives in the flocking control problem are leader tracking, collision avoidance, and speed consensus.

To formally define the cooperative control problem, let us consider a flock of N agents navigating on a 2D plane. The agents communicate over an undirected graph which can be either fully or partially connected with equal or different edge weights. A fully connected graph is a graph where each pair of agents (nodes) can communicate directly between each other through only one edge (communication channel), otherwise the graph is called

partially connected. The overall goal here is to guide the flock to satisfy a set of constraints or objectives. The three specific objectives will be demonstrated in detail later. Note that the leader is not limited to be a physical agent. It may represent a virtual time-dependent signal or trajectory that would command the agents' behavior, for example. The problem can be cast in a trajectory-tracking or follower-leader framework, promoting useful applications in fields such as UAV monitoring and surveillance, area coverage, and mobile sensor networks.

Since the proposed system focuses on surface mobile flock control, the agents are set to navigate in a 2D plane. The motion of each agent i is governed by

$$\begin{aligned} \boldsymbol{p}_{k+1}^i &= \boldsymbol{p}_k^i + T\,\boldsymbol{q}_k^i \\ \boldsymbol{q}_{k+1}^i &= \boldsymbol{q}_k^i + T\,\boldsymbol{u}_k^i \end{aligned}$$

where $\boldsymbol{p}_k^i = [x_k^i \;\; y_k^i]^T \in \mathbb{R}^2$ and $\boldsymbol{q}_k^i = [v_k^{i,x} \;\; v_k^{i,y}]^T \in \mathbb{R}^2$ denote the position and linear velocity, respectively, of agent $i \in \{1, 2, \ldots, N\}$ at discrete-time index $k \in \mathbb{N}$, T is the sampling time period, and $\boldsymbol{u}_k^i = [u_k^{i,x}, u_k^{i,y}]^T \in \mathbb{R}^2$ is the control signal commanding the linear acceleration in the x- and y-directions. The control signal of each agent is calculated by the proposed control algorithm, except the one for the leader $\ell \in \{1, \ldots, N\}$, which is generated by an independent command generator. The leader is considered as the tracking object by the followers while they do not have any information about the leader's future behavior. Therefore, each agent is only able to get location data of its tracking object from the communication graph for the current time step. The location measurements of the flock are set to be available for each agent assuming a fully or partially connected undirected communication graph.

The aim of the flock is to satisfy the required constrains simultaneously. Recall the constrains or objectives are

1. Tracking the leader.
2. Keeping a safe distance from the neighboring agents.
3. Reaching a velocity consensus among all the agents.

The latter requirement involves an information layer among the agents. The agents have to manage the occasionally conflicting nature of such goals to reach the best possible compromise. They collectively search for a decision making policy that balances the local and team objectives. The local objective for each agent is to avoid colliding with its neighbors while staying close to the flock's focus area; while the team objectives involve

reaching consensus on velocity for the flock and successfully following the leader. Thus, the above three objectives can be formally described by the following respective relations for each agent $i \in \{1, \dots, N\} \setminus \{\ell\}$:

$$\lim_{k\to\infty} \left\| \boldsymbol{p}_k^i - \boldsymbol{p}_k^\ell \right\| = 0 \tag{4.1a}$$

$$\lim_{k\to\infty} \left| \zeta_k^i - \zeta_k^j \right| \geq d, \ \forall \zeta \in \{x, y\}, \ \forall j \in \mathcal{N}_i \tag{4.1b}$$

$$\lim_{k\to\infty} v_k^{i,\zeta} = v^{*,\zeta} \tag{4.1c}$$

where $\mathcal{N}_i$ denotes the set of neighboring agents to agent i, d is a safety distance that each agent should try to maintain from its neighbors, $v^{*,\zeta}$ is a consensus speed that all the agents are expected to align with, and $\zeta \in \{x, y\}$ refers to either the x or y component.

4.2 Control Structure

The control signal of each agent generated from the proposed control system is composed of three (possibly conflicting) decisions reflecting the different objectives. The overall control command for each agent $i \in \{1, \dots, N\} \setminus \{\ell\}$ is formulated as an aggregate of three auxiliary signals, as follows

$$u_k^{i,\zeta} = u_{t,k}^{i,\zeta} + u_{s,k}^{i,\zeta} + u_{c,k}^{i,\zeta}. \tag{4.2}$$

The control structure is expected to reach a balance between all the three objectives to meet the goal.

The first term $u_{t,k}^{i,\zeta}$ represents the tracking or navigation control signal, whose goal is to guide the agents to satisfy global objective (4.1a). It is defined as a function of the agent's and leader's positions, $u_{t,k}^{i,\zeta} = f_{t,\zeta}^i(\zeta_k^i, \zeta_k^\ell)$, $\zeta \in \{x, y\}$, for some function $f_{t,\zeta}^i$, which handles the conflicts pertaining to these positions. By applying only this signal, the agent should eventually track the leader.

The separation or collision avoidance control $u_{s,k}^{i,\zeta}$, of agent i, is applied to avoid colliding with any agent j in its neighbourhood $\mathcal{N}_i$ along the ζ direction. This is accomplished by maintaining a safety distance from its neighboring agents, and so satisfying local objective (4.1b). The control signal depends on the agent's position relative to its neighbours.

It is set as

$$u_{s,k}^{i,\zeta} = f_{s,\zeta}^{i}(\zeta_k^i, \zeta_k^j) = \frac{\sum_{j\in\mathcal{N}_i} u_{s,k}^{ij,\zeta}}{|\mathcal{N}_i|} \tag{4.3}$$

for some function $f_{s,\zeta}^{i}$, where $|\mathcal{N}_i|$ denotes the cardinality of $\mathcal{N}_i$ and $u_{s,k}^{ij,\zeta}$ refers to the opinion or partial separation control signal taken by agent i towards each agent $j \in \mathcal{N}_i$. More details about the calculation of this signal is provided later in this chapter.

The last objective (4.1c), is a global objective which pertains to guiding each of the agents to reach consensus on a common velocity. This is achieved by the consensus control signal $u_{c,k}^{i,\zeta} = f_{c,\zeta}^{i}(v_k^{i,\zeta}, v_k^{j,\zeta})$, $\forall j \in \mathcal{N}_i$, for some function $f_{c,\zeta}^{i}$, which acts on the velocities of the agent i and each of its neighbouring agents j.

The block diagram shown in Figure 4.1 illustrates the aggregate control scheme followed by each agent $i \in \{1, \ldots, N\} \setminus \{\ell\}$. At each time step, the system calculates the three auxiliary control signals in (4.2) from the flock's current state and sends the aggregate signal to agent i. After applying the control command, agent i feeds its new state back to the system to keep the flock's state updated.

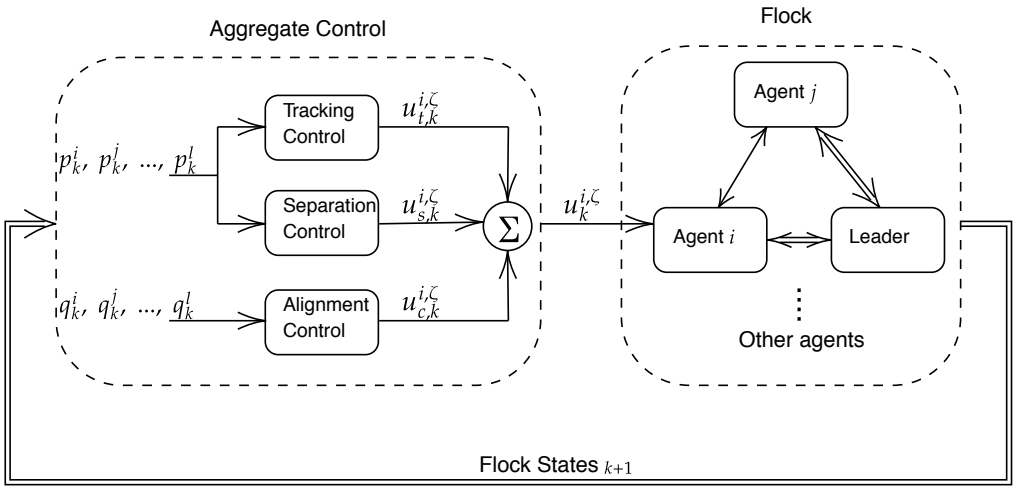


Figure 4.1: Aggregate control scheme followed by each agent $i \in \{1, \ldots, N\} \setminus \{\ell\}$

4.3 Summary

The general concept of the flocking control problem is described in this context by controlling a flock of agents to meet three objectives: leader tracking, collision avoidance, and speed consensus. The leader is not limited to a physical agent but may also represent a virtual time-dependent signal or a virtual trajectory. The control structure has its main control function composed of three auxiliary signals. Each auxiliary signal corresponds to an objective to be achieved. The output of the function provides an overall control signal for each agent to meet all the objectives as closely as possible.

Chapter 5

Value Iteration and Fixed Communication Graph Approach

This chapter demonstrates the complete computations of each control signal through three techniques: a value iteration approach in RL using critic-actor structure, an adaptive FIS, and fixed communication graph topology. They are addressed in Sections 5.1 to 5.3. This Fuzzy-RL adaptive learning scheme is simulated with two different scenarios to validate its performance. In Section 5.4.1, initial settings including agent velocity, position and speed limitations are listed. The reward and fuzzy membership functions of the adaptive fuzzy system are also introduced in the section with mathematical equations. Other simulation parameters, such as learning parameters, matrix and sampling time are also listed. Sections 5.4.2 and 5.4.3 examine the performance of the two scenarios while benchmarking the Fuzzy-RL against a similar technique reported in the literature.

5.1 Tracking Control Mechanism - Value Iteration

This section details the architecture of the Value Iteration tracking control mechanism which is responsible for the computation of the tracking signal $u_{t,k}^{i,\zeta}$. In Section 5.1.1, a RL structure is proposed to provide an optimal solution framework for the problem. Then a Value Iteration approach is advised to solve a set of coupled Bellman optimality equations in real-time. This is accomplished to adapt tracking strategies in a distributed fashion without any prior knowledge of the agents' dynamics. In Section 5.1.2, an actor-critic structure based on neural network is employed to implement the Value Iteration approach.

5.1.1 Optimization Framework

In order to achieve the objective of tracking the leader, an RL framework is built to provide the solution for this optimization problem. The key to constructing the framework is the derivation of the temporal difference equations (Bellman equations) along with the required optimality conditions. The ultimate objective of the framework is to converge to the best policies or control strategies for each follower so that the leader is tracked.

To that end, let $\boldsymbol{E}_k^{i,\zeta} = [\zeta_k^i - \zeta_k^\ell,\ \zeta_{k-1}^i - \zeta_{k-1}^\ell,\ \zeta_{k-2}^i - \zeta_{k-2}^\ell]^T \in \mathbb{R}^3$, for $\zeta \in \{x, y\}$, be an error signal vector corresponding to agent i at time index k, representing a time window of the last three tracking error measurements between the agent's position and that of the leader along the x- and y-directions. The choice of the error window size of $\boldsymbol{E}_k^{i,\zeta}$ depends on the complexity of the problem to solve. In our case, a window containing the last three error measurements happened to provide a satisfactory compromise between accuracy and computational complexity. In order for each agent i to measure the quality of its tracking control strategy $u_{t,k}^{i,\zeta}$, a performance index $\chi^{i,\zeta} = \sum_{k=0}^{\infty} U_k^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta})$ is proposed, where $U_k^{i,\zeta}$ is an objective function that is designed to be quadratic and convex

$$U_k^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta}) = \frac{1}{2}\left[\boldsymbol{E}_k^{i,\zeta T}\boldsymbol{Q}^i\boldsymbol{E}_k^{i,\zeta} + R^i\left(u_{t,k}^{i,\zeta}\right)^2\right] \tag{5.1}$$

where $\mathbf{0} < \boldsymbol{Q}^i \in \mathbb{R}^{3\times 3}$ and $0 < R^i \in \mathbb{R}$ are some weighting factors corresponding to the tracking error vector and the control signal, respectively. When applied to matrices, the notations "$> \mathbf{0}$" and "$\geq \mathbf{0}$" denote positive definite and positive semi-definite matrices, respectively.

Motivated by the structure of the performance index $\chi^{i,\zeta}$ and cost function (5.1), a solving value function $V^{i,\zeta}$ is formulated as

$$V^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta}) = \frac{1}{2}\begin{bmatrix} \boldsymbol{E}_k^{i,\zeta T} & u_{t,k}^{i,\zeta} \end{bmatrix}\boldsymbol{H}^{i,\zeta}\begin{bmatrix} \boldsymbol{E}_k^{i,\zeta} \\ u_{t,k}^{i,\zeta} \end{bmatrix}$$

such that,

$$\boldsymbol{H}^{i,\zeta} \equiv \begin{bmatrix} \boldsymbol{H}^{i,\zeta}_{\boldsymbol{E}^{i,\zeta}\boldsymbol{E}^{i,\zeta}} & \boldsymbol{H}^{i,\zeta}_{\boldsymbol{E}^{i,\zeta}u_t^{i,\zeta}} \\ \boldsymbol{H}^{i,\zeta}_{u_t^{i,\zeta}\boldsymbol{E}^{i,\zeta}} & \boldsymbol{H}^{i,\zeta}_{u_t^{i,\zeta}u_t^{i,\zeta}} \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

where $\boldsymbol{H}^{i,\zeta} > \mathbf{0}$, $\boldsymbol{H}^{i,\zeta}_{u_t^{i,\zeta}u_t^{i,\zeta}} \in \mathbb{R}$, and $\boldsymbol{H}^{i,\zeta}_{u_t^{i,\zeta}\boldsymbol{E}^{i,\zeta}} \in \mathbb{R}^{1\times 3}$. Note that the structure of the solving value function is a significant factor for an RL algorithm to decide on the best strategies [5].

With the chosen quadratic structure, the leader-follower optimization problem is reduced to finding the optimal solving value functions and consequently computing the optimal control gain using Bellman optimality principles.

The solving value function results in the following temporal difference (Bellman) equation

$$V^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta}) = U_k^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta}) + V^{i,\zeta}(\boldsymbol{E}_{k+1}^{i,\zeta}, u_{t,k+1}^{i,\zeta}). \tag{5.2}$$

Applying the optimality principle by taking $\arg\min_{u_{t,k}^{i,\zeta}} \left(V^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta})\right)$ yields the optimal strategy-to-follow that is given by

$$u_{t,k}^{i,\zeta(o)} = -\left(\boldsymbol{H}_{u_t^{i,\zeta} u_t^{i,\zeta}}^{i,\zeta}\right)^{-1} \boldsymbol{H}_{u_t^{i,\zeta} \boldsymbol{E}^{i,\zeta}}^{i,\zeta} \boldsymbol{E}_k^{i,\zeta}. \tag{5.3}$$

The optimization process aims at minimizing the performance index $\chi_0^{i,\zeta}$ by optimizing some learning weight of the RL system. This iterative process can lead the obtained policy to practically converge towards the optimal policy although it may not reach the theoretical value. Employing the optimal policy in Bellman equation yields the Bellman optimality relation

$$V^{i,\zeta(o)}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta(o)}) = U_k^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta(o)}) + V^{i,\zeta(o)}(\boldsymbol{E}_{k+1}^{i,\zeta}, u_{t,k+1}^{i,\zeta(o)}) \tag{5.4}$$

This equation is solved simultaneously by each agent i so that it can eventually converge to an optimized tracking policy that would enable it to follow the leader.

In this work, a two-step technique, known as Value Iteration, is applied to solve the Bellman optimality (5.4) using the optimal policy (5.3) [94]. This is achieved by iteratively evaluating the Bellman optimality

$$V^{i,\zeta(r+1)}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta}) = U_k^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, u_{t,k}^{i,\zeta(r)}) + V^{i,\zeta(r)}(\boldsymbol{E}_{k+1}^{i,\zeta}, u_{t,k+1}^{i,\zeta(r)})$$

and the optimal strategy

$$u_{t,k}^{i,\zeta(r+1)} = -\left(\boldsymbol{H}_{u_t^{i,\zeta} u_t^{i,\zeta}}^{i,\zeta(r)}\right)^{-1} \boldsymbol{H}_{u_t^{i,\zeta} \boldsymbol{E}^{i,\zeta}}^{i,\zeta(r)} \boldsymbol{E}_k^{i,\zeta} \tag{5.5}$$

concurrently till convergence [5]. This procedure leads to a converging nondecreasing sequence of solving value functions $0 \leq V^{i,\zeta(0)} \leq V^{i,\zeta(1)} \leq \cdots \leq V^{i,\zeta(r)} \leq \cdots \leq V^{i,\zeta(o)}$.

5.1.2 Neural Network Approximation

In online adaptive control problems with continuous state-action space, such as the one in hand, it is impossible to compute the optimal value function defined in (5.4) and the optimal tracking strategy (5.3). To mitigate this problem, both quantities are approximated. An actor-critic approach is employed by each agent to execute the Value Iteration process. The actor is realized by a neural network approximator to estimate the optimal tracking strategy (5.3). Another neural network is designed to implement the critic which approximates the optimal value function (5.4).

The structures of the actor and critic approximators are motivated by the those of the optimal policy $u_{t,k}^{i,\zeta(o)}$ and optimal value function $V^{i,\zeta(o)}$. To this end, the optimal policy is approximated as $\hat{u}_{t,k}^{i,\zeta} = \boldsymbol{\omega}^{i,\zeta}\boldsymbol{E}_k^{i,\zeta}$, where $\boldsymbol{\omega}^{i,\zeta} \in \mathbb{R}^{1\times 3}$ is the actor approximation weight vector. Similarly, the optimal value function $V^{i,\zeta(o)}$ is estimated by

$$\hat{V}^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, \hat{u}_{t,k}^{i,\zeta}) = \frac{1}{2}\begin{bmatrix} \boldsymbol{E}_k^{i,\zeta^T} & \hat{u}_{t,k}^{i,\zeta} \end{bmatrix} \boldsymbol{\Omega}^{i,\zeta} \begin{bmatrix} \boldsymbol{E}_k^{i,\zeta} \\ \hat{u}_{t,k}^{i,\zeta} \end{bmatrix} \tag{5.6}$$

such that,

$$\boldsymbol{\Omega}^{i,\zeta} \equiv \begin{bmatrix} \boldsymbol{\Omega}^{i,\zeta}_{\boldsymbol{E}^{i,\zeta}\boldsymbol{E}^{i,\zeta}} & \boldsymbol{\Omega}^{i,\zeta}_{\boldsymbol{E}^{i,\zeta}\hat{u}_t^{i,\zeta}} \\ \boldsymbol{\Omega}^{i,\zeta}_{\hat{u}_t^{i,\zeta}\boldsymbol{E}^{i,\zeta}} & \boldsymbol{\Omega}^{i,\zeta}_{\hat{u}_t^{i,\zeta}\hat{u}_t^{i,\zeta}} \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

where $\boldsymbol{\Omega}^{i,\zeta} > \mathbf{0}$, $\boldsymbol{\Omega}^{i,\zeta}_{\hat{u}_t^{i,\zeta}\hat{u}_t^{i,\zeta}} \in \mathbb{R}$, and $\boldsymbol{\Omega}^{i,\zeta}_{\hat{u}_t^{i,\zeta}\boldsymbol{E}^{i,\zeta}} \in \mathbb{R}^{1\times 3}$.

