**Learning and Design of Fuzzy Logic Controllers For Pursuit-Evasion
Differential Games**

by

Sameh Farahat Ibrahim Desouky, M.Sc

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

July 2010

© Copyright 2010, Sameh Farahat Ibrahim Desouky

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-70547-6
Our file *Notre référence*
ISBN: 978-0-494-70547-6

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

■ ■ ■
Canada

Abstract

This thesis presents different learning techniques for robotic applications. We show how to automatically and adaptively tune the input and the output parameters of a fuzzy logic controller in different situations using different types of learning. We focus on three robotic applications; the wall-following case, the single pursuit-evasion case and the multiple pursuit-evasion case.

In this thesis, we focus on two learning types. The first type is supervised learning in which we need input/output data or an expert to learn from. Under this type, we use genetic algorithms and gradient-based techniques to tune the input and the output parameters of fuzzy logic controllers. We propose a new technique that uses input/output training data obtained from a PD controller "an expert" to iteratively tune the input and the output parameters of fuzzy logic controllers using genetic algorithms. The number of rules in the rule-base is also reduced. The proposed technique is applied to a wall-following mobile robot. We also propose another technique that uses input/output training data obtained from a PD controller to iteratively tune the input and the output parameters of fuzzy logic controllers using gradient-based techniques. The proposed technique is applied to a pursuit-evasion game.

The second learning type is reward-based learning in which we do not have an output data set or an expert to learn from. Instead, we need a reward or a punishment function for evaluating the learning model. Under this type, we use genetic algorithms

and reinforcement learning methods to tune the input and the output parameters of fuzzy logic controllers. We propose a new technique that combines genetic algorithms with reinforcement learning methods to tune the input and the output parameters of fuzzy logic controllers. The proposed technique is applied to different pursuit-evasion differential games starting from a single pursuit-evasion case in which only the pursuer self-learns its control strategy and ending with decentralized learning in which n pursuers and n evaders with different capabilities can self-learn their control strategies. We also propose a new technique that uses reinforcement learning methods to tune the input and the output parameters of fuzzy logic controllers. We also present a convergence proof for the model-building Q-learning algorithm with eligibility traces.

Contents

Abstract	iii
1 Introduction	1
1.1 Overview	1
1.2 The Main Problems and Objective	2
1.3 Thesis Organization	5
1.4 Literature Overview	7
1.4.1 Genetic algorithms	8
1.4.2 Adaptive network fuzzy inference system	9
1.4.3 Reinforcement learning	10
1.4.4 Differential games	11
1.5 Thesis Contributions	12
2 Survey on Different Learning Algorithms	15
2.1 Introduction	15
2.2 Fuzzy Logic Controller	15
2.2.1 Membership function	16
2.2.2 Fuzzy inference system	18
2.2.3 Defuzzification	19
2.2.4 Scaling factor	20

2.3	Genetic Algorithms	21
2.4	Adaptive Network Fuzzy Inference System	22
2.4.1	Mathematical description	23
2.4.2	Hybrid learning rule	28
2.5	Reinforcement Learning	31
2.5.1	Markov decision process problem	32
2.5.2	Q(λ)-learning	35
2.5.3	Actor-critic learning	37
2.5.4	Action selection	39
2.6	Summary	40
3	Iterative Genetic Based Fuzzy Logic Controller	41
3.1	Introduction	41
3.2	Model Structure and Control	42
3.2.1	Wall-following model	42
3.2.2	PD controller design	43
3.2.3	Fuzzy logic controller design	44
3.3	Iterative Genetic Based Fuzzy Logic Controller	45
3.3.1	Coding the parameters of a fuzzy logic controller using GAs	47
3.4	Computer Simulation	49
3.5	Results	51
3.6	Summary	57
4	Iterative Adaptive Network Fuzzy Inference System	59
4.1	Introduction	59
4.2	Model Structure and Control	60

