Multi-Robot Learning in the Guarding a Territory Game

by

Chidozie Vincent Analikwu, B.Eng.

A thesis submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario
July, 2016

©Copyright

Chidozie Vincent Analikwu, 2016

The undersigned hereby recommends to the
Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis

Multi-Robot Learning in the Guarding a Territory Game

submitted by **Chidozie Vincent Analikwu, B.Eng.**

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

Professor Howard Schwartz, Thesis Supervisor

Professor Yvan Labiche, Chair,
Department of Systems and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

July, 2016

Abstract

In this thesis, we explore reinforcement learning in the game of guarding a territory which is played in the continuous domain. We make the assumption that the players have no a priori knowledge of their optimal behaviors. Therefore, we apply reinforcement learning to train the players to find their optimal behaviors. To our knowledge, this is the first investigation of both the invader and the guard learning simultaneously. In addition, we look at the possibility of an invader which is superior (faster) to a group of guards. To determine the optimal solution of the game when the players have different speeds and evaluate the players learning performance, we apply the Apollonius circle approach. This is the first application of the Apollonius circle approach to the guarding a territory game that we know of. Simulation results from this study show that the players are able to learn their optimal strategies simultaneously.

Acknowledgments

I owe my deepest gratitude to my thesis supervisor Prof. Howard Schwartz for his patient guidance and support throughout my graduate studies. This thesis would not have been possible without his enthusiasm, advice and continuous encouragement.

It is a pleasure to thank my friends Vidya David, Funmilayo Adaramola and Ifaz Haider for all the suggestions and advice they gave me to improve this thesis. I would also like to thank my friend Ikenna Ezenwajiaku for his frequent help and support.

I also want to express my appreciation to the professors, staff and colleagues I had the opportunity to interact with during the course of my graduate studies. Thanks also go to my colleagues Ahmad Atiya and Mostafa Awheda for their friendship and support.

Finally, many thanks and gratitude to my family for their endless support, love, and encouragement.

Table of Contents

Abstract	iii
Acknowledgments	iv
Table of Contents	v
List of Figures	ix
1 Introduction	1
1.1 Overview	1
1.2 Motivation	3
1.3 Contributions	6
1.4 Thesis Organization	6
2 Literature Review	8
2.1 Reinforcement Learning	9
2.2 The Differential Game of Guarding a Territory	12
2.3 Reinforcement Learning in the Guarding a Territory Game	14
2.4 Summary	15
3 Reinforcement Learning	17
3.1 Discounted Future Reward	18

3.2	Markov Decision Processes	21
3.3	Dynamic Programming	25
3.3.1	Value Iteration	26
3.4	Temporal Difference Learning	31
3.4.1	Q-Learning	32
3.5	Actor–Critic Methods	34
3.6	Summary	35
4	The Guarding a Territory Problem	36
4.1	A Brief Introduction to Differential Games	37
4.2	The Apollonius Circle	39
4.3	The Optimal Strategies of the Players	40
4.3.1	Two-Player Game: An Invader Against a Guard With the Same Speed	41
4.3.2	Two-Player Game: An Inferior Invader Against a Superior Guard	43
4.3.3	Three-Player Game: A Superior Invader Against Two Inferior Guards	45
4.4	Problem Formulation	48
4.4.1	Two-Player Game: An Invader Against a Guard With the Same Speed	49
4.4.2	Two-Player Game: An Inferior Invader Against a Superior Guard	50
4.4.3	Three-Player Game: A Superior Invader Against Two Inferior Guards	51
4.5	Summary	52
5	Fuzzy Reinforcement Learning	53

5.1	A Brief Introduction to Fuzzy Logic	54
5.1.1	Fuzzy Set Operations	55
5.1.2	Fuzzy Reasoning	57
5.1.3	Fuzzy Logic Controller	60
5.2	Fuzzy Controller Design for the Guarding a Territory Problem	62
5.3	Fuzzy Actor-Critic Learning	67
5.4	Fuzzy Q-Learning	70
5.5	Comparison of the Fuzzy Q-Learning with the Fuzzy Actor-Critic Learning	73
5.6	Reward Function	74
5.6.1	Reward Function for the Invader	75
5.6.2	Reward Function for the Guard	76
5.7	Summary	77
6	Simulation Results	79
6.1	Two Player Game: An Invader Against a Guard With the Same Speed	81
6.1.1	Possible Capture	82
6.1.2	Possible Reaching	87
6.1.3	Discussion	91
6.2	Two Player Game: An Inferior Invader Against a Superior Guard . .	92
6.2.1	Possible Capture	93
6.2.2	Possible Reaching	96
6.2.3	Discussion	98
6.3	Three-Player Game: A Superior Invader Against Two Inferior Guards	100
6.3.1	Possible Capture	103
6.3.2	Possible Reaching	107

6.3.3	Discussion	109
6.4	Summary	110
7	Conclusion	111
7.1	Contributions	112
7.2	Future Work	113
	List of References	115

List of Figures

3.1	The agent-environment interaction scheme.	18
3.2	An example 1×8 grid world.	20
3.3	An example of Value Iteration (a) 4×4 grid world (b) 16 states of the grid world.	27
3.4	The optimal policy.	29
3.5	The optimal policy after we modified the dynamics of the environment.	31
3.6	The actor-critic scheme.	34
4.1	The Apollonius circle AC constructed for an inferior invader I against a superior guard G	40
4.2	The optimal strategy of an invader I against a guard G with the same speed.	41
4.3	The optimal strategy of an invader I against a guard G with the same speed. The perpendicular bisector line (PBL) is constructed.	42
4.4	The optimal strategy of an inferior invader I against a superior guard G	43
4.5	The optimal strategy of an inferior invader I against a superior guard G . The Apollonius circle AC is constructed.	44
4.6	The optimal strategy of a superior invader I against two inferior guards G_1 and G_2	45

4.7	The strategy of a superior invader I against two inferior guards G_1 and G_2 . The Apollonius circles AC_1 and AC_2 are constructed.	46
4.8	The optimal strategy of a superior invader I against two inferior guards G_1 and G_2 when the environment is bounded.	47
4.9	Two-player game setup: An invader against a guard with the same speed.	49
4.10	Two-player game setup: An inferior invader against a superior guard.	50
4.11	Three-player game setup: A superior invader against two inferior guards.	51
5.1	The fuzzy logic controller (FLC) scheme.	60
5.2	The inputs to the controller of the invader.	63
5.3	The membership functions for (a) the angle variable θ (b) the Manhattan distance variable d	65
5.4	The fuzzy actor-critic learning (FACL) scheme for the guarding a territory problem.	68
5.5	The fuzzy Q-learning (FQL) scheme for the guarding a territory problem.	71
5.6	The guard's reward function description.	77
6.1	The game setup for an invader against a guard with the same speed—possible capture.	82
6.2	FQL simulation: The trajectory plot for an invader against a guard with the same speed—possible capture (a) the first epoch (b) the final epoch.	83
6.3	FACL simulation: The trajectory plot for an invader against a guard with the same speed—possible capture (a) the first epoch (b) the final epoch.	85
6.4	The average payoff plot for an invader against a guard with the same speed—possible capture.	86

6.5	The average termination time plot for an invader against a guard with the same speed—possible capture.	86
6.6	The game setup for an invader against a guard with the same speed—possible reaching.	88
6.7	FQL simulation: The trajectory plot of the final epoch for an invader against a guard with the same speed—possible reaching.	88
6.8	FACL simulation: The trajectory plot of the final epoch for an invader against a guard with the same speed—possible reaching.	89
6.9	The average payoff plot for an invader against a guard with the same speed—possible reaching.	90
6.10	The average termination time plot for an invader against a guard with the same speed—possible reaching.	90
6.11	The game setup for an inferior invader against a superior guard—possible capture.	94
6.12	FACL simulation: The trajectory plot for an inferior invader against a superior guard—possible capture (a) the first epoch (b) the final epoch.	95
6.13	The average payoff plot for an inferior invader against a superior guard—possible capture.	96
6.14	The game setup for an inferior invader against a superior guard—possible reaching.	97
6.15	FACL simulation: The trajectory plot of the final epoch for an inferior invader against a superior guard—possible reaching.	97
6.16	The average payoff plot for an inferior invader against a superior guard—possible reaching.	98

6.17	FACL simulation: The trajectory plot for an inferior invader against a superior guard with the test location for the invader set at $[8.5, 6.5]$, and the test location for the guard set at $[8.5, 26.5]$ —possible capture (a) the game setup (b) the first simulation result.	99
6.18	The game setup for a superior invader against two inferior guards—possible capture.	103
6.19	FACL simulation: The trajectory plot of the final epoch for a superior invader against two inferior guards with guard 2 located at coordinate $[25, 25]$ absent.	104
6.20	The average payoff plot for a superior invader against two inferior guards with guard 2 located at coordinate $[25, 25]$ absent.	104
6.21	FACL simulation: The trajectory plot for a superior invader against two inferior guards with guard 2 located at coordinate $[25, 25]$ present—possible capture (a) the first epoch (b) the final epoch. . . .	106
6.22	The average payoff plot for a superior invader against two inferior guards with guard 2 located at coordinate $[25, 25]$ present—possible capture.	107
6.23	The game setup for a superior invader against two inferior guards—possible reaching.	108
6.24	FACL simulation: The trajectory plot of final epoch for a superior invader against two inferior guards—possible reaching.	108
6.25	The average payoff plot for a superior invader against two inferior guards—possible reaching.	109

Chapter 1

Introduction

1.1 Overview

Autonomous robots are intelligent machines that do not require explicit human guidance to achieve set goals [1]. These self-reliant robots are able to perceive the state of their environment, reason about it and take actions to achieve certain objectives. In a dynamic environment, autonomous robots can adapt their behavior through learning algorithms to achieve optimum performance. Autonomous robots are also called intelligent agents. When more than one robot is deployed for a specific task, they are commonly referred to as intelligent multi-robot (or multi-agent) systems. The application of intelligent multi-robot systems can be found in exploration [2], [3], security missions [4], [5], intelligent transportation [6], and many other applications.

Intelligent multi-robot systems will play a major role in future security missions. Recently, Knightscope, a security company based in California, developed an autonomous security robot called the *K5 Autonomous Data Machine* [7]. The K5 utilizes a high definition camera along with various sensors and lasers to autonomously navigate and gather information about its environment. The information gathered is transmitted to a security data center for real time alerts and forensics. The optimal

strategy for multi-robot systems, such as the K5, to guard a secured area can be studied under the framework of the guarding a territory game.

The guarding a territory game was first introduced by Isaacs [8]. In the game, an invader tries to get as close as possible to the territory which is defended by a guard. The goal of the guard is to intercept the invader as far as possible from the territory. Practical applications of the game can be found in missile defense, border security and other security applications. The payoff in the guarding a territory game is the distance between the invader and the territory at game termination. The conflict of interest between the invader, who wants to minimize the payoff, and the guard, who wants to maximize the payoff, makes the guarding a territory problem challenging. If the players have no a priori knowledge of their optimal strategies, they will have to learn these while playing the game.

Reinforcement learning is concerned with how an agent can learn to behave rationally in an unknown environment to achieve its goal. In this scenario, the agent has no explicit supervisor to guide its learning process. However, the agent is able to perceive the state of its environment and take action to influence its environment. The environment returns a numerical reward which is the consequence of the action taken by the agent. The reward takes the form of a credit or a punishment for performing a particular action. The learning process of the agent is thus guided by the reward. The goal of the agent is to learn a policy that maximizes its long-term cumulative reward. The policy is a function that indicates which action the agent should choose in any given state.

The policy can be in the form of a look-up table, if the state and action spaces are discrete and small in number. However, if the state and action spaces are large or in the continuous domain, it is intractable to maintain a look-up table. Function approximators, such as the fuzzy logic controller (FLC), can be used to represent the

continuous state and action spaces in an effective way [9], [10]. FLCs are universal approximators that can estimate any continuous function to any degree of accuracy [11]. Reinforcement learning combined with FLCs can be used to solve problems in the continuous domain. When reinforcement learning is combined with FLC, the approach is commonly called *fuzzy reinforcement learning* [12], [13].

In this thesis, we apply fuzzy reinforcement learning to the game of guarding a territory which is played in the continuous domain. We make the assumption that the players have no a priori knowledge of their optimal behaviors. Therefore, the players will obtain this after learning. To our knowledge, this is the first work to investigate the possibility of all the players (the invader and the guard) simultaneously learning in the game of guarding a territory.

1.2 Motivation

Isaacs' [8] guarding a territory game is a differential game played in the continuous domain. In the game, a guard tries to intercept an invader as far as possible from the territory. The goal of the invader is to avoid capture and reach the territory if it can or, if this is not possible, to get as close as possible to the territory before capture occurs. Isaacs [8] introduced the perpendicular bisector approach to determine the optimal solution of the game when the players have the same speed.

The guarding a territory game has been investigated previously in the literature [14], [15]. In [14], a FLC was used to approximate the specific location of the invader. In [15], a FLC was used to reason the strategy of the invader to enable the guard choose the most effective strategy to capture the invader. However, in these publications [14], [15] the players are assumed to have a priori knowledge of their optimal behaviors. In many real-world applications of the game, for example

autonomous multi-robot systems for protecting critical infrastructure, the robots may not have a priori knowledge of their optimal behaviors. Therefore, the robots would need to learn and adapt to challenging and unpredictable environments.

The guarding a territory game has been investigated previously as a learning problem in the literature [16], [17]. Lu [16] used reinforcement learning to train only the guard in the game. Simulation results in [16] demonstrated that the guard was able to learn its optimal behavior to capture the invader in an effective way. Raslan [17] modified the approach in [16] to investigate the possibility of only the invader learning in the game. Simulation results in [17] showed that the invader was able to learn its optimal behaviors to counter different strategies chosen by the guard. The work done in [16], [17] are important contributions to the field of machine learning. However, to our knowledge, no work has investigated the possibility of both players (the invader and the guard) simultaneously learning in the game.

When the players have the same speed, the optimal solution of the guarding a territory game can be determined using the perpendicular bisector approach [8]. However, when the players do not have the same speed, the optimal solution of the game cannot be determined using this same approach. There has been little research done in the literature to determine the optimal solution of the game when the players do not have the same speed. Given the many applications of the guarding a territory game in security operations, this is an important scenario to explore. Therefore, we are motivated to study the optimal solution of the game when the players have different speeds.

Our research led us to the pursuit-evasion game, which is well-studied in the literature [18], [19], [20], [21]. The pursuit-evasion game is also a differential game introduced by Isaacs [8]. We observed that the optimal solution of the pursuit-evasion game was mainly determined using the Apollonius circle formed by the pursuer and

the evader. We investigated whether the approach can be used to determine the optimal solution of the game of guarding a territory. We discovered that the Apollonius circle approach can effectively be used to determine the optimal solution of the guarding a territory game when the players do not have the same speed.

In this thesis, we introduce the Apollonius circle approach to determine the optimal solution of the game of guarding a territory when the players do not have the same speed. The Apollonius circle approach makes use of the Apollonius circle formed by the invader and the guard to determine the theoretical solution of the game. To our knowledge, this is the first work to apply the Apollonius circle to the game of guarding a territory. Although, previously, this technique has been applied in the pursuit-evasion game [8], [18], [19]. Practical applications of the Apollonius circle approach in the real-world can be found in developing powerful missile defense system, collision avoidance system and many other applications.

The main focus of this thesis is to investigate the use of reinforcement learning to train the players in the game of guarding a territory. Different from the work done in [16], where only the guard was learning, and in [17], where only the invader was learning, we investigate the possibility of both players simultaneously learning in the game. Moreover, in our formulation, the players can have different speeds. As such, we make use of the optimal solution given by the Apollonius circle approach to evaluate the learning performance of the players.

Furthermore, we investigate a more complex problem which involves two guards working jointly to stop a superior (or a faster) invader from reaching the territory. All three players have no a priori knowledge of their optimal behaviors and therefore are simultaneously learning in the game. To our knowledge, this is the first work to investigate how a group of guards can work together to capture a superior invader. Practical applications of the proposed approach in the real-world can be found in

developing adaptive multi-robot systems that are able to neutralize a threat (persons or robots) at the farthest possibility from the target.

1.3 Contributions

The main contributions of this thesis are:

1. We introduce the Apollonius circle approach to determine the optimal solution of the game of guarding a territory when the players do not have the same speed.
2. We modify the approach in [16], which considered only the guard learning, and in [17], which considered only the invader learning, to investigate the possibility of both players (the invader and the guard) simultaneously learning in the game.
3. We investigate a more complex problem which involves two guards working jointly to defend the territory against a superior invader. All three players in this scenario are simultaneously learning.
4. We design instantaneous reward functions for the learning process of the players.

1.4 Thesis Organization

The thesis is organized as follows:

- **Chapter 2** provides relevant historical background and highlights the main challenges faced by reinforcement learning when the state and action spaces are

large or in the continuous domain. Furthermore, we introduce the theory of differential games, focusing on the guarding a territory game.

- **Chapter 3** provides background information on reinforcement learning. We briefly review the Markov decision process, dynamic programming and temporal difference learning methods.
- **Chapter 4** provides background information on differential games. We introduce the Apollonius circle and subsequently present the Apollonius circle approach. We formulate three problems that we make use of to evaluate the learning process of the players.
- **Chapter 5** provides a brief introduction to fuzzy logic. We provide details of the design of the controller for the guarding a territory game. We review the FQL algorithm and the FACL algorithm. We also provide specifics for the design of the reward functions.
- **Chapter 6** presents the simulation results that demonstrate the performance of the learning agents. We simulate and compare the FQL algorithm with the FACL algorithm. Detail analyses and discussions of the simulation results will be provided.
- **Conclusion** provides the summary of our work and future directions.

Chapter 2

Literature Review

In this chapter, we introduce the basic principles behind reinforcement learning. We present one of the most popular model-free methods for solving reinforcement learning problems called *Q-learning*. We highlight the main challenges faced by Q-learning when the state and action spaces are large or in the continuous domain. We also discuss function approximation techniques for dealing with the problem of continuous state and action spaces, and briefly survey real-world applications of these techniques in the field of robotics. In addition, we introduce the theory of differential games, with a focus on the guarding a territory game. Finally, we provide an overview of recent studies that have explored reinforcement learning in the guarding a territory game.

This chapter is organized as follows. Section 2.1 introduces the basic principles behind reinforcement learning and discusses the main challenges faced by reinforcement learning when solving problems with continuous domains. In Section 2.2, we present differential game theory, focusing on the guarding a territory game. In Section 2.3, we present and discuss recent works that have explored reinforcement learning in the guarding a territory game.