A gradient descent approach is applied for the online training of the neural networks. The target approximation values of the optimal strategy and the optimal value function can be expressed, respectively, as

$$\begin{aligned} \tilde{u}_{t,k}^{i,\zeta} &= -\left(\boldsymbol{\Omega}^{i,\zeta}_{\hat{u}_t^{i,\zeta}\hat{u}_t^{i,\zeta}}\right)^{-1} \boldsymbol{\Omega}^{i,\zeta}_{\hat{u}_t^{i,\zeta}\boldsymbol{E}^{i,\zeta}} \boldsymbol{E}_k^{i,\zeta} \\ \tilde{V}_k^{i,\zeta} &= U_k^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, \hat{u}_{t,k}^{i,\zeta}) + \hat{V}^{i,\zeta}(\boldsymbol{E}_{k+1}^{i,\zeta}, \hat{u}_{t,k+1}^{i,\zeta}) \end{aligned}$$

As a result, the actor and critic approximation errors are, respectively,

$$\begin{aligned} \varepsilon_{t,k}^{i,\zeta(actor)} &= \frac{1}{2}\left(\hat{u}_{t,k}^{i,\zeta} - \tilde{u}_{t,k}^{i,\zeta}\right)^2 \\ \varepsilon_{t,k}^{i,\zeta(critic)} &= \frac{1}{2}\left(\hat{V}^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, \hat{u}_{t,k}^{i,\zeta}) - \tilde{V}_k^{i,\zeta}\right)^2 \end{aligned}$$

Applying the gradient along the direction minimizing the errors yields the following weight update laws

$$\boldsymbol{\omega}^{i,\zeta(r+1)} = \boldsymbol{\omega}^{i,\zeta(r)} - \rho_a \left(\varepsilon_{t,k}^{i,\zeta(actor)} \boldsymbol{E}_k^{i,\zeta} \right)^{(r)} \tag{5.7a}$$

$$\boldsymbol{\Omega}^{i,\zeta(r+1)} = \boldsymbol{\Omega}^{i,\zeta(r)} - \rho_c \left(\varepsilon_{t,k}^{i,\zeta(critic)} \boldsymbol{Z}_{t,k}^{i,\zeta^T} Z_{t,k}^{i,\zeta} \right)^{(r)} \tag{5.7b}$$

where $\boldsymbol{Z}_{t,k}^{i,\zeta} = [\, \boldsymbol{E}_k^{i,\zeta^T} \;\; \hat{u}_{t,k}^{i,\zeta} \,]$, r is the iteration index of the weight update loop, and $0 < \rho_a, \rho_c < 1$ are the learning rates of the actor and critic, respectively. It is important to notice that the designed RL algorithm does not use any information about the dynamics of the agents. The strategies followed by each follower depend only on its own location measurements and those of the leader.

5.2 Separation Control Mechanism

The separation control protocol aims to prevent agents from colliding with each other by imposing a safety distance between the agents. The RL protocol is designed such that it penalizes agents that are closer to or further from each other than they should. Each agent i bases its decisions on on its proximity from its neighboring agents $j \in \mathcal{N}_i$, as described by (4.3). The mechanism is realized through a zero-order TS FIS with an online adaptation capability based on an actor-critic scheme. The synergistic integration of fuzzy logic and connectionist modeling theories has been exploited in the past to provide an approximate dynamic programming solution of the separation control problem [8, 39, 97]. The main merit of such an adaptive fuzzy-RL scheme is its ability to approximate the highly nonlinear system's input-output relationship, using attractive-repulsive potential functions, on which it builds its optimized policy. Furthermore, it solves the granularity issue associated to the output space of classical RL approaches by converting it from a discrete action space to a continuous one.

5.2.1 Zero-Order TS Fuzzy Inference System

One of the most salient features of fuzzy logic is its ability to control complex systems characterized by dynamic uncertainties without the need for their precise mathematical models. It does so by incorporating human-like expertise, in the form of if-then rules for the control of such ill-defined systems. Without loss of generality, consider a zero-order TS

fuzzy system with A inputs, a single output and P rules. Then, each rule $p \in \{1, \ldots, P\}$ is of the form

$$\text{If } \theta_1 \text{ is } \Gamma_{m_1}^{(p)}, \ldots, \text{ and } \theta_A \text{ is } \Gamma_{m_A}^{(p)} \text{ , then } u^{(p)} = \phi^{(p)},$$

where input θ_a, $a \in \{1, \ldots, A\}$, is an antecedent fuzzy variable, $\Gamma_{m_a}^{(p)}$ is a linguistic label associated to fuzzy set m_a, $u^{(p)}$ is a consequent fuzzy variable, and $\phi^{(p)} \in \mathbb{R}$ is a singleton. The firing strength of rule p is computed by

$$\Psi^{(p)} = \left[\prod_{a=1}^{A} \eta^{\Gamma_{m_a}^{(p)}}(\theta_a)\right] \Bigg/ \sum_{p=1}^{P} \left(\prod_{a=1}^{A} \eta^{\Gamma_{m_a}^{(p)}}(\theta_a)\right) \tag{5.8}$$

where $\eta^{\Gamma_{m_a}^{(p)}}$ is a membership function associated to $\Gamma_{m_a}^{(p)}$. The fuzzy engine's inferenced output is computed through a defuzzification process as

$$u^{\text{Fuzzy}} = \sum_{p=1}^{P} \Psi^{(p)} \phi^{(p)} \tag{5.9}$$

5.2.2 Adaptive Critics Fuzzy System

A zero-order TS fuzzy system is adopted by each agent i to compute a separation signal $u_{s,k}^{ij,\zeta}$ to control its proximity to each agent $j \in \mathcal{N}_i$. The fuzzy engine takes a single input fuzzy variable $\tilde{\zeta}_k^{ij} = \left|\zeta_k^i - \zeta_k^j\right| - d$, where d is the desired safety distance and $\zeta \in \{x, y\}$. With such a single-input single-output fuzzy system, rule p becomes of the form

$$\text{If } \tilde{\zeta}_k^{ij} \text{ is } \Gamma_{m_1}^{(p)} \text{ , then } u_{s,k}^{ij,\zeta(p)} = \phi^{ij,\zeta(p)}$$

The firing strength $\Psi_k^{ij,\zeta(p)}$ of rule p and the defuzzified output $u_{s,k}^{ij,\zeta}$ of the fuzzy system are computed as specified by (5.8) and (5.9), respectively, for $A = 1$.

In order to design an adaptive law to tune the consequent membership functions $\phi^{ij,\zeta(p)}$ of each rule p, the quality of the taken actions are assessed using a value function, which is approximated by a critic neural network

$$S_k^{ij,\zeta} = \sum_{p=1}^{P} \Psi_k^{ij,\zeta(p)} \Phi^{ij,\zeta(p)}$$

with $\Phi^{ij,\zeta(p)}$ being the critic weight corresponding to rule p. A temporal difference equation can be formed in terms of the value function $S_k^{ij,\zeta}\left(\tilde{\zeta}_k^{ij}\right)$ and a reward function $\mathcal{R}_k^{ij,\zeta}$ as

$$S_k^{ij,\zeta}(\tilde{\zeta}_k^{ij}) = \mathcal{R}_k^{ij,\zeta} + S_{k+1}^{ij,\zeta}\left(\tilde{\zeta}_{k+1}^{ij}\right)$$

The learning process of the actor and critic weights rely on such temporal difference equations. The control objective here is to maximize instant rewards $\mathcal{R}_k^{ij,\zeta}$ based on the distance $\tilde{\zeta}_k^{ij}$, $\forall k$. Hence, the temporal difference error is represented by

$$\mathcal{T}_k^{ij,\zeta} = S_k^{ij,\zeta}(\tilde{\zeta}_k^{ij}) - \left(\mathcal{R}_k^{ij,\zeta} + S_{k+1}^{ij,\zeta}\left(\tilde{\zeta}_{k+1}^{ij}\right)\right)$$

The actor and critic weights are updated based on information about the temporal difference error $\mathcal{T}_k^{ij,\zeta}$ using the TS fuzzy system framework. This results in an adaptive fuzzy system where the consequences of the rules are continuously updated, until convergence is achieved, to cope with changes in the dynamic environment. Applying a gradient-based descent approach, the update law of the actor weights is then

$$\begin{aligned}
\phi^{ij,\zeta(p)(r+1)} &= \phi^{ij,\zeta(p)(r)} - \alpha_a \left(\operatorname{sign}\left(\mathcal{T}_k^{ij,\zeta}\right) \frac{\partial u_s^{ij,\zeta}}{\partial \phi^{ij,\zeta(p)}}\right)^{(r)} \\
&= \phi^{ij,\zeta(p)(r)} - \alpha_a \left(\operatorname{sign}\left(\mathcal{T}_k^{ij,\zeta}\right) \Psi^{ij,\zeta(p)}\right)^{(r)}
\end{aligned} \tag{5.10}$$

where $0 < \alpha_a < 1$ is a learning rate. Similarly, the adaptive law of the critic weights is derived from the temporal difference evaluation $\mathcal{T}_k^{ij,\zeta}$ as

$$\begin{aligned}
\Phi^{ij,\zeta(p)(r+1)} &= \Phi^{ij,\zeta(p)(r)} - \alpha_c \left(\left(\mathcal{T}_k^{ij,\zeta}\right) \frac{\partial S_k^{ij,\zeta}}{\partial \Phi^{ij,\zeta(p)}}\right)^{(r)} \\
&= \Phi^{ij,\zeta(p)(r)} - \alpha_c \left(\left(\mathcal{T}_k^{ij,\zeta}\right) \Psi^{ij,\zeta(p)}\right)^{(r)}
\end{aligned} \tag{5.11}$$

for a learning rate $0 < \alpha_c < 1$. This adaptive critics structure is implemented by each agent i in each direction $\zeta \in \{x, y\}$ to provide an aggregate separation policy $u_{s,k}^{i,\zeta}$ defined by

$$u_{s,k}^{i,\zeta} = \frac{\sum_{j\in\mathcal{N}_i}\sum_{p=1}^{P} \Psi_k^{ij,\zeta(p)} \phi^{ij,\zeta(p)}}{|\mathcal{N}_i|}. \tag{5.12}$$

It is important not to mix up the actor-critic approximators corresponding to the tracking and separation control mechanisms. Each of the two has its own actor-critic neural network structures.

5.3 Consensus Control Mechanism

The flock of agents is guided to reach a consensus on a common flock velocity using a communication layer designated by a graph topology. An undirected fully connected graph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ is adopted for this purpose, where $\mathcal{N} = \{\delta_i\}_{i=1,\dots,|\mathcal{N}|}$ is the set of nodes of cardinality $|\mathcal{N}|$ and $\mathcal{E} = \{(\delta_i, \delta_j) \in \mathcal{N}^2\}$ is the set of edges representing the communication links between the agents [29]. We will denote the connectivity weights associated to every edge $(\delta_i, \delta_j) \in \mathcal{E}$ by c_{ij}. Note that due to the undirected nature of the graph, $c_{ij} = c_{ji}$, $\forall j \neq i$, while $c_{ii} = 0$. The local consensus protocol followed by each agent i is set to

$$u_{c,k}^{i,\zeta} = -\sum_{j \in \mathcal{N}_i} c_{ij}(v_k^{i,\zeta} - v_k^{j,\zeta}). \tag{5.13}$$

Hence, the consensus control decisions for all the agents can be presented collectively by

$$\boldsymbol{u}_{c,k}^{\zeta} = -\boldsymbol{L}\boldsymbol{v}_k^{\zeta},$$

where $\boldsymbol{u}_{c,k}^{\zeta} = [\, u_{c,k}^{1,\zeta} \;\; u_{c,k}^{2,\zeta} \;\; \dots \;\; u_{c,k}^{N,\zeta} \,]^T$, $\boldsymbol{v}_k^{\zeta} = [\, v_k^{1,\zeta} \;\; v_k^{2,\zeta} \;\; \dots \;\; v_k^{N,\zeta} \,]^T$, and $\boldsymbol{L}$ is the graph Laplacian. Let $\boldsymbol{C} = [c_{ij}] \in \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ be the graph's adjacency matrix and $\boldsymbol{C}^d = [c_{ii}^d] \in \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ be a square diagonal matrix where

$$c_{ii}^d = \sum_{j \in \mathcal{N}_i} c_{ij}.$$

Then,

$$\boldsymbol{L} = \boldsymbol{C}^d - \boldsymbol{C}.$$

The convergence speed of the consensus process to a common velocity is governed by the second eigenvalue associated with Fiedler Eigenvector of the graph Laplacian $\boldsymbol{L}$ [67]. As a result, the convergence is directly affected by the graph weights and topology.

The overall Value Iteration procedure is detailed in Algorithm 5.1.

Algorithm 5.1 Fuzzy-Reinforcement Learning Solution

Input:

Membership function $\eta^{\Gamma_{m_1}^{(p)}}$
Weighting matrices $\boldsymbol{Q}^i$ and R^i
Number of search iterations $\mathcal{T}_n$
Learning rates ρ_a, ρ_c, α_a, and α_c
Adjacency matrix $\boldsymbol{C}$ and graph Laplacian $\boldsymbol{L}$
Initial tracking error vector $\boldsymbol{E}_0^{i,\zeta}$ and distance $r_0^{ij(0)}$
Initial weights $\boldsymbol{\omega}^{i,\zeta(0)}, \boldsymbol{\Omega}^{i,\zeta(0)}, \phi^{ij,\zeta(p)^{(0)}}$, and $\Phi^{ij,\zeta(p)^{(0)}}$
Convergence error ξ and size of a moving window $\mathcal{W}$

Output:

Tuned weights $\boldsymbol{\omega}^{i,\zeta(*)}, \boldsymbol{\Omega}^{i,\zeta(*)}, \phi^{ij,\zeta(p)^{(*)}}$, and $\Phi^{ij,\zeta(p)^{(*)}}$

stop $\leftarrow$ false
$r \leftarrow 0$
RL_weights_converged $\leftarrow$ false
FZ_weights_converged $\leftarrow$ false
while $r < \mathcal{T}_n$ **and** stop = false **do**
    Calculate the strategies $\hat{u}_{t,k}^{i,\zeta(r)}, \hat{u}_{s,k}^{i,\zeta(r)}, u_{c,k}^{i,\zeta(r)}$
    Use (4.2) to get $u_k^{i,\zeta(r)}$ and then apply to the agent
    Find $U_k^{i,\zeta}(\boldsymbol{E}_k^{i,\zeta}, \hat{u}_{t,k}^{i,\zeta(r)})$ and $\hat{V}^{i,\zeta(r)}(\boldsymbol{E}_k^{i,\zeta}, \hat{u}_{t,k}^{i,\zeta(r)})$
    Calculate $r_k^{ij(r)}, \mathcal{R}_k^{ij,\zeta(r)}$, $\Psi^{ij,\zeta(p)}(r_k^{ij(r)})$, and $S_k^{ij,\zeta}(r_k^{ij(r)})$
    Compute $\boldsymbol{E}_{(k+1)}^{i,\zeta}, \hat{u}_{t,(k+1)}^{i,\zeta(r)}, \hat{u}_{s,(k+1)}^{i,\zeta(r)}$, and $u_{c,(k+1)}^{i,\zeta(r)}$
    Find $\hat{V}^{i,\zeta(r)}(\boldsymbol{E}_{(k+1)}^{i,\zeta}, \hat{u}_{t,(k+1)}^{i,\zeta(r)})$ and $S_{(k+1)}^{ij,\zeta}(r_{(k+1)}^{ij(r)})$
    Calculate $\varepsilon_{t,k}^{i,\zeta(actor)(r)}$, $\varepsilon_{t,k}^{i,\zeta(critic)(r)}$, and $\mathcal{T}_k^{ij,\zeta(r)}$
    $r \leftarrow r + 1$
    Update $\boldsymbol{\omega}^{i,\zeta(r)}, \boldsymbol{\Omega}^{i,\zeta(r)}, \phi^{ij,\zeta(p)^{(r)}}$, and $\Phi^{ij,\zeta(p)^{(r)}}$

(to be continued)

Algorithm 5.1 Fuzzy-Reinforcement Learning Solution (continued)

if $r > \mathcal{W}$ **then**
  if $\left\|\boldsymbol{\omega}^{i,\zeta\,(r+1-l)} - \boldsymbol{\omega}^{i,\zeta(r-l)}\right\|$ **and** $\left\|\boldsymbol{\Omega}^{i,\zeta\,(r+1-l)} - \boldsymbol{\Omega}^{i,\zeta(r-l)}\right\| \leq \xi, \forall l \in \{0, 1, \ldots, \mathcal{W}\}$, **then**
    $\boldsymbol{\omega}^{i,\zeta(*)} \leftarrow \boldsymbol{\omega}^{i,\zeta(r+1)}$
    $\boldsymbol{\Omega}^{i,\zeta(*)} \leftarrow \boldsymbol{\Omega}^{i,\zeta(r+1)}$
    RL_weights_converged $\leftarrow$ true
  end if
  if $\left\|\phi^{ij,\zeta(p)(r+1-l)} - \phi^{ij,\zeta(p)(r-l)}\right\|$ **and** $\left\|\Phi^{ij,\zeta(p)(r+1-l)} - \Phi^{ij,\zeta(p)(r-l)}\right\| \leq \xi,\ \forall l \in \{0, 1, \ldots, \mathcal{W}\}$, **then**
    $\phi^{ij,\zeta(p)(*)} \leftarrow \phi^{ij,\zeta(p)(r+1)}$
    $\Phi^{ij,\zeta(p)(*)} \leftarrow \Phi^{ij,\zeta(p)(r+1)}$
    FZ_weights_converged $\leftarrow$ true
  end if
end if
if RL_weights_converged **and** FZ_weights_converged **then**
  stop $\leftarrow$ true
end if
end while
return $\boldsymbol{\omega}^{i,\zeta(*)}, \boldsymbol{\Omega}^{i,\zeta(*)}, \phi^{ij,\zeta(p)(*)}$, and $\Phi^{ij,\zeta(p)(*)}$

5.4 Result and Discussion

5.4.1 Setup

The performance of the algorithm is validated through Matlab simulations where a total of 21 mobile agents are used. One of them is set as a leader which is commanded from an external independent source to navigate in a circular trajectory defined by

$$x_k^\ell = 5\cos(0.03k) \qquad\qquad y_k^\ell = 5\sin(0.03k)$$

The 20 other agents are followers which are distributively controlled by the Fuzzy-RL algorithm to achieve the goals casted by (4.1). The simulation of the flock of 21 agents is performed in a synchronous manner, in the sense that they all receive and apply their different command signals simultaneously, at a frequency of 1 kHz. The agents' initial positions and velocities are randomly selected within the ranges of $[-5, 5]$ m and $[0, 1]$ m/s,

respectively. The linear velocity and acceleration limits for each agent including the leader can be found in Table 5.1. Robots in this simulation are considered as omnidirectional particles with no orientation, inertia, or volume.