4.2.1	Pursuit-evasion model	60
4.2.2	PD controller design	63
4.2.3	Fuzzy logic controller design	63
4.3	Iterative Adaptive Network Fuzzy Inference System	65
4.3.1	Tuning the parameters of a fuzzy logic controller using ANFIS	67
4.4	Computer Simulation	67
4.5	Results	69
4.6	Summary	72
5	On the Convergence of Model-building Q-learning with Eligibility	
	Traces	73
5.1	Introduction	73
5.2	Model-building Q-learning with Eligibility Traces	75
5.3	Boundedness Proof	78
5.4	Convergence Proof	81
5.5	Summary	87
6	Q(λ)-Learning Based Genetic Fuzzy Logic Controller	88
6.1	Introduction	88
6.2	Problem Statement	89
6.3	Pursuit-Evasion Model	91
6.4	Q(λ)-learning Based Genetic Fuzzy Logic Controller	93
6.4.1	Fuzzy logic controller design	93
6.4.2	Learning in phase 1	95
6.4.3	Learning in phase 2	97
6.4.4	Reward-based genetic algorithm	100

6.5	Computer Simulation and Results	101
6.5.1	Learning time	102
6.5.2	Evader using a simple control strategy	104
6.5.3	Evader using the advantage of higher maneuverability	111
6.5.4	Learning in a multi-robot system	113
6.6	Summary	118
7	Q(λ)-Learning Fuzzy Inference System	120
7.1	Introduction	120
7.2	Q(λ)-learning Fuzzy Inference System	122
7.3	Computer Simulation and Results	127
7.3.1	Evader using a simple control strategy	127
7.3.2	Evader using the advantage of higher maneuverability	131
7.3.3	Learning in a multi-robot system	133
7.4	Summary	136
8	Learning in Multi-Player Pursuit-Evasion Games	140
8.1	Introduction	140
8.2	Multi-Robot Reinforcement Learning	141
8.3	Minimax Q(λ)-learning Compared to Independent Q(λ)-learning . . .	143
8.4	Learning in Two-Pursuer One-Evader Game	151
8.4.1	Computer simulation and results	155
8.4.2	Pursuers have more information about the evader	160
8.5	Learning in n -Pursuer n -Evader Games	165
8.5.1	Building the Q-tables	166
8.5.2	Constructing the reward function	167

8.5.3	Computer simulation and results	169
8.6	Summary	174
9	Conclusion, Contributions and Publications	176
9.1	Conclusion	176
9.2	Thesis Contributions	178
9.3	Publications	179
	References	181

List of Figures

2.1	Fuzzy design methodology.	16
2.2	Trapezoidal MF.	17
2.3	Gaussian MF.	18
2.4	Crossover operation.	21
2.5	Mutation operation.	22
2.6	Structure of a FIS used in ANFIS Model.	23
2.7	Agent-environment interaction in RL.	31
2.8	Replacing trace.	36
2.9	Actor-critic structure.	38
3.1	Wall-following model.	42
3.2	Block diagram of a PD controller system.	43
3.3	Block diagram of a FLC system.	44
3.4	The proposed IGBFLC.	46
3.5	Trapezoidal MF with modified coefficients.	49
3.6	Unit step response of the PD controller for different values of K_p and K_d	50
3.7	Average fitness value for different combinations of the iteration numbers and the population sizes.	52

3.8	Error resulting from the PD controller (solid line) compared to error resulting from the FLC obtained from stage 1 (dotted line).	52
3.9	Error resulting from the PD controller (solid line) compared to error resulting from the PD controller parallel with the FLC obtained from stage 1 (dotted line).	53
3.10	Error resulting from the PD controller (solid line) compared to error resulting from the FLC obtained from iteration 1 of stage 2 (dotted line).	54
3.11	Error resulting from the PD controller (solid line) compared to error resulting from the PD controller parallel with the FLC obtained from iteration 1 of stage 2 (dotted line).	54
3.12	Error resulting from the PD controller (solid line) compared to error resulting from the FLC obtained from iteration 2 of stage 2 (dotted line).	55
3.13	Error resulting from the PD controller (solid line) compared to error resulting from the PD controller parallel with the FLC obtained from iteration 2 of stage 2 (dotted line).	55
3.14	Path of the mobile robot to follow a wall using the PD controller (solid line) compared to the IGBFLC (dotted line).	56
4.1	Pursuit-evasion model.	61
4.2	Block diagram of a PD controller system.	63
4.3	Block diagram of a FLC system.	64
4.4	The proposed IANFIS.	65
4.5	Unit step response of the PD controller for different values of K_p and K_d	68
4.6	Tuned MFs for the inputs using the proposed IANFIS.	70