2.1 Reinforcement Learning

The idea behind learning through trial-and-error stems from research in animal behavior and psychology. One of the earliest objective analyses of learning in animals was performed by Edward Thorndike in his 1898 doctoral dissertation [22], [23]. The series of experiments he conducted on animals (primarily cats) demonstrated that animals can use trial-and-error to learn tasks [23], [24]. This learning process requires that the animals receive reinforcement, such as food, for executing the desired behavior [24]. The reinforcement strengthens the behavior and therefore increases the likelihood that the animal will execute the right action (or sequence of actions) under similar circumstances.

However, it was Richard Bellman [25] and others who developed the mathematical framework known as the Markov Decision Process (MDP) to study and analyze sequential decision problems. In this framework, an agent (or decision maker) interacts with the environment by executing actions and observing their outcomes. Actions transform the environment and give rise to a scalar reward (or reinforcement) signal. The state transition and the reward can be probabilistic in some cases. The state transition is usually defined using a transition function, whereas, the reward distribution is given by a reward function. The goal of the agent is to learn a policy that maximizes its long-term cumulative reward. In this context, we define an optimal policy as the policy that yields the most reward. Bellman [25] also introduced the Value Iteration method to compute optimal policies given a perfect model of the environment. In 1960, Ron Howard [26] developed the Policy Iteration method for solving MDPs. Value Iteration and Policy Iteration are generally referred to as Dynamic Programming (DP) methods [27], [28].

If the complete dynamics of the MDP is known, one can apply any DP method to

determine the optimal policy. However, in many real-world problems, the transition function and the reward function are not known in advance. In this scenario, the agent must explore the environment to learn which action yields the most reward. The agent must also exploit what it already knows in order to improve its performance in the long run. As such, in order to perform well, the agent must find a balance between exploring the unknown environment and exploiting its previous experiences. Multi-armed bandit problems, which model the exploration-exploitation dilemma in machine learning, have been studied extensively in the literature [27], [29].

Two options are possible when solving MDPs with incomplete information. The first one is to learn the model (i.e. the transition function and the reward function) by sampling the environment and collecting statistics. Once the model is approximately correct, any of the DP methods can be applied to compute the optimal policy. This approach is commonly known as model-based reinforcement learning [28], [30]. The second option is to incrementally estimate the value function based on the immediate reward and the estimated value of the next state. The value function represents the long-term cumulative reward the agent can expect to gain starting from that state. Almost all reinforcement learning algorithms compute optimal policies by learning value functions [27], [28]. The second approach is commonly known as model-free reinforcement learning [28], [30]. In this section, we will focus on model-free learning methods.

Temporal difference (TD) learning was developed by Sutton [31] and others as an approach to learn directly from raw experience. TD methods perform bootstrapping, in the sense that they update previous estimates on the basis of new estimates [27]. Some of the convergence properties of TD learning have been proved in the literature [31], [32], [33]. Barto et al. [34] and others developed the actor-critic architecture which uses a separate memory structure to represent the decision mechanism (actor)

independent of the value function estimator (critic). Actor-critic methods combine trial-and-error learning with TD learning to solve the prediction problem (i.e. estimating the value function).

Christopher Watkins [35], in his 1989 doctoral dissertation, developed the Q-learning algorithm which successfully combined TD learning and trial-and-error learning to solve MDPs with incomplete information. Q-learning was a major breakthrough in reinforcement learning. The basic idea in Q-learning is to iteratively learn Q-values (or action-value functions) based on the immediate reward and the estimated Q-values of the next state. Watkins and Dayan [36] proved that Q-learning converges to the optimum Q-values regardless of the policy (or exploration strategy) being followed. For this reason, Q-learning is sometimes referred to as an off-policy learning method [27], [37]. Q-learning does not require the complete knowledge of the environment's dynamics in order to determine the optimal policy. As such, Q-learning is a model-free learning method [30], [37].

Reinforcement learning methods, such as Q-learning [35], explicitly store all the Q-values (i.e. state-action pairs) in tabular form. This approach is suitable for problems with small finite state and action spaces. However, for problems with continuous state and action spaces, the tabular representation will require a priori discretization of the continuous spaces. As such, if a finely grained discretization is done so as to obtain good accuracy, this will result in heavy memory requirements and slow learning procedures [38], [39]. In reinforcement learning, this problem is well-known as the *curse of dimensionality* [40], [41]. The term *curse of dimensionality* was coined by Bellman when he faced a similar challenge while exploring dynamic optimization problems in high-dimensional spaces [25], [42].

The combination of reinforcement learning methods with function approximators often appears in the literature as a popular technique to deal with large or continuous

spaces [9], [27], [39], [41], [43]. Function-approximation techniques, including various fuzzy logic controllers (FLC) [41], [44], [45], [46], and neural networks [43], [47], have been used to represent the continuous state and action spaces with considerable success. The FLC has two significant advantages when used as a function-approximation technique: First, the FLC permits human heuristic knowledge to be incorporated in the control design; Secondly, the learning process can be localized to only the consequent parameters [10], [46]. The combination of the FLC with reinforcement learning is sometimes called *fuzzy reinforcement learning* [12], [13].

In recent years, there has been increasing interest in function approximation methods for reinforcement learning, especially in the field of robotics to solve real-world problems [48], [49], [50], [51], [52], [53]. Duan et al. showed in [48] that fuzzy neural networks can be used with reinforcement learning to make a *Pioneer II* robot learn basic navigation. Cerebellar Model Articulation Controller (CMAC) was applied in [49] to enable a biped robot learn to walk. Gaskett et al. [50] applied neural networks to make a *Nomad 200* robot learn a wandering behavior and visual servoing. Gullapalli et al. [53] applied neural networks to get a *Zebra Zero* robot to learn a peg-in-hole insertion task. These studies demonstrate the success and progress of function approximation techniques.

2.2 The Differential Game of Guarding a Territory

The theory of differential games was developed in the 1950s by Rufus Isaacs [8], who was then with the Mathematics Department of the RAND Corporation. Differential game theory provides a mathematical framework to study strategic interactions between multiple participants in systems which are driven by differential equations. Usually, the participants have conflicting aims (zero-sum games). The participants

are often referred to as players or agents. The theory of differential games brings together the principles of game theory, the calculus of variations, and control theory to study and analyze a wide variety of conflict problems [8]. These include the pursuit-evasion games, the guarding a territory game and many other problems of conflict. In this section, we will focus on the guarding a territory game.

Isaacs' [8] guarding a territory game is a differential game played in the continuous domain. In its simplest form, the game consist of two players: the invader and the guard. The goal of the guard is to intercept the invader as far as possible from the territory. Whereas, the goal of the invader is to avoid capture and get as close as possible to the territory. The payoff in the guarding a territory game is the distance between the invader and the territory at the end of the game. Therefore, it is rational that the guard finds the optimal strategy that maximizes the payoff while taking into consideration that the invader is doing the exact opposite. This conflict of interest between the players make the guarding a territory problem challenging.

The guarding a territory game has been investigated previously in the literature [14], [15], [54]. Hsia et al. [14] considered the differential game problem where the guard has imprecise knowledge of the location of the invader. A fuzzy controller was designed and optimized for the guard to locate the invader's position. Lee et al. [15] proposed a fuzzy reasoning algorithm to determine the invader's strategy based on the observed behavior of the invader over a time period. Once the invader's strategy has been reasonably inferred, the guard can then choose the most effective strategy to capture the invader. Lee et al. [54] provides an in-depth study and analysis of the guarding a territory problem with two guards and two invaders. They show that in certain arrangements of the players and the location of the territory it is possible for the invader to reach the territory.

The work done in [14], [15], [54] are important contributions to the field of differential games. However, the authors of the studies assumed that the players have a priori knowledge of their optimal behaviors. In other words, it was assumed that once the players are able to obtain precise information (or reliable estimates) of the location of their opponent and the territory, the direct application of the perpendicular bisector approach (proposed by Isaacs [8]) can be used to determine the players' optimal behavior. This assumption may not hold for all applications of the game in real-life. This is because the real-world is dynamic and highly demanding. In many real-world applications of the game, such as autonomous multi-robot systems for protecting critical infrastructure, it is difficult to specify in advance the optimal behavior for all possible situations the robots may encounter. For this reason, learning algorithms can be used to enable the robots learn, improve, and adapt to their operating environment in real time.

2.3 Reinforcement Learning in the Guarding a Territory Game

Previous studies in [16], [17] have investigated the guarding a territory game as a learning problem. Lu [16] investigated the possibility of only the guard learning in the game. In his formulation, both players (the invader and the guard) have equal kinematics. The game environment was assumed to be a two-dimensional bounded space with no obstacles. Lu employed holonomic robots as the players of the game. He applied two fuzzy reinforcement learning algorithms (the fuzzy Q-learning (FQL) and the fuzzy actor-critic learning (FACL)) to the guarding a territory problem. Simulation results obtained by Lu demonstrated that the guard was able to learn its optimal behavior to effectively capture the invader using both learning algorithms.

Raslan [17] modified the approach in [16] to investigate the possibility of only the invader learning. He increased the complexity of the guarding a territory problem by setting the guard to be 20 percent faster than the invader. Therefore, in this scenario, the guard is superior to the invader. Furthermore, Raslan employed non-holonomic robots as the players of the game. Similar to [16] he applied the FQL and the FACL algorithm to the guarding a territory problem. Simulation and experimental results obtained by Raslan demonstrated that the invader was able to learn its optimal behaviors to counter different strategies chosen by the guard.

The work done in [16], [17] are important contributions to the field of machine learning and differential games. However, to our knowledge, no work has yet investigated the possibility of both the invader and the guard simultaneously learning. Furthermore, it appears that a faster (or a superior) invader has so far not been investigated in the literature. It is not safe to assume that in all real-life scenarios the guards will always be faster than the intruder (or the invader). Therefore, there is a need to investigate the possibility of a scenario where the invader is faster than the guards. This setup may require multiple guards to work jointly to defend the territory against the faster invader, making the problem more complex. The remaining chapters of this thesis are devoted to exploring this and related scenarios.

2.4 Summary

In this chapter, we provided relevant historical background and highlighted the main challenges faced by reinforcement learning when solving problems with continuous domains. We discussed function approximation techniques for dealing with the problem of large or continuous state and action spaces. We also provided a number of

successful applications of function approximators in reinforcement learning. Furthermore, we introduced the theory of differential games, using the guarding a territory game as our primary example. We then briefly surveyed recent research that have applied reinforcement learning to the guarding a territory game.

Chapter 3

Reinforcement Learning

Reinforcement learning is a type of machine learning that can be used to train an agent to optimize some reward function through interaction with the initially unknown environment. Reinforcement learning is different from supervised learning. In supervised learning, the agent learns with the desired input-output pattern provided by an external supervisor [27]. However, in reinforcement learning, there is no external supervisor to guide the learning process of the agent. The agent has no knowledge of the desired output. Therefore, the agent must explore its available actions to discover which actions yield the most reward. The agent must also exploit what it already knows in order to improve its performance in the long run. The overall goal of the agent is to learn a policy (a mapping from states to actions) that maximizes its long-term cumulative reward [27].

Suppose the agent interacts with the environment in discrete time steps $t = 0, 1, 2, \dots$. At each time step t , the agent is presented with a state $x_t \in X$, where X is the state space. Based on the current state x_t , the agent executes an action $a_t \in A$, where A is the action space. The action a_t transforms the environment and a new state x_{t+1} is achieved. The environment returns a reward $r_{t+1} \in \mathbb{R}$, the consequence of the action taken by the agent. The reward is a scalar signal that indicates how

Figure 3.1: The agent-environment interaction scheme.

good or bad it is to take action a_t in state x_t . The agent has to retain a memory of the actions that resulted in the most reward from previously encountered situations. The history of past actions and observations forms a wealth of information, which can be used by the agent to make better future decisions and improve its performance over time. The agent-environment interaction scheme is shown in Fig. 3.1.

This chapter provides background information on reinforcement learning. In Section 3.1, we introduce the concept of discounted future rewards. Section 3.2 presents the Markov Decision Processes (MDP). In Section 3.3, we provide an introduction to dynamic programming. Temporal difference learning method is presented in Section 3.4. In Section 3.5, we discuss the actor-critic learning methods.

3.1 Discounted Future Reward

In reinforcement learning, the goal of the agent is to maximize the sum of future rewards R_t . If the reinforcement learning task has an associated terminal state, the cumulative reward R_t has a finite value and is given as

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T \quad (3.1)$$

where T is the terminal time. Reinforcement learning task with an associated terminal state is called an episodic task [27]. However, not all reinforcement learning tasks have an associated terminal state. Some tasks continue indefinitely. Reinforcement learning tasks that continue indefinitely are referred to as a continuous tasks [27]. The cumulative reward R_t for a continuous task have an infinite value and is given as

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots = \sum_{k=0}^{\infty} r_{t+k+1} \quad (3.2)$$

A fundamental concept in reinforcement learning to solve problems that continue indefinitely is the concept of discounted future rewards. An exponential discounting factor γ ($0 \leq \gamma \leq 1$) is introduced to the reward function given in (3.2) to obtain the modified reward function given in (3.3)

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (3.3)$$

The cumulative reward R_t for a continuous task have a finite value as long as $\gamma < 1$ and the reward sequence r_k is bounded [27]. The discounting factor γ also determines the present value of future rewards [27]. If γ is steeply discounted, the agent will strongly prefer immediate rewards to rewards later in the future. As γ approaches 1, the agent will more strongly consider rewards that can be obtained in the future [27].

Example 2.1: We take an example of a 1×8 grid world shown in Fig. 3.2, to illustrate the concept of discounted future reward. The goal of the agent in this scenario, is to exit the grid through the door that maximizes its cumulative reward. To simplify the problem, we make the assumption that state transitions are deterministic. The agent is presented with only two policies and has to decide which of these policies is the optimum one (i.e. maximizes its cumulative reward). The two policies are: always go

Figure 3.2: An example 1×8 grid world.

right and always go left. We can represent this two policies in the form

$$\text{Always go right } (\pi_R) : \pi(x, a) \quad a = \text{Right } \forall x \in X \quad (3.4)$$

$$\text{Always go left } (\pi_L) : \pi(x, a) \quad a = \text{Left } \forall x \in X \quad (3.5)$$

The cells of the grid represent the states of the environment. The game has 8 states numbered 1 to 8 starting from the left, as shown in Fig. 3.2. State x_1 and x_8 are the terminal states of the game. The agent's initial position is set at state x_3 . A reward of +10 is given if the agent reaches state x_1 . A reward of +30 is given if the agent reaches state x_8 . A reward of -1 is given for any transition between states other than the terminal states. We set $\gamma = 0.7$. The cumulative reward obtained from following the two policies is computed as follows:

$$\begin{aligned} R^{\pi_R} &= -1 + 0.7^1(-1) + 0.7^2(-1) + 0.7^3(-1) + 0.7^4(+30) \\ &= 4.7 \end{aligned} \quad (3.6)$$

$$\begin{aligned} R^{\pi_L} &= -1 + 0.7^1(+10) \\ &= 6.0 \end{aligned} \quad (3.7)$$

From this example, we observe that the agent will prefer the policy π_L over π_R because it achieves the maximum cumulative reward of 6.0 given in (3.7). As such, in

this context, we regard π_L as the *optimal policy*. In this thesis, we assign the notation π^* for an optimal policy. Similarly, we refer to any agent that follows its optimal policy as a *rational agent*. If we set $\gamma = 1$, the agent would prefer policy π_R over π_L , because this achieves the maximum cumulative reward of 26 given in (3.8).

$$\begin{aligned} R^{\pi_R} &= -1 + (-1) + (-1) + (-1) + (+30) \\ &= 26.0 \end{aligned} \tag{3.8}$$

$$\begin{aligned} R^{\pi_L} &= -1 + (+10) \\ &= 9.0 \end{aligned} \tag{3.9}$$

This examples shows that as the value of γ gets smaller, the agent becomes more myopic, preferring rewards sooner than later. As γ approaches 1, the agent becomes more farsighted, preferring rewards that can be obtained in the future.

3.2 Markov Decision Processes

The Markov property requires that the next state x_{t+1} depend only on the current state x_t and action a_t and not on the history of previous successive states and actions leading to x_t [27]. In other words, the current state x_t has all the information necessary to decide the next action a_t . The Markov property can be written in the form [27]

$$Pr\{x_{t+1} = x' \mid x_t, a_t\} \quad \forall x, x' \in X, a \in A \tag{3.10}$$

where $Pr\{x_{t+1} = x' \mid x_t, a_t\}$ is the probability of transitioning to state x' after executing action a_t in state x_t . We observe in Equation (3.10) that the next state x_{t+1} depends only on the current state x_t and action a_t , and not on the history of previous

successive states and actions $(x_{t-1}, a_{t-1}, x_{t-2}, a_{t-2}, \dots, x_0, a_0)$.

Any reinforcement learning task that satisfies the Markov property is called a Markov Decision Process (MDP) [27]. The MDP is called a *finite MDP* if the state and action spaces are finite [27]. Any MDP can be characterized by the 5-tuple [9], [55]

$$\{X, A, \gamma, T(x, a, x'), R(x, a, x')\} \quad (3.11)$$

where X is the state space, A is the action space, γ ($0 \leq \gamma \leq 1$) is an exponential discounting factor, $T(\cdot)$ is the transition probability also called the transition function and $R(\cdot)$ is the reward function. We define the transition function $T(x, a, x')$ as the probability of transitioning to state x' after executing action a in state x given as [27]

$$T(x, a, x') = \Pr\{x_{t+1} = x' \mid x_t, a_t\} \quad (3.12)$$

The transition function satisfies

$$\sum_{x'} T(x, a, x') = 1 \quad \forall x, x' \in X, a \in A \quad (3.13)$$

Similarly, we define the reward function $R(x, a, x')$ as the expected value of the next reward after executing action a in state x and transitioning to state x' given as [27]

$$R(x, a, x') = E\{r_{t+1} = r \mid x_{t+1} = x', a_t = a, x_t = x\} \quad (3.14)$$

The reward function immediately provides the agent with information of how good or bad a given action a_t is in state x_t . The goal of the agent is not to maximize the immediate rewards but to maximize the long-term cumulative reward. The agent uses the instantaneous rewards to compute a state-value function for every state $x \in X$.

The state-value function indicates the long-term desirability of any given state [27]. The agent has to take the state-value function into more consideration in deciding future actions in order to perform well in the long-term.