Table 5.1: Simulation Limits

Parameter	Value
Linear speed limit	$\pm 3\,\mathrm{m/s}$
Linear acceleration limit	$\pm 2\,\mathrm{m/s^2}$

A reward function $\mathcal{R}_k^{ij,\zeta}$ is designed to give the highest value for maintaining the target separation distance while penalizing the actions violating this constraint

$$\mathcal{R}_k^{ij,\zeta} = \begin{cases} -\tilde{\zeta}_k^{ij}, & \text{if } \tilde{\zeta}_k^{ij} > 0 \\ 3, & \text{if } \tilde{\zeta}_k^{ij} = 0 \\ \frac{3\tilde{\zeta}_k^{ij}}{d}, & \text{if } \tilde{\zeta}_k^{ij} < 0 \end{cases}$$

When $\tilde{\zeta}_k^{ij} > 0$ or $\tilde{\zeta}_k^{ij} < 0$, agent i is penalized by getting a negative reward because the separation distance is too far or too close from its neighbour agent j. It gets a positive reward value of 3 if the separation distance is kept right at the desired distance d.

The fuzzy logic engine is designed with triangular membership functions as follows:

$$\eta^{\Gamma_{m_1}}(\tilde{\zeta}_k^{ij}) = \begin{cases} 0, & \text{if } \tilde{\zeta}_k^{ij} < t_l \\ \frac{\tilde{\zeta}_k^{ij} - t_l}{t_c - t_l}, & \text{if } t_l \leq \tilde{\zeta}_k^{ij} \leq t_c \\ \frac{t_h - \tilde{\zeta}_k^{ij}}{t_h - t_c}, & \text{if } t_c \leq \tilde{\zeta}_k^{ij} \leq t_h \\ 0, & \text{if } \tilde{\zeta}_k^{ij} > t_h \end{cases} \tag{5.14}$$

Five symmetric membership functions designed from (5.14) are used in both of the simulation scenarios. They are LN, N, Z, P, and LP as demonstrated in Figure 5.1. As we can see from the figure, the membership functions are centered around $t_c \in \{-6, -3, 0, 3, 6\}$ with $t_l = t_h = 3$. The input universe of the fuzzy logic controller is set to $(-6, 6)$. If the input is outside this range, it gets capped to the end value of the universe. The variable $\tilde{\zeta}_k^{ij} = \left|\zeta_k^i - \zeta_k^j\right| - d$ is used as the single fuzzy control input, which means that $\tilde{\zeta}_k^{ij}$ is the only antecedent fuzzy variable for each fuzzy rule. The five fuzzy rules are designed as

follows, where the consequent of the fuzzy rules $\phi^{ij,\zeta(p)}$ is the actor learning weight:

$$\begin{aligned}
&\text{If } \left|\zeta_k^i - \zeta_k^j\right| - d \text{ is LN Then } u_{s,k}^{ij,\zeta(1)} = \phi^{ij,\zeta(1)} \\
&\text{If } \left|\zeta_k^i - \zeta_k^j\right| - d \text{ is N Then } u_{s,k}^{ij,\zeta(2)} = \phi^{ij,\zeta(2)} \\
&\text{If } \left|\zeta_k^i - \zeta_k^j\right| - d \text{ is Z Then } u_{s,k}^{ij,\zeta(3)} = \phi^{ij,\zeta(3)} \\
&\text{If } \left|\zeta_k^i - \zeta_k^j\right| - d \text{ is P Then } u_{s,k}^{ij,\zeta(4)} = \phi^{ij,\zeta(4)} \\
&\text{If } \left|\zeta_k^i - \zeta_k^j\right| - d \text{ is LP Then } u_{s,k}^{ij,\zeta(5)} = \phi^{ij,\zeta(5)}
\end{aligned}$$

These settings reflect the action and state spaces for each agent while the consequences of the fuzzy rules are decided online using the adaptive actor-critic structures.

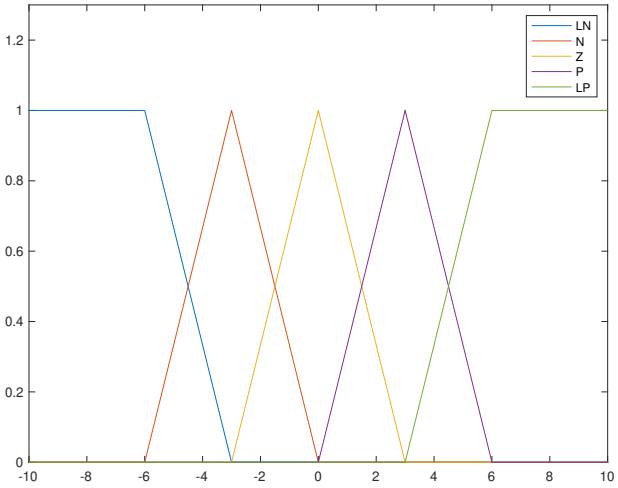


Figure 5.1: Symmetric Triangular Membership Function

The remaining simulation parameters are given in Table 5.2.

The neighborhood $\mathcal{N}_i$, for each follower i, is defined to be $\mathcal{N}_i = \{\delta_j\}_{j=1,\ldots,N} \setminus \{\delta_i, \delta_\ell\}$. This makes $|\mathcal{N}_i| = 19$, since $N = 21$. In order to quantitatively assess the performance of the proposed algorithm, the following objective measures are adopted: The average separation error for each follower i at time index k is measured as

$$O_{s,k}^i = \left[\sum_{j\in\mathcal{N}_i} \left(\left\|\boldsymbol{p}_k^j - \boldsymbol{p}_k^i\right\| - d\right)\right] \Big/ |\mathcal{N}_i|.$$

Table 5.2: Simulation Parameters

Parameter	Value	Parameter	Value
$\boldsymbol{Q}^i$	$I_{3\times3}$	R^i	1
ρ_c	10^{-7}	ρ_a	10^{-2}
α_a	0.1	α_c	0.05
T	0.1s	d	2m

The overall average tracking error, separation error, and velocity are measured respectively as

$$O_{t,k} = \left(\sum_{i\neq\ell} \left\|\boldsymbol{p}_k^i - \boldsymbol{p}_k^\ell\right\|\right)/(N-1),$$
$$O_{s,k} = \left(\sum_{i\neq\ell} O_{s,k}^i\right)/(N-1),$$
$$O_{v,k} = \left(\sum_{i\neq\ell} \left\|\boldsymbol{q}_k^i\right\|\right)/(N-1).$$

The tracking and separation error measurements consider an overall performance for all the agents except the leader. This is realized by taking the sum of all the followers' errors at time step k and then divided by the total number of followers which is $N-1$. A similar concept is applied when calculating the followers' average velocity.

5.4.2 Scenario # 1 - Circular Trajectory Tracking

Two test cases are considered in the evaluation of this algorithm. The first one assigns a mobile agent as a leader and commands it to navigate in a circular trajectory to examine the agents overall performance in compromising the competing objectives: tracking, separation and consensus. The results are shown in Figures 5.2(a) to 5.2(d). The performance is compared to a similar technique introduced by Gu et al. in [29]. Similar to the proposed method's structure, Gu's method also uses distributed control technique to achieve each of the objectives. For the leader tracking objective, linear tracking approach is used in Gu's method. Fuzzy control is applied to achieve the goal of collision avoidance. Note that the difference between this strategy and the proposed fuzzy system is that the latter

is compacted with RL critics scheme which brings adaptability and learning ability to the proposed system. The third objective is approached by Gu using the same strategy as in this thesis, which is consensus control policy. Gu's method is simulated with the exact same scenario to present a clear comparison to the proposed method.

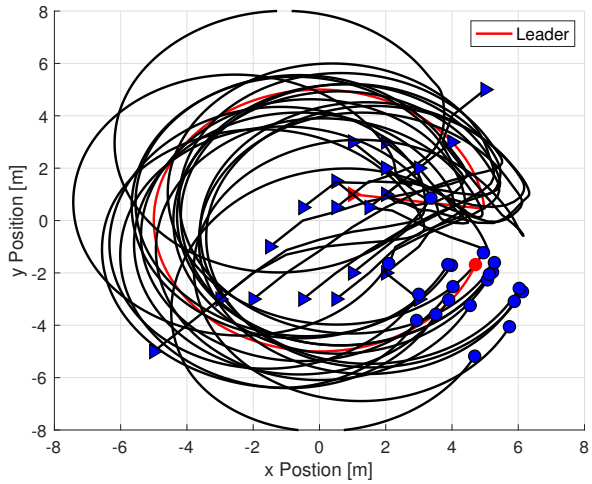
Fig. 5.2(a) reveals the paths of the agents, where the start and end locations are indicated by a triangle and a circle, respectively. The agents follow a circular trajectory while making a compromise to keep each agent apart from the other during the entire simulation time. The overall average flock-performances related to the tracking error, separation error, and velocity of the mobile agents and their standard deviations at each instance k are illustrated in Figs. 5.2(b), 5.2(c), and 5.2(d), represented by red lines and light red shaded areas, respectively against those obtained by [29], represented by green lines and light green shaded areas. As is evident from the tracking error plot in Fig. 5.2(b), the agents follow the leader keeping an almost constant overall average separation distance. This is well aligned with the ability of the adaptive learning scheme to preserve an overall average zero separation error as shown in Fig. 5.2(c).Finally, the agents reach a consensus on a common flock velocity as shown in Fig. 5.2(d).

It is interesting to notice that with the adaptive Fuzzy-RL method, not only do all the signals reach their respective steady states faster than with Gu's method, they do so within less than 2 s. Although this comes at the expense of a larger variation (standard deviation) for the average tracking and average separation errors, the agents reach a consensus on almost the exact same velocity with practically a nil standard deviation compared to Gu's method (Fig. 5.2(d)). The proposed method led to a significantly lower average tracking and average separation errors than Gu's controller. The average separation error with the adaptive Fuzzy-RL algorithm rapidly converged to practically zero, outperforming Gu's algorithm whose error converged to 1 m. This means that while maintaining the competing objectives, the desired overall separation is achieved among the agents.

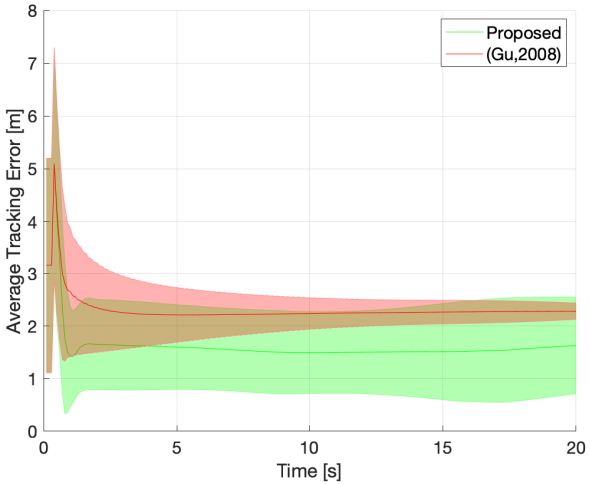
5.4.3 Scenario # 2 - Dynamic Missions

The second simulation is designed to test the online ability of the Fuzzy-RL controller to adapt to dynamic disturbances while the agents are in action. To that end, the flock starts off as in the first simulation, but with the following changes

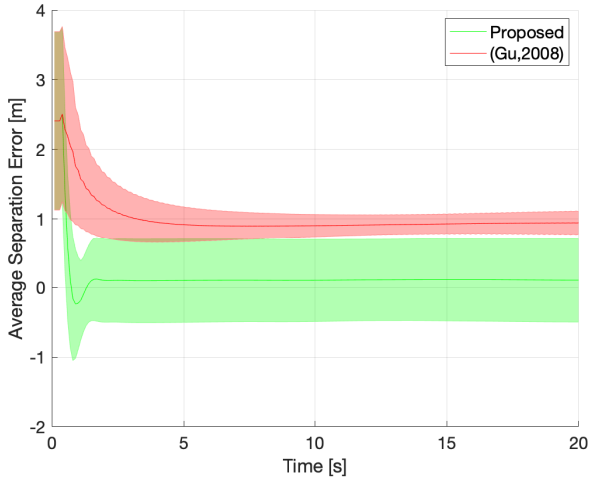
1. at time $t = 10\,\text{s}$, 4 of the 20 followers are decommissioned
2. at $t = 20\,\text{s}$, the leader changes its trajectory from a circular to a linear trajectory defined by $v_k^{\ell,x} = 1.7321$, $v_k^{\ell,y} = 1\,\text{m/s}$ (a linear motion with a heading of 30°)



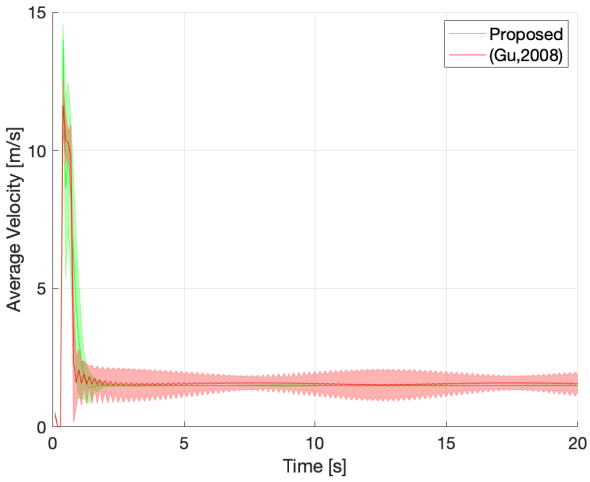
(a) Phase plane plot



(b) Average tracking error $O_{t,k}$



(c) Average separation error $O_{s,k}$



(d) Average follower velocity $O_{v,k}$

Figure 5.2: Simulation results of Scenario 1

3. at $t = 30\,\text{s}$, the safety distance d is changed from 2 to 2.5 m.

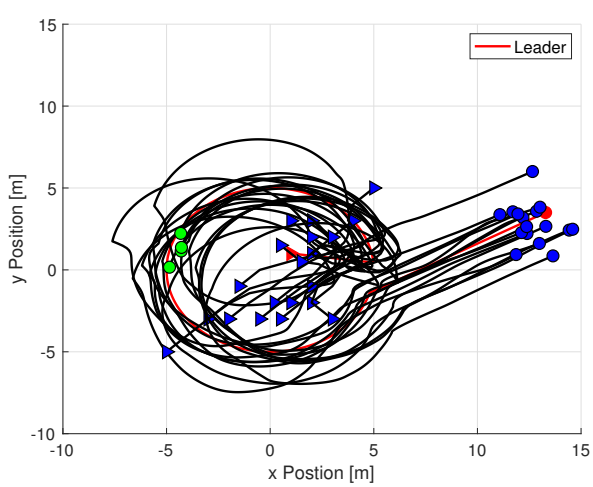
The first and third scenario modification aim to examine the control system's adaptability of maintaining same separation distance when the neighbourhood of some agents is suddenly changed, or a different desired separation distance is commanded during the mission. The followers are expected to quickly adjust the flock formation after the change happens and maintain the required separation distance. The second change of the scenario challenges the control system's tracking ability. With the uncertainty of the leader's dynamic motion, the followers should always be able to track and follow their target.

The results of this scenario are depicted in Figures 5.3(a) to 5.3(d). They are not compared with Gu's algorithm this time because it was not designed to be adaptive to dynamic variations. Its feedback policies are based on fixed control gains. The 4 agents decommissioned at $t = 10\,\text{s}$ are identified in green in Fig. 5.3(a). The figure demonstrates how the followers are able to globally track the leader regardless of its sudden change in trajectory or the number of active agents. The average tracking error, separation error, and velocity, rapidly converged right after each disturbance. Their standard deviations were not much affected by the changes in the simulation conditions. Nevertheless, the average separation error was the signal that is most affected by varying the safety distance from 2 to 2.5 m at $t = 30\,\text{s}$. It went from almost zero right before the change to about $-0.75\,\text{m}$ after it. On the other hand, Figure 5.3(d) shows that the standard deviation of the consensus velocity converges to an insignificant value after each disturbance.

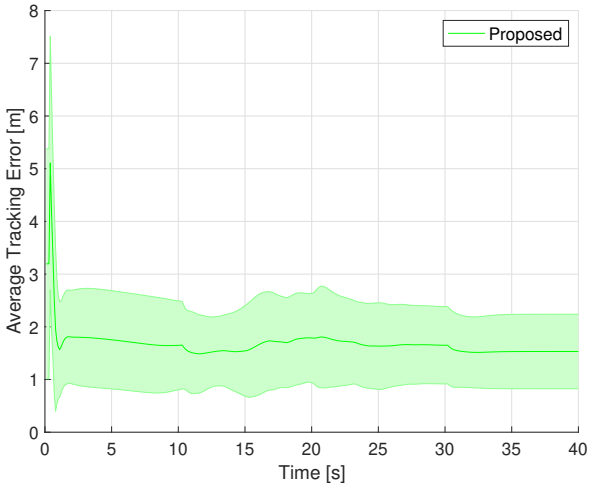
5.5 Summary

The neural network critic-actor approximator is used to compute the navigation signal for leader tracking through the value iteration methodology. The separation control system combines a fuzzy system with another critic-actor neural network to increase its adaptability. The consensus control signal is computed using a communication layer designated by a graph topology with fixed connectivity strength.