4.7	Tuned MFs for the inputs using the proposed IGBFLC.	70
6.1	Pursuit-evasion model.	92
6.2	The proposed QLBGFLC technique.	94
6.3	Block diagram of a FLC system.	95
6.4	A FLC coded into a chromosome.	99
6.5	Crossover process in GAs.	99
6.6	Mean values of the average capture times for different numbers of steps.	106
6.7	Average learning time for different numbers of steps.	106
6.8	Mean values of the average capture times for different numbers of episodes.	107
6.9	Average learning time for different numbers of episodes.	108
6.10	MFs for the inputs after tuning using the proposed QLBGFLC. . . .	110
6.11	Paths of the pursuer (ooo) and the evader (+++) using the proposed QLBGFLC.	114
6.12	MFs for the inputs of the pursuer after tuning using the proposed QLBGFLC.	115
6.13	MFs for the inputs of the evader after tuning using the proposed QL- BGFLC.	116
6.14	Paths of the pursuer and the evader (solid line) using the $Q(\lambda)$ -learning only compared to the classical control strategies (dotted line). . . .	117
6.15	Paths of the pursuer and the evader (solid line) using the reward-based GA compared to the classical control strategies (dotted line). . . .	117
6.16	Paths of the pursuer and the evader (solid line) using the proposed QLBGFLC compared to the classical control strategies (dotted line).	118

7.1	The proposed QLFIS technique.	122
7.2	Initial MFs of the FLC inputs.	128
7.3	MFs for the inputs after tuning using the proposed QLFIS.	131
7.4	Paths of the pursuer (ooo) and the evader (+++) using the proposed QLFIS.	133
7.5	MFs for the inputs of the pursuer after tuning using the proposed QLFIS.	134
7.6	MFs for the inputs of the evader after tuning using the proposed QLFIS.	135
7.7	Paths of the pursuer and the evader (solid line) using the $Q(\lambda)$ -learning only compared to the classical control strategies (dotted line).	136
7.8	Paths of the pursuer and the evader (solid line) using the reward-based GA compared to the classical control strategies (dotted line).	136
7.9	Paths of the pursuer and the evader (solid line) using the technique proposed by Dai et al. [1] compared to the classical control strategies (dotted line).	137
7.10	Paths of the pursuer and the evader (solid line) using the proposed QLBGFLC compared to the classical control strategies (dotted line).	137
7.11	Paths of the pursuer and the evader (solid line) using the proposed QLFIS compared to the classical control strategies (dotted line).	138
8.1	Average capture time using the IQL algorithm (solid line) compared to the minimax $Q(\lambda)$ -learning algorithm (dotted line).	150
8.2	Average total reward using the IQL algorithm (solid line) compared to the minimax $Q(\lambda)$ -learning algorithm (dotted line).	150
8.3	Two-Pursuer one-evader model.	152

8.4	Paths of the pursuers and the evader after learning. The initial position of the evader is (0,0) and the pursuers starts at different initial positions. The capture time for each game is also shown.	159
8.5	Two-pursuer one-evader model with more information.	160
8.6	Description of the angles α and β	161
8.7	Paths of the pursuers and the evader after learning when the pursuers get more information about the evader. The initial position of the evader is (0,0) and the pursuers starts at different initial positions. The capture time for each game is also shown.	164
8.8	Paths of the pursuers and the evaders after learning for $n = 2$ compared with the classical control strategies for the single pursuit-evasion game.	171
8.9	Paths of the pursuers and the evaders after learning when the wrong coupling is chosen for $n = 2$	172
8.10	Paths of the pursuers and the evaders after learning for $n = 3$ compared with the classical control strategies for the single pursuit-evasion game.	174

List of Tables

2.1	Fuzzy decision table.	25
2.2	Learning procedures for the hybrid rule.	28
3.1	Initial Fuzzy decision table.	45
3.2	Tuned MF parameters for the input e	56
3.3	Tuned MF parameters for the input $\dot{e}$	57
3.4	Tuned MF parameters for the output u	57
3.5	Scaling factors.	57
3.6	Fuzzy decision table after tuning.	57
4.1	Fuzzy decision table.	64
4.2	Initial fuzzy decision table.	69
4.3	Values of GA parameters.	69
4.4	Fuzzy decision table after tuning using the proposed IANFIS.	70
4.5	Fuzzy decision table after tuning using the proposed IGBFLC.	71
4.6	Capture time, in seconds, using the different techniques.	71
4.7	Capture time, in seconds, using different initial positions for the evader.	72
6.1	Values of GAs parameters.	109
6.2	Fuzzy decision table after tuning using the proposed QLBGFLC.	110