We define the state-value function $V^\pi(x)$ acting under the policy π as the expected cumulative reward starting from state x and following policy π thereafter given as [27]

$$\begin{aligned}
V^\pi(x) &= E_\pi\{R_t \mid x_t = x\} \\
&= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid x_t = x\right\} \\
&= E_\pi\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid x_t = x\right\} \\
&= E_\pi\{r_{t+1} + \gamma V^\pi(x') \mid x_t = x\} \\
&= \sum_a \pi(x, a) \sum_{x'} T(x, a, x') \{R(x, a, x') + \gamma V^\pi(x')\} \quad (3.15)
\end{aligned}$$

where the notation x' refers to the next state, $\pi(x, a)$ is the probability assigned for taken action a in state x under policy π and $E_\pi\{\}$ denotes the expected value given that the agent acts under the policy π . The policy π satisfies

$$\sum_a \pi(x, a) = 1 \quad \forall x \in X, a \in A \quad (3.16)$$

Similarly, we define the action-value function $Q^\pi(x, a)$ acting under the policy π as the expected cumulative reward starting from state x , executing action a and

thereafter following policy π given as [27]

$$\begin{aligned}
 Q^\pi(x, a) &= E_\pi\{R_t \mid x_t = x, a_t = a\} \\
 &= E_\pi\{r_{t+1} + \gamma V^\pi(x) \mid x_t = x, a_t = a\} \\
 &= \sum_{x'} T(x, a, x') \{R(x, a, x') + \gamma V^\pi(x')\}
 \end{aligned} \tag{3.17}$$

Equation (3.15) and (3.17) are called the Bellman equation for the state-value function $V^\pi(x)$ and the action-value function $Q^\pi(x, a)$ respectively, acting under policy π [27]. The action-value function $Q(x, a)$ determines the value of each state-action pair also known as the Q-values. If the agent always chooses the actions with the highest estimated Q-values, the agent is said to be following the *greedy strategy* [9], [30].

We define the optimal state-value function $V^*(x)$ as the maximum expected cumulative reward starting from state x and thereafter acting optimally given as

$$V^*(x) = \max_{\pi} V^\pi(x) \tag{3.18}$$

Given that we are taking the maximum over all policies in (3.18), a policy π' is said to be better than or equal to a policy π (i.e. $\pi' \geq \pi$) if and only if its expected cumulative reward is greater than or equal to that of π for all states, given as [27]

$$V^{\pi'}(x) \geq V^\pi(x) \quad \forall x \in X \tag{3.19}$$

The policy π that maximizes the expected cumulative reward in the long run is called the optimal policy π^* [9].

Similarly, we define the optimal action-value function $Q^*(x, a)$ as the maximum expected cumulative reward starting from state x , executing action a and thereafter

acting optimally given as

$$\begin{aligned} Q^*(x, a) &= \max_{\pi} Q^{\pi}(x, a) \\ &= \sum_{x'} T(x, a, x') \{R(x, a, x') + \gamma V^*(x')\} \end{aligned} \quad (3.20)$$

The optimal state-value function $V^*(x)$ can be written in terms of the optimal action-value function $Q^*(x, a)$ as follows [27]

$$\begin{aligned} V^*(x) &= \max_a Q^*(x, a) \\ &= \max_a \sum_{x'} T(x, a, x') \{R(x, a, x') + \gamma V^*(x')\} \end{aligned} \quad (3.21)$$

The action-value function $Q^*(x, a)$ given in (3.20) can also be written as

$$Q^*(x, a) = \sum_{x'} T(x, a, x') \left\{ R(x, a, x') + \gamma \max_{a'} Q^*(x', a') \right\} \quad (3.22)$$

Equation (3.21) and (3.22) are called the Bellman optimality equation for the state-value function $V(x)$ and the action-value function $Q(x, a)$ respectively [27].

3.3 Dynamic Programming

Dynamic programming (DP) refers to a class of algorithms that can be used to compute optimal policies given a perfect model of the environment [27]. DP algorithms convert the Bellman equation given in (3.21) into update rules and then recursively solve the resulting equation for the optimal state-value function $V^*(x)$. DP methods perform bootstrapping, in the sense that they update previous estimates on the basis of new estimates [27]. In the literature, the two most common DP methods are the Value Iteration and the Policy Iteration [27], [28].

Algorithm 3.1 Value Iteration Algorithm

-
- 1: Initialize $V_0(x)$ arbitrary $\forall x \in X$.
 - 2: **repeat** for each time step t
 - 3: Set $\Lambda = 0$
 - 4: **for** each $x \in X$ **do**
 - 5: $v = V_t(x)$
 - 6: $V_{t+1}(x) = \max_a \sum_{x'} T(x, a, x') \{R(x, a, x') + \gamma V_t(x')\}$
 - 7: $\Lambda = \max(\Lambda, |v - V_{t+1}(x)|)$
 - 8: **end for**
 - 9: **until** $\Lambda \leq \Omega$ (Ω is a small positive number)
-

Obtain the deterministic policy π using:

$$\pi(x) = \arg \max_a \sum_{x'} T(x, a, x') \{R(x, a, x') + \gamma V_t(x')\}$$

3.3.1 Value Iteration

The Value Iteration is a DP algorithm that recursively solves the Bellman equation given in (3.21). Value Iteration requires perfect information of the state transition probabilities and the rewards. Value Iteration converts the Bellman equation given in (3.21) into an update rule given by

$$V_{t+1}(x) = \max_a \sum_{x'} T(x, a, x') \{R(x, a, x') + \gamma V_t(x')\} \quad (3.23)$$

The initial estimates $V_0(x)$ can be set as zeros for all state $x \in X$ or to any arbitrary value. Value Iteration recursively solves Equation (3.23) for the optimal state-value function $V^*(x)$. A typical terminating condition for the Value Iteration algorithm is to test the quantity $\max_{x \in X} |V_{t+1} - V_t|$ after each iteration and stop when it is less than a specified tolerance Ω [27]. The deterministic policy $\pi(x)$ computed by Value Iteration can be obtained using

$$\pi(x) = \arg \max_a \sum_{x'} T(x, a, x') \{R(x, a, x') + \gamma V_t(x')\} \quad (3.24)$$

Figure 3.3: An example of Value Iteration (a) 4×4 grid world (b) 16 states of the grid world.

for all $x \in X$, where $\arg \max_a$ is the action a in state x that maximizes the state-value function. Value Iteration converges to the optimal policy for discounted finite MDPs [27]. The Value Iteration algorithm is summarized in Algorithm 3.1 [27].

Example 2.2: We take an example of a 4×4 grid shown in Fig. 3.3, to illustrate Value Iteration for a finite MDP. The cells of the grid represents the states of the environment. There are 16 states in total, as shown in Fig. 3.3b. State x_1 is not used in this game, hence it is grayed out in Fig. 3.3. State x_2 and x_5 are the terminal states of the game, as shown in Fig. 3.3a. At each state, the agent can take four possible actions which are: *left* ($\leftarrow$), *down* ($\downarrow$), *right* ($\rightarrow$) and *up* ($\uparrow$). In this example, state transitions are stochastic. Executing any one of the possible actions moves the agent one cell in the direction of the specified action with probability 0.6 or one cell sideways, either to the left cell or to the right cell, with equal probability 0.2. For example,

Table 3.1: The Value Iteration result.

State	Actions				Value
	Left	Down	Right	Up	
1	—	—	—	—	—
2	—	—	—	—	—
3	-1.11	8.77	10.93	5.92	10.93
4	10.32	11.19	10.46	10.03	11.19
5	—	—	—	—	—
6	19.88	20.55	10.63	2.92	20.55
7	16.21	15.55	12.33	12.00	16.21
8	13.35	13.08	11.81	11.37	13.35
9	24.32	20.42	21.54	26.68	26.68
10	21.53	18.24	16.57	19.05	21.53
11	17.50	15.31	13.62	15.22	17.50
12	14.36	13.26	12.67	12.95	14.36
13	20.46	19.09	19.00	21.75	21.75
14	19.05	17.15	16.15	18.49	19.05
15	16.39	14.79	13.63	15.39	16.39
16	13.94	12.99	12.62	13.21	13.94

executing the action ($\uparrow$) in state x_{11} will move the agent to state x_7 with probability 0.6, to state x_{10} with probability 0.2 or to state x_{12} with probability 0.2. Any action that would lead the agent off the grid will leave the agent in its current location. A reward of -10 is given if the agent reaches the terminal state x_2 . A reward of +30 is given if the agent reaches the terminal state x_5 . A reward of 0 is given for any transition between states other than the terminal states. We set $\gamma = 0.9$, in this example. Therefore, in this scenario, future rewards are discounted.

The goal of this example is to illustrate the use of Value Iteration to obtain a deterministic optimal policy that will guide the agent to reach the goal state x_5 . We

Figure 3.4: The optimal policy.

set $\Omega = 0.0001$ and run the Value Iteration algorithm. After 41 iterations, the algorithm terminates and the result is shown in Table 3.1. The optimal policy obtained is illustrated in Fig. 3.4. There are important observations in Fig. 3.4. We observe in state x_6 that the agent prefers to take action ($\downarrow$) to action ($\leftarrow$), because executing action ($\leftarrow$) in state x_6 may lead the agent to the bad state x_2 with probability 0.2. The agent not willing to risk a 20 percent chance of moving to the bad state x_2 , prefers to take action ($\downarrow$) in state x_6 which may lead to taking more steps to reach the goal state x_5 . A similar observation is made in state x_3 , as shown in Fig. 3.4. The agent prefers to take action ($\rightarrow$) to action ($\downarrow$), because executing action ($\downarrow$) in state x_3 may lead the agent to the bad state x_2 with probability 0.2. The agent not willing to risk moving to the bad state x_2 prefers to take action ($\rightarrow$) in state x_3 .

However, if we modify the dynamics of the environment such that executing any one of the possible actions moves the agent one cell in the direction of the specified action with probability 0.8 or one cell sideways, either to the left cell or the right cell, with equal probability 0.1, we obtain a new policy as shown in Fig. 3.5. The

Table 3.2: The Value Iteration result after we modified the dynamics of the environment.

State	Actions				Value
	Left	Down	Right	Up	
1	—	—	—	—	—
2	—	—	—	—	—
3	-4.62	16.01	15.19	12.01	16.01
4	14.69	16.39	14.96	14.72	16.39
5	—	—	—	—	—
6	25.25	22.98	16.79	-3.06	25.25
7	21.58	19.62	16.88	15.49	21.58
8	18.72	17.26	16.66	15.43	18.72
9	26.03	22.87	23.29	28.85	28.85
10	25.05	20.59	19.95	22.74	25.05
11	21.76	18.20	17.35	19.50	21.76
12	18.93	16.28	16.89	17.14	18.93
13	22.87	22.28	20.89	25.03	25.03
14	22.28	20.08	18.50	22.07	22.28
15	19.78	17.83	16.36	19.25	19.78
16	17.52	15.97	15.90	16.99	17.52

Value Iteration result after modifying the dynamics of the environment is shown in Table 3.2. There are important observations in Fig. 3.5. Based on the new dynamics of the environment, the agent is willing to risk the action ($\leftarrow$) in state x_6 to reach the goal state x_5 in fewer steps. The agent is willing to risk action ($\leftarrow$), because executing action ($\leftarrow$) in state x_6 now has a small 10 percent chance of leading the agent to the bad state x_2 and a high 80 percent chance of leading the agent to the goal state x_5 . A similar observation is made in state x_3 . The agent is willing to risk action ($\downarrow$) to reach the goal state in fewer steps.

Figure 3.5: The optimal policy after we modified the dynamics of the environment.

This example demonstrates how the dynamics of the environment determine the shape of the optimal policy. The more uncertain the environment, the more cautious the agent becomes, preferring to take the risk-free path. This may require the agent to take more steps to reach the goal state. However, the more predictable the environment, the more greedy the agent becomes, preferring to take the shortest path to reach the goal state.

3.4 Temporal Difference Learning

Temporal Difference (TD) learning methods are algorithmic methods that make use of the outcomes of the agent's interactions with the environment to determine the optimal policy. TD methods are different from DP methods in that they do not require the complete dynamics of an MDP in order to determine the optimal policy [27]. The simplest TD method, known as TD(0), uses the immediate outcome of the agent's

interaction with the environment to form a *target* in the form [27]

$$\text{Target} = r_{t+1} + \gamma V_{t+1}(x') \quad (3.25)$$

The difference between the *target* and the previous estimate is used to compute an error signal Δ . This error signal Δ is called the TD error and is given as

$$\Delta = r_{t+1} + \gamma V_{t+1}(x') - V_t(x) \quad (3.26)$$

The TD error Δ is used to update the previous estimate. The update rule is given as

$$V_{t+1}(x) = V_t(x) + \alpha[r_{t+1} + \gamma V_{t+1}(x') - V_t(x)] \quad (3.27)$$

where α ($0 \leq \alpha \leq 1$) is the learning rate. The learning rate should be sufficiently small and adequately decreased over time as the performance of the agent improves. Similar to DP Methods, TD methods are called bootstrapping methods because they update previous estimates on the basis of new estimates [27]. In the literature, the most common TD method is the Q-learning method.

3.4.1 Q-Learning

Q-learning was first introduced by Christopher Watkins [35]. Q-learning is a TD learning method that recursively solves the Bellman equation given in Equation (3.22). Q-learning is useful when the state transition probabilities and the rewards are uncertain. Q-learning uses each new experience of the agent to immediately form a target in the form

$$\text{Target} = r_{t+1} + \gamma \max_{a'} Q_{t+1}(x', a') \quad (3.28)$$

Algorithm 3.2 Q-Learning Algorithm

-
- 1: Initialize $Q_0(x, a)$ arbitrarily $\forall x \in X, a \in A$.
 - 2: Initialize x .
 - 3: **repeat** for each time step t
 - 4: Choose action a based on policy derived from Q (e.g. ϵ -greedy).
 - 5: Take action a , observe next state x' and reward r .
 - 6: Estimate the action-value function $Q_{t+1}(x, a)$ in (3.29)
 - 7: Set $x = x'$.
 - 8: **until** x meets a terminating condition
-

The update rule for the Q-learning method is given as

$$\begin{aligned}
 Q_{t+1}(x, a) &= Q_t(x, a) + \alpha \Delta \\
 &= Q_t(x, a) + \alpha [r_{t+1} + \gamma \max_{a'} Q_{t+1}(x', a') - Q_t(x, a)] \quad (3.29)
 \end{aligned}$$

where Δ is the TD error and α is the learning rate. It has been proven that for any finite MDP, the Q-learning algorithm converges to the optimum Q-values with probability one so long as the state-action pairs are visited infinite times and the learning rate is adequately decreased over time [36].

In the literature, the most common approach to make action selections in Q-learning is the ϵ -greedy action selection policy [9]. The ϵ -greedy policy selects a random action $a^r \in A$ with probability ϵ or the greedy action $a^g \in A$ with probability $1 - \epsilon$ given as [9]

$$a_t = \begin{cases} a^r & \text{with prob } (\epsilon) \\ a^g & \text{with prob } (1 - \epsilon) \end{cases} \quad (3.30)$$

where the greedy action a^g is the action that maximizes the action-value function $Q_t(x, a)$ for the current state x_t defined as

$$a^g = \arg \max_{a \in A} Q_t(x, a) \quad (3.31)$$

Figure 3.6: The actor-critic scheme.

The Q-learning algorithm is summarized in Algorithm 3.2 [9], [27].

3.5 Actor–Critic Methods

The actor-critic methods are TD methods that have a separate memory structure to represent the decision mechanism (actor) independent of the value function estimator (critic) [27]. The actor-critic scheme is shown in Fig. 3.6. The actor decides the action to take at any time instant, while the critic evaluates the outcome of the action to determine whether the agent performed better or worse than expected [27]. A TD error is computed based on (3.32)

$$\Delta = r_{t+1} + \gamma V_{t+1}(x') - V_t(x) \quad (3.32)$$

The TD error is used to adapt the actor and the critic. The structure of the actor-critic method provides the means to combine function approximators such as the fuzzy logic controller (FLC) with the TD learning method to solve problems with

continuous state and action spaces. Various reinforcement learning algorithms such as the fuzzy actor-critic learning (FACL) algorithm and the fuzzy Q-learning (FQL) algorithm have been derived by combining the actor-critic methods with the FLC. We provide more details of the FACL algorithm and the FQL algorithm in Chapter 5.

3.6 Summary

In this chapter, we provided a brief introduction to reinforcement learning. We reviewed the concept of discounted future rewards. We provided an example to illustrate the effect of discounting future rewards on the optimal policy. We introduced the mathematical framework of MDP. We explained the notion of the reward function, the transition function, the state-value function and the action-value function.

We presented DP methods as an approach to compute optimal policies given a perfect model of the environment. We presented Value Iteration as one of the DP methods. We provided an example to illustrate Value Iteration for a finite MDP. In addition, we introduced TD learning as an approach to compute optimal policies given we do not have complete knowledge of the state transition probabilities and the rewards. We then presented Q-learning and actor-critic methods as members of TD methods.

The concepts and terminologies presented in this chapter will be used in Chapter 5 to design our control system which with the aid of reinforcement learning will guide our learning agents to determine their optimal strategies.

Chapter 4

The Guarding a Territory Problem

The guarding a territory game was first introduced by Isaacs [8]. In the game, an invader tries to get as close as possible to the territory which is defended by a guard. The goal of the guard is to intercept the invader as far as possible from the territory. The game has practical real-world applications which include protecting borders from unlawful entries, and localizing and neutralizing environmental threats.

When the players have the same speed, the optimal solution of the guarding a territory game can be determined using the perpendicular bisector approach [8]. However, when the players do not have the same speed, the optimal solution of the game cannot be determined using this same approach. In this chapter, we introduce the Apollonius circle approach to determine the optimal solution of the guarding a territory game when the players do not have the same speed.

Furthermore, we formulate three problems that we make use of in Chapter 6 to evaluate the learning process of the players in the game of guarding a territory. In our formulation, the players can have different speeds. In other words, the guard can be faster than the invader or the invader can be faster than the guard.

This chapter is organized as follows. We provide background information on differential games in Section 4.1. In Section 4.2, we briefly review the Apollonius

circle. In Section 4.3, we present the Apollonius circle approach to determine the optimal strategies of the players when the players do not have the same speed. In Section 4.4, we formulate three problems that we make use of in Chapter 6 to evaluate the learning process of the players in the game of guarding a territory.

4.1 A Brief Introduction to Differential Games

In Isaacs' [8] differential game between two players, the system dynamics are given by

$$\dot{\bar{x}} = f(t, \bar{x}, \bar{u}_1, \bar{u}_2), \quad \bar{x}(t_0) = \bar{x}_0 \quad (4.1)$$

where $\bar{x} \in \mathbb{R}^h$ is the state vector of dimension h , t_0 is the start time of the game, $\bar{u}_1$ and $\bar{u}_2$ are the strategies played by player 1 and player 2 respectively, and $f(\cdot)$ determines the dynamics of the system. The payoff, represented as $P(\bar{u}_1, \bar{u}_2)$, is given in the form [8]

$$P(\bar{u}_1, \bar{u}_2) = \int_{t_0}^{t_f} Q(t, \bar{x}, \bar{u}_1, \bar{u}_2) dt + H(t_f, \bar{x}(t_f)) \quad (4.2)$$

where t_f is the terminal time (or the first time the states $\bar{x}(t)$ intersect a given final condition), $Q(\cdot)$ is the integral payoff, $H(\cdot)$ is the payoff at the terminal time and the functions $Q(\cdot)$ and $H(\cdot)$ are chosen in order to achieve an objective.