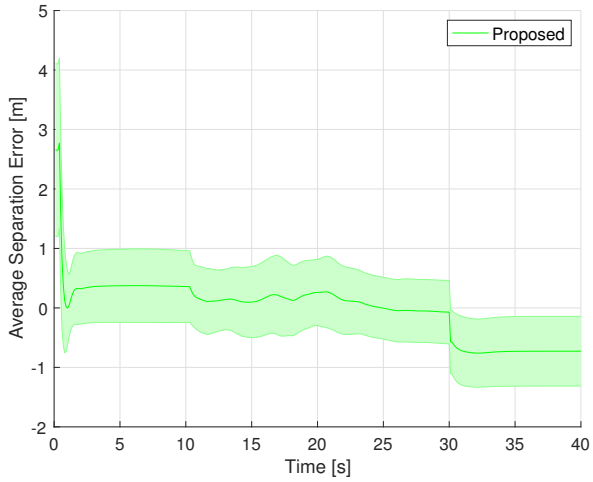
This chapter validates and demonstrates the Fuzzy-RL adaptive learning algorithm with two simulation scenarios that are implemented in Matlab. The first scenario requires the followers to track a simple circular motion and the result is compared with Gu's method. The proposed algorithm shows improved results. The second scenario involves multiple dynamic changes such as decommissioned followers and changed trajectory.



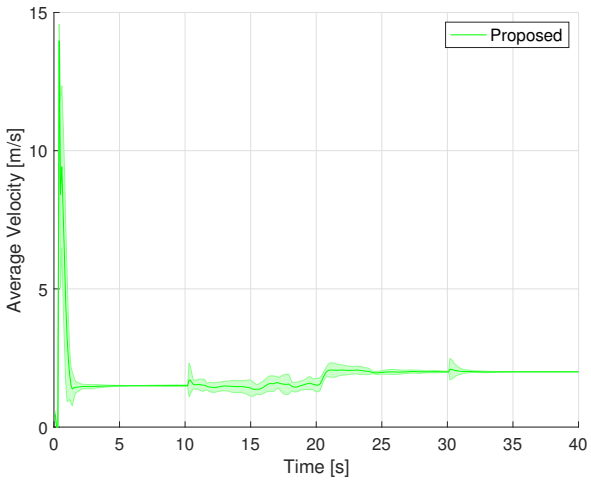
(a) Phase plane plot



(b) Average tracking error $O_{t,k}$



(c) Average separation error $O_{s,k}$



(d) Average follower velocity $O_{v,k}$

Figure 5.3: Simulation results of Scenario 2

Chapter 6

Policy Iteration Approach With Time-Varying Communication Graph

This chapter proposes a Fuzzy-RLS Policy Iteration approach for each agent to compute the optimal control signal among the three competing objectives as discussed in Equation (4.1). Using the new approach, the tracking control signal $u_{t,k}^{i,\zeta}$ and the consensus control signal $u_{c,k}^{i,\zeta}$ are generated differently than in Chapter 5. The tracking subsystem utilizes RLS methodology to implement the Policy Iteration techniques, and the consensus subsystem extended from Section 5.3 with time-varying edge weights improves the communication graph's connectivity, while the adaptive fuzzy system for the separation is kept the same as demonstrated in Section 5.2.

A realistic robotic simulator named "CoppeliaSim" is introduced in Section 6.3. The simulator is discussed with its characteristics, models, scenes, and API functions. It gives a better visualization of the proposed Fuzzy-RL algorithm's performance when simulating in a near real-world environment. Four simulation scenarios are designed and demonstrated in Sections 6.4.1 to 6.4.4 to validate the performance of the Fuzzy-RLS control approach while benchmarking against the Fuzzy-RL technique. Detailed discussions are provided on comparing the adaptability and stability between the RL Policy Iteration and Value Iteration techniques.

6.1 Tracking Control Mechanism - Policy Iteration

The section introduces another model-free Policy Iteration process by using RLS to compute the tracking control signal $u_{t,k}^{i,\zeta}$ for each agent i to satisfy objective (4.1a). This tracking mechanism is also accomplished in real-time without requiring the dynamics of the leader or any of the followers.

The Least Mean Squares (LMS) approximator is another possible candidate for computing the estimated policy. Unlike the RLS method, the LMS technique is non-iterative, more robust, less computationally complex, but is characterized by a higher space complexity. As such, the LMS cannot be applied in online control algorithms. It is worth mentioning that Kalman filter and its variants are typically applicable to estimate the state of the system and cannot be used for policy approximation, as it is the case for the RLS and LMS methods.

6.1.1 Recursive Least Square

The Policy Iteration solution is implemented in two steps. First, a given policy is evaluated. Second, the tracking strategy-to-follow is improved. Therefore, the RLS approach solves (5.2) for the optimal value $\boldsymbol{H}^{i,\zeta}$ or strategy using value function approximation since it is not possible to solve Bellman optimality equation analytically.

Let

$$\boldsymbol{\varphi}_k^{i,\zeta} = \bar{\boldsymbol{x}}_k^{i,\zeta} - \bar{\boldsymbol{x}}_{k+1}^{i,\zeta},$$

where

$$\bar{\boldsymbol{x}}^{i,\zeta} = [0.5z_1^2 \; z_1z_2 \; z_1z_3 \; z_1z_4 \; 0.5z_2^2 \; z_2z_3 \; z_2z_4 \; 0.5z_3^2 \; z_3z_4 \; 0.5z_4^2]$$

and z_1 to z_4 correspond to the elements of vector $[\,\boldsymbol{E}^{i,\zeta^T} \;\; \tilde{u}_t^{i,\zeta}\,]$ which is a part of the approximated value function (5.6). Also, let $\boldsymbol{\vartheta}^{i,\zeta} \in \mathbb{R}^{10\times 1}$ be the entries in the symmetric solution matrix $\boldsymbol{\Omega}^{i,\zeta}$ associated with the vector $\bar{\boldsymbol{x}}^{i,\zeta}$. Hence, it is required to solve the following equation

$$\boldsymbol{\varphi}_k^{i,\zeta}\boldsymbol{\vartheta}^{i,\zeta} = U_k^{i,\zeta}. \tag{6.1}$$

This is done using RLS technique for each agent i. Then, the approximated optimal strategy-to-follow is calculated by (5.5) after reconstructing $\boldsymbol{\Omega}^{i,\zeta}$ back from $\boldsymbol{\vartheta}^{i,\zeta}$, such that

the improved control strategy follows

$$\tilde{u}_{t,k}^{i,\zeta} = -\left(\boldsymbol{\Omega}_{\hat{u}_t^{i,\zeta}\hat{u}_t^{i,\zeta}}^{i,\zeta}\right)^{-1} \boldsymbol{\Omega}_{\hat{u}_t^{i,\zeta}\boldsymbol{E}^{i,\zeta}}^{i,\zeta} \boldsymbol{E}_k^{i,\zeta}$$

The RLS approach solves for the unknown weights $\boldsymbol{\vartheta}_k^{i,\zeta} \in \mathbb{R}^{10\times 1}$ in real-time as follows [10], without requiring any prior knowledge about the agents' dynamics

$$\begin{aligned}
\boldsymbol{\vartheta}_k^{i,\zeta} &= \boldsymbol{\vartheta}_{k-1}^{i,\zeta} + \boldsymbol{G}_k^{i,\zeta}\left(U_k^{i,\zeta} - \boldsymbol{\varphi}_k^{i,\zeta}\boldsymbol{\vartheta}_{k-1}^{i,\zeta}\right) \\
\boldsymbol{G}_k^{i,\zeta} &= \boldsymbol{P}_{k-1}^{i,\zeta}\boldsymbol{\varphi}_k^{i,\zeta^T}\left(1 + \boldsymbol{\varphi}_k^{i,\zeta}\boldsymbol{P}_{k-1}^{i,\zeta}\boldsymbol{\varphi}_k^{i,\zeta^T}\right)^{-1} \\
\boldsymbol{P}_k^{i,\zeta} &= \left(\boldsymbol{I} - \boldsymbol{G}_k^{i,\zeta}\boldsymbol{\varphi}_k^{i,\zeta}\right)\boldsymbol{P}_{k-1}^{i,\zeta}
\end{aligned}$$

where $\boldsymbol{G}^{i,\zeta} \in \mathbb{R}^{10\times 1}$ is a gain adaptation vector, $\boldsymbol{P}^{i,\zeta} \in \mathbb{R}^{10\times 10}$ is a covariance matrix, and $\boldsymbol{I}$ is the identity matrix. A Policy Iteration solution using the RLS method is shown in Algorithm 6.1. It is executed simultaneously by each agent i.

6.2 Consensus Control Mechanism - Time Varying

The objective of cohesiveness as expressed in (4.1c) is achieved using a similar technique as demonstrated in Section 5.3. The same consensus protocol (3.2) is applied but the connection strength $c_{i,j}$ which is also considered as the edge weight in this case is time-varied.

The graph connectivity weights are decided using a scalar pump function $\gamma_a(\zeta^{ij})$, such that

$$\gamma_a(\zeta^{ij}) = \begin{cases} 1 & ,\zeta^{ij} \in [0, a) \\ \frac{1}{2}\left[1 + \cos\left(\pi\frac{\zeta^{ij}-a}{r-a}\right)\right] & ,\zeta^{ij} \in [a, r) \\ 0 & ,\text{otherwise} \end{cases}$$

where $\zeta^{ij} = \zeta^i - \zeta^j$ is the relative distance between agents i and j. Note that the output of the scalar pump function also takes into account a connectivity communication range r between the agents, which somewhat contributes to the separation control objective [66]. The edge or connectivity weights are calculated as

$$c_{ij}(\zeta^{ij}) = \gamma_a\left(\left\|\zeta^{ij}\right\|_\alpha / \|r\|_\alpha\right) \in [0, 1),\ j \neq i.$$

Algorithm 6.1 RLS-Based PI Tracking Mechanism

Input:

Weighting matrices $\boldsymbol{Q}^i$ and R^i
Number of search iterations $\mathcal{T}_n$
Initial tracking error vector $\boldsymbol{E}_0^{i,\zeta}$
Initial weights $\boldsymbol{\Omega}^{i,\zeta(0)}$ and the corresponding $\boldsymbol{\vartheta}_0^{i,\zeta}$
Initial $\boldsymbol{P}_0^{i,\zeta}$ and $\boldsymbol{G}_0^{i,\zeta}$
Convergence error ξ and size of a moving window $\mathcal{W}$

Output:

Tuned weights $\boldsymbol{\Omega}^{i,\zeta(*)}$

$k \leftarrow 0$
RLS_weights_converged $\leftarrow$ false
while $k < \mathcal{T}_n$ **and** RSL_weights_converged = false **do**
    Compute control signal $\tilde{u}_{t,k}^{i,\zeta}$ and apply it to agent i
    Find $U_k^{i,\zeta}(\boldsymbol{Z}_k^{i,\zeta}, \tilde{u}_{t,k}^{i,\zeta})$ and $\bar{\boldsymbol{x}}_k^{i,\zeta}$
    Calculate $\boldsymbol{E}_{(k+1)}^{i,\zeta}, \tilde{u}_{t,k+1}^{i,\zeta}$
    Find $\bar{\boldsymbol{x}}_{k+1}^{i,\zeta}$ and calculate $\boldsymbol{\varphi}_k^{i,\zeta}$
    $k \leftarrow k + 1$
    Calculate $\boldsymbol{P}_k^{i,\zeta}$ and $\boldsymbol{G}_k^{i,\zeta}$
    Find $\boldsymbol{\vartheta}_k^{i,\zeta}$, then update $\boldsymbol{\Omega}_k^{i,\zeta}$
    if $k > \mathcal{W}$ **and** $\left\| \boldsymbol{\vartheta}^{i,\zeta\,(k+1-l)} - \boldsymbol{\vartheta}^{i,\zeta(k-l)} \right\| \leq \xi$, $\forall l \in \{0, 1, \ldots, \mathcal{W}\}$, **then**
        $\boldsymbol{\Omega}^{i,\zeta(*)} \leftarrow \boldsymbol{\Omega}^{i,\zeta(k+1)}$
        RSL_weights_converged $\leftarrow$ true
    end if
end while
return $\boldsymbol{\Omega}^{i,\zeta(*)}$

The α-norm is defined by

$$\left\|\zeta^{ij}\right\|_\alpha = \frac{1}{\mu}\left[\sqrt{1 + \mu\|\zeta^{ij}\|^2} - 1\right],$$

where μ is a positive real scalar. This control strategy is adaptive to the agents formation, where the connectivity between the agents can vary in time according to the agents proximity to one another.

6.3 CoppeliaSim Overview

The proposed Fuzzy-RL algorithm is validated with similar simulation scenarios in a robotic simulator - CoppeliaSim. A main control system is implemented in Matlab as the controller for each robots, and it is connected to CoppeliaSim through provided built-in API functions. The two software communicate using corresponding IP addresses through the network.

CoppeliaSim is a robotic simulation framework that can be operated in an integrated development environment. Each robot object or machine model can be controlled individually via an embedded script, a plugin, remote API clients, or a custom solution. The simulator's structure is based on a distributed control architecture which makes it versatile, scalable, and general-purpose [83]. These characteristics makes CoppeliaSim become an ideal framework for multi-robot applications, for the purpose of not only robotic simulation but also designing and modeling. Controllers used to navigate or operate the robots in the framework can be written in various programming languages, including C/C++, Python, Java, Lua, Matlab, among other languages.

CoppeliaSim can be either used as a stand-alone application or embedded into a main client application: combining the low/high-level functionalities to obtain higher level functionalities. This allows for a multitude of applications including rapid algorithm development, system verification, rapid prototyping, and deployment for cases such as safety/remote monitoring, training and education, hardware control, and factory automation simulation [83].

The physical simulation engines provided in CoppeliaSim, which are also formally named as "dynamics modules", support the simulation involving multi-object interactions. These dynamics modules can simulate close to real-world object interaction scenarios. Objects including robot models in CoppeliaSim are able to fall from higher positions, collide with other objects, or bounce around. It also allows manipulators to grasp objects and vehicles bumping in a realistic fashion on a rough terrain.

6.3.1 Models and Scenes

A scene without a main script in CoppeliaSim can be seen as a simulation's initialization or set up. As one of the most important elements in a scene, the robots or models to be simulated are always included in a scene object. Additionally, a scene can also contain the following elements: cameras and views, an environment, a floor as the base, and a main script. A view is associated with a camera object. It displays the simulation from

the sight of the camera. The environment represents the condition of the simulation and is composed by properties like ambient light, fog, background color, etc. Floor is the ground type of a scene. Different scenes may require different types of floors depending on the simulation of the robots. If a scene is associated with a main script, then it is able to run designed simulations scenarios instead of just representing a set up. Figure 6.1 demonstrates an example of a CoppeliaSim scene. It is a built-in example and can be loaded directly from the software scene examples folder. The main model to be simulated in this scene is the motorbike shown on the floor. The floor is designed as a concrete road so that the simulation is close to a real world scenario. The view is obtained from a camera that is located above the floor inside the scene.

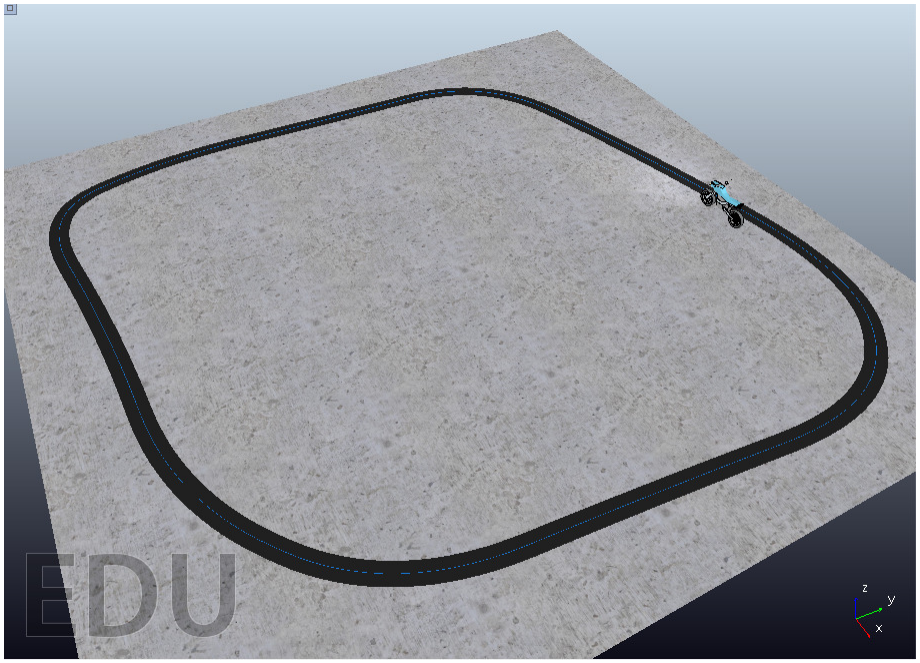


Figure 6.1: Scene Example of Motorbike Simulation

A model in CoppeliaSim is a sub-element of a scene. It represents the combination of one or a few robots with other objects. It is formally defined in CoppeliaSim as a selection of scene objects built on a same hierarchy tree, where the base of this tree is designed to be a model base. A model can not be simulated only by itself but has to be contained in a scene to be able to operate from commands or scripts. Figure 6.2 shows two model examples on a different floor type. The two models are robotic arms that can perform multi degree of freedom operations. A more complex model which is constructed as a

combination of multiple models and objects can be seen as an industrial machine.