6.3	Capture time, in seconds, for different evader initial positions and learning time, in seconds, for the different techniques.	111
6.4	Values of GAs parameters.	113
6.5	Fuzzy decision table for the pursuer after tuning using the proposed QLBGFLC.	115
6.6	Fuzzy decision table for the evader after tuning using the proposed QLBGFLC.	116
6.7	Learning time, in seconds, for the different techniques.	118
7.1	Initial fuzzy decision table.	129
7.2	Fuzzy decision table after tuning using the proposed QLFIS.	131
7.3	Capture time, in seconds, for different evader initial positions and learning time, in seconds, for the different techniques.	132
7.4	Fuzzy decision table for the pursuer after tuning using the proposed QLFIS.	134
7.5	Fuzzy decision table for the evader after tuning using the proposed QLFIS.	135
7.6	Learning time, in seconds, for the different techniques.	138
8.1	Values of GAs parameters.	158
8.2	Discretized actions ($n = 2$).	166
8.3	Discretized actions ($n = 3$).	167
8.4	Dimensions of the different Q-tables.	167
8.5	All possible coupling choices.	168
8.6	Parameters of mobile robots ($n = 2$).	170
8.7	Parameters of $Q(\lambda)$ -learning ($n = 2$).	170

8.8	Values of GAs parameters ($n = 2$).	170
8.9	The state-action values in Q_{CF} ($n = 2$).	170
8.10	Parameters of mobile robots ($n = 3$).	172
8.11	Parameters of $Q(\lambda)$ -learning ($n = 3$).	172
8.12	Values of GAs parameters ($n = 3$).	173
8.13	The state-action values in Q_{CF} ($n = 3$).	173
8.14	The capture times of all possible coupling for $n = 3$	174

Nomenclature

ANFIS	adaptive network fuzzy inference system.
BP	back-propagation.
COA	center of area defuzzification method.
DP	dynamic programming.
FLC	fuzzy logic controller.
GA	genetic algorithm.
GD	gradient descent approach.
IANFIS	iterative adaptive network fuzzy inference system.
IGBFLC	iterative genetic based fuzzy logic controller.
IQL	independent $Q(\lambda)$ -learning.
LoS	line-of-sight.
LSE	least square error.
MARL	multi-agent reinforcement learning.
MAS	multi-agent system.
MDP	Markov decision process.
MG	Markov game.

MRS	multi-robot system.
MSE	mean square error.
NN	neural network.
QLBGFLC	$Q(\lambda)$ -learning based fuzzy logic controller.
QLFIS	$Q(\lambda)$ -learning fuzzy inference system.
RL	reinforcement learning.
SG	stochastic game.
TD	temporal difference.
TSK FIS	Takagi-Sugeno-Kang fuzzy inference system.

Chapter 1

Introduction

1.1 Overview

In recent years, the concept of the fuzzy control, as one type of the intelligent control, is being widely used especially in robotic applications. The reason is that the classical control requires a mathematical model and needs a mathematical analysis for the controller design. Also, the inaccuracy of the mathematical model of the controller degrades the performance of that controller. In addition, the complexity of the environment makes it difficult to a priori design a good controller. On the other hand, fuzzy logic controllers can deal with uncertainty, imprecision, imperfect description and environment changes. To design an intelligent robot, one cannot use preprogrammed robots due to the complexity of the environment. The robot must learn and adapt its behavior by interacting with the changing environment to improve its performance.

Learning and control become two important and challenging problems in robotic systems. In this thesis, we constrained ourselves to learning the input and the output

parameters of fuzzy logic controllers. In addition, we focus on the wall-following mobile robot system, the single pursuit-evasion differential game and the multiple pursuit-evasion differential game.

In a pursuit-evasion game, one robot (a pursuer) is pursuing another robot (an evader). The pursuer wants to catch the evader in minimum time while the evader tries to escape from the pursuer. We are interested in the pursuit-evasion differential game because it can be considered as a general problem for many other robotic problems such as obstacle avoidance, leader-follower, wall-following and path planning. Also, the optimal solution for a simple version of the pursuit-evasion game is known. Therefore, we can compare the performance of the different learning techniques proposed in the thesis to the known optimal solutions. In addition, most of the work done for the pursuit-evasion differential game in the literature (as we will see in Section 1.4.4) did not use any type of learning except from the work done by the author himself or from the work done by his colleagues.