For a two-player zero-sum differential game, the players choose their strategies, such that, one player is driven to maximize the payoff and the other player is driven to minimize it. If we assume that player 1 with strategy $\bar{u}_1$ wants to maximize the payoff and player 2 with strategy $\bar{u}_2$ wants to minimize it, then there exist a value of

the game $\widehat{V}$ such that [56]

$$P(\bar{u}_1, \bar{u}_2^*) \leq P(\bar{u}_1^*, \bar{u}_2^*) = \widehat{V} \leq P(\bar{u}_1^*, \bar{u}_2) \quad \forall \bar{u}_1, \bar{u}_2 \quad (4.3)$$

where $\bar{u}_1^*$, $\bar{u}_2^*$ are the optimal strategies of player 1 and player 2 respectively. Therefore, the objective of the game is to find the strategies $\bar{u}_1^*$, $\bar{u}_2^*$, to achieve the value of the game $\widehat{V}$.

The guarding a territory game, is a zero-sum differential game. Adopting unicycle robots as our players in the game, the dynamics of any one of the players (the invader or the guard) is defined as [57]

$$\dot{x} = V \cos \theta \quad (4.4)$$

$$\dot{y} = V \sin \theta \quad (4.5)$$

$$\dot{\theta} = \omega \quad (4.6)$$

where V is the linear speed of the robot, ω is the angular speed of the robot, $[x, y]$ is the geometrical location of the robot and θ is the orientation of the robot with respect to the global x -axis.

In order to simplify the problem, the territory T is assumed to be a circle with center C_T and radius R_T . The payoff $P(u_G, u_I)$ of the game is the Euclidean distance between the position of the invader I and the territory T at game termination defined as

$$P(u_G, u_I) = \sqrt{(x_I(t_f) - x_{C_T})^2 + (y_I(t_f) - y_{C_T})^2} - R_T$$

where u_G and u_I are the strategies of the guard and the invader respectively. Since the payoff implies how close the invader could get to the territory at game termination, it is rational that the invader choose its strategy u_I to minimize the payoff and the

guard u_G to maximize it. Our research is focused on investigating the use of learning algorithms to train the invader and the guard in order to find their respective optimal strategies u_I^* , u_G^* .

We made certain assumptions in the game. The game environment is obstacle-free. The game is played in a bounded space, such that if any player attempts to move out of the game area, the player will remain at its current position. The players have perfect information of the geometrical location of the other players. All the players have a constant speed, but not necessary the same speed. At any time instant t , the players have perfect knowledge of the control decisions chosen by their opponents prior to t , but none regarding the future control decisions their opponents will choose.

4.2 The Apollonius Circle

Consider the Apollonius circle AC formed by the invader I and the guard G as shown in Fig. 4.1. In this scenario, the guard is faster than the invader (i.e. $V_G > V_I$). In this thesis, we refer to any player with speed greater than that of the other players as a superior player. Otherwise, it is an inferior player. The Apollonius circle will always encircle the inferior player. If we pick any point M_1 , M_2 , on the Apollonius circle as shown in Fig. 4.1, both players can reach that point at the same time, given their individual speed. Let λ be the ratio of the speeds of the players, such that

$$\lambda = \frac{V_G}{V_I} = \frac{|GM_1|}{|IM_1|} = \frac{|GM_2|}{|IM_2|} \quad (4.7)$$

where $|GM_1|$ denotes the Euclidean distance between the position of the guard G and the point M_1 . The center of the Apollonius circle C_{AC} and its radius R_{AC} can be

Figure 4.1: The Apollonius circle AC constructed for an inferior invader I against a superior guard G .

computed using

$$C_{AC} = \left(\frac{x_G - \lambda^2 x_I}{1 - \lambda^2}, \frac{y_G - \lambda^2 y_I}{1 - \lambda^2} \right) \quad (4.8)$$

$$R_{AC} = \frac{\lambda \sqrt{(x_G - x_I)^2 + (y_G - y_I)^2}}{1 - \lambda^2} \quad (4.9)$$

respectively [18], [19].

4.3 The Optimal Strategies of the Players

As previously mentioned, when the players do not have the same speed, the Apollonius circle approach can be applied to determine the optimal strategies of the players. In

Figure 4.2: The optimal strategy of an invader I against a guard G with the same speed.

this section, we briefly review the perpendicular bisector approach and subsequently present the Apollonius circle approach.

4.3.1 Two-Player Game: An Invader Against a Guard With the Same Speed

Consider the scenario presented in Fig. 4.2. In this scenario, there are two players in the game: an invader and a guard. The invader is located at position I and the guard at position G . The invader and the guard have the same speed. According to Isaacs [8], the optimal strategies of the players can be determined by constructing the perpendicular bisector line (PBL) between the players, as shown in Fig. 4.3. The region below the line PBL is the region the invader can reach before the guard. Likewise, the region above the line PBL is the region the guard can reach before the

Figure 4.3: The optimal strategy of an invader I against a guard G with the same speed. The perpendicular bisector line (PBL) is constructed.

invader.

The point O_1 is the closest point to the territory located on the line PBL , as shown in Fig. 4.3. Assuming both players play rationally, the point O_1 is the closest the invader can get to the territory as well as the farthest from the territory the guard can intercept the invader. Therefore, the optimal strategy of both players in this scenario is to move to the point O_1 . We refer to the point O_1 in this context as the “optimum point”. The optimum point O_1 must be recomputed at each time instant. The value of the game $\hat{V}$ is therefore given as

$$\hat{V} = |O_1 C_T| - R_T \quad (4.10)$$

The invader and the guard must find their respective strategies $u_I = u_I^*$, $u_G = u_G^*$ to achieve the value of the game $\hat{V}$. Should one of the players fail to find its optimal

Figure 4.4: The optimal strategy of an inferior invader I against a superior guard G .

strategy, its opponent should be able to achieve a more favorable payoff than the value of the game assuming the opponent plays rationally.

4.3.2 Two-Player Game: An Inferior Invader Against a Superior Guard

Consider the scenario presented in Fig. 4.4. In this scenario, there are two players in the game: an invader and a guard. The invader is located at position I and the guard at position G . The guard is superior to the invader. The Apollonius circle formed by the invader and the guard is shown in Fig. 4.5. The Apollonius circle will encircle the invader, since the invader is the inferior player. The region inside the Apollonius circle is the region the invader can reach before the guard. In this scenario, the optimal strategies of the players can be determined by constructing a line $\overleftrightarrow{C_{AC}C_T}$ that passes

Figure 4.5: The optimal strategy of an inferior invader I against a superior guard G . The Apollonius circle AC is constructed.

through the center of the Apollonius circle C_{AC} and the center of the territory C_T , as shown in Fig. 4.5. If we solve for the intersection points between the line $\overleftrightarrow{C_{AC}C_T}$ and the Apollonius circle AC , we obtain two points O_1 and O_2 , as shown in Fig. 4.5.

The optimal strategy of both players in this scenario is to move to the point O_2 . The point O_2 is the closest the invader can get to the territory as well as the farthest from the territory the guard can intercept the invader. We refer to the point O_2 in this context as the optimum point. The optimum point O_2 must be recomputed at each time instant. The value of the game $\hat{V}$ is therefore given as

$$\hat{V} = |O_2C_T| - R_T \quad (4.11)$$

The invader and the guard must find their respective strategies $u_I = u_I^*$, $u_G = u_G^*$ to achieve the value of the game $\hat{V}$. Should one of the players fail to find its optimal

Figure 4.6: The optimal strategy of a superior invader I against two inferior guards G_1 and G_2 .

strategy, its opponent should be able to achieve a more favorable payoff than the value of the game assuming the opponent plays rationally.

4.3.3 Three-Player Game: A Superior Invader Against Two Inferior Guards

Consider the scenario presented in Fig. 4.6. In this scenario, there are two guards in the game against a single invader. The invader is located at position I , the first guard (i.e. guard 1) is located at position G_1 and the second guard (i.e. guard 2) is located at position G_2 . The invader is superior to the two guards. The Apollonius circle formed by the invader and each guard is shown in Fig. 4.7. As shown in Fig. 4.7, AC_1 denotes the Apollonius circle formed between the invader and guard 1, while AC_2 denotes the Apollonius circle formed between the invader and guard 2.

Figure 4.7: The strategy of a superior invader I against two inferior guards G_1 and G_2 . The Apollonius circles AC_1 and AC_2 are constructed.

In this scenario, the optimal strategies of the players can be determined by constructing a line $\overleftrightarrow{C_{AC_1}C_T}$ that passes through the center of the Apollonius circle C_{AC_1} and the center of the territory C_T , as shown in Fig. 4.7. Similarly, we construct a second line $\overleftrightarrow{C_{AC_2}C_T}$ that passes through the center of the second Apollonius circle C_{AC_2} and the center of the territory C_T , as shown in Fig. 4.7.

If we solve for the intersection points between the line $\overleftrightarrow{C_{AC_1}C_T}$ and the Apollonius circle AC_1 , we obtain two points O_1 and O_2 , as shown in Fig. 4.7. Similarly, if we solve for the intersection points between the line $\overleftrightarrow{C_{AC_2}C_T}$ and the Apollonius circle AC_2 we obtain two other points O_3 and O_4 , as shown in Fig. 4.7. Each of these points O_1 , O_2 , O_3 and O_4 will be considered as the possible optimum point. In addition, we also consider the intersection points of the Apollonius circles as the possible optimum

Figure 4.8: The optimal strategy of a superior invader I against two inferior guards G_1 and G_2 when the environment is bounded.

point. The Apollonius circles intersect at points O_5 and O_6 , as shown in Fig. 4.7.

We observe in Fig. 4.7 that the closest point to the territory located on the Apollonius circle which is possible for the invader to reach is the point O_6 . However, we cannot yet conclude that the point O_6 is the optimal point. This is because as the players move towards the point O_6 , the radius of the Apollonius circle diminishes since the radius is a function of the distance between the invader and the guard, and of the ratio of their speeds (i.e. λ). Therefore, in some scenarios it may be possible for the invader to go around the Apollonius circle to achieve a more favorable payoff, taking into consideration that there are only two guards in the game.

In order to simplify the problem, we assume that the environment is bounded such that it is impossible for the invader to go around any of the Apollonius circles. This setup is shown in Fig. 4.8. Therefore, for this scenario, we define the optimal point as the point O_6 where both Apollonius circles intersect. The optimal strategy

of all three players in this scenario is to move to the point O_6 . The point O_6 is the closest the invader can get to the territory as well as the farthest from the territory the guards can intercept the invader. The optimum point O_6 must be recomputed at each time instant. The value of the game is therefore given as

$$\widehat{V} = |O_6 C_T| - R_T \quad (4.12)$$

The invader, guard 1 and guard 2 must find their respective strategies $u_I = u_I^*$, $u_{G_1} = u_{G_1}^*$ and $u_{G_2} = u_{G_2}^*$, to achieve the value of the game $\widehat{V}$. Should any one of the guards fail to find its optimal strategy, this may lead to adjacent Apollonius circles no longer intersecting each other (i.e. an opening between the adjacent Apollonius circles). A rational invader should be able to exploit this opening between adjacent Apollonius circles to achieve a more favorable payoff than the value of the game. Likewise, should the invader fail to find its optimal strategy, the guards should be able to achieve a more favorable payoff than the value of the game assuming they play rationally.

4.4 Problem Formulation

In this section, we formulate three problems that we make use of in Chapter 6 to evaluate the learning process of the players in the game of guarding a territory. In our formulation, all the players are simultaneously learning.

Figure 4.9: Two-player game setup: An invader against a guard with the same speed.

4.4.1 Two-Player Game: An Invader Against a Guard With the Same Speed

In this scenario, there are two players in the game (an invader and a guard) as shown in Fig. 4.9. Both players have the same speed. The game area was chosen as a square area of size 30×30 units, as shown in Fig. 4.9. The goal of this problem is to investigate the possibility of both players simultaneously learning given the players have the same speed. Since both players have the same speed, we make use of the optimal solution given by the perpendicular bisector approach to evaluate the performance of the players. kindly note that the dimensions of the robots shown in Fig. 4.9 does not represent the size of the robot nor the type of robot used in our simulation. The robots shown in Fig. 4.9 are used only for illustrative purposes.

Figure 4.10: Two-player game setup: An inferior invader against a superior guard.

4.4.2 Two-Player Game: An Inferior Invader Against a Superior Guard

In this scenario, there are two players in the game (an invader and a guard) as shown in Fig. 4.10. We set the guard to be 30 percent faster than the invader. As such, the guard is superior to the invader. Since the guard is superior to the invader, we would expect a rational guard to capture the invader farther from the territory, than it is able to when it has the same speed as the invader. The goal of this problem is to investigate if the guard can learn that it is faster than the invader by playing the game. As such, move to capture the invader at a farther distance from the territory. We would also expect the invader to try to reach the territory if it is possible to do so or, if this is not possible, to get as close as possible to the territory before capture occurs. Since both players do not have the same speed in this scenario, we make

Figure 4.11: Three-player game setup: A superior invader against two inferior guards.

use of the optimal solution given by the Apollonius circle approach to evaluate the performance of the players.

4.4.3 Three-Player Game: A Superior Invader Against Two Inferior Guards

In this scenario, there are two guards in the game against a single invader, as shown in Fig. 4.11. We increase the complexity of the guarding a territory problem by setting the invader to be 30 percent faster than the two guards. As such, the invader is superior to the two guards. The goal of this problem is to investigate the possibility of the two guards working jointly to intercept the superior invader before it reaches the territory. We also expect the invader to learn its optimal behavior to avoid capture

and reach the territory if it can or, if this is not possible, to get as close as possible to the territory before capture occurs. Since the players do not have the same speed, we make use of the optimal solution given by the Apollonius circle approach to evaluate the performance of the players.

4.5 Summary

In this chapter, we provided a brief introduction to the differential game of guarding a territory. We introduced the Apollonius circle and subsequently presented the Apollonius circle approach to determine the optimal solution of the game of guarding a territory when the players do not have the same speed. We formulated three problems that we make use of in Chapter 6 to evaluate the learning process of the players in the game of guarding a territory.

Chapter 5

Fuzzy Reinforcement Learning

An agent can learn its desired behavior in a dynamic environment using reinforcement learning techniques. The objective of the agent in this scenario, is to learn a policy that maximizes the value of every state. The policy can be in the form of a look-up table, if the state and action spaces are discrete and small in number. However, if the state and action spaces are large or in the continuous domain, it is intractable to maintain a look-up table. Function approximators such as the fuzzy logic controller (FLC) can be used to represent the continuous state and action spaces in an effective way [9], [10]. A class of reinforcement learning methods called the actor-critic methods can be combined with the FLC to solve problems in the continuous domain [9], [10]. When reinforcement learning is combined with FLC, the approach is commonly called *fuzzy reinforcement learning* [12], [13].

In this thesis, we apply two fuzzy reinforcement learning algorithms to the game of guarding a territory. The two fuzzy reinforcement learning algorithms are called the fuzzy actor-critic learning (FACL) algorithm [10] and the fuzzy Q-learning (FQL) algorithm [58]. Both algorithms are members of the actor-critic reinforcement learning methods. The two algorithms make use of the FLC to implement both the actor and the critic. The FQL and FACL algorithms have been previously applied to the

guarding a territory game in [16], [17]. However, in [16] only the guard was learning and in [17] only the invader was learning. In this thesis, we investigate the possibility of the all the players simultaneously learning in the game.

This chapter is organized as follow. In Section 5.1, we provide a brief introduction to fuzzy logic. Section 5.2 provides details of the design of the controller for the guarding a territory problem. We present the FACL algorithm in Section 5.3 and the FQL algorithm in Section 5.4. We compare the FQL algorithm and the FACL algorithm in Section 5.5. In Section 5.6, we provide specifics for the design of the reward functions for the learning process of the players.

5.1 A Brief Introduction to Fuzzy Logic

Fuzzy logic introduced by Zadeh [59] provides a procedure for representing and manipulating human heuristic knowledge to model real-world problems. Fuzzy logic extends conventional Boolean logic (true or false) to allow for partial truth in modeling problems which make use of imprecise information. Applications of fuzzy logic can be found in anti-lock braking system [60], [61], elevator control [62], [63], automatic transmission systems [64], process temperature control [65] and other applications.

Fuzzy logic relies on fuzzy sets and fuzzy rules for reasoning and decision making. A fuzzy set is a set that permits its member elements to have a partial degree of membership. This is in contrast to a classical set (or a crisp set), where each element either “belongs” or “does not belong” to the set [66]. Fuzzy sets are defined on a universe of discourse. The universe of discourse (or simply universe) is the collection of all possible set of elements x considered as input (or output) to the fuzzy system. If we denote the universe by X , the fuzzy set A in X is a collection of ordered pairs

given as [67]

$$A = [(x, \mu_A(x)) \mid x \in X] \quad (5.1)$$

where $\mu_A(x) : X \rightarrow [0, 1]$ is the membership function (MF). The function $\mu(x)$ provides the membership grade of x in X . In the literature, the most frequently used membership functions are the singleton MF, the triangular MF, the trapezoidal MF and the Gaussian MF [66].

Fuzzy rules are a collection of linguistic statements (IF-THEN statements) that relates the inputs to the outputs for a fuzzy system [67]. The rules take the form

$$\text{IF } x \text{ is } A \text{ THEN } y \text{ is } B$$

where A and B are fuzzy sets defined on the universe X and Y respectively. In the literature, “ x is A ” is commonly referred to as the antecedent or premise, while “ y is B ” is commonly referred to as the consequence or conclusion [67]. Some examples of fuzzy IF-THEN rules include

- If the lights go out, sound the alarm.
- If the temperature is warm, switch on the air conditioner.
- If the fruit is ripe and looks delicious, place it in the shopping cart.