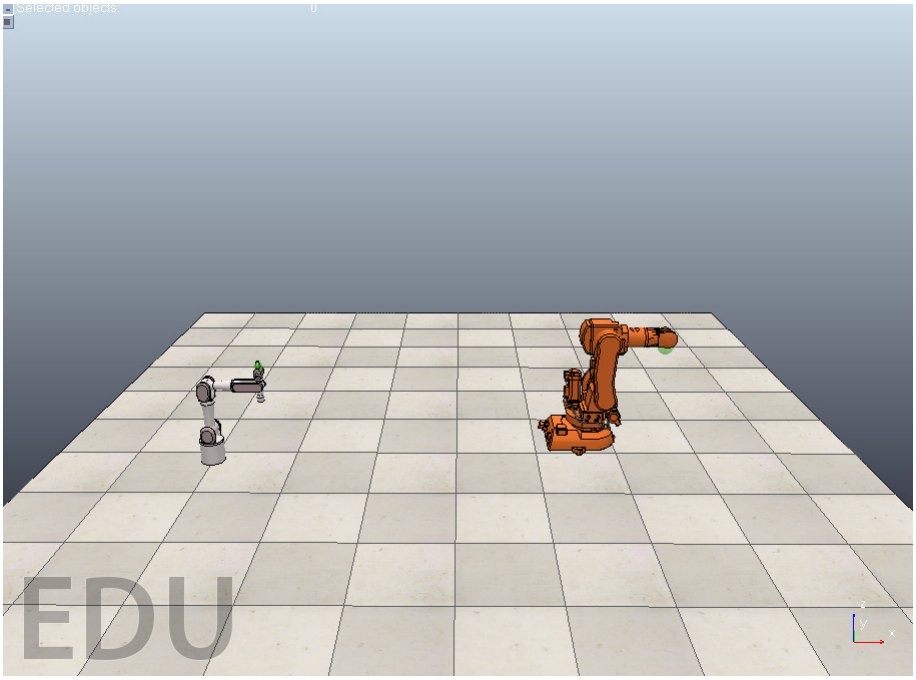


Figure 6.2: Model Examples of Robotic Arms

Pioneer 3-DX is a small lightweight two-wheel two-motor differential drive robot. It is one of the most popular research robots nowadays because of its modest and balanced size combined with reasonable hardware [111]. This makes it one of the most suitable in-door navigation drive robots. Pioneer 3-DX is available as a robot model in CoppeliaSim. It is controlled by two motors, one on each of the drive wheels. The controllers proposed in this thesis are simulated on Pioneer 3-DX to give a clear and close to real world demonstration of how they would perform in a real world scenario. Figure 6.3 shows a pioneer 3-DX robot in CoppeliaSim loaded on a floor.

6.3.2 Remote API clients - Matlab

The remote Application Programming Interface (API) in CoppeliaSim allows interactions between itself or its simulations with an external entity via network communication. This is composed by remote API server services and remote API clients. The client side can be any hardware as long as it is able to virtually embed small footprint codes (C/C++,

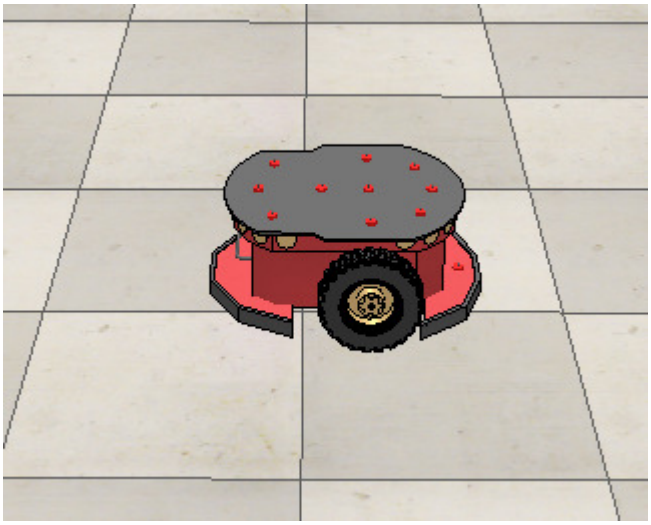


Figure 6.3: Pioneer-3DX Robot

Python, Java, Matlab, and Rubi) and allows remote function calling, as well as fast data streaming back and forth. The remote API feature provides functionality and availability on multiple platforms and its small footprint makes its easy to use.

Using the feature of remote API, simulations can be done by connecting CoppeliaSim with a control device. The device can be either local or connected through the network. For instance, a scene with multiple Pioneer 3-DX robots without a main control script represents a simulation set up, and a computer running Matlab scripts to send and receive data from CoppeliaSim is seen as the controller. Usually, the interface functions are fetching time interval status of a robot and then reapplying new commands to the robot's motors. Pioneer 3-DX in this case is commanded with angular speeds on each wheel to perform the desired operation.

6.3.3 Simulation Implementation

An auxiliary script is created to utilize the Fuzzy-RL system that is already implemented in Matlab and demonstrated in the previous section. This auxiliary script functions as an interface between Matlab and CoppeliaSim by calling the provided API functions. Firstly, it sets up the connection for them through network using IP address. After the two software are successfully connected, some API remote functions are called to obtain each agent's (Pioneer-3DX) associated handles as the initialization. The handles include the robot

itself, its left and right motor's identities. During each time interval, every agent's status is updated by running the corresponding API remote function with its identity/handle. The status of an agent is returned including its position and speeds in x and y directions. This information is then fed as input to the Fuzzy-RL control system that computes new wheel speed commands to be applied on the robot's wheels.

The implemented Fuzzy-RL adaptive learning system in Matlab uses linear x and y speeds for each agent while the Pioneer-3DX robot only takes rotational wheel speeds as the control command. Therefore a transformation from the robot's linear and angular velocities to the wheels angular velocities is required. This is performed as follow:

$$w_{k+1}^{i,R} = \frac{2v_{k+1}^{i} + d\alpha_{k+1}^{i}}{2r} \qquad\qquad w_{k+1}^{i,L} = \frac{2v_{k+1}^{i} - d\alpha_{k+1}^{i}}{2r}$$

where $w_{k+1}^{i,R}$ and $w_{k+1}^{i,R}$ are the angular speeds of the robot's right and left wheel, respectively; v_{k+1}^{i} and α_{k+1}^{i} the the robot's linear and angular speeds, respectively, r is the radius of the wheel, and d is the distance between both wheels. The linear velocity is calculated from its x-component and the heading of the robot, as

$$v_{k+1}^{i} = \frac{v_{k+1}^{i,x}}{\cos\theta_{k+1}^{i}},$$

where θ_{k+1}^{i} is the robot's heading on the 2D plane, and is calculated from the x-y coordinates of the linear velocity as follow:

$$\theta_{k+1}^{i} = \arctan\left(\frac{v_{k+1}^{i,y}}{v_{k+1}^{i,x}}\right).$$

The angular velocity α_{k+1}^{i} is the rate of change of the robot's headings over two time intervals.

$$\alpha_{k+1}^{i} = \frac{\theta_{k+1}^{i} - \theta_{k}^{i}}{T}$$

where T is the sampling time period.

6.4 Results and Discussion

Four simulation scenarios are designed with a system of 10 Pioneer-3DX mobile robots to validate the proposed Policy Iteration approach in Algorithm 6.1, where one robot plays the

role of a leader while the others act as followers. The simulation is realized in CoppeliaSim and compared with the Fuzzy-RL Value Iteration approach discussed in Chapter 5.

All of the scenarios share the same simulation parameter settings: the weighting matrices of the utility function are set to $\boldsymbol{Q}^i = 0.0001 \times \boldsymbol{I}_{3\times 3}$ and $R^i = 0.01$, $\forall i$. The RLS simulation parameters are taken as $\mathcal{T}_n = 400$ and $\boldsymbol{P}_0^{i,\gamma} = 100 \times \boldsymbol{I}_{10}$, $\forall i$. The parameters of the position-dependent adjacency function are fixed to $a = 1$, $r = 3.5\,\text{m}$, and $\mu = 0.5$. The speed and acceleration limits and other Pioneer-3DX specifications are displayed in Table 6.1. Each of the simulation scenarios is run for about 30 s.

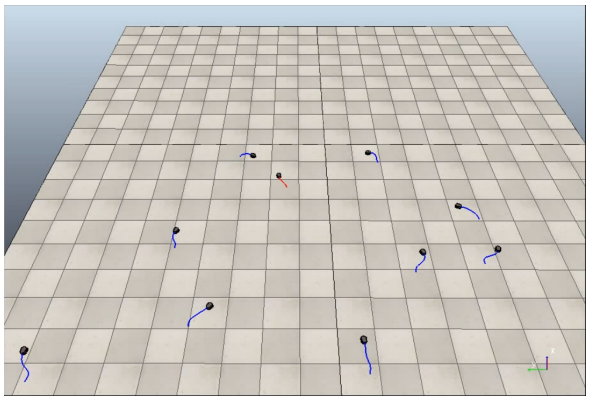
Table 6.1: Physical Specifications of Pioneer-3DX

Parameter	Value
Linear speed limit	$\pm 1.2\,\text{m/s}$
Linear acceleration limit	$\pm 2\,\text{m/s}^2$
Angular speed limit	$\pm 150\,^\circ/\text{s}$
Angular acceleration limit	$\pm 150\,^\circ/\text{s}^2$
Wheel diameter $2r$	0.195 m
Distance between wheels d	0.33 m

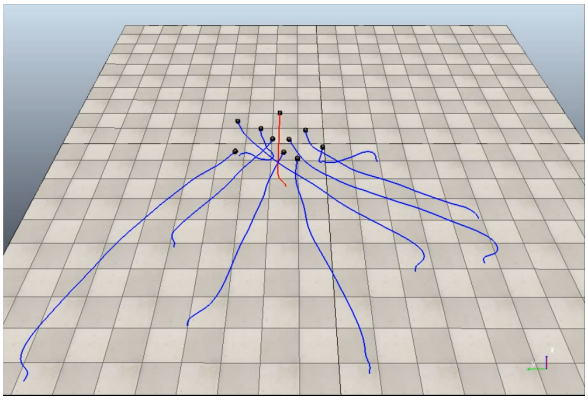
6.4.1 Scenario # 1 - Linear Trajectory

The trajectory of the flock is shown in Figure 6.4, where the leader's trajectory is marked in red and the followers' trajectories are shown in blue. The desired separation distance between each agent is taken as $d = 2\,\text{m}$. During the 30 s simulation, the leader assumes only a linear trajectory and is commanded with a linear velocity of 0.4 m/s (the angular velocity is $0\,^\circ/\text{s}$).

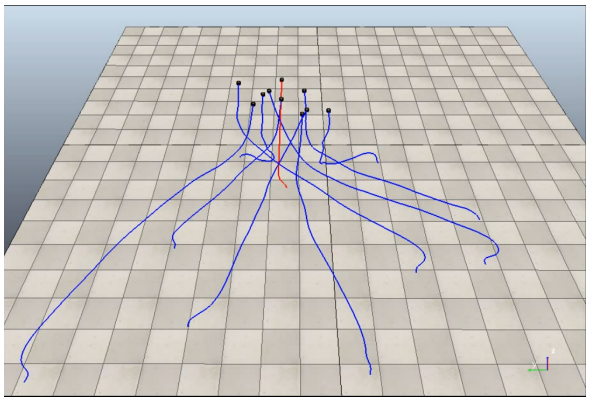
Initially, the robots are scattered randomly in the environment with the leader placed around the center of the flock, as seen in Figure 6.4(a). Then the leader starts to move in the desired path and the followers are commanded by the optimal control signal computed among all the three competing objectives from their Fuzzy-RLS systems. Each follower is controlled by a Fuzzy-RLS system that is independent from other agents' control systems. The followers keep adjusting their positions during the simulation to avoid collision between other agents but also try to eliminate the tracking error by staying close to the leader, as shown in Figures 6.4(b) to 6.4(d).



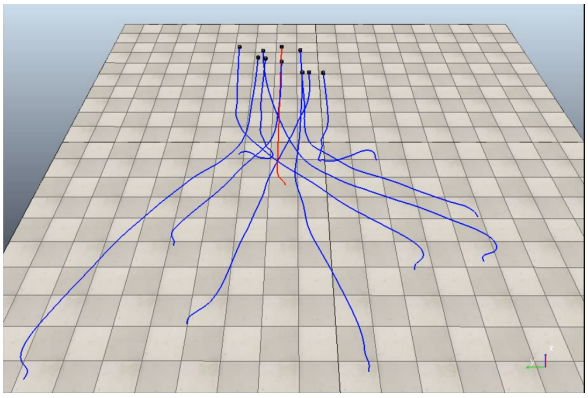
(a) $t = 1\,\mathrm{s}$



(b) $t = 12\,\mathrm{s}$



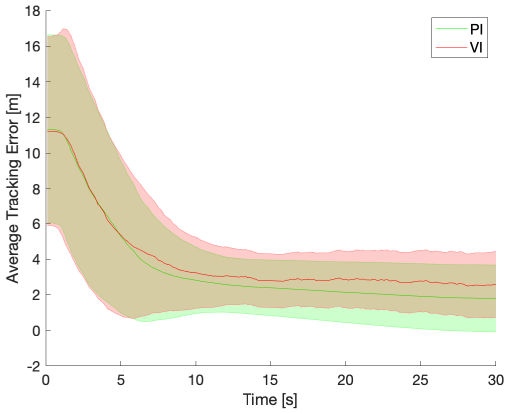
(c) $t = 21\,\mathrm{s}$



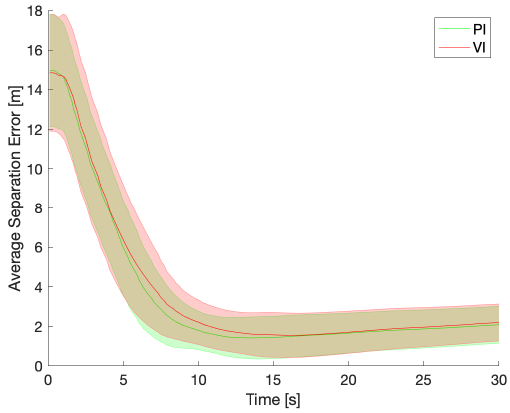
(d) $t = 30\,\mathrm{s}$

Figure 6.4: Simulation results of linear trajectory

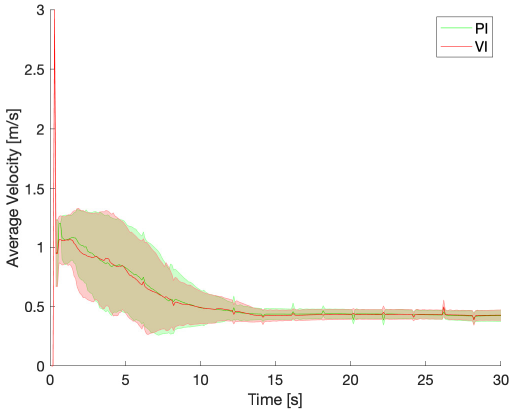
For a more quantitative assessment, the proposed control algorithm's performance is contrasted against the Fuzzy-RL Value Iteration approach introduced in Chapter 5 (and in [78]). The results are shown in Figure 6.5, where the solid lines represent the mean value and the shaded areas indicate the standard deviation among all the followers. It is clear from Figure 6.5(c) that the followers' average velocity stabilizes after around 10 s, and that in this aspect, the performance of the Policy Iteration approach is relatively close to that of the Value Iteration. As for the average tracking error, it is noticed that it is generally smaller with the Policy Iteration techniques, as revealed in Figure 6.5(a). One can also observe how it is faster to converge towards the end of the simulation than with the Value Iteration method. The same remark is also applicable to the average separation error between the followers and their neighbors, as shown in Figure 6.5(b).



(a) Average tracking error $O_{t,k}$



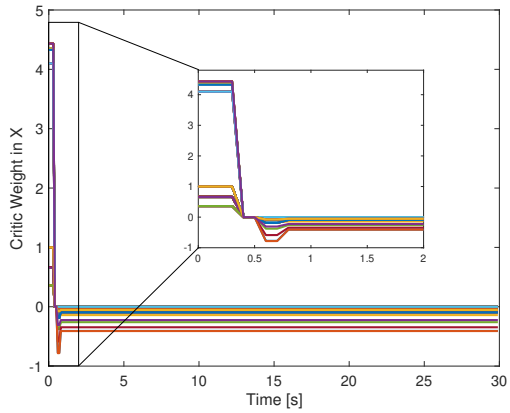
(b) Average separation error $O_{s,k}$



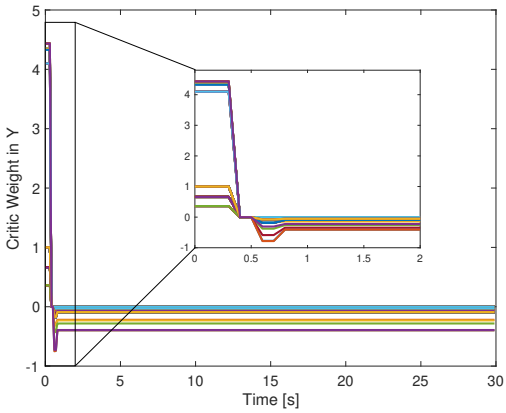
(c) Average follower velocity $O_{v,k}$

Figure 6.5: Comparison with the Value Iteration approach of Chapter 5 [78]

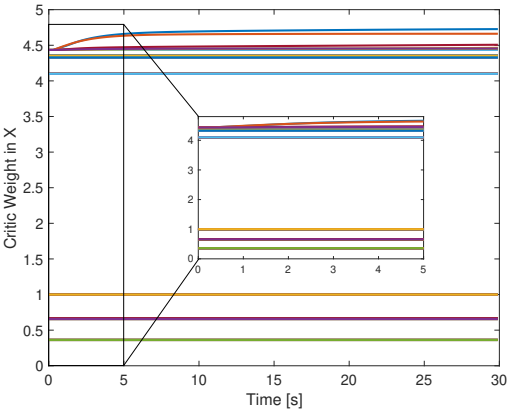
It is also worth to mention another observation about the proposed RLS-based Policy Iteration tracking algorithm, which is its faster critic weight convergence along the x and y-directions when compared to its Value Iteration counterpart presented in Chapter 5 (and in [78]). Figure 6.6 shows that the weights of the former method converge in less than 1 s whereas the Value Iteration algorithm takes about 5 s to stabilize its critic weights. This reflects a higher ability to adapt to time-varying graph topology in the flock decision process, which is encoded in the Policy Iteration method, rather than relying on a fully connected-graph topology as it is the case with the Value Iteration approach.