1.2 The Main Problems and Objective

In robotic systems, a robot must learn and adapt its behavior based on the changing environment. In our work, we assume first that we have an expert to learn from. When an expert is available or input/output training data can be easily obtained, then, supervised learning techniques are quite enough to be used. However, the problem in using supervised learning techniques is that the performance of the learned fuzzy logic controller depends on the performance of the expert.

To solve this problem, we propose a new technique in which a supervised learning technique is used iteratively until we get a learned fuzzy logic controller that can

outperform the expert itself. In addition, the proposed technique is also used to reduce the number of rules in the rule base of the fuzzy logic controller. That results in a simpler controller design. The proposed technique is implemented using two different learning techniques; genetic algorithms (in Chapter 3) and gradient-based techniques (in Chapter 4). The proposed technique is also tested using two different applications; the wall-following mobile robot (in Chapter 3) and the pursuit-evasion game (in Chapter 4).

In the previous paragraph, we assumed that we have an expert to learn from. Now, we assume the case in which the expert is not available or the input/output training data is hard or expensive to obtain. In this case, supervised learning techniques cannot be used. Therefore, we think of reward-based techniques that do not need any a priori knowledge about the output data. Instead, a reward/punishment function needs to be defined. In this thesis, we are interested in two different reward-based techniques which are reward-based genetic algorithms and reinforcement learning.

Genetic algorithms search directly for the desired controller without the need to estimate a value function that is responsible for mapping from the state space to the action space. Although a genetic algorithm, as a powerful optimization technique, can find the desired controller, it takes a long time searching for that controller. On the other hand, reinforcement learning methods estimate first the value function and accordingly can estimate the desired controller. Although a reinforcement learning method can estimate the value function in a short time, the estimated value function is not enough to get the desired controller from.

To solve the long searching time problem of genetic algorithms and the low performance problem of reinforcement learning methods, we propose a new technique that combines genetic algorithms with reinforcement learning methods to increase

the performance of the learned controller and to reduce the searching time in the learning process. The proposed technique is used to automatically tune the input and the output parameters of fuzzy logic controllers online through interaction with the changing environment. The proposed technique is applied to different versions of pursuit-evasion differential games (see Chapter 6 and Chapter 8). As reinforcement learning methods are used in the discrete space, the proposed technique is also used in the discrete space. To use the proposed technique for continuous spaces (as in our case), one should first discretize these spaces.

Another problem that we face is the need to learn in the continuous space directly, without discretizing the continuous state and action spaces. A continuous space means that we have an infinite number of states and an infinite number of actions. As a result, a problem known as the curse of dimensionality will appear. To solve this problem, we propose a new technique that makes use of function approximations with reinforcement learning methods. The proposed technique is applied to different versions of pursuit-evasion differential games (see Chapter 7).

When we upgrade from single pursuit-evasion games into multiple pursuit-evasion games, we face some problems such as communication complexity and the coordination problem. To avoid communication complexity, we use independent learning in which each robot learns its control strategy without sharing any information (e.g. rewards, states and actions) with the other robots in the system. When we use independent learning in our proposed techniques, the performance of the learned fuzzy logic controllers approaches the performance of optimal solutions (see Chapter 8).

To solve the coordination problem, we propose a decentralized approach to learn coordination between robots. Therefore, the multiple pursuit-evasion game can be decomposed into single pursuit-evasion games (see Chapter 8).

Finally, we can state that our objective in this thesis is to design fuzzy logic controllers, for robotic applications, that can automatically and adaptively learn their parameters in different situations. These fuzzy controllers should be robust enough to withstand the difficulties in the environment.

1.3 Thesis Organization

This thesis research is organized as follows:

- **Chapter 2** presents some basic terminologies for fuzzy logic controller, supervised learning algorithms (such as genetic algorithms and adaptive network fuzzy inference systems) and reinforcement learning methods (such as Q-learning).
- **Chapter 3** presents a novel learning technique that combines fuzzy logic controllers with genetic algorithms. The proposed technique is called *iterative genetic based fuzzy logic controller* (IGBFLC). The proposed IGBFLC is based on supervised learning therefore, we make use of a PD controller as an expert to collect data examples for the learning process. The proposed IGBFLC is applied to a wall-following mobile robot. Computer simulation and results show that the performance of the proposed technique outperforms the performance of the PD controller itself.
- **Chapter 4** generalizes the idea proposed in Chapter 3 by presenting it in a different learning technique that combines fuzzy logic controller with gradient-based techniques. The proposed technique is called *iterative adaptive network fuzzy inference system* (IANFIS) which is applied to a pursuit-evasion game.