5.1.1 Fuzzy Set Operations

The basic operations on fuzzy sets are the complement operation, the intersection operation and the union operation. The complement or negation $\bar{A}$ of a fuzzy set A

in X , is defined as

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad (5.2)$$

The generalized intersection and union operations on two fuzzy sets A and B are binary operations that aggregate the membership function of the two sets A and B into a single membership function C [66]. This mapping is in the form

$$f : [0, 1] \times [0, 1] \rightarrow [0, 1] \quad (5.3)$$

The intersection of two fuzzy sets is generally carried out using a binary function called the triangular norm (T -norm). The T -norm operator applied to two fuzzy sets A and B , results in a fuzzy set C given as

$$\mu_C(x) = \mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)) \quad (5.4)$$

The two most frequently used T -norm operators are the minimum and the algebraic product defined as follows:

$$\text{Minimum:} \quad T(a, b) = \min(a, b) \quad (5.5)$$

$$\text{Algebraic product:} \quad T(a, b) = a \cdot b \quad (5.6)$$

Similarly, the union of two fuzzy sets is generally carried out using a binary function called the triangular conorm (S -norm). The S -norm operator applied to two fuzzy sets A and B , results in a fuzzy set C given as

$$\mu_C(x) = \mu_{A \cup B}(x) = S(\mu_A(x), \mu_B(x)) \quad (5.7)$$

The two most frequently used *S-norm* operators are the maximum and the algebraic sum defined as follows:

$$\text{Maximum:} \quad S(a, b) = \max(a, b) \quad (5.8)$$

$$\text{Algebraic sum:} \quad S(a, b) = a + b - ab \quad (5.9)$$

5.1.2 Fuzzy Reasoning

Fuzzy reasoning, also known as approximate reasoning, is an inference procedure that relies on a set of fuzzy IF-THEN rules for decision making [67]. Consider the generalized *modus ponens*

$$\text{premise 1 (rule):} \quad \text{IF } x \text{ is } A \text{ THEN } y \text{ is } B, \quad (5.10)$$

$$\text{premise 2 (fact):} \quad \text{IF } x \text{ is } A', \quad (5.11)$$

$$\text{conclusion:} \quad y \text{ is } B'. \quad (5.12)$$

where $\{A, A'\}$ and $\{B, B'\}$ are fuzzy sets defined on the universe X and Y respectively. The relationship given in (5.10) is called a fuzzy implication $R : A \rightarrow B$ and it is defined in the Cartesian product space $X \times Y$ [66]. The fuzzy implication can be determined using

$$\mu_R(x, y) = \mu_{A \rightarrow B}(x, y) = T(\mu_A(x), \mu_B(y)) \quad (5.13)$$

The output fuzzy set B' in (5.12) can be determined using the *maximum-T-norm* (*max-T*) compositional rule of inference given as

$$\mu_{B'}(y) = \max_{x \in X} T[\mu_{A'}(x), \mu_R(x, y)] \quad (5.14)$$

In the literature, the most frequently used fuzzy inference engine is the product inference engine. The product inference engine makes use of the individual rule based inference method, the *algebraic product* given in (5.6) for all the *T-norm* operators and the *maximum* given in (5.8) for all the *S-norm* operators [11]. In addition, the product inference engine aggregates multiple rules using the union combination. Applying the product inference engine, we can rewrite Equation (5.14) as follows

$$\mu_{B'}(y) = \max_{x \in X} [\mu_{A'}(x) \cdot \mu_A(x) \cdot \mu_B(y)] \quad (5.15)$$

In the case of fuzzy IF-THEN statements with L fuzzy rules and m antecedents given as

$$\begin{array}{lll} \text{rule 1} & : & \text{IF } x_1 \text{ is } A_{11} \text{ and } x_2 \text{ is } A_{12} \cdots \text{ and } x_m \text{ is } A_{1m} \text{ THEN } y \text{ is } B_1 \\ & & \vdots \\ & & \vdots \\ \text{rule } L & : & \text{IF } x_1 \text{ is } A_{L1} \text{ and } x_2 \text{ is } A_{L2} \cdots \text{ and } x_m \text{ is } A_{Lm} \text{ THEN } y \text{ is } B_L \end{array}$$

for any given input $(A'_1, A'_2, \dots, A'_m)$, the output fuzzy set B' can be obtained using the product inference engine as follows

$$\mu_{B'}(y) = \max_{j=1, \dots, L} \left\{ \max_{x \in X} \left[\prod_{k=1}^m \mu_{A'_k}(x_k) \cdot \mu_{A_{jk}}(x_k) \cdot \mu_{B_j}(y) \right] \right\} \quad (5.16)$$

We provide an example to illustrate how one can determine the output fuzzy set using the product inference engine. Given two fuzzy IF-THEN rules with two

antecedents for each rule such that

$$\text{premise 1 (rule 1):} \quad \text{IF } x_1 \text{ is } A_{11} \text{ and } x_2 \text{ is } A_{12} \text{ THEN } y \text{ is } B_1, \quad (5.17)$$

$$\text{premise 2 (rule 2):} \quad \text{IF } x_1 \text{ is } A_{21} \text{ and } x_2 \text{ is } A_{22} \text{ THEN } y \text{ is } B_2, \quad (5.18)$$

$$\text{premise 3 (fact):} \quad \text{IF } x_1 \text{ is } A'_1 \text{ and } x_2 \text{ is } A'_2, \quad (5.19)$$

$$\text{conclusion:} \quad y \text{ is } B'. \quad (5.20)$$

the output fuzzy set $\mu_{B'_1}(y)$ for rule 1 is given as

$$\mu_{B'_1}(y) = \max_{x \in X} T[\mu_{A'_1}(x_1), \mu_{A'_2}(x_2), \mu_{B_1}(x, y)] \quad (5.21)$$

$$= \max_{x \in X} [\mu_{A'_1}(x_1) \cdot \mu_{A'_2}(x_2) \cdot \mu_{A_{11}}(x_1) \cdot \mu_{A_{12}}(x_2) \cdot \mu_{B_1}(y)] \quad (5.22)$$

$$= \max_{x \in X} \left[\prod_{k=1}^2 \mu_{A'_k}(x_k) \cdot \mu_{A_{1k}}(x_k) \cdot \mu_{B_1}(y) \right] \quad (5.23)$$

where the T -norm operator used is the algebraic product. Similarly, the output fuzzy set $\mu_{B'_2}(y)$ for rule 2 is given as

$$\mu_{B'_2}(y) = \max_{x \in X} \left[\prod_{k=1}^2 \mu_{A'_k}(x_k) \cdot \mu_{A_{2k}}(x_k) \cdot \mu_{B_2}(y) \right] \quad (5.24)$$

The product inference engine uses the union combination (i.e. S -norm operator) to aggregate multiple rules. Therefore, the overall output fuzzy set $\mu_{B'}(y)$ is given as

$$\mu_{B'}(y) = S \{ \mu_{B'_1}(y), \mu_{B'_2}(y) \} \quad (5.25)$$

$$= \max_{j=1,2} \left\{ \max_{x \in X} \left[\prod_{k=1}^2 \mu_{A'_k}(x_k) \cdot \mu_{A_{jk}}(x_k) \cdot \mu_{B_j}(y) \right] \right\} \quad (5.26)$$

Figure 5.1: The fuzzy logic controller (FLC) scheme.

5.1.3 Fuzzy Logic Controller

The two most commonly used fuzzy models in the literature are the Mamdani fuzzy model [68] and the Sugeno fuzzy model [69], [70]. The Sugeno fuzzy model offers a less computationally complex method to determine the output (crisp valued signal) from the fuzzy logic controller (FLC). In this thesis, we implemented the controller of the mobile robots using the Sugeno fuzzy model. We provide details of the design of the controller in Section 5.2. However, in this section, we briefly introduce the Sugeno fuzzy model.

The FLC scheme is shown in Fig. 5.1. It consist of four main components. These are: the fuzzy rule base, the fuzzifier, the fuzzy inference engine and the defuzzifier. The fuzzy rule base is the collection of all fuzzy conditional statements (IF-THEN statements) that relate the inputs to the outputs. The fuzzy rules in the Sugeno fuzzy model take the form [67]

$$\text{IF } x_1 \text{ is } A_1 \text{ and } x_2 \text{ is } A_2 \text{ THEN } y = f(x_1, x_2) \quad (5.27)$$

where A_1 in X_1 and A_2 in X_2 are fuzzy sets in the antecedent, while $y = f(x_1, x_2)$ is a crisp function in the consequent. If the function f is a constant, the fuzzy model is referred to as a *zero-order Sugeno fuzzy model* [67]. Whereas, if the function $f(x_1, x_2)$ is a first-order polynomial, the fuzzy model is referred to as a *first-order Sugeno fuzzy model* [67].

The fuzzifier converts the inputs to the controller to fuzzy sets. In the literature, the most common method to fuzzify the inputs is the singleton method [66]. The singleton method maps a real-valued input $x^* \in \mathbb{R}$ into a fuzzy singleton A' in X , with a membership function defined by [11]

$$\mu_{A'}(x) = \begin{cases} 1 & \text{if } x = x^* \\ 0 & \text{otherwise} \end{cases} \quad (5.28)$$

The fuzzy inference engine computes the degree of fulfillment of each rule in the fuzzy rule base. The most common fuzzy inference engine used in the Sugeno fuzzy model is the product inference engine.

The defuzzifier computes the overall crisp output of the controller using the weighted average method. Given a zero-order Sugeno fuzzy model with L rules and m input variables, the crisp output y can be obtained using [9], [11]

$$y = \frac{\sum_{j=1}^L \left(\prod_{k=1}^m \mu_{A'_{jk}}(x_k) \right) c_j}{\sum_{j=1}^L \left(\prod_{k=1}^m \mu_{A'_{jk}}(x_k) \right)} \quad (5.29)$$

where c_j is the constant consequent for rule j .

5.2 Fuzzy Controller Design for the Guarding a Territory Problem

In this section, we present details of the design of the controller for the guarding a territory game. We simplify the design of the controller by making a temporary assumption that only two-players (an invader and a guard) are present in the game. In Section 6.3, details of the controller design for the three-player game (two guards and one superior invader) will be provided.

In this thesis, we make use of the fuzzy logic controller (FLC) for three main purposes:

1. To represent the continuous state space.
2. To generate the continuous control action.
3. To predict the value of every state.

With the aid of reinforcement learning, we tune only the output parameters of the FLC. However, in other works [44], [45], the input parameters of the FLC has also been adapted. We propose to tune only the consequent parameters of the FLC in order to simplify the computational complexity of the system.

The design of the FLC essentially involves:

1. Choosing the controller model.
2. Choosing the method for fuzzification and defuzzification, and the type of fuzzy inference engine.
3. Choosing the inputs to the controller.
4. Designing the fuzzy sets and the fuzzy rules.

Figure 5.2: The inputs to the controller of the invader.

In our design, the FLC is implemented as a zero-order Sugeno fuzzy system. We make use of the singleton method for fuzzification, the product inference engine for the fuzzy inference engine and the weighted average method for defuzzification. The choice of the model of the FLC, and the methods for fuzzification and defuzzification are based on a priori knowledge of the problem.

We now define the inputs to the controller. We propose making use of state variables that describe the state of the environment to define the inputs to the controller. Three state variables were chosen to define the inputs to the controller of the invader. The first variable θ_{IT} is the angle between the global x -axis and the invader's line of sight towards the territory, as shown in Fig. 5.2. Such that $\theta_{IT} \in [-\pi, \pi]$. The variable θ_{IT} describe the position of the territory with respect to the invader. The second variable d_{IG}^x and the third variable d_{IG}^y are the components of the Manhattan distance between the invader and the guard, as shown in Fig. 5.2. Such that d_{IG}^x and $d_{IG}^y \in [-30, 30]$. The variables d_{IG}^x and d_{IG}^y describe the position of the guard with

respect to the invader. Therefore, the vector of state variables $\bar{x}_I$ that define the inputs to the controller of the invader is given as

$$\bar{x}_I = [\theta_{IT}, d_{IG}^x, d_{IG}^y] \quad (5.30)$$

where

$$d_{IG}^x = x_I - x_G \quad (5.31)$$

$$d_{IG}^y = y_I - y_G \quad (5.32)$$

These are the same inputs used in [17] for the learning invader. However, in [17] only the invader was learning.

Similarly, three state variables were chosen to define the inputs to the controller of the guard. The first variable θ_{GT} is the angle between the global x -axis and the guard's line of sight towards the territory. Such that $\theta_{GT} \in [-\pi, \pi]$. The second variable d_{GI}^x and the third variable d_{GI}^y are the components of the Manhattan distance between the guard and the invader. Such that d_{GI}^x and $d_{GI}^y \in [-30, 30]$. Therefore, the vector of state variables $\bar{x}_G$ that define the inputs to the controller of the guard is given as

$$\bar{x}_G = [\theta_{GT}, d_{GI}^x, d_{GI}^y] \quad (5.33)$$

We make use of triangular membership functions to define all the membership functions (MF) in the various input domains. We define 10 fuzzy sets for the angle variable θ , as shown in Fig. 5.3a. The variable θ describe the inputs θ_{IT} and θ_{GT} . In Fig. 5.3, N denotes *negative*, P denotes *positive*, L denotes *large*, M denotes *medium*, S denotes *small* and ZE denotes *zero*. For example, the fuzzy set *negative small*

(a)

(b)

Figure 5.3: The membership functions for (a) the angle variable θ (b) the Manhattan distance variable d .

small (NSS) in Fig. 5.3a, is given by the membership function

$$\mu_{NSS}(\theta) = \text{triangle} \left(\theta; -\frac{\pi}{3}, -\frac{\pi}{9}, \frac{\pi}{9} \right) \quad (5.34)$$

where $-\pi/3 < -\pi/9 < \pi/9$ define the locations of the three corners of the triangular MF. Similarly, we define 7 fuzzy sets for the Manhattan distance variable d , as shown in Fig. 5.3b. The variable d describe the inputs d_{IG}^x , d_{IG}^y , d_{GI}^x and d_{GI}^y .

The fuzzy sets used in our implementation are similar to those used in [17]. However, we defined an additional fuzzy set for the angle variable θ and two additional fuzzy sets for the Manhattan distance variable d , as opposed to those used in [17]. These additional fuzzy sets improved the performance of the learning agents towards the boundary of the game area.

Since the FLC is implemented as a zero-order Sugeno fuzzy system, any rule l in the fuzzy rule base will take the form

$$\text{IF } \theta \text{ is PSS, } d^x \text{ is NM, and } d^y \text{ is NL, THEN } \phi_t^l(\bar{x}_t) = c^l \quad (5.35)$$

where $\phi_t^l(\bar{x}_t)$ is the output of rule l at time step t and c^l is the constant consequent parameter for rule l . All possible combinations of the fuzzy sets defined in the various input domains will result in a total of $10 \times 7 \times 7 = 490$ fuzzy rules.

The degree of fulfillment (or firing strength) $\psi_t^l(\bar{x}_t)$ of any rule l will take the form

$$\psi_t^l(\bar{x}_t) = T[\mu_{PSS}(\theta), \mu_{NM}(d^x), \mu_{NL}(d^y)]$$

where $T[.]$ implies the T-norm operator and $\mu_{PSS}(\theta)$ provides the membership grade of the input variable θ . We make use of the singleton method for fuzzification, the product inference engine for the fuzzy inference engine, and the weighted average method for defuzzification. Therefore, the overall crisp output $\phi_t(\bar{x}_t)$ of the controller

is given as

$$\begin{aligned}\phi_t(\bar{x}_t) &= \frac{\sum_{l=1}^L \psi_t^l c^l}{\sum_{l=1}^L \psi_t^l} \\ &= \sum_{l=1}^L \varphi_t^l c^l\end{aligned}\tag{5.36}$$

where L is the total number of fuzzy rules and φ_t^l is the normalized degree of fulfillment for rule l defined as

$$\varphi_t^l = \frac{\psi_t^l}{\sum_{l=1}^L \psi_t^l}\tag{5.37}$$

5.3 Fuzzy Actor-Critic Learning

The fuzzy actor-critic learning (FACL) scheme for the guarding a territory problem is shown in Fig. 5.4. The dotted circle enclosing the critic blocks in Fig. 5.4, signifies that we are referring to the same critic. We abstracted the robot and its environment to the system block, as shown in Fig. 5.4.

The role of the actor is to generate the continuous control action $U_t(\bar{x}_t)$. In our implementation, $U_t(\bar{x}_t)$ translates to the desired heading angle of the robot. To generate the correct control actions, the actor maintains a vector W which has as many components as the number of fuzzy rules. The components of the vector W are used as the consequent parameters in the defuzzification procedure for the actor in (5.36). Therefore, the output of the actor is given as

$$U_t(\bar{x}_t) = \sum_{l=1}^L \varphi_t^l W_t^l\tag{5.38}$$

Figure 5.4: The fuzzy actor-critic learning (FACL) scheme for the guarding a territory problem.

A key issue of reinforcement learning is balancing exploration and exploitation. To cause sufficient exploration of the action space in the FACL algorithm, a Gaussian white noise parameter N_t with mean zero and variance σ_t^2 is added to the continuous control action $U_t(\bar{x}_t)$ to produce the final control action $U'_t(\bar{x}_t)$, as shown in Fig. 5.4. The adjusted control action $U'_t(\bar{x}_t)$ is used to run the game.

The role of the critic is to predict the sum of future discounted reward (or the state-value function) $V_t(\bar{x}_t)$. The critic is able to make a good estimate of the state-value function by maintaining a vector ζ which has as many components as the number of fuzzy rules. The components of the vector ζ are used as the consequent parameters in the defuzzification procedure for the critic in (5.36). Therefore, the output of the critic is given as

$$V_t(\bar{x}_t) = \sum_{l=1}^L \varphi_t^l \zeta_t^l \quad (5.39)$$

The external reinforcement signal r_{t+1} received from the environment after executing action $U_t(\bar{x}_t)$ in state x_t and transitioning to state x_{t+1} is used to compute a

temporal difference (TD) error Δ_{t+1} given by

$$\Delta_{t+1} = r_{t+1} + \gamma V_{t+1}(\bar{x}_{t+1}) - V_t(\bar{x}_t) \quad (5.40)$$

where $\gamma \in [0, 1]$ is the discounting factor. We provide details of the external reinforcement signal r_{t+1} in Section 5.6.

The TD error is used to adapt the actor and the critic. To prevent large adaption steps of the actor in the wrong direction, a more stable way to adapt the actor is to use only the sign of the TD error Δ_{t+1} and the exploration N_t [46]. The adaptation law for the actor is therefore given as

$$W_{t+1}^l = W_t^l + \left\{ \beta \operatorname{sign}[N_t \Delta_{t+1}] \frac{\partial U_t(\bar{x}_t)}{\partial W_t^l} \right\} \quad (5.41)$$

where $\beta \in [0, 1]$ is the learning rate of the actor. The adaptation law for the critic is given as

$$\zeta_{t+1}^l = \zeta_t^l + \left\{ \alpha \Delta_{t+1} \frac{\partial V_t(\bar{x}_t)}{\partial \zeta_t^l} \right\} \quad (5.42)$$

where $\alpha \in [0, 1]$ is the learning rate of the critic. The partial derivate in (5.41) and (5.42) is easily determined from (5.38) and (5.39) respectively to be

$$\frac{\partial U_t(\bar{x}_t)}{\partial W_t^l} = \frac{\partial V_t(\bar{x}_t)}{\partial \zeta_t^l} = \varphi_t^l \quad (5.43)$$

The FACL algorithm is summarized in Algorithm 5.1 [9]. In Chapter 6, we provide details of how we set the values of the learning factors in the FACL algorithm.