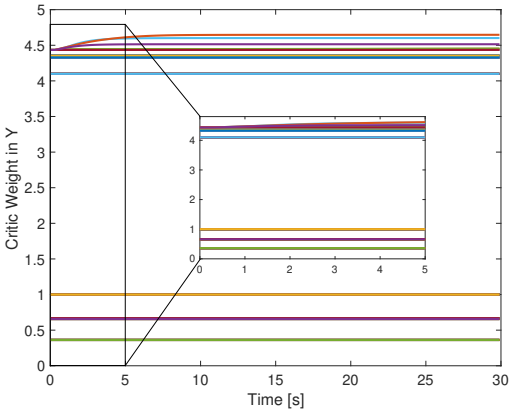
(a) PI (x)



(b) PI (y)



(c) VI (x)



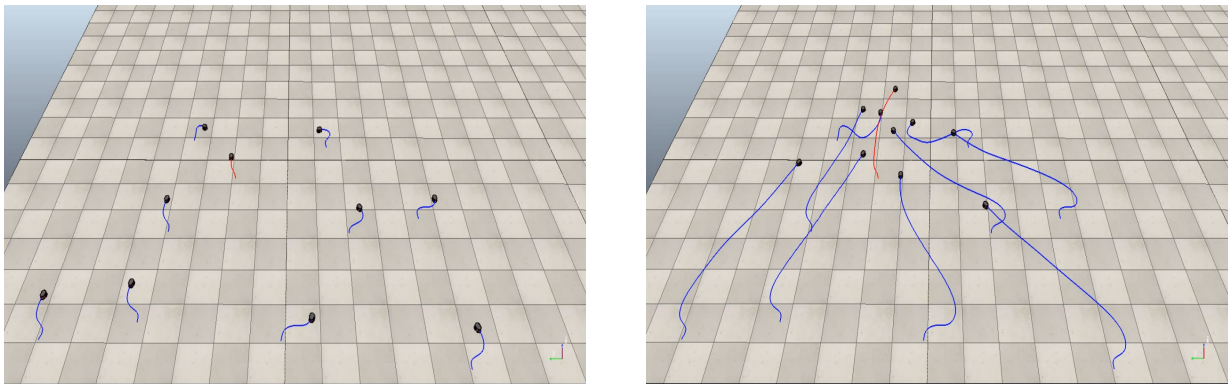
(d) VI (y)

Figure 6.6: Tracking critic weights of the proposed PI technique and the VI approach of Chapter 5 [78] along the x and y-directions

6.4.2 Scenario # 2 - Circular Trajectory

Scenario 2 is designed to examine the Fuzzy-RLS system's adaptability and stability in a more challenging case. In this scenario, the leader assumes a circular trajectory and is commanded with a faster linear velocity of $0.8\,\mathrm{m/s}$ and an angular velocity of $-3.3\,^{\circ}/\mathrm{s}$ during the $30\,\mathrm{s}$ simulation. The desired separation distance between each agent is taken as $d = 2.5\,\mathrm{m}$.

Figure 6.7 shows the trajectory of the flock. The robots have similar initial placements as the previous scenario, as shown in Figure 6.7(a). From Figures 6.7(b) and 6.7(c), we can observe that the agents have successfully followed the leader and also maintained collision avoidance between each other. At the end of the simulation, the agents have achieved the formation of a flock easily during a circular trajectory with a larger separation distance, as presented in Figure 6.7(d).



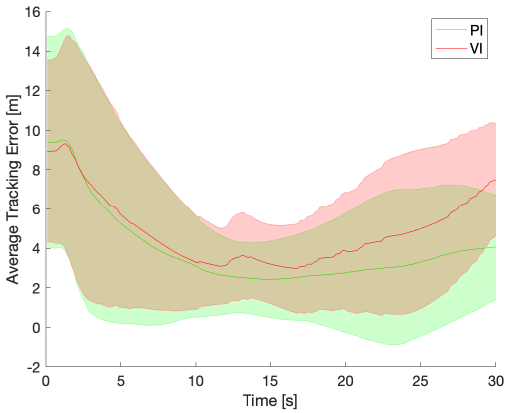
(a) $t = 1\,\mathrm{s}$ (b) $t = 7\,\mathrm{s}$

(c) $t = 16\,\mathrm{s}$ (d) $t = 30\,\mathrm{s}$

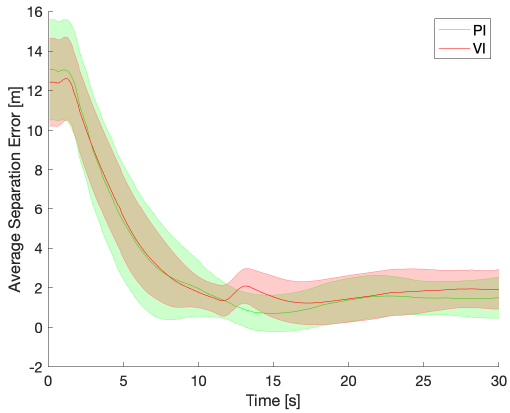
Figure 6.7: Simulation results of circular trajectory

For a more detailed evaluation of the proposed Fuzzy-RLS Policy Iteration control approach, we again perform the comparison against the Fuzzy-RL Value Iteration algorithm. The result is demonstrated in Figure 6.8. Figure 6.8(c) shows that at the beginning of the simulation the two approaches have similar velocity convergence characteristic, which is as observed in Scenario 1. However, during the second half of the simulation, where the followers in both approaches converged to the same linear velocity as the leader, the

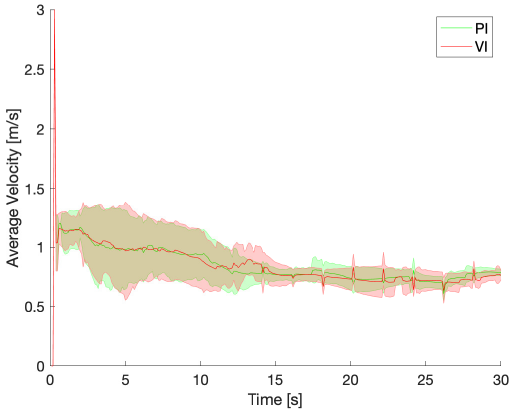
Policy Iteration approach achieves a relative smaller variance in velocity. This can be seen as a reflection of the time-varying graph topology which improves the flock's stability and velocity convergence. Figure 6.8(a) clearly shows that the tracking error with Policy Iteration technique is generally smaller than that of the Value Iteration. The Policy Iteration method demonstrates a better robustness in this case. For the separation error, it can be observed that the Policy Iteration gets a higher variance at the beginning of the simulation but a slightly smaller error and variance at the end, as shown in Figure 6.8(b).



(a) Average tracking error $O_{t,k}$



(b) Average separation error $O_{s,k}$

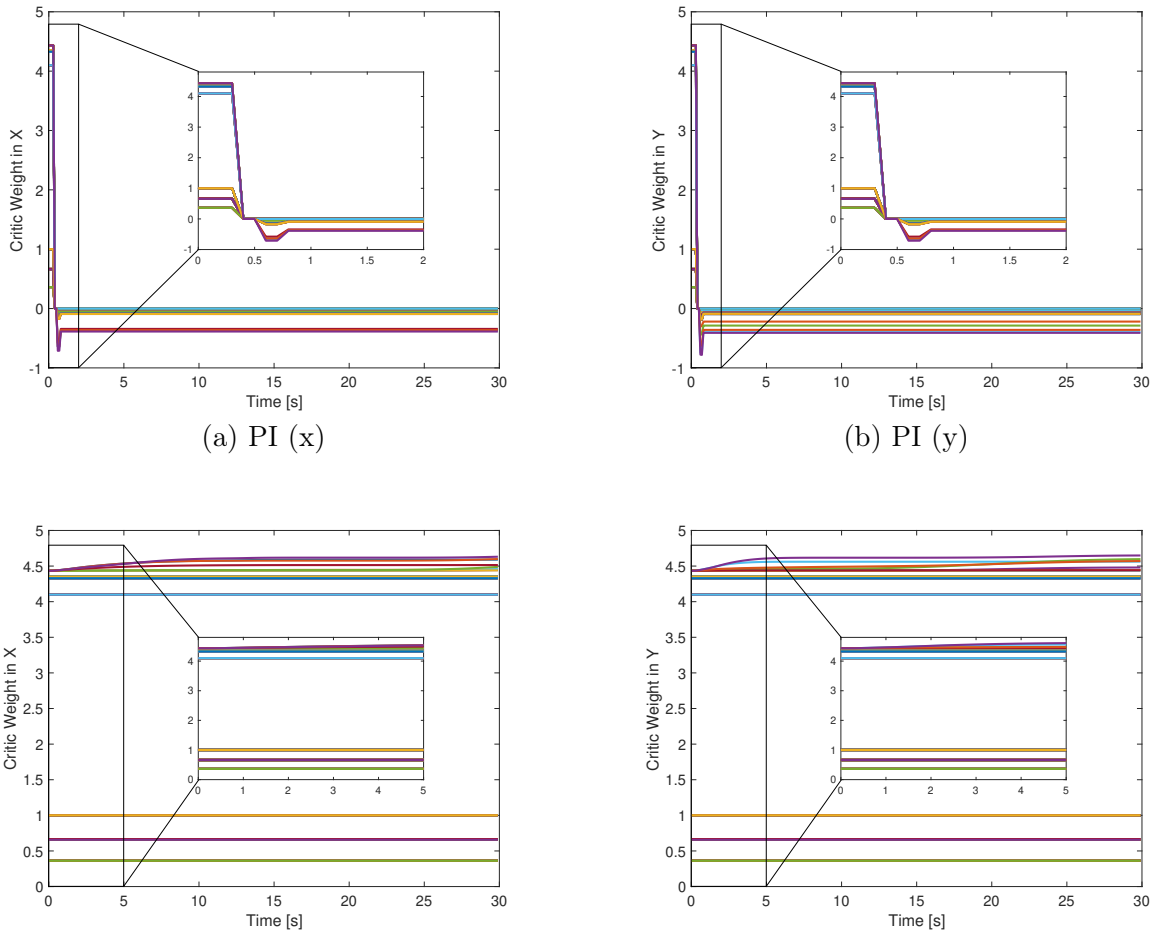


(c) Average follower velocity $O_{v,k}$

Figure 6.8: Comparison with the Value Iteration approach of Chapter 5 [78]

Similar to scenario 1, the proposed RLS-based Policy Iteration algorithm demonstrates a faster critic weight convergence along the x and y-directions when compared to its Value

Iteration counterpart. Figure 6.9 shows the Policy Iteration approach converges in less than 1 s while the Value Iteration approach takes a longer time.



(a) PI (x)

(b) PI (y)

(c) VI (x)

(d) VI (y)

Figure 6.9: Tracking critic weights of the proposed PI technique and the VI approach of Chapter 5 [78] along the x and y-directions

6.4.3 Scenario # 3 - Linear Trajectory with Joining Agents

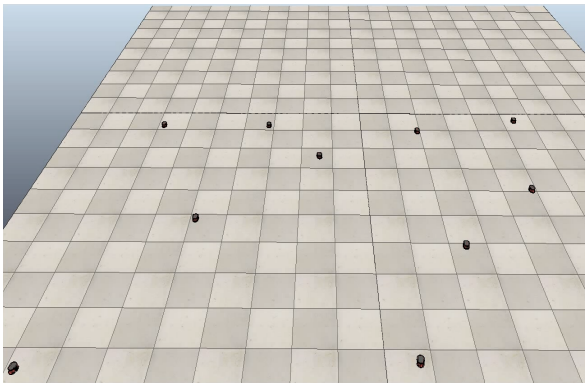
Scenario 3 aims to test the Fuzzy-RLS system's adaptability and robustness with dynamic changes, such as agents joining to the flock while in action. In this scenario, the leader

assumes the same linear trajectory as the one used in scenario 1 (Section 6.4.1). The linear velocity and the desired separation distance between the agents are set to 0.4 m/s and $d = 2.5$ m, respectively.

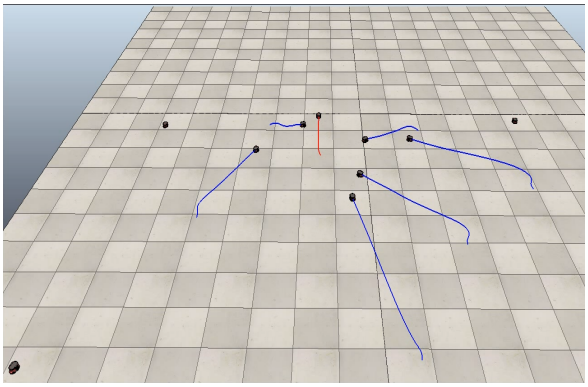
Figure 6.10 demonstrates the complete trajectory of the flock during the simulation. From Figure 6.10(a), we can see that the agents are initially placed on the 2D plane and about to move toward the leader. There are three agents that are decommissioned during the first 10 s of the simulation and shown in green trajectories. Six other follower agents receive the control signal from the beginning and try to achieve the flocking objectives as usual, and it is shown in Figure 6.10(b) that the three agents are remaining at their initial positions. After 10 s, the decommissioned followers start to proceed with the control signals and try to join to the moving flock quickly. As demonstrated in Figure 6.10(c), just a few seconds after the three agents start to join the mission, they move in a faster speed than other agents and try to form a new flock as soon as possible. This is a reflection of the control system's strong ability in computing accurate corresponding control signals in a short time with regards to each agent's current state and keeping the converged agents remain at their steady states. In Figure 6.10(d), it can be seen that a new flock is formed.

The detailed evaluation of the proposed Fuzzy-RLS Policy Iteration control approach against the Fuzzy-RL Value Iteration is demonstrated in Figure 6.11. As demonstrated in all the figures, The Value Iteration approach has generally similar performance as the Policy Iteration. Comparing to the previous scenarios, we can observe a larger variance of tracking and separation error during the first 10 s of the simulation. This is because the system still considers the decommissioned agents as part of the flock from the beginning of the simulation so that we can have a clear performance comparison on the time before and after the agents joining the flock. It can also be seen that in Figure 6.11(a) and Figure 6.11(b), the convergence rate is faster after 10 s of the simulation, which further explains that the decommissioned agents have been taken into account at the beginning and the system quickly converged to the steady state values in about 10 s after the agents join the flock. From the two figures, we can also see that after entering the steady state, the Policy Iteration obtains a slightly smaller variance in average tracking error and a smaller average separation error.

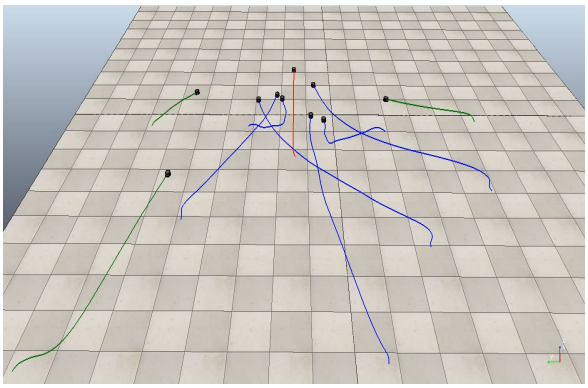
As shown in Figure 6.12, the Policy Iteration still performs a much faster weight convergence characteristic while the Value Iteration demonstrates a slower convergence rate comparing to previous scenarios due to the decommissioned agents at the beginning.



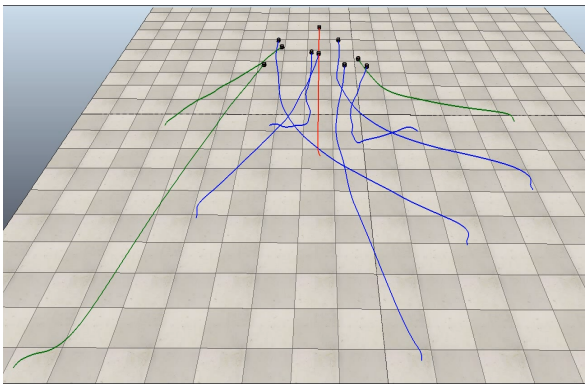
(a) $t = 1\,\mathrm{s}$



(b) $t = 8\,\mathrm{s}$



(c) $t = 18\,\mathrm{s}$



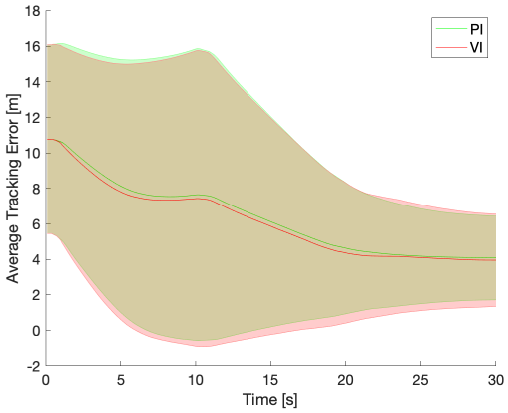
(d) $t = 30\,\mathrm{s}$

Figure 6.10: Simulation results of joining agents

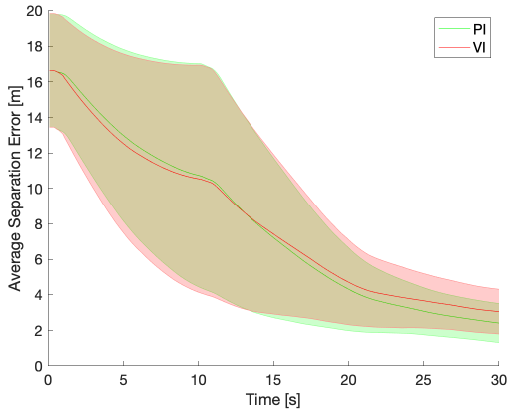
6.4.4 Scenario # 4 - Circular Trajectory with Leader Change

Scenario 4 is designed with another dynamic change by switching of the leader, to further test the Fuzzy-RLS system's adaptability and stability. In this scenario, the leader follows a similar circular trajectory as shown in scenario 2 in Section 6.4.2. The linear and angular velocities are set to be $0.82\,\mathrm{m/s}$ and $-3.5\,^{\circ}/\mathrm{s}$, respectively, for the leader. The desired separation distance between the agents is set as $d = 2.5\,\mathrm{m}$. Note that after half of the simulation, the leader shown in a red trajectory is deleted from the flock and no longer transmits data to the other agents. During the second half of the simulation, one of the followers is selected as the new leader and follows the same velocities as the previous leader.