Computer simulation and results show also that the performance of the proposed technique outperforms the performance of the PD controller itself.

- **Chapter 5** presents a convergence proof for the model-building Q-learning algorithm with eligibility traces.
- **Chapter 6** presents a novel learning technique that combines fuzzy logic controllers and genetic algorithms with $Q(\lambda)$ -learning. The proposed technique is called a *$Q(\lambda)$ -learning based genetic fuzzy logic controller (QLBGFLC)*. The proposed QLBGFLC is based on reinforcement learning therefore, we do not need a data set or an expert to learn from. We only need to construct a reward function that is used instead. The proposed QLBGFLC is applied to different versions of pursuit-evasion games. The proposed technique is used for discrete spaces. Computer simulation and results show that the proposed technique outperforms all the compared techniques.
- **Chapter 7** presents a novel learning technique that combines fuzzy logic controllers with $Q(\lambda)$ -learning. The proposed technique is called a *$Q(\lambda)$ -learning fuzzy inference system (QLFIS)*. As in Chapter 6, the proposed QLFIS is based on reinforcement learning. The proposed QLFIS is applied to different versions of pursuit-evasion games. The proposed QLFIS is used for the continuous domains. Computer simulation and results show that the proposed technique outperforms all the compared techniques in performance and learning time.
- **Chapter 8** presents a comparison between two different $Q(\lambda)$ -learning algorithms which are the minimax Q-learning and the independent Q-learning. This chapter also presents learning in a two-pursuer one-evader game. We show that

both the pursuers and the evader can simultaneously self-learn these control strategies using the proposed QLBGFLC. We also show that as the pursuers know more information about the evader, they can learn better control strategies. Finally, an extension from learning in a single pursuit-evasion game to learning in a multiple pursuit-evasion game is introduced. In the multiple pursuit-evasion game, all the robots are heterogeneous (i.e. the robots have different velocities and different steering angles).

- **Chapter 9** summarizes the ideas introduced in this thesis and mentions the resulting publications.

1.4 Literature Overview

Fuzzy logic controllers (FLCs) are currently being used in engineering applications [2, 3] especially for plants that are complex and ill-defined [4, 5] and plants with high uncertainty in the knowledge about its environment such as autonomous mobile robotic systems [6, 7]. However, FLC has a drawback of finding its knowledge base which is based on a tedious and unreliable trial and error process. To overcome this drawback one can use supervised learning. Supervised learning needs a teacher or input/output training data. The meaning of supervised learning is that the desired "target" output values are given, from an expert, for each given input pattern.

In this thesis, we examine two supervised learning techniques. They are genetic algorithms and adaptive networks. However, in many practical cases the model is totally or partially unknown and it is difficult or expensive and in some cases impossible to get training data. In such cases it is preferable to use reinforcement learning.

1.4.1 Genetic algorithms

Genetic algorithms (GAs) were first proposed by Holland [8] in 1973. Applications of GAs have been studied by many researchers since early 1990s. GAs have been used to overcome the difficulty and the complexity in tuning the parameters of FLCs such as membership functions, scaling factors and control rules [9].

Wang [10], Herrera et al. [11] and Lekova et al. [12] generated fuzzy rules from given desired input/output data pairs. The number of the generated rules is the same as the number of the given data pairs so they assigned a degree to each generated rule to eliminate the redundant rules. They used GAs so as to make the FLC behaves as closely as possible to the expert but not better. Magdalena and Monasterio [13] and Magdalena and Monasterio [14] tuned the normalization limits of the variables and modified the rule bases without reduction. Linkens and Nyongesa [15] derived a GA scheme that produced an acceptable set of control rules without any reduction. Chin and Qi [16] used a GA to tune the output parameters of a FLC and to reduce the number of rules which were reduced by 23%. Kubota et al. [17] proposed a perception-based genetic algorithm (PerGA) technique to tune the output parameters of a FLC that is applied to an obstacle avoidance problem. Ming et al. [18] used a GA to tune the parameters of the input membership functions for a FLC that is applied to an obstacle avoidance problem. Wu et al. [19] generated fuzzy rules, without reduction and tuned the parameters of the membership functions using a fuzzy genetic map (FGM) to prevent the condition that causes too many binary digits for encoding the membership chromosome. de Abreu and Ribeiro [20] and Desouky [21] tuned the parameters of Gaussian membership functions and the parameters of fuzzy rules using GAs. Chiou and Lan [22] used sequential learning for both rules and membership functions. They tuned rules and used them to tune membership functions and vice