Algorithm 5.1 FACL algorithm

-
- 1: Initialize $V(.) = 0$, $W^l = 0$, $\zeta^l = 0$ for $l = 1, \dots, L$.
 - 2: **for** each time step t **do**
 - 3: Obtain the inputs $\bar{x}_t$.
 - 4: Compute the control action $U_t(\bar{x}_t)$ in (5.38).
 - 5: Estimate the state-value function $V_t(\bar{x}_t)$ in (5.39).
 - 6: Use $U_t(\bar{x}_t)$ to run the game for the current time step t .
 - 7: Obtain the new inputs $\bar{x}_{t+1}$ and the reward r_{t+1} at time $t + 1$.
 - 8: Estimate the state-value function $V_t(\bar{x}_{t+1})$ in (5.39).
 - 9: Compute the TD error Δ_{t+1} in (5.40).
 - 10: Adapt W_{t+1}^l in (5.41) and ζ_{t+1}^l in (5.42) for $l = 1, \dots, L$.
 - 11: **end for**
-

5.4 Fuzzy Q-Learning

The FQL scheme for the guarding a territory problem is shown in Fig. 5.5. The dotted circle enclosing the critic blocks in Fig. 5.5, signifies that we are referring to the same critic. We abstracted the robot and its environment to the system block, as shown in Fig. 5.5.

The FQL algorithm maintains a Q-table, similar to the Q-learning algorithm [35]. In the FQL algorithm, the action space A is a discrete set of actions. In our implementation, eight discrete actions were chosen to span the possible turning angles of the robot. We found the action set given in [9] suitable for our application. The action set A is defined as

$$A = [\pi, \frac{-3\pi}{4}, \frac{-\pi}{2}, \frac{-\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}] \quad (5.44)$$

The role of the actor is to generate the continuous control action $\tilde{U}_t(\bar{x}_t)$. In our implementation, $\tilde{U}_t(\bar{x}_t)$ translates to the desired heading angle of the robot. Given that the action space in the FQL algorithm is a discrete set of actions, the most

Figure 5.5: The fuzzy Q-learning (FQL) scheme for the guarding a territory problem.

common approach in the literature to cause sufficient exploration of the action space is to use the ϵ -greedy policy [9]. The ϵ -greedy policy selects a random action $a^r \in A$ with probability ϵ or the greedy action $a^g \in A$ with probability $1 - \epsilon$ given as [9]

$$a^l = \begin{cases} a^r & \text{with prob}(\epsilon) \\ a^g & \text{with prob}(1 - \epsilon) \end{cases} \quad (5.45)$$

where the greedy action a^g is the action with the maximum Q-value (or action-value) for the associated rule l in the Q-table. Let $q(l, a)$ denote the Q-value for rule l and action $a \in A$, the greedy action a^g is defined as

$$a^g = \arg \max_{a \in A} q(l, a) \quad (5.46)$$

In this thesis, we refer to the actor in the FQL algorithm as the ϵ -greedy actor. The ϵ -greedy actor uses the actions selected by the ϵ -greedy policy as the consequent parameter in the defuzzification procedure in (5.36). Therefore, the output of the

ϵ -greedy actor is given as

$$\tilde{U}_t(\bar{x}_t) = \sum_{l=1}^L \varphi_t^l a_t^l \quad (5.47)$$

The control action $\tilde{U}_t(\bar{x}_t)$ is used to run the game.

The role of the critic in the FQL algorithm is to estimate the action-value function $Q_t(\bar{x}_t)$. In this thesis, we refer to the critic in the FQL algorithm as the *q-critic*. The *q-critic* uses the Q-values associated with the actions selected by the ϵ -greedy policy as the consequent parameters in the defuzzification procedure in (5.36). Therefore, the output of the *q-critic* is given as

$$Q_t(\bar{x}_t) = \sum_{l=1}^L \varphi_t^l q_t(l, a) \quad (5.48)$$

The external reinforcement signal r_{t+1} received from the environment after executing action $\tilde{U}_t(\bar{x}_t)$ in state x_t and transitioning to state x_{t+1} is used to compute a TD error $\tilde{\Delta}_{t+1}$ given by

$$\tilde{\Delta}_{t+1} = r_{t+1} + \gamma Q_{t+1}^*(\bar{x}_{t+1}) - Q_t(\bar{x}_t) \quad (5.49)$$

where $\gamma \in [0, 1]$ is the discounting factor and $Q_{t+1}^*(\bar{x}_{t+1})$ is the maximum expected cumulative reward from the next state $\bar{x}_{t+1}$ given as

$$Q_{t+1}^*(\bar{x}_{t+1}) = \sum_{l=1}^L \varphi_{t+1}^l \max_{a \in A} q_t(l, a) \quad (5.50)$$

The TD error is used to adapt the *q-critic*. The adaptation law for the *q-critic* is given as

$$q_{t+1}(l, a) = q_t(l, a) + \eta \tilde{\Delta}_{t+1} \frac{\partial Q_t(\bar{x}_t)}{\partial q_t(l, a)} \quad (5.51)$$

where $\eta \in [0, 1]$ is the learning rate of the *q-critic*. The partial derivate in (5.51) is

Algorithm 5.2 FQL algorithm

-
- 1: Initialize $Q(\cdot) = 0$, $q(l, a) = 0$, for all $l = 1, \dots, L$ and $a \in A$.
 - 2: **for** each time step t **do**
 - 3: Obtain the inputs $\bar{x}_t$.
 - 4: Choose an action for each rule l based on (5.45).
 - 5: Compute the control action $\tilde{U}_t(\bar{x}_t)$ in (5.47).
 - 6: Estimate the action-value function $Q_t(\bar{x}_t)$ in (5.48).
 - 7: Use $\tilde{U}_t(\bar{x}_t)$ to run the game for the current time step t .
 - 8: Obtain the new inputs $\bar{x}_{t+1}$ and the reward r_{t+1} at time $t + 1$.
 - 9: Estimate the action-value function $Q_t^*(\bar{x}_{t+1})$ in (5.50).
 - 10: Compute the TD error $\tilde{\Delta}_{t+1}$ in (5.49).
 - 11: Adapt $q(l, a)$ in (5.51) for $l = 1, 2, \dots, L$.
 - 12: **end for**
-

easily determined from (5.48) to be

$$\frac{\partial Q_t(\bar{x}_t)}{\partial q_t(l, a)} = \varphi_t^l \quad (5.52)$$

The FQL algorithm is summarized in Algorithm 5.2 [9]. In Chapter 6, we provide details of how we set the values of the learning factors in the FQL algorithm.

5.5 Comparison of the Fuzzy Q-Learning with the Fuzzy Actor-Critic Learning

The FQL algorithm and the FACL algorithm are members of the actor-critic reinforcement learning methods. Both algorithms make use of two controllers, one to implement the actor and the other to implement the critic. The major difference between the FQL algorithm and the FACL algorithm lies in the fact that the FQL algorithm requires a discrete set of actions to determine the continuous control action, which is not required in the FACL algorithm.

The fact that the FQL algorithm requires a discrete set of action is essentially

a drawback to the use of the FQL algorithm. This is because one may not have a priori knowledge of the domain of the action space nor how large the action space is. As such, it becomes difficult to discretize the action space. Moreover, if the domain of the action space is known, a fine discretization of the action space will make the dimension of the Q-table large, which will result in large memory storage and slow learning procedures [71]. On the other hand, a coarse discretization of the action space will result in poor performance because the generated action will not be smooth [71].

In Chapter 6, we apply the two algorithms to the game of guarding a territory. We simulate and compare the performance between the FQL algorithm and the FACL algorithm.

5.6 Reward Function

In the guarding a territory game, one may use the payoff obtained at game termination to reward the performance of the learning agents. However, delaying the reward so that it is only received at game termination will result in a slow learning procedure and will also make the learning process difficult. This is because by game termination the agents would have taken many actions. As such, it would be difficult to know which actions were good and should be rewarded, and which were bad and should be penalized. In reinforcement learning, this problem is called the temporary credit assignment problem [27], [28].

To improve the learning process of the agents, one may introduce instantaneous rewards to the environment [72], [73]. Instantaneous rewards provide immediate reinforcement to compensate for the delayed reward. As a result, the agents would learn instantaneously which actions they take are good and which are bad. The reward

function must be designed such that it reflects the goal of the agents. To simplify the design of the reward function, we make a temporary assumption that only two-players (an invader and a guard) are present in the game. In Section 6.3, details of the design of the reward function for the three-player game (two guards and one superior invader) will be provided.

5.6.1 Reward Function for the Invader

We found the reward function applied to the invader in [17], for a two-player game, suitable for our application. The reward function R_{t+1}^I for the invader I has two components. The first component δ_{IT} gives a positive reward if the invader moves closer to the territory T and a negative reward otherwise, given as

$$\delta_{IT} = \text{dist}_{IC_T}(t) - \text{dist}_{IC_T}(t+1) \quad (5.53)$$

where $\text{dist}_{IC_T}(t)$ denotes the Euclidean distance between the invader I and the center of the territory C_T at time step t . The second component δ_{IG} gives a positive reward if the invader moves farther from the guard G and a negative reward otherwise, given as

$$\delta_{IG} = \text{dist}_{IG}(t+1) - \text{dist}_{IG}(t) \quad (5.54)$$

Putting the two components together results in the reward function given in (5.55)

$$R_{t+1}^I = D\delta_{IT} + J\delta_{IG} \quad (5.55)$$

where D and J are constants that determine the importance attached to the first and the second component of the reward function respectively.

A trade-off between moving farther from the guard and moving closer to the

territory has to be reached. For example, if we choose D as a large number, this will make the invader move towards the territory while ignoring the guard. On the other hand, if we choose J as a large number, this will make the invader overly afraid of the guard. As a result, the invader will prefer a longer path to reach the territory. In Chapter 6, we provide details of how we set the values of these constants D and J .

5.6.2 Reward Function for the Guard

The reward function for the guard R_{t+1}^G is designed using a method similar to the one in Section 5.6.1. The reward function R_{t+1}^G has two components. The first component δ_{GI} gives a positive reward if the guard moves closer to the invader and a negative reward otherwise, given as

$$\delta_{GI} = \text{dist}_{GI}(t) - \text{dist}_{GI}(t+1) \quad (5.56)$$

The second component δ_{GT} gives a positive reward if the guard moves closer to the territory and a negative reward otherwise, given as

$$\delta_{GT} = \text{dist}_{GT}(t) - \text{dist}_{GT}(t+1) \quad (5.57)$$

Putting the two components together results in the reward function given in (5.58)

$$R_{t+1}^G = P\delta_{GI} + M\delta_{GT} \quad (5.58)$$

where P and M are constants that determine the importance attached to the first and the second component of the reward function respectively.

The reward function for the guard is illustrated in Fig. 5.6. A trade-off between moving closer to the territory and moving closer to the invader must be reached. For

Figure 5.6: The guard's reward function description.

example, if we choose P as a large number, the guard will directly pursue after the invader, which is an irrational strategy well studied in [17]. On the other hand, if we choose M as a large number, this will make the guard ignore the invader and move towards the territory. In Chapter 6, we provide details of how we set the values of the constants P and M .

5.7 Summary

In this chapter, we provided a brief introduction to fuzzy logic. We discussed the basic fuzzy set operations which include the complement operation, the union operation and the intersection operation. We described the fuzzy inference procedure, using the product inference engine as our primary example. We then introduced the fuzzy logic controller, with a focus on the Sugeno fuzzy model. We presented details of the design of the controller for the guarding a territory problem which include the inputs to the controller, the fuzzy sets and the fuzzy rules. In addition, we introduced and compared two fuzzy reinforcement learning algorithms: the FQL algorithm and the

FACL algorithm. Finally, we provided details of the design of the reward functions for the players' learning process.

Chapter 6

Simulation Results

In this chapter, we apply the fuzzy reinforcement learning methods described in Chapter 5 to the guarding a territory game. We make use of the problem set described in Section 4.4 to simulate and evaluate the performance of the players. In our simulations, all the players in the game (the invader and the guards) are simultaneously learning. We set the total simulation time for an epoch to 60 *seconds*. An epoch describes a single run of the game, which starts when the players are in their initial positions and ends when the game meets a terminating condition. The game terminates if the simulation timer for an epoch expires, the invader reaches the territory or the invader is within the capture radius R_C of the guards. We set $R_C = 2$ *units* for all the guards.

For the FACL algorithm, we set the learning rates $\beta_0 = 0.05$ in (5.41), $\alpha_0 = 0.1$ in (5.42). Since we do not make use of terminal rewards in the game, we propose to use a small discount factor $\gamma = 0.5$. This is done to make the agent focus on maximizing the instantaneous rewards. We set the variance of the exploration noise $\sigma_0^2 = 1$. The

decay rates of the learning factors after each epoch k are set as follows:

$$\beta_{k+1} = 0.994^k \beta_0 \quad (6.1)$$

$$\alpha_{k+1} = 0.999^k \alpha_0 \quad (6.2)$$

$$\sigma_{k+1}^2 = 0.994^k \sigma_0^2 \quad (6.3)$$

Similarly, for the FQL algorithm, we set the learning rate $\eta_0 = 0.1$ in (5.51). We set the ϵ -greedy policy in (5.45) to $\epsilon_0 = 0.3$. The discount factor is set to $\gamma = 0.5$. The decay rates of the learning factors after each epoch k are set as follows:

$$\epsilon_{k+1} = 0.95^k \epsilon_0 \quad (6.4)$$

$$\eta_{k+1} = 0.999^k \eta_0 \quad (6.5)$$

The Kinematic equations of the players given in Equation (4.4)–(4.6) were solved in simulation using the second-order Runge-Kutta method [57]. The sampling time is set to 10 *milliseconds*. The players have a constant linear speed throughout the game. However, the angular speed ω_t of any one of the players is given as

$$\omega_t = K(u_t - \theta_t) \quad (6.6)$$

where K is the proportional gain, u_t is the desired heading angle obtained from the controller and θ_t is the current heading angle of the player relative to the global x -axis. We set $K = 1$. For the FACL algorithm, u_t is determined using Equation (5.38), whereas, for the FQL algorithm, u_t is determined using Equation (5.47).

This chapter is organized as follows. In Section 6.1, we evaluate and compare the FACL algorithm with the FQL algorithm using the scenario described in Section 4.4.1. In Section 6.2, we evaluate the performance of the players using the scenario described

in Section 4.4.2. Section 6.3 describes the inputs and the rewards for the three-player game. Furthermore, we evaluate the performance of the players in the three-player game using the scenario described in Section 4.4.3.

6.1 Two Player Game: An Invader Against a Guard With the Same Speed

In this scenario, there are two players in the game (an invader and a guard) as described in Section 4.4.1. Both players in the game have the same speed. We set the speed of the players to $V_I = V_G = 1 \text{ unit/second}$. For the reward function of the invader given in Equation (5.55), we set the parameter $D = 2$ and $J = 1$. The parameter D describes the importance the invader attaches to moving towards the territory, while J describes the importance the invader attaches to moving farther away from the guard. The value of the parameter D was chosen to be twice the value of the parameter J so that the invader places greater importance on moving towards the territory.

Similarly, for the reward function of the guard given in Equation (5.58), we set the parameter $P = 1.1$ and $M = 1$. The parameter P describes the importance the guard attaches to moving towards the invader, while M describes the importance the guard attaches to moving towards the territory. Both components of the reward function in Equation (5.58) will work jointly to guide the guard to determine the optimal point in-between the invader and the territory that maximizes the capture distance of the invader from the territory, as illustrated in Fig. 5.6.

Our objectives in this scenario are as follows:

1. To investigate the possibility of both the invader and the guard simultaneously

Figure 6.1: The game setup for an invader against a guard with the same speed—possible capture.

learning given that both players have the same speed.

2. To evaluate and compare the performance between the FQL algorithm and the FACL algorithm.

6.1.1 Possible Capture

The game setup is shown in Fig. 6.1. The invader's initial position is set at $[5, 5]$. The guard's initial position is set at $[5, 25]$. The initial orientation of both players is set to 0 *radians*. The center of the territory C_T is located at $[20, 20]$ and the radius $R_T = 3$ *units*. Also shown in Fig. 6.1 is the perpendicular bisector line (*PBL*) that defines the regions the players can reach before their opponent. As depicted by line *PBL* in Fig. 6.1, the territory is not within the reachable region of the invader. As such, the guard should be able to intercept the invader before the invader reaches the territory, assuming both players learn to play rationally.

(a)

(b)

Figure 6.2: FQL simulation: The trajectory plot for an invader against a guard with the same speed—possible capture (a) the first epoch (b) the final epoch.

FQL Simulation

We run the simulation for 500 epochs of training using the FQL algorithm. The trajectories of the players at the first and final epochs are shown in Fig. 6.2. We

observe in Fig. 6.2a that at the first epoch, the players make exploratory movements in search of their optimal strategies. The trajectories of the players at the final epoch are shown in Fig. 6.2b. We observe in Fig. 6.2b that the invader learns to play rationally and tries to reach the territory. However, ultimately, the guard learns its optimal behavior to capture the invader before it can reach the territory.

FACL Simulation

Similarly, we run the simulation for 500 epochs of training using the FACL algorithm. The trajectories of the players at the first and final epochs are shown in Fig. 6.3. We observe at the first epoch shown in Fig. 6.3a that the players play irrationally. This is due to the fact that both players have no a priori knowledge of their optimal behaviors. However, after successfully tuning the controllers of the players, we observe at the final epoch shown in Fig. 6.3b that the invader learns to play rationally and tries to reach the territory. However, the guard learns its optimal strategy to intercept the invader before it can reach the territory.

Average Payoff

In this thesis, we make use of the payoff obtained at game termination to evaluate the performance of the learning agents. We run the simulation for 10 learning trials using both the FQL algorithm and the FACL algorithm. A single learning trial describes one complete cycle of the game which contains 500 training epochs. The payoff averaged over 10 learning trials is shown in Fig. 6.4. In Fig. 6.4, we plotted the average payoff for both the FQL algorithm and the FACL algorithm. This is done to increase the ease of comparing between the two algorithms. The average payoff obtained using the FQL algorithm converges to 3.12 *units*. Similarly, the average payoff obtained

(a)

(b)

Figure 6.3: FACL simulation: The trajectory plot for an invader against a guard with the same speed—possible capture (a) the first epoch (b) the final epoch.

using the FACL algorithm converges to 3.28 *units*. In order to compare the players' performance to the optimal strategy given by the perpendicular bisector approach, we run one test epoch where both players apply their optimal strategy (with no learning).

Figure 6.4: The average payoff plot for an invader against a guard with the same speed—possible capture.

Figure 6.5: The average termination time plot for an invader against a guard with the same speed—possible capture.