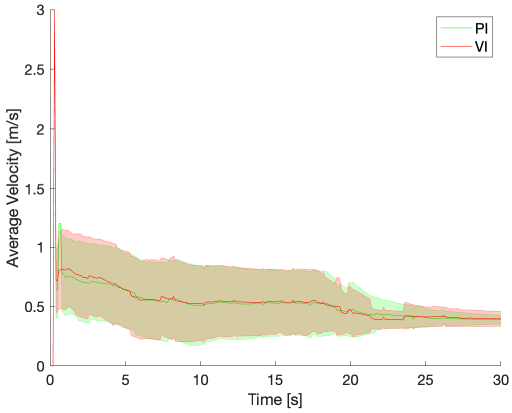
Figure 6.13 shows the captures of the simulation during different times. From Figure 6.13(a) we can see that the agents are initially placed on the 2D plane and about to



(a) Average tracking error $O_{t,k}$



(b) Average separation error $O_{s,k}$

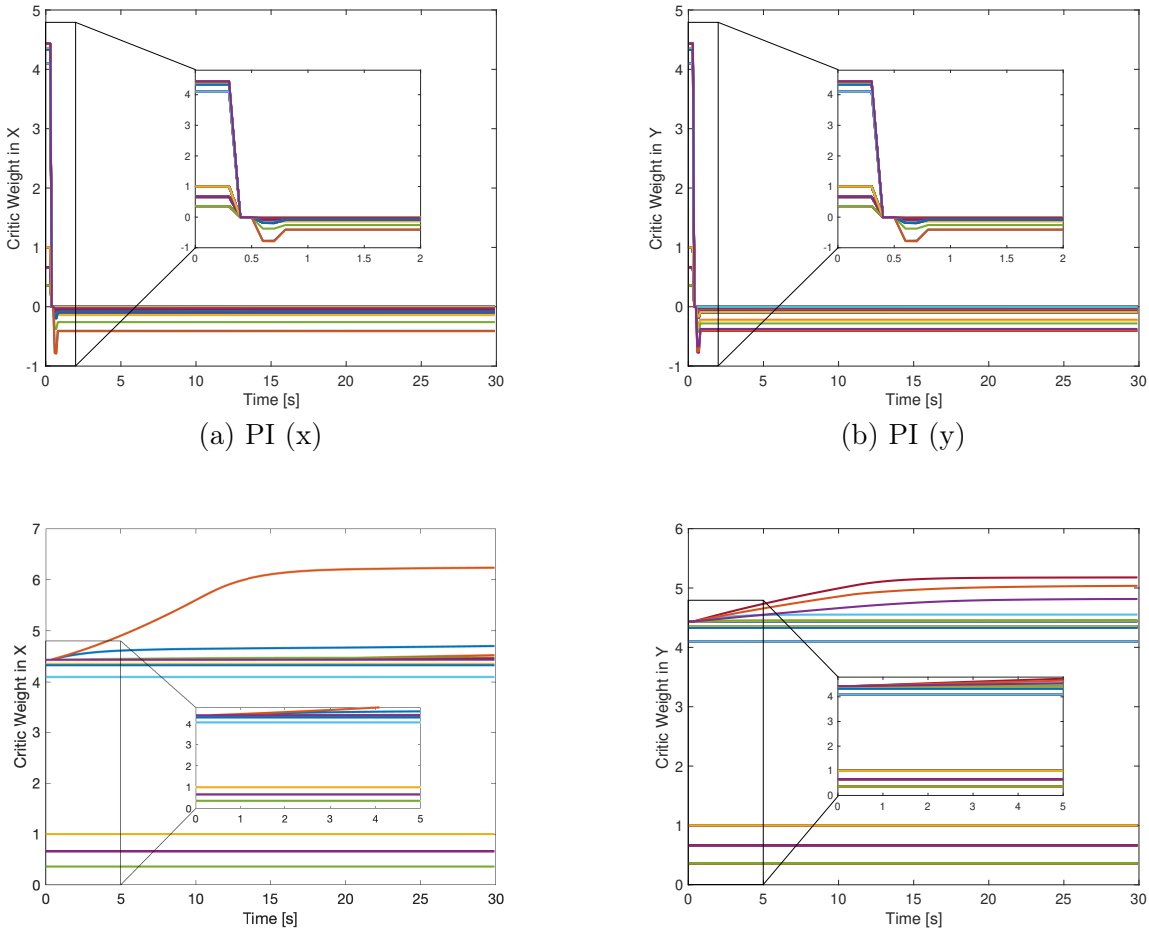


(c) Average follower velocity $O_{v,k}$

Figure 6.11: Comparison with the Value Iteration approach of Chapter 5 [78]

proceed with the control commands to achieve the flocking objectives. Until 15 s after the simulation, the flock behaves normally without any dynamic change. Starting at 16 s, the leader with red trajectory separates itself from the flock and stops at the position, and one of the followers shown in green trajectory starts to play the role of the new leader. Figure 6.13(c) clearly demonstrates that the followers' trajectories have a rapid change towards the agent shown in green after the leader is switched. At the end of the simulation, the agents can adapt to the dynamic change and remain the flock formation.

Figure 6.14 presents a detailed comparison between the proposed Policy Iteration and

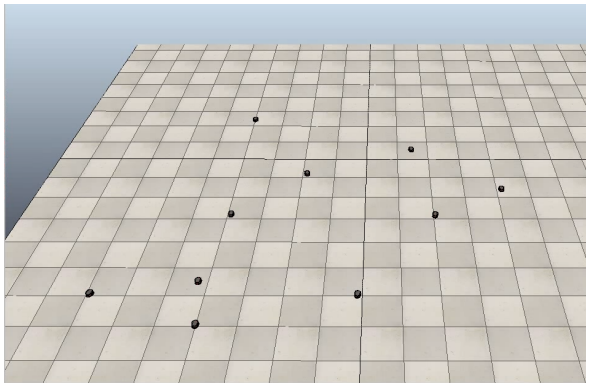


(a) PI (x) (b) PI (y)

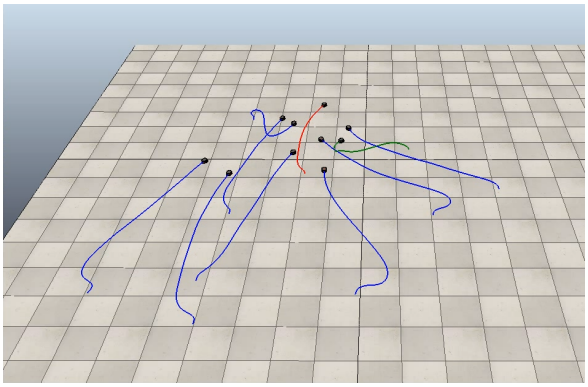
(c) VI (x) (d) VI (y)

Figure 6.12: Tracking critic weights of the proposed PI technique and the VI approach of Chapter 5 [78] along the x and y-directions

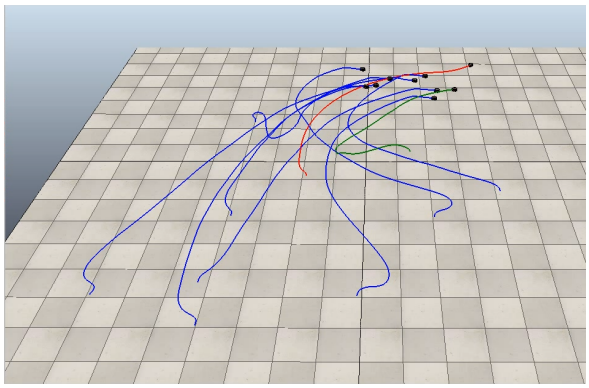
Value Iteration approaches with the scenario. Overall, the Policy Iteration approach demonstrates a more stable performance than the Value Iteration from all the three figures. In Figure 6.14(a) and Figure 6.14(c), we can see there exits a notable adaptation happened at 15 s during the simulation, where it is a clear reflection of the leader change. After the change of the leader, the agents are able to adjust their positions rapidly to keep satisfying the flock constraints and converge to their steady states. The Policy Iteration approach with time varying communication graph still reveals a better system robustness against the Value Iteration algorithm. As demonstrated in Figure 6.14(c), when the two systems reach



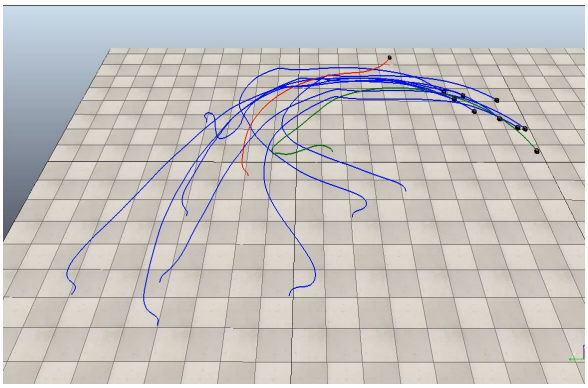
(a) $t = 1\,\mathrm{s}$



(b) $t = 8\,\mathrm{s}$



(c) $t = 18\,\mathrm{s}$

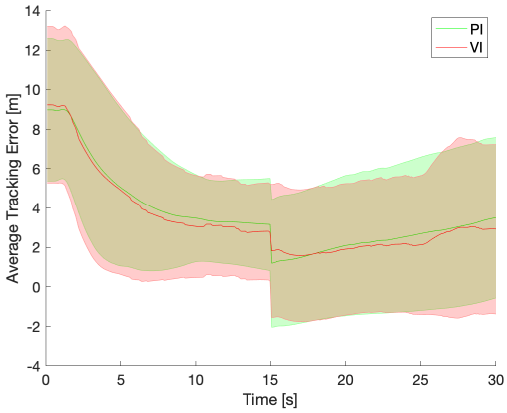


(d) $t = 30\,\mathrm{s}$

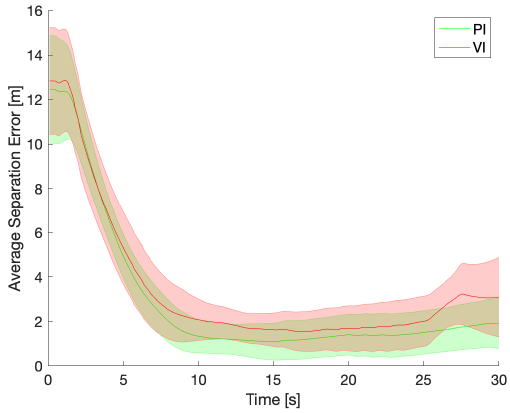
Figure 6.13: Simulation results of leader change

the steady states, the Policy Iteration has less fluctuation and variance especially during the 23 s to 30 s interval. In addition, the average separation error of the Policy Iteration has a smaller and more stable value, as shown in Figure 6.14(b)

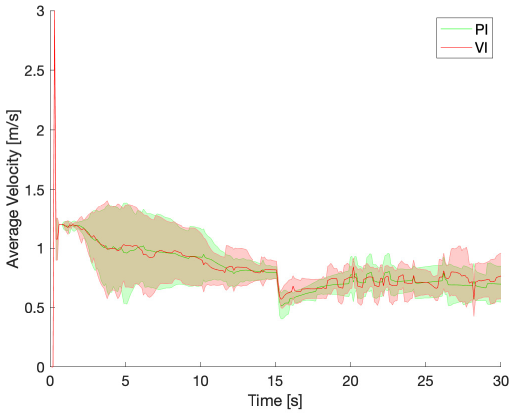
As shown in Figure 6.15, the Policy Iteration still performs a much faster weight convergence characteristic while the Value Iteration demonstrates a relatively slower convergence rate.



(a) Average tracking error $O_{t,k}$



(b) Average separation error $O_{s,k}$

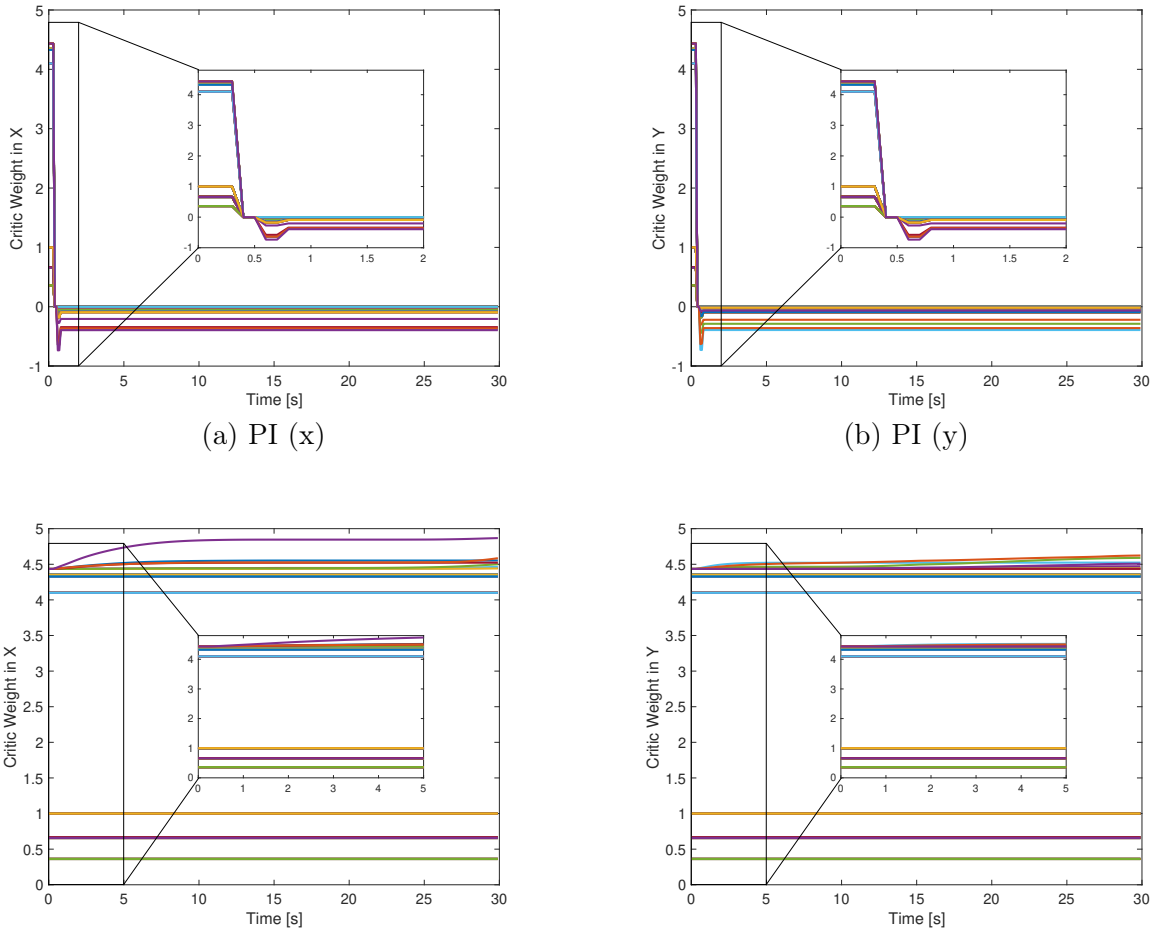


(c) Average follower velocity $O_{v,k}$

Figure 6.14: Comparison with the Value Iteration approach of Chapter 5 [78]

6.5 Summary

This chapter introduces a Fuzzy-RLS control mechanism based on Policy Iteration to solve the flocking motion problem in real-time without knowing the dynamics of the agents or those of the formation. A robotics simulation software engine is employed with a flock of Pioneer-3DX mobile robots to validate the performance of the proposed solution in different scenarios when applied in a near real-world environment. The results are contrasted against the previously proposed Fuzzy-RL approach. The Fuzzy-RLS solution based on Policy



(a) PI (x)

(b) PI (y)

(c) VI (x)

(d) VI (y)

Figure 6.15: Tracking critic weights of the proposed PI technique and the VI approach of Chapter 5 [78] along the x and y-directions

Iteration and time-varying graph topology exhibit better convergence characteristics when compared with the Fuzzy-RL process that is based on Value Iteration and a fully connected-graph topology.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

The thesis proposes two adaptive distributed online Fuzzy-RLS/Fuzzy-RL approaches for the autonomous control of flock systems. The control algorithms try to simultaneously compromise three objectives: 1) tracking a leader; 2) avoiding collisions by maintaining a predetermined safety distance between the neighboring agents; and 3) reaching a velocity consensus among the flock.

The Fuzzy-RL algorithm guides the flock towards a leader by applying Value Iteration techniques in a reinforcement learning scheme using actor-critic structure that is only dependent on the agent position feedback signals. The collision avoidance is realized by means of an adaptive FIS that is based on its own reinforcement learning process. The integration of the traditional fuzzy system with a neural network obtains the adaptability of self-learning by automatically tuning the FIS's parameters. Finally, a graph-based protocol is employed to enable consensus on a common flock velocity with fixed edge weights between each of the agents. In addition to its flexible and simple structure, the proposed Fuzzy-RL algorithm is shown in the simulations to possess a number of other salient features, such as fast convergence and online adaptability to dynamic disturbances. More specifically, it proved to be relatively insensitive to the sudden change in the leader's trajectory, number of active agents, and the target separation distance between the agents. The algorithm's superiority with respect to the set of objectives is demonstrated against a similar technique proposed in the literature.

The Fuzzy-RLS approach as an extension to the Fuzzy-RL algorithm is also introduced and simulated in the thesis. The algorithm performs an enhanced navigation ability to

track the target by employing Policy Iteration techniques using the RLS approximation strategy. The adaptive FIS system used for collision avoidance is kept the same as the Fuzzy-RL approach. An improved communication graph implemented from a time-varying edge weight graph topology helps the agents reach a velocity consensus smoothly by taking the relative distance as the subsystem's input. Applying the Fuzzy-RLS approach, four simulation scenarios are demonstrated in a realistic robotic simulator - CoppeliaSim. The algorithm's simulation result shows enhanced tracking convergence comparing to the Fuzzy-RL method from both the tracking error and weight convergence analysis.

7.2 Future Work

To further extend the current research, a few of avenues can be taken. One of the potentially most important enhancements is improving the FIS's convergence time with a time-varying safety distance. The convergence of the separation control takes a relatively longer time than the tracking control in the presence of dynamic changes, such as a varying safety distance. It might be very rewarding to find some remedy to this phenomenon.

References

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems*, pages 1–8, 2007.

[2] 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)*, pages 32–37. IEEE, 2017.

[3] 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)*, pages 111–116, 2017.

[4] Mohammed Abouheaf, Wail Gueaieb, and Frank Lewis. Model-free gradient-based adaptive learning controller for an unmanned flexible wing aircraft. *Robotics*, 7(4):66, October 2018.

[5] Mohammed Abouheaf, Frank L. Lewis, Magdi S. Mahmoud, and Dariusz G. Mikulski. Discrete-time dynamic graphical games: Model-free reinforcement learning solution. *Control Theory and Technology*, 13(1):55–69, February 2015.

[6] Mohammed Abouheaf, Shuzheng Qu, Wail Gueaieb, Rami Abielmona, and Moufid Harb. Responding to illegal activities along the canadian coastlines using reinforcement learning. *IEEE Instrumentation and Measurement Magazine*, 24(2):118–126, April 2021.

[7] Nouara Achour and Karim Braikia. An MDP-based approach oriented optimal policy for path planning. In *2010 International Conference on Machine and Web Intelligence*, pages 205–210, 2010.

[8] Tolulope Akinbulire, Howard Schwartz, Rafael Falcon, and Rami Abielmona. A reinforcement learning approach to tackle illegal, unreported and unregulated fishing. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, November 2017.

[9] Plamen P. Angelov and Dimitar P. Filev. An approach to online identification of takagi-sugeno fuzzy models. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):484–498, 2004.

[10] Karl J Åström and Björn Wittenmark. *Adaptive Control*. Courier Corporation, 2013.