versa. The learned FLC is applied to a leader-follower problem. Mucientes et al. [23] and Mucientes et al. [7] used iterative rule learning (IRL) technique to tune the parameters of the input and the output membership functions. The learned FLC is applied to a wall-following problem.

1.4.2 Adaptive network fuzzy inference system

An Adaptive network fuzzy inference system (ANFIS) is a gradient-based technique that was originally introduced by Jang [24]. It is a combination of fuzzy inference systems and adaptive networks to tune the input and/or the output parameters of fuzzy logic controllers from data samples. Next, we will show why several authors from different fields prefer to use the ANFIS model.

In the electrochemical field, Entchev and Yang [25] used ANFIS on a solid oxide fuel cell system (SOFC). Their work shows that the ANFIS model is able to predict and optimize system performance faster than using conventional models and the system is modeled with high accuracy. Wu et al. [26] show that it is feasible to setup a model of a SOFC stack based on ANFIS and hence avoiding the complicated differential equations that are used to describe the stack. In chemistry, Nucci et al. [27] compared ANFIS with multi-layer perceptrons and the results show that ANFIS has better performance to infer cell concentrations. In electronics, Daldaban et al. [3] used ANFIS for estimating the phase inductance of switched reluctance motors. The results show that ANFIS is in agreement with the finite element method (FEM). The benefit from using ANFIS here is that the FEM method is mathematically complex and requires a very large number of computations. That makes ANFIS a very powerful approach to use instead. In aerospace, Güney [28] used ANFIS for tracking a single manoeuvring target in the cluttered environment. The work done shows that

ANFIS speeds up the target tracking and has better accuracy when compared with conventional algorithms such as the nearest neighbor Kalman filter and the probabilistic data association filter. In addition, conventional algorithms have complex iterative processes.

From the previous paragraph we can see that ANFIS has several advantages. ANFIS can simply tune the input and the output parameters of FLCs effectively. ANFIS is used to avoid facing complex mathematical calculations. These advantages make us think of using ANFIS in this thesis.

1.4.3 Reinforcement learning

Reinforcement learning (RL) is a computational approach to learning through interaction with the environment [29, 30]. The main advantages of RL is that it does not need either an expert or a training data set (i.e. prior knowledge about the environment). All that it needs is rewards/punishments for the agent to learn. RL is suitable for intelligent robot control [31] especially in the fields of autonomous mobile robots [32–38] and game playing [39].

Tesauro [39] applied RL to learning the backgammon game. It is shown that the model could learn without any a priori knowledge. Crites and Barto [40] applied RL to solve the problem of elevator scheduling. Their results show the power of RL on this very large scale stochastic dynamic problem. Benbrahim [41] and Benbrahim and Franklin [42] developed a new RL algorithm to solve complex control problems. They applied this algorithm to a biped robot which learned dynamic walking without any previous knowledge about its dynamic model. Nie and Haykin [43] used RL to solve the problem of dynamic channel assignment by interacting with the mobile communication environment. Stone and Sutton [44] applied RL to real and simulated

robots which try to learn how to play soccer.

Limited studies have applied RL alone but its use with other learning algorithms has increased. Dai et al. [1] used an RL approach to tune the parameters of a FLC. This approach is applied to a single case of one robot following another along a straight line. Ye et al. [45] and Er and Deng [46] proposed a hybrid learning approach that combines a neuro-fuzzy system with RL in a two-phase structure applied to an obstacle avoidance mobile robot. In phase 1, supervised learning is used to tune the parameters of a FLC then in phase 2, RL is employed so that the system can re-adapt to a new environment. The limitation in their approach is that if the training data are hard or expensive to obtain then supervised learning can not be applied. Xiao et al. [47] overcame this limitation by using Q-learning as an expert to obtain training data. Then the training data are used to tune the weights of an artificial neural network controller that is applied to a mobile robot path planning problem.