The optimal strategy gives the payoff value of 3.21 *units*, as shown in Fig. 6.4. We observe in Fig. 6.4 that there is only a minor difference, less than 0.10 *units*, between

the average payoff obtained using the learning algorithms and the payoff given by the optimal strategy. This demonstrates that the players are able to learn their optimal strategies using both learning algorithms.

We also observe in Fig. 6.4 that although both algorithms converge to the optimal strategy, the FACL algorithm converges before the FQL algorithm. The FACL algorithm converges after 300 epochs of training, as shown in Fig. 6.4, whereas, the FQL algorithm converges after 380 epochs of training. This demonstrates that the FACL algorithm converges faster than the FQL algorithm in this scenario.

Average Termination Time

To further compare the FQL algorithm with the FACL algorithm we plotted the average termination time of both algorithms over 10 learning trials, as shown in Fig. 6.5. We observe in Fig. 6.5 that the average termination time of the FQL algorithm is 17.48 *seconds* after convergence. Similarly, the average termination time of the FACL algorithm is 16.87 *seconds* after convergence. We observe in Fig. 6.5 that there is only a minor difference, approximately 0.61 *seconds*, between the average termination time of the FQL algorithm and the FACL algorithm. Although both algorithms perform effectively well, in this scenario the FACL algorithm converges faster than the FQL algorithm, as shown in Fig. 6.5.

6.1.2 Possible Reaching

In this scenario illustrated in Fig. 6.6, we bring the invader forward to the location [9, 9] to investigate whether the invader can learn its optimal strategy to reach the territory when it is possible to do so. As depicted by line *PBL* in Fig. 6.6, the territory is within the reachable region of the invader. Regardless of what the guard does

Figure 6.6: The game setup for an invader against a guard with the same speed—possible reaching.

Figure 6.7: FQL simulation: The trajectory plot of the final epoch for an invader against a guard with the same speed—possible reaching.

in this scenario, the invader should be able to reach the territory if it learns to play rationally.

Figure 6.8: FACL simulation: The trajectory plot of the final epoch for an invader against a guard with the same speed—possible reaching.

FQL Simulation

We run the simulation for 500 epochs of training using the FQL algorithm. The trajectories of the learning agents at the final epoch are shown in Fig. 6.7. We observe in Fig. 6.7 that the invader is able to learn its optimal behavior to reach the territory.

FACL Simulation

Similarly, we run the simulation for 500 epochs of training using the FACL algorithm. The trajectories of the learning agents at the final epoch are shown in Fig. 6.8. We observe in Fig. 6.8 that the invader learns to play rationally. The invader avoids capture and reaches the territory.

Average Payoff

The payoff averaged over 10 learning trials is shown Fig. 6.9. In Fig. 6.9, we plotted

Figure 6.9: The average payoff plot for an invader against a guard with the same speed—possible reaching.

Figure 6.10: The average termination time plot for an invader against a guard with the same speed—possible reaching.

the average payoff for both the FQL algorithm and the FACL algorithm. We observe in Fig. 6.9 that the average payoff for both the FQL algorithm and the FACL

algorithm converges to zero as expected. Also, in this scenario, we observe that the FACL algorithm converges faster than the FQL algorithm. The FACL algorithm converges after 40 epochs of training, whereas, the FQL algorithm converges after 60 epochs of training.

Average Termination Time

The average termination time for both the FQL algorithm and the FACL algorithm over 10 learning trials is shown in Fig. 6.10. We observe in Fig. 6.10 that the average termination time of the FQL algorithm is 14.54 *seconds* after convergence. Similarly, the average termination time of the FACL algorithm is 14.36 *seconds* after convergence. Both algorithms are able to perform well in this scenario. However, the simulation results demonstrate that the FACL algorithm converges faster than the FQL algorithm in this scenario.

6.1.3 Discussion

The simulation results presented in this section demonstrated that the players were able to learn their optimal strategies using both learning algorithms. Both algorithms performed effectively well in all the scenarios presented in this section. We evaluated and compared both algorithms based on the average payoff obtained at game termination and the average termination time. We observed that both algorithms achieved similar results. However, in the scenarios described in this section, we observed that the FACL algorithm converges faster than the FQL algorithm. This may be due to the fact that the FQL algorithm makes use of a discrete set of actions. As such, the FQL algorithm will require more training examples to update all the values in the

state-action function. Since fast convergence is a sought after property in reinforcement learning, especially when solving problems where the state and action spaces are large or in the continuous domain, we conclude that the FACL algorithm is the best algorithm for our application.

6.2 Two Player Game: An Inferior Invader Against a Superior Guard

In this scenario, there are two players in the game (an invader and a guard) as described in Section 4.4.2. We set the speed of the invader to $V_I = 1.0 \text{ units/second}$, while the speed of the guard is set to $V_G = 1.3 \text{ units/second}$. Therefore, the guard is 30 percent faster than the invader. For the reward function of the invader given in Equation (5.55), we set the parameters $D = 2$ and $J = 1$. Similarly, for the reward function of the guard given in Equation (5.58), we set the parameters $P = 1.4$ and $M = 1$.

In this section, we apply only the FACL algorithm. Each epoch is carried out in two phases: a training phase where the learning occurs and a test phase where the players apply what they have learned. In the training phase, we set the players' initial positions at random in specified training regions, whereas in the test phase, we set the players' initial positions at fixed locations. We record only the payoff obtained in the test phase. We average the payoff over 10 learning trials. In this scenario, a single learning trial describes one complete cycle of the game which contains 500 training epochs and 500 test epochs. The procedure is summarized in Algorithm 6.1 [17]. In Algorithm 6.1, k and p denote the current index of the epoch and the learning trial respectively. The objective of the Test-Train algorithm is to investigate whether the training performed in one area can be applied to other regions.

Algorithm 6.1 Train-Test Algorithm

```

1: Initialize  $p = 1, k = 1$ .
2: while  $p \leq 10$  do
3:   Initialize  $W^l = 0, \zeta^l = 0$  for  $l = 1, \dots, L$ .
4:   while  $k \leq 500$  do
5:     Run training epoch with the players' initial positions assigned at
       random in training area.
6:     Adapt  $W_{t+1}^l$  in (5.41) and  $\zeta_{t+1}^l$  in (5.42) for  $l = 1, \dots, L$ .
7:     Run test epoch with the players' initial positions fixed at assigned
       test locations.
8:     Record the payoff achieved.
9:     Update the learning factors in (6.1)–(6.3).
10:     $k = k + 1$ .
11:   end while
12:   Record the average payoff achieved.
13:    $p = p + 1$ .
14: end while

```

The objectives of this section are as follows:

1. To investigate the possibility of both the invader and the guard simultaneously learning given that the guard is superior to the invader.
2. To investigate if the parameters learned in one area can also be used in other regions (by applying the Test-Train Algorithm).

6.2.1 Possible Capture

The game setup is shown in Fig. 6.11. The training regions for the players are illustrated in Fig. 6.11. We set the test location for the invader at $[5, 5]$, while the test location for the guard is set at $[5, 25]$. We set the initial orientation of both players to 0 *radians*. The center of the territory C_T is located at $[20, 20]$ and the radius $R_T = 3$ *units*. Also shown in Fig. 6.11 is the Apollonius circle that describe the region the invader can reach before the guard. We observe in Fig. 6.11 that the territory is not within the reachable region of the invader. As such, the guard should

Figure 6.11: The game setup for an inferior invader against a superior guard—possible capture.

be able to intercept the invader before the invader reaches the territory, assuming both players learn to play rationally.

We run the simulation for 500 epochs using the Test-Train algorithm. The trajectories of the players at the first and final epochs are shown in Fig. 6.12. We observe at the first epoch shown in Fig. 6.12a that the players make exploratory movements in search of their optimal strategies. The trajectories of the players at the final epoch are shown in Fig. 6.12b. We observe in Fig. 6.12b that the invader learns to play rationally and tries to reach the territory. However, the guard learns its optimal behavior to capture the invader before it can reach the territory. Given that the guard is superior to the invader, the guard is able to capture the invader at a farther distance from the territory, than it was able to in the simulation results obtained previously in Section 6.1.1.

The payoff averaged over 10 learning trials is shown in Fig. 6.13. The average payoff converges to 6.64 *units*. In order to compare the players' performance to the

(a)

(b)

Figure 6.12: FACL simulation: The trajectory plot for an inferior invader against a superior guard—possible capture (a) the first epoch (b) the final epoch.

optimal strategy given by the Apollonius circle approach, we run one test epoch where both players apply their optimal strategy (with no learning). The optimal strategy gives the payoff value of 6.52 *units*, as shown in Fig. 6.13. We observe in Fig. 6.13 that

Figure 6.13: The average payoff plot for an inferior invader against a superior guard—possible capture.

there is only a minor difference, approximately 0.12 *units*, between the average payoff obtained using the learning algorithm and the payoff given by the optimal strategy. This demonstrates that both players are able to learn their optimal strategies using the FACL algorithm.

6.2.2 Possible Reaching

In this scenario illustrated in Fig. 6.14, we bring the invader forward to the location $[14, 8]$ to investigate whether the invader can learn its optimal behavior to reach the territory when it is possible to do so. As depicted by the Apollonius circle in Fig. 6.14, the territory is within the reachable region of the invader. Therefore, If the invader learns to play rationally, it should be able to reach the territory.

We run the simulation for 500 Epochs. The trajectories of the players obtained at the final epoch are shown in Fig. 6.15. We observe in Fig. 6.15 that the invader learns its optimal behavior to effectively avoid capture and reach the territory.

Figure 6.14: The game setup for an inferior invader against a superior guard—possible reaching.

Figure 6.15: FACL simulation: The trajectory plot of the final epoch for an inferior invader against a superior guard—possible reaching.

The payoff averaged over 10 learning trials is shown in Fig. 6.16. We observe in Fig. 6.16 that the average payoff converges to zero as expected and this occurs at the

Figure 6.16: The average payoff plot for an inferior invader against a superior guard—possible reaching.

80th epoch. This demonstrate that the invader is able to learn its optimal strategy after 80 epochs.

6.2.3 Discussion

The simulation results in this section demonstrated that the FACL algorithm was able to determine the optimal strategies of the players in an effective way. We observed in Section 6.2.1 that the guard with no a priori knowledge of the speed of the invader, was able to learn that it was 30 percent faster than the invader. As such, the guard was able to capture the invader at a farther distance from the territory, than it was able to in the simulation results obtained previously in Section 6.1, when both players had the same speed. We also observed in Section 6.2.2 that when the game is setup such that it is possible for the invader to reach the territory, the invader would learn its optimal behavior to avoid capture and reach the territory.

Figure 6.17: FACL simulation: The trajectory plot for an inferior invader against a superior guard with the test location for the invader set at $[8.5, 6.5]$, and the test location for the guard set at $[8.5, 26.5]$ —possible capture (a) the game setup (b) the first simulation result.

The simulation results presented in this section also demonstrated that the training in one area can be applied to other regions. As shown in Fig. 6.11, the test locations for the players are fixed locations within the specified area. Specifically, the test locations for the invader and the guard are 1.5 *units* distance away from their respective training regions. As demonstrated by the simulation results in Section 6.2.1, the agents were able to perform optimally in the test locations although they had received no previous training in these locations.

To explore if the parameters learned in the specified area can also be used in regions outside the area, we also selected random test locations that are 1 *unit* distance away from the specified area. One of the scenarios we made use of in our investigation is illustrated in Fig. 6.17a. The present simulation was run 10 times using the tuned parameters obtained for the different learning trials in the simulation run in Section 6.2.1. The trajectories of the players for the first trial are shown in Fig. 6.17b. The average payoff obtained over the 10 trials was 4.10 *units* with a standard deviation of 0.24 *units*. The optimal solution, as given by the Apollonius circle approach, is 3.99 *units*. Therefore, we observe that the performance of the players in this scenario came close to the optimal solution.

6.3 Three-Player Game: A Superior Invader Against Two Inferior Guards

In this scenario, there are three players in the game (two guards and one superior invader) as described in Section 4.4.3. We now provide details of the controller inputs and the rewards for the three-player game.

For the inputs to the controller of the invader in the three-player game, we propose the use of five state variables. As previously illustrated in Section 5.2, the first variable

θ_{IT} describe the position of the territory with respect to the invader. The second variable $d_{IG_1}^x$ and the third variable $d_{IG_1}^y$ describe the position of guard 1, G_1 , with respect to the invader. Similarly, the fourth variable $d_{IG_2}^x$ and the fifth variable $d_{IG_2}^y$ describe the position of guard 2, G_2 , with respect to the invader. Therefore, the vector of state variables $\bar{x}_I$ that define the inputs to the controller of the invader is given as

$$\bar{x}_I = [\theta_{IT}, d_{IG_1}^x, d_{IG_1}^y, d_{IG_2}^x, d_{IG_2}^y] \quad (6.7)$$

For the inputs to the controller of any one of the guards, we propose the use of three state variables. The first state variable describe the position of the territory with respect to the guard and the other two state variables describe the position of the invader with respect to the guard. We take for example the inputs to the controller of guard 1. The first variable θ_{G_1T} describe the position of the territory with respect to guard 1. The second variable $d_{G_1I}^x$ and the third variable $d_{G_1I}^y$ describe the position of the invader with respect to guard 1. Therefore, the vector of state variables $\bar{x}_{G_1}$ and $\bar{x}_{G_2}$ that define the inputs to the controller of guard 1 and guard 2 are given as

$$\bar{x}_{G_1} = [\theta_{G_1T}, d_{G_1I}^x, d_{G_1I}^y] \quad (6.8)$$

$$\bar{x}_{G_2} = [\theta_{G_2T}, d_{G_2I}^x, d_{G_2I}^y] \quad (6.9)$$

respectively. Our objective in choosing three state variables to define the inputs to the controller of the guards is to investigate whether the guards are able to learn in a decentralized manner. In other words, it was done to investigate the possibility of both guards jointly working to capture the invader with no form of communication between the guards.

We now describe the fuzzy sets defined in the various input domains. For the angle variable θ , we make use of the fuzzy sets defined in Fig. 5.3a. The variable θ describe

the inputs: θ_{IT} , θ_{G_1T} and θ_{G_2T} . Similarly, for the Manhattan distance variable d , we make use of the fuzzy sets defined in Fig. 5.3b. The variable d describe the inputs: $d_{IG_1}^x$, $d_{IG_1}^y$, $d_{IG_2}^x$, $d_{IG_2}^y$, $d_{G_1I}^x$, $d_{G_1I}^y$, $d_{G_2I}^x$, and $d_{G_2I}^y$. For the invader, we apply a total of $10 \times 7 \times 7 \times 7 \times 7 = 24010$ fuzzy rules. However, for each of the guards, we apply a total of $10 \times 7 \times 7 = 490$ fuzzy rules.

We now describe the reward function for the three-player game. We modify the reward function for the invader given in (5.55) for the two-player game so that the invader can learn using the information of the relative positions of the two guards. The modified reward function is given as

$$R_{t+1}^I = D\delta_{IT} + J_1\delta_{IG_1} + J_2\delta_{IG_2} \quad (6.10)$$

where D , J_1 , J_2 are constants that determine the importance attached to the various components of the reward function. The component δ_{IT} is defined in Equation (5.53), whereas, the components δ_{IG_1} and δ_{IG_2} are defined in Equation (5.54). We set the parameters $D = 3.0$, $J_1 = 1.0$ and $J_2 = 1.0$. For the guards, we apply the reward function given in Equation (5.58) to both guard 1 and guard 2. We set the parameters $P = 1.1$ and $M = 1.0$.

In this section, we apply only the FACL algorithm. We run all the simulations in this section using the Test-Train algorithm described in Algorithm 6.1. We set the speed of the guards to $V_{G_1} = V_{G_2} = 1 \text{ unit/second}$, while the speed of the invader is set to $V_I = 1.3 \text{ units/second}$. Therefore, the invader is 30 percent faster than the guards.

Our objective in this section are as follows:

1. To investigate the possibility of two guards and a single invader simultaneously learning given that the invader is superior to both guards.

Figure 6.18: The game setup for a superior invader against two inferior guards—possible capture.

2. To investigate the possibility of two guards jointly working to defend the territory against a superior invader even though they do not directly share any information.

6.3.1 Possible Capture

The game setup is shown in Fig. 6.18. The training regions for the players are also illustrated in Fig. 6.18. We set the test location for Guard 1 at $[5, 5]$ and its initial orientation to 0 *radians*. The test location for Guard 2 is set at $[25, 25]$ and its initial orientation to $-\pi/2$ *radians*. Similarly, we set the test location for the invader at $[5, 25]$ and its initial orientation to 0 *radians*. The center of the territory C_T is located at $[22, 8]$ and the radius $R_T = 3$ *units*. The Apollonius circles constructed in Fig. 6.18 describe the regions the guards can reach before the invader. Although the Apollonius circles do not completely encircle the territory, the guards should be able

Figure 6.19: FACL simulation: The trajectory plot of the final epoch for a superior invader against two inferior guards with guard 2 located at coordinate $[25, 25]$ absent.

Figure 6.20: The average payoff plot for a superior invader against two inferior guards with guard 2 located at coordinate $[25, 25]$ absent.

to intercept the invader before the invader reaches the territory. This is possible if the guards exploit their 2 *units* capture radius.

We run the simulation for 500 epochs with guard 2 located at $[25, 25]$ absent. This is done to investigate whether a single guard can defend the territory in this scenario. The trajectories of the learning agents at the final epoch are shown in Fig. 6.19. We observe in Fig. 6.19 that guard 1 alone cannot stop the superior invader from reaching the territory. As shown in Fig. 6.19, the guard made an attempt to move as close as possible to the invader to effect capture. However, the invader learned its optimal strategy to aim for the region of the territory that it can reach before the guard. The average payoff over 10 learning trials is shown in Fig. 6.20. We observe in Fig. 6.20 that, on average, the learning process is complete after 100 epochs. From the 100th epoch onwards, the invader is able to reach the territory regardless of what the single guard did.

We introduce guard 2 into the game and run the simulation for 500 epochs. The trajectories of the learning agents at the first and final epochs are shown in Fig. 6.21. We observe at the first epoch shown in Fig. 6.21a that the players play irrationally. This is due to the fact that all three players have no a priori knowledge of their optimal behaviors. However, after successfully tuning the controllers of the players, we observe at the final epoch shown in Fig. 6.21b that the two guards are able to jointly work to capture the superior invader. The guards are able to implicitly cooperate to capture the invader although there is no direct communication between the guards. This is possible because the flow of information of the presence of a second guard in the game is transmitted to the other guard through the motion of the invader. In other words, guard 1 has implicit information about the presence of guard 2 in the game through the motion of the invader.