[11] Ying Bai and Dali Wang. *Fundamentals of Fuzzy Logic Control — Fuzzy Sets, Fuzzy Rules and Defuzzifications*, pages 17–36. Springer London, London, 2006.

[12] Alec Banks, Jonathan Vincent, and Chukwudi Anyakoha. A review of particle swarm optimization. part i: Background and development. *Natural Computing*, 6(4):467–484, July 2007.

[13] Chandreyee Bhowmick, Laxmidhar Behera, Amit Shukla, and Hamad Karki. 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*, pages 166–171, 2016.

[14] Tao Bian and Zhong-Ping Jiang. Reinforcement learning and adaptive optimal control for continuous-time nonlinear systems: A value iteration approach. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–10, 2021.

[15] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[16] Lucian Buşoniu, Damien Ernst, Bart De Schutter, and Robert Babuška. Online least-squares policy iteration for reinforcement learning control. In *Proceedings of the 2010 American Control Conference*, pages 486–491. IEEE, 2010.

[17] Young Hwan Chang, Claire Tomlin, and Karl Hedrick. Biologically-inspired coordination of multiple uavs using sliding mode control. In *Proceedings of the 2011 American Control Conference*, pages 4123–4128, 2011.

[18] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement

learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6015–6022, 2019.

[19] Peter C. Y. Chen and Aun-Neow Poo. Engineering, artificial intelligence in. In Hossein Bidgoli, editor, *Encyclopedia of Information Systems*, pages 141–155. Elsevier, New York, 2003.

[20] Mingli Chiang, Ansheng Liu, and Lichen Fu. Reinforcement learning control for consensus of the leader-follower multi-agent systems. In *2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS)*, pages 1152–1157, 2018.

[21] Michael Defoort, Thierry Floquet, Annemarie Kokosy, and Wilfrid Perruquetti. Sliding-mode formation control for cooperative autonomous mobile robots. *IEEE Transactions on Industrial Electronics*, 55(11):3944–3953, 2008.

[22] Chao Deng and Guang-Hong Yang. Distributed adaptive fuzzy control for nonlinear multiagent systems under directed graphs. *IEEE Transactions on Fuzzy Systems*, 26(3):1356–1366, 2018.

[23] Dimiter Driankov and Alessandro Saffiotti. *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, volume 61. Physica, 2013.

[24] Yong Duan and Hexu Xin. Fuzzy reinforcement learning and its application in robot navigation. In *2005 International Conference on Machine Learning and Cybernetics*, volume 2, pages 899–904, 2005.

[25] Meng Joo Er and Yang Gao. Robust adaptive control of robot manipulators using generalized fuzzy neural networks. *IEEE Transactions on Industrial Electronics*, 50(3):620–628, 2003.

[26] Ku Ge and Jin Cheng. Review on flocking control. *Social Informatics and Telecommunications Engineering*, 327, 2020.

[27] Atef Gharbi and Samir BEN Ahmed. Fuzzy-logic multi-agent system. In *Computer Science & Information Technology (CS & IT)*. Academy & Industry Research Collaboration Center (AIRCC), May 2014.

[28] Joseph A. Goguen and Lotfi A. Zadeh. Similarity relations and fuzzy orderings. *Information Sciences*, 3(2):177–200, 1971.

[29] Dongbing Gu and Huosheng Hu. Using fuzzy logic to design separation function in flocking algorithms. *IEEE Transactions on Fuzzy Systems*, 16:826 – 838, September 2008.

[30] Udit Halder and Biswadip Dey. Biomimetic algorithms for coordinated motion: Theory and implementation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5426–5432, 2015.

[31] K Han, J Lee, and Youdan Kim. Unmanned aerial vehicle swarm control using potential functions and sliding mode control. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 222(6):721–730, 2008.

[32] Zhicheng Hou and Isabelle Fantoni. Distributed leader-follower formation control for multiple quadrotors with weighted topology. In *2015 10th System of Systems Engineering Conference (SoSE)*, pages 256–261. IEEE, 2015.

[33] Jiangping Hu and Gang Feng. Distributed tracking control of leader–follower multi-agent systems under noisy measurement. *Automatica*, 46(8):1382–1387, 2010.

[34] Adrianto R. Ibrahim, Widyawardana Adiprawita, and Riyanto T. Bambang. Analysis on consensus-like protocol for multi-agent system. In *2013 International Conference on Robotics, Biomimetics, Intelligent Computational Systems*, pages 83–87, 2013.

[35] Bianca Innocenti, Beatriz López, and Joaquim Salvi. A multi-agent architecture with cooperative fuzzy control for a mobile robot. *Robotics and Autonomous Systems*, 55(12):881–891, 2007.

[36] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2961–2970. PMLR, June 2019.

[37] Mahdi Jadaliha, Joonho Lee, and Jongeun Choi. Adaptive control of multi-agent systems for finding peaks of unknown fields. In *Dynamic Systems and Control Conference*, volume 44182, pages 623–630, 2010.

[38] Endra Joelianto and Albert Sagala. Swarm tracking control for flocking of a multi-agent system. In *2012 IEEE Conference on Control, Systems Industrial Informatics*, pages 75–80, 2012.

[39] Lionel Jouffe. Fuzzy inference system learning by reinforcement methods. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 28(3):338–355, 1998.

[40] Harikumar Kandath, J. Senthilnath, and Suresh Sundaram. Mutli-agent consensus under communication failure using actor-critic reinforcement learning. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1461–1465, 2018.

[41] Revanth Konda, Hung Manh La, and Jun Zhang. Decentralized function approximated q-learning in multi-robot systems for predator avoidance. *IEEE Robotics and Automation Letters*, 5(4):6342–6349, 2020.

[42] Cezary Kownacki and Daniel Ołdziej. Fixed-wing UAVs flock control through cohesion and repulsion behaviours combined with a leadership. *International Journal of Advanced Robotic Systems*, 13(1), 2016.

[43] Nikolaus Kriegeskorte and Tal Golan. Neural network models and deep learning. *Current Biology*, 29(7):R231–R236, April 2019.

[44] Gireesh Kumar T. and Vinodh P. Vijayan. A multi-agent optimal path planning approach to robotics environment. In *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*, volume 1, pages 400–404, 2007.

[45] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[46] Shu-Hung Leung and Henry C.F. So. Gradient-based variable forgetting factor RLS algorithm in time-varying environments. *IEEE Transactions on Signal Processing*, 53(8):3141–3150, 2005.

[47] Frank L. Lewis, Draguna Vrabie, and Vassilis L. Syrmos. *Optimal Control*. WILEY, February 2012.

[48] Chungui Li, Meng Wang, and Shuhong Yang. Incremental least squares policy iteration in reinforcement learning for control. In *2008 International Conference on Machine Learning and Cybernetics*, volume 4, pages 2010–2014, 2008.

[49] Hongliang Li, Derong Liu, and Ding Wang. Neural-network-based optimal control for discrete-time nonlinear systems using general value iteration. In *Proceedings of the 31st Chinese Control Conference*, pages 2932–2937, 2012.

[50] Jun Li and Lianghao Ji. Optimal consensus control for second-order discrete-time multi-agent systems: Using online policy iteration algorithm. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 2210–2216, 2020.

[51] Zhongkui Li, Guanghui Wen, Zhisheng Duan, and Wei Ren. Designing fully distributed consensus protocols for linear multi-agent systems with directed graphs. *IEEE Transactions on Automatic Control*, 60(4):1152–1157, 2015.

[52] W-S Lin, C-L Huang, and M-K Chuang. Hierarchical fuzzy control for autonomous navigation of wheeled robots. *IEE Proceedings-Control Theory and Applications*, 152(5):598–606, 2005.

[53] Micheal L. Littman. Markov decision processes. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*, pages 9240–9242. Pergamon, Oxford, 2001.

[54] Huagang Liu, Hao Fang, Yutian Mao, Hu Cao, and Rui Jia. Distributed flocking control and obstacle avoidance for multi-agent systems. In *Proceedings of the 29th Chinese Control Conference*, pages 4536–4541, 2010.

[55] Zongchun Liu, Yantao Tian, and Mao Yang. Neighborhood description and flocking behavior analysis of swarm robots based on fuzzy logic. In *2010 2nd International Conference on Advanced Computer Control*, volume 1, pages 25–29, 2010.

[56] Kenzo Lobos-Tsunekawa, Francisco Leiva, and Javier Ruiz-del Solar. Visual navigation for biped humanoid robots using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(4):3247–3254, 2018.

[57] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6252–6259, 2018.

[58] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6382–6393, Red Hook, NY, USA, 2017. Curran Associates Inc.

[59] Biao Luo, Yin Yang, Huaining Wu, and Tingwen Huang. Balancing value iteration and policy iteration for discrete-time control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(11):3948–3958, 2020.

[60] Kamyar Mehran. Takagi-sugeno fuzzy modeling for process control. *Industrial Automation, Robotics and Artificial Intelligence (EEE8005)*, 262, 2008.

[61] Martijn R. K. Mes and Arturo Pérez Rivera. Approximate dynamic programming by practical examples. In *Markov Decision Processes in Practice*, pages 63–101. Springer, 2017.

[62] Mehran Mesbahi and Magnus Egerstedt. *Graph Theoretic Methods in Multiagent Networks*, volume 33. Princeton University Press, 2010.

[63] Salama A. Mostafa, Aida Mustapha, Mazin Abed Mohammed, Mohd Sharifuddin Ahmad, and Moamin A. Mahmoud. A fuzzy logic control in adjustable autonomy of a multi-agent system for an automated elderly movement monitoring application. *International Journal of Medical Informatics*, 112:173–184, 2018.

[64] Roger Ng. Garment modelling by fuzzy logic. In Arun Majumdar, editor, *Soft Computing in Textile Engineering*, Woodhead Publishing Series in Textiles, pages 271–293. Woodhead Publishing, 2011.

[65] Thang Nguyen, Thanh-Trung Han, and Hung Manh La. Distributed flocking control of mobile robots by bounded feedback. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 563–568, 2016.

[66] Reza Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, 2006.

[67] Reza Olfati-Saber and Richard Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, September 2004.

[68] Sami Oweis, Subra Ganesan, and Ka C Cheok. Server based control flocking for aerial-systems. In *IEEE International Conference on Electro/Information Technology*, pages 314–319, 2014.

[69] Constantin Paleologu, Jacob Benesty, and Silviu Ciochina. A robust variable forgetting factor recursive least-squares algorithm for system identification. *IEEE Signal Processing Letters*, 15:597–600, 2008.

[70] Bo Pang and Zhongping Jiang. Adaptive optimal control of linear periodic systems: An off-policy value iteration approach. *IEEE Transactions on Automatic Control*, 66(2):888–894, 2021.

[71] Aditya A. Paranjape, Soon-Jo Chung, Kyunam Kim, and David Hyunchul Shim. Robotic herding of a flock of birds using an unmanned aerial vehicle. *IEEE Transactions on Robotics*, 34(4):901–915, 2018.

[72] Young Joon Park, Young Jae Lee, and Seoung Bum Kim. Cooperative multi-agent reinforcement learning with approximate model learning. *IEEE Access*, 8:125389–125400, 2020.

[73] Thomy Phan, Kyrill Schmid, Lenz Belzner, Thomas Gabor, Sebastian Feld, and Claudia Linnhoff-Popien. Distributed policy iteration for scalable approximation of cooperative multi-agent policies, 2019.

[74] Warren B. Powell. Approximate dynamic programming: Lessons from the field. In *2008 Winter Simulation Conference*, pages 205–214. IEEE, 2008.

[75] Xuewei Qi, Yadan Luo, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew Barth. Deep reinforcement learning enabled self-learning control for energy efficient driving. *Transportation Research Part C: Emerging Technologies*, 99:67–81, February 2019.

[76] Wei Qinglai, Liu Derong, and Song Ruizhuo. Optimal self-learning cooperative control for continuous-time heterogeneous multi-agent systems. In *2015 34th Chinese Control Conference (CCC)*, pages 3005–3010, 2015.

[77] Huaxin Qiu and Haibin Duan. Multiple UAV distributed close formation control based on in-flight leadership hierarchies of pigeon flocks. *Aerospace Science and Technology*, 70:471–486, 2017.

[78] Shuzheng Qu, Mohammed Abouheaf, Wail Gueaieb, and Davide Spinello. An adaptive fuzzy reinforcement learning cooperative approach for the autonomous control of flock systems. In *IEEE International Conference on Robotics and Automation (ICRA 2021)*, pages 1–7, Xi'an, China, May 2021. (Accepted: Mar. 2021).

[79] Shuzheng Qu, Mohammed Abouheaf, Wail Gueaieb, and Davide Spinello. A policy iteration approach for flock motion control. In *IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pages 1–7, October 2021. (Accepted: Sep. 2021).

[80] Zhihua Qu, Jing Wang, and Richard A. Hull. Cooperative control of dynamical systems with application to autonomous vehicles. *IEEE Transactions on Automatic Control*, 53(4):894–911, 2008.

[81] Rouzbeh Razavi, S. Klein, and Holger Claussen. Self-optimization of capacity and coverage in lte networks using a fuzzy reinforcement learning approach. In *21st Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1865–1870, 2010.

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

[83] Eric Rohmer, Surya P. N. Singh, and Marc Freese. V-REP: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013.

[84] Michael T Rosenstein, Andrew G Barto, Jennie Si, Andy Barto, and Warren Powell. Supervised actor-critic reinforcement learning. *Learning and Approximate Dynamic Programming: Scaling Up to the Real World*, pages 359–380, 2004.

[85] Axel Rottmann and Wolfram Burgard. Adaptive autonomous control using online value iteration with gaussian processes. In *2009 IEEE International Conference on Robotics and Automation*, pages 2106–2111, 2009.

[86] Basant Kumar Sahu, Madan M. Gupta, and Bidyadhar Subudhi. Fuzzy separation potential function based flocking control of multiple AUVs. In *2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS)*, pages 1429–1434, 2013.

[87] Bhasker Sharma and Neeta Awasthy. State feedback controller design via T-S fuzzy model. In *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, volume 4, pages 176–181, 2009.

[88] Jan Sikora and Renata Wagnerová. Overview of reinforcement learning and its application in control theory. In *2020 21th International Carpathian Control Conference (ICCC)*, pages 1–4, 2020.

[89] Harpreet Singh, Madan M. Gupta, Thomas Meitzler, Zeng-Guang Hou, Kum K. Garg, Ashu M. G. Solo, and Lotfi A. Zadeh. Real-life applications of fuzzy logic. *Advances in Fuzzy Systems*, 2013:1–3, 2013.

[90] Jorge M. Soares, A. Pedro Aguiar, António M. Pascoal, and Alcherio Martinoli. A distributed formation-based odor source localization algorithm - design, implementation, and wind tunnel evaluation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1830–1836, 2015.

[91] Christian Speck and Donald J. Bucci. Distributed UAV swarm formation control via object-focused, multi-objective sarsa. In *2018 Annual American Control Conference*, pages 6596–6601, 2018.

[92] Chun-Yi Su and Yu Stepanenko. Adaptive control of a class of nonlinear systems with fuzzy logic. *IEEE Transactions on Fuzzy Systems*, 2(4):285–294, 1994.

[93] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[94] Richard S. Sutton, Andrew G. Barto, and Ronald J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, 1992.

[95] Ardalan Vahidi, Anna Stefanopoulou, and Huei Peng. Recursive least squares with forgetting for online estimation of vehicle mass and road grade: Theory and experiments. *Vehicle System Dynamics - VEH SYST DYN*, 43:31–55, January 2005.

[96] John Valasek, James Doebbler, M. D. Tandale, and Andrew J. Meade. Improved adaptive–reinforcement learning control for morphing unmanned air vehicles. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):1014–1020, 2008.

[97] W. M. van Buijtenen, G. Schram, Robert Babuska, and H. B. Verbruggen. Adaptive fuzzy control of satellite attitude by reinforcement learning. *IEEE Transactions on Fuzzy Systems*, 6(2):185–194, May 1998.

[98] Xue-Song Wang, Yu-Hu Cheng, and Jian-Qiang Yi. A fuzzy actor–critic reinforcement learning network. *Information Sciences*, 177(18):3764–3781, 2007.

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

[100] Qinglai Wei, Derong Liu, and Hanquan Lin. Value iteration adaptive dynamic programming for optimal control of discrete-time nonlinear systems. *IEEE Transactions on Cybernetics*, 46(3):840–853, 2016.

[101] Guoxing Wen, C. L. Philip Chen, Shuzhi Sam Ge, Hongli Yang, and Xiaoguang Liu. Optimized adaptive nonlinear tracking control using actor–critic reinforcement learning strategy. *IEEE Transactions on Industrial Informatics*, 15(9):4969–4977, 2019.

[102] Paul Werbos. ADP: Goals, opportunities and principles. *Handbook of Learning and Approximate Dynamic Programming*, 2:3–44, 2004.

[103] Paul J. Werbos. Reinforcement learning and approximate dynamic programming (RLADP)-foundations, common misconceptions, and the challenges ahead. *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, pages 1–30, 2013.

[104] Joohyun Woo, Chanwoo Yu, and Nakwan Kim. Deep reinforcement learning-based controller for path following of an unmanned surface vehicle. *Ocean Engineering*, 183:155–166, 2019.

[105] Peter Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29:9–20, 03 2008.

[106] Xin Xu, Dewen Hu, and Xicheng Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007.

[107] Jichen Yang, Qishao Lu, and Juezhi Chen. Flocking of multi-agent systems with non-smooth control. In *2009 Fifth International Conference on Natural Computation*, volume 5, pages 556–560, 2009.

[108] Hui Yu and Tiecheng Zhang. Center guided flocking motion of multi-agent using fuzzy logic. In *2009 International Conference on Computational Intelligence and Software Engineering*, pages 1–4, 2009.

[109] Hui Yu, Tiecheng Zhang, and Jigui Jian. Flocking with obstacle avoidance based on fuzzy logic. In *IEEE ICCA 2010*, pages 1876–1881, 2010.

[110] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

[111] Safdar Zaman, Wolfgang Slany, and Gerald Steinbauer. ROS-based mapping, localization and autonomous navigation using a pioneer 3-DX robot and their relevant issues. In *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, pages 1–5, 2011.

[112] Jin Zhao, Jianjing Wang, Huajun Zhang, and Wei Yang. Reinforcement learning based self-constructing fuzzy neural network controller for ac motor drives. In *2011 6th IEEE Conference on Industrial Electronics and Applications*, pages 913–918, 2011.

[113] Anton Zhilenkov, Sergei Chernyi, and Andrey Firsov. Autonomous underwater robot fuzzy motion control system with parametric uncertainties. *Designs*, 5(1), 2021.