1.4.4 Differential games

Modern game theory was first introduced by Neuman and Morgenstern [48] in 1947. In their book, the focus is on economics. Then in 1950's game theory and optimal control theory were combined to form what is called differential games [49]. A differential game is a game that is played in continuous time with continuous state and action spaces. The pursuit-evasion differential game is one application of differential games in which a pursuer tries to catch an evader in minimum time whereas the evader tries to escape from the pursuer. As we mentioned before, the pursuit-evasion game can be considered as a general application for many other applications such as obstacle avoidance, path planning, leader-follower and wall-following.

Vidal et al. [50] investigated a multi-robot pursuit-evasion game. The model consists of a combination of aerial and ground vehicles. However, the unmanned vehicles are not learning. They just do the actions they received from a central computer system. Li et al. [51] propose a hierarchical software structure to multi-player pursuit-evasion game. However, the robots do not learn and do not adapt for a changing environment. Ishiwaka et al. [52] discuss the use of reinforcement learning in the multi-agent pursuit problem. Some ideas are proposed on a relatively simple pursuit game using identical agents. The individual agents simply learn a particular pursuit strategy but do not adapt to their own environment. In addition, the authors do not use a realistic robot model or robot control structure. Givigi et al. [53] used RL to tune the output parameters of a FLC in a pursuit-evasion game.

To the best of our knowledge, most of the work done in the pursuit-evasion differential game in the literature did not use any type of learning except from the work done by the author himself or from the work done by his colleagues in [53].

1.5 Thesis Contributions

The work done in this thesis introduces several contributions:

1. A novel technique is proposed to combine fuzzy logic controllers with genetic algorithms that we called an *iterative genetic based fuzzy logic controller* (IG-BFLC).
2. We also use the same idea of this technique to combine fuzzy inference systems with gradient-based techniques resulting in an *iterative adaptive network fuzzy inference system* (IANFIS).

3. A proof of convergence for the model-building Q-learning algorithm with eligibility traces is presented.
4. A novel technique is proposed to combine fuzzy logic controllers and genetic algorithms with reinforcement learning methods that we called a *$Q(\lambda)$ -learning based genetic fuzzy logic controller (QLBGFLC)*.
5. A novel technique is proposed to combine fuzzy logic controllers with $Q(\lambda)$ -learning that we called a *$Q(\lambda)$ -learning fuzzy inference system (QLFIS)*.
6. We apply the techniques proposed in this thesis to different robotic problems such as wall-following applications and different pursuit-evasion games.
7. We investigate a number of different scenarios and approaches for the pursuit-evasion game. We introduce learning in different pursuit-evasion games with pursuers and evaders that have different capabilities (heterogeneous robots). The idea of individual robots adapting their behavior based on their capabilities is new to the literature (see Section 6.5.3 and Section 7.3.2).
8. We solve the curse of dimensionality problem (large state and action spaces) using the proposed QLBGFLC and the proposed QLFIS (see Section 6.4, Section 7.2 and Section 8.4.1).
9. We reduce the communication complexity by using independent learning techniques in combination with the proposed QLBGFLC technique. By using independent learning, the robots do not need to share information such as rewards, states and actions between each other (see Section 8.4.1 and Section 6.5.4, Section 7.3.3, Section 8.4.1 and Section 8.5).

-
10. A decentralized learning approach is proposed for learning in the n -pursuer n -evader game. In this approach, the robots can successfully learn coordination between each other.

Chapter 2

Survey on Different Learning Algorithms

2.1 Introduction

In this chapter we introduce some basic terminologies about fuzzy logic controller in Section 2.2. We illustrate the different learning algorithms that we use in this thesis such as genetic algorithms in Section 2.3, adaptive network fuzzy inference system in Section 2.4 and reinforcement learning in Section 2.5.

2.2 Fuzzy Logic Controller

A typical design methodology for a fuzzy logic controller (FLC) usually follows the iterative steps shown in Figure 2.1 [15]. The first step in the FLC design procedure is to analyze the model dynamics. The second step is to define and indicate the boundaries of the fuzzy universe of discourse and the number of membership functions

within it. The third step is to decide a fuzzification and a defuzzification strategies to map a real data set into the fuzzy domain and vice versa. In the fourth and final step, the control rules are derived. In the next subsections