The payoff averaged over 10 learning trials is shown in Fig. 6.22. The average payoff converges to 5.34 *units*. In order to compare the player's performance to the optimal strategy given by the Apollonius circle approach, we run one test epoch where

(a)

(b)

Figure 6.21: FACL simulation: The trajectory plot for a superior invader against two inferior guards with guard 2 located at coordinate $[25, 25]$ present—possible capture (a) the first epoch (b) the final epoch.

all three players apply their optimal strategy (with no learning). The optimal strategy gives the payoff value of 5.13 *units*, as shown in Fig. 6.22. We observe in Fig. 6.22 that

Figure 6.22: The average payoff plot for a superior invader against two inferior guards with guard 2 located at coordinate $[25, 25]$ present—possible capture.

there is only a minor difference, approximately 0.21 *units*, between the average payoff obtained using the learning algorithm and the payoff given by the optimal strategy. This demonstrates that all three players are able to learn their optimal strategies using the FACL algorithm.

6.3.2 Possible Reaching

In this scenario illustrated in Fig. 6.23, we bring the invader forward to the location $[7, 23]$ to investigate whether the invader can learn its optimal behavior to reach the territory when it is possible to do so. We observe in Fig. 6.23 that based on the new test location of the invader, there is a wide opening between the adjacent Apollonius circles making it possible for the invader to reach the territory if it learns to play rationally.

We run the simulation for 500 epochs using the Test-Train algorithm. The trajectories of the learning agents at the final epoch are shown in Fig. 6.24. We observe

Figure 6.23: The game setup for a superior invader against two inferior guards—possible reaching.

Figure 6.24: FACL simulation: The trajectory plot of final epoch for a superior invader against two inferior guards—possible reaching.

in Fig. 6.24 that the invader learned its optimal behavior by exploiting the opening between the adjacent Apollonius circles to reach the territory.

Figure 6.25: The average payoff plot for a superior invader against two inferior guards—possible reaching.

The payoff averaged over 10 learning trials is shown in Fig. 6.25. We observe in Fig. 6.25 that the FACL algorithm converges to zero as expected and this occurs at the 160th epoch. This demonstrates that the FACL algorithm determined the optimal strategies of the players in an effective way.

6.3.3 Discussion

When designing the controller inputs for the three-player game, we initially thought that the guards would require explicit information about the position of the other guard. However, as the simulation results demonstrates, the information about the existence of the second guard is implicitly transmitted through the motion of the invader. As such, the information about the location of the second guard is not required and that drastically simplified the computational complexity of the system.

The simulation results presented in this section also demonstrated that the learning algorithm was able to determine the optimal strategies of all three players in an

effective way. We observe in Section 6.3.1 that a single guard was not able to stop the superior invader from reaching the territory. However, when we introduced an additional guard to the game, both guards were able to work jointly to intercept the invader before it can reach the territory. The guards were able to implicitly cooperate to capture the superior invader although there is no direct communication between the guards. This demonstrates that the players were able to learn in a decentralized manner. We also observe in Section 6.3.2 that when the game is setup such that it is possible for the invader to reach the territory, the invader is able to learn its optimal behavior to exploit the opening between the adjacent Apollonius circles to reach the territory.

6.4 Summary

In this chapter, we evaluated and compared the performance of the FQL algorithm with the FACL algorithm. Our simulation results demonstrated that both learning algorithms were able to guide the players to determine their optimal strategies. However, in the scenarios described in this chapter, we observed that the FACL algorithm converges faster than the FQL algorithm.

In both the two-player game and the three-player game described in this chapter, the FACL algorithm performed effectively well in determining the optimal strategies of the players. In Section 6.3, we also demonstrated the potential for the FACL algorithm to be used as a completely decentralized learning algorithm.

Chapter 7

Conclusion

In this thesis, we investigated the use of reinforcement learning to train the players (the invader and the guards) in the game of guarding a territory. We made use of multi-robot systems as our players in the game. We demonstrated that multi-robot systems with conflicting interests can learn their optimal strategies against their opponents using reinforcement learning techniques.

We designed a control system to guide the players in the game. Our control system was designed using the fuzzy logic controller (FLC). With the aid of reinforcement learning we tuned the control system to improve the performance of the players. The combination of the FLC with reinforcement learning resulted in two fuzzy reinforcement learning methods known as the fuzzy Q-learning (FQL) and the fuzzy actor-critic learning (FACL).

We applied both fuzzy reinforcement learning techniques to the guarding a territory game. Simulation results demonstrated that both the FQL and the FACL can be used to train the players. However, in the scenarios described in this thesis, we observed that the FACL converges faster than the FQL. Therefore, we concluded that the FACL algorithm is the best algorithm for our application.

In many applications of the guarding a territory game in the real-world, the participants may not have the same speed. As such, we investigated the other possibility of the game where the players have different speeds. We introduced the Apollonius circle approach to determine the theoretical solution of the game when the players do not have the same speed. We evaluated the players' learning performance using the optimal solution given by the Apollonius circle approach.

We first considered the possibility where the guard is faster than the invader. Our simulation results demonstrated that our designed control system was able to guide both players to achieve their optimal performance although the players have no a priori knowledge of the speed of their opponent.

Furthermore, we considered the second possibility of the game where the invader is faster than two guards. Simulation results demonstrated that the two guards were able to learn their optimal strategies to defend the territory in an effective way although there was no direct communication between the two guards. Our designed control system was able to guide all three players in this scenario to achieve their optimal performance.

In conclusion, the simulation results demonstrates that our designed control system with reinforcement learning is able to effectively guide multi-robot systems to determine their optimal strategies against their opponent in the game of guarding a territory.

7.1 Contributions

In summary, these are the contributions presented in this thesis:

1. We introduced the Apollonius circle approach to determine the optimal solution of the game of guarding a territory when the players do not have the same speed.

We illustrated in Chapter 4 how one can apply the Apollonius circle approach to determine the theoretical solution of the game.

2. We investigated the possibility of both players (the invader and the guard) learning simultaneously in the game. Simulation results in Section 6.1 and Section 6.2 demonstrated that our designed control system was able to guide the players to determine their optimal strategies against their opponent.
3. We investigated how two guards can implicitly cooperate with each other to defend the territory against a superior invader. Simulation results in Section 6.3 demonstrated that our designed control system was able to guide all three players to achieve their desired performance.
4. We designed instantaneous reward functions for the learning process of the players. Our simulation results demonstrated that our choice of reward functions met the performance requirements.

7.2 Future Work

The work presented in this thesis opens up new future research directions for multi-robot reinforcement learning in the game of guarding a territory. They are as follows.

1. In the future, one can add more players to the game to investigate the more general scenario of multiple invaders against multiple guards. It will be interesting to investigate how behaviors will emerge in the game. How can the team of guards effectively collaborate to defend the territory against the invaders.
2. The players in the current game had a constant speed. It will be an interesting problem to investigate the possibility of the players being able to change their

speed in the game. That is to incorporate acceleration and deceleration in the game.

3. The current game did not consider the maneuverability of the players. In the future, one may investigate how a team of guards can work jointly to defend the territory against an invader who can make sharp turns. In other words, the invader has more maneuverability than the guards.
4. The current game is played in the two-dimensional space. In the future, one may add an extra dimension to the game. In other words, study the game in three-dimensional space. This will be highly beneficial especially as the future looks towards flying drones for major security operations.
5. In the current game, there are no obstacle in the game environment. In the future, one may investigate how the players can find the best path around an obstacle to achieve their optimum performance.
6. In this thesis, the majority of the training was performed within a specific area. Therefore, future studies should attempt to increase the regions used for training as well as investigate further possibilities.

List of References

- [1] G. A. Bekey, *Autonomous robots: from biological inspiration to implementation and control*. Cambridge, Massachusetts: MIT Press, 2005.
- [2] G. Antonelli, *Underwater robots*. Springer, 2014, vol. 96.
- [3] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hahnel, D. Montemerlo, A. Morris, Z. Omohundro, and C. Reverte, “Autonomous exploration and mapping of abandoned mines,” *IEEE Robotics Automation Magazine*, vol. 11, no. 4, pp. 79–91, Dec 2004.
- [4] R. Abielmona, E. M. Petriu, M. Harb, and S. Wesolkowski, “Mission-driven robotic intelligent sensor agents for territorial security,” *IEEE Computational Intelligence Magazine*, vol. 6, no. 1, pp. 55–67, Feb 2011.
- [5] M. Tanas, W. Holubowicz, A. Adamczyk, and G. Taberski, “The TALOS project EU wide robotic border guard system,” in *Proceedings of the 16th International Conference on Methods and Models in Automation and Robotics (MMAR)*, Aug 2011, pp. 336–340.
- [6] M. Parent and G. Gallais, “Intelligent transportation in cities with CTS,” in *Proceedings of the 5th IEEE International Conference on Intelligent Transportation Systems*, 2002, pp. 826–830.
- [7] (2016) Knightscope inc. [Online]. Available: <http://www.knightscope.com/>.
- [8] R. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley and Sons, 1965.
- [9] H. M. Schwartz, *Multi-Agent Machine Learning: A Reinforcement Approach*. John Wiley and Sons, 2014.

- [10] L. Jouffe, “Actor-critic learning based on fuzzy inference system,” in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, 1996, pp. 339–344.
- [11] L.-X. Wang, *A course in fuzzy systems and control*. Upper Saddle River, New Jersey: Prentice Hall, 1997.
- [12] M. Andrecut and M. K. Ali, “Fuzzy reinforcement learning,” *International Journal of Modern Physics C*, vol. 13, no. 05, pp. 659–674, 2002.
- [13] E. Duman, M. Kaya, and E. Akin, *A multi-agent fuzzy-reinforcement learning method for continuous domains*. Berlin, Heidelberg: Springer, 2005, vol. 3690, pp. 306–315.
- [14] K. H. Hsia and J. G. Hsieh, “A first approach to fuzzy differential game problem: guarding a territory,” *Fuzzy Sets and Systems*, vol. 55, no. 2, pp. 157–167, 1993.
- [15] Y. S. Lee, K. H. Hsia, and J. G. Hsieh, “A strategy for a payoff-switching differential game based on fuzzy reasoning,” *Fuzzy Sets and Systems*, vol. 130, no. 2, pp. 237–251, 2002.
- [16] X. Lu, “Multi-agent reinforcement learning in games,” Ph.D. dissertation, Carleton University, Ottawa, 2012.
- [17] H. Raslan, “Reinforcement learning in differential games: A learning invader for the guarding a territory game,” Master’s thesis, Carleton University, Ottawa, 2015.
- [18] F. Bao-Fu, P. Qi-Shu, H. Bing-Rong, D. Lei, Z. Qiu-Bo, and Z. Zhaosheng, “Research on high speed evader vs. multi lower speed pursuers in multi pursuit-evasion games,” *Information Technology Journal*, vol. 11, no. 8, pp. 989–997, 2012.
- [19] S. Jin and Z. Qu, “Pursuit-evasion games with multi-pursuer vs. one fast evader,” in *Proceedings of the 8th World Congress on Intelligent Control and Automation (WCICA)*, 2010, pp. 3184–3189.
- [20] L. A. Petrosjan, *Differential games of pursuit*. World Scientific, 1993, vol. 2.
- [21] P. J. Nahin, *Chases and escapes: the mathematics of pursuit and evasion*. Princeton University Press, 2012.

- [22] E. L. Thorndike, "Animal intelligence: An experimental study of the associative processes in animals." *The Psychological Review: Monograph Supplements*, vol. 2, no. 4, 1898.
- [23] B. R. Hergenhahn, *An introduction to the history of psychology*. Wadsworth Publishing, 2009.
- [24] B. G. Galef, "Edward thorndike: Revolutionary psychologist, ambiguous biologist," *American Psychologist*, vol. 53, no. 10, pp. 1128–1134, 1998.
- [25] R. E. Bellman, *Dynamic Programming*. Princeton, New Jersey: Princeton University Press, 1957.
- [26] R. A. Howard, *Dynamic programming and Markov processes*. Cambridge, Massachusetts: MIT Press, 1960.
- [27] R. S. Sutton and A. G. Barto, *Reinforcement Learning : An Introduction*. Cambridge, Massachusetts: MIT Press, 1998.
- [28] M. van Otterlo and M. Wiering, *Reinforcement Learning and Markov Decision Processes*. Berlin, Heidelberg: Springer, 2012, vol. 12, pp. 3–42.
- [29] D. A. Berry and B. Fristedt, *Bandit problems: sequential allocation of experiments*. Chapman and Hall, 1985.
- [30] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, no. 1, pp. 237–285, May 1996.
- [31] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [32] P. Dayan, "The convergence of TD(λ) for general λ ," *Machine Learning*, vol. 8, no. 3, pp. 341–362, 1992.
- [33] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.
- [34] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, Sept 1983.

- [35] C. J. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, University of Cambridge, England, 1989.
- [36] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
- [37] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ: Prentice Hall Press, 2010.
- [38] S. Sathiya Keerthi and B. Ravindran, “A tutorial survey of reinforcement learning,” *Sadhana*, vol. 19, no. 6, pp. 851–889, 1994.
- [39] X. Xu, L. Zuo, and Z. Huang, “Reinforcement learning algorithms with function approximation: Recent advances and applications,” *Information Sciences*, vol. 261, pp. 1–31, 2014.
- [40] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
- [41] L. Jouffe, “Fuzzy inference system learning by reinforcement methods,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 28, no. 3, pp. 338–355, Aug 1998.
- [42] R. E. Bellman, *Adaptive control processes: A guided tour*. Princeton, New Jersey: Princeton University Press, 1961.
- [43] H. V. Hasselt and M. A. Wiering, “Reinforcement learning in continuous action spaces,” in *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, April 2007, pp. 272–279.
- [44] S. F. Desouky and H. M. Schwartz, “Self-learning fuzzy logic controllers for pursuit-evasion differential games,” *Robotics and Autonomous Systems*, vol. 59, no. 1, pp. 22 – 33, 2011.
- [45] M. D. Awgheda and H. M. Schwartz, “A fuzzy reinforcement learning algorithm with a prediction mechanism,” in *Proceedings of the 22nd Mediterranean Conference on Control and Automation (MED)*, 2014, pp. 593–598.
- [46] W. M. Buijtenen, G. Schram, R. Babuska, and H. B. Verbruggen, “Adaptive fuzzy control of satellite attitude by reinforcement learning,” *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 2, pp. 185–194, 1998.

- [47] G. A. Rummery and M. Niranjan, “On-line Q-learning using connectionist systems,” Cambridge University Engineering Department, CUED/F-INFENG/TR 166, September 1994.
- [48] Y. Duan, B. Cui, and H. Yang, *Robot Navigation Based on Fuzzy RL Algorithm*. Berlin, Heidelberg: Springer, 2008, pp. 391–399.
- [49] H. Benbrahim and J. A. Franklin, “Biped dynamic walking using reinforcement learning,” *Robotics and Autonomous Systems*, vol. 22, no. 3, pp. 283–302, 1997.
- [50] C. Gaskett, L. Fletcher, and A. Zelinsky, “Reinforcement learning for a vision based mobile robot,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, 2000, pp. 403–409.
- [51] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, “Reinforcement learning for robot soccer,” *Autonomous Robots*, vol. 27, no. 1, pp. 55–73, 2009.
- [52] R. Hafner and M. Riedmiller, “Reinforcement learning on an omnidirectional mobile robot,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, 2003, pp. 418–423.
- [53] V. Gullapalli, J. A. Franklin, and H. Benbrahim, “Acquiring robot skills via reinforcement learning,” *IEEE Control Systems*, vol. 14, no. 1, pp. 13–24, 1994.
- [54] Y. S. Lee, K. H. Hsia, and J. G. Hsieh, “A problem of guarding a territory with two invaders and two defenders,” in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, 1999, pp. 863–868.
- [55] M. L. Littman, “Value-function reinforcement learning in Markov games,” *Cognitive Systems Research*, vol. 2, no. 1, pp. 55–66, 2001.
- [56] A. Friedman, *Differential games*. Wiley, 1971.
- [57] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2009.
- [58] P. Y. Glorennec and L. Jouffe, “Fuzzy Q-learning,” in *Proceedings of the 6th IEEE International Conference on Fuzzy Systems*, vol. 2, Jul 1997, pp. 659–662.
- [59] L. Zadeh, “Fuzzy sets,” *Information and Control*, vol. 8, no. 3, pp. 338 – 353, 1965.

- [60] J. H. Park, D. H. Kim, and Y. J. Kim, "Anti-lock brake system control for buses based on fuzzy logic and a sliding-mode observer," *KSME International Journal*, vol. 15, no. 10, pp. 1398–1407, 2001.
- [61] Y. Yazicioglu and Y. S. Unlusoy, "A fuzzy logic controlled anti-lock braking system (ABS) for improved braking performance and directional stability," *International Journal of Vehicle Design*, vol. 48, no. 3-4, pp. 299–315, 2008.
- [62] J. Jamaludin, N. Rahim, and W. P. Hew, "Development of a self-tuning fuzzy logic controller for intelligent control of elevator systems," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 8, pp. 1167–1178, 2009.
- [63] C. Kim, K. A. Seong, H. Lee-Kwang, and J. O. Kim, "Design and implementation of a fuzzy elevator group control system," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 28, no. 3, pp. 277–287, 1998.
- [64] S. Sakaguchi, I. Sakai, and T. Haga, "Application of fuzzy logic to shift scheduling method for automatic transmission," in *Proceedings of the 2nd IEEE International Conference on Fuzzy Systems*, vol. 1, 1993, pp. 52–58.
- [65] P. Isomursu and T. Rauma, "A self-tuning fuzzy logic controller for temperature control of superheated steam," in *Proceedings of the 3rd IEEE Conference on Fuzzy Systems, IEEE Word Congress on Computational Intelligence*, vol. 3, Jun 1994, pp. 1560–1563.
- [66] F. O. Karray and C. W. De Silva, *Soft Computing and Intelligent Systems Design: Theory, Tools, and Applications*. Pearson, Addison Wesley, 2004.
- [67] J.-S. Jang and C.-T. Sun, *Neuro-fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.
- [68] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1 – 13, 1975.
- [69] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transactions Systems, Man and Cybernetics*, vol. 15, no. 1, pp. 116–132, 1985.
- [70] M. Sugeno and G. Kang, "Structure identification of fuzzy model," *Fuzzy Sets and Systems*, vol. 28, no. 1, pp. 15 – 33, 1988.

- [71] X. Dai, C.-K. Li, and A. B. Rad, “An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 3, pp. 285–293, Sept 2005.
- [72] V. Gullapalli and A. G. Barto, “Shaping as a method for accelerating reinforcement learning,” in *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, 1992, pp. 554–559.
- [73] J. Randløv and P. Alstrøm, “Learning to drive a bicycle using reinforcement learning and shaping,” in *Proceedings of the 15th International Conference on Machine Learning*, 1998, pp. 463–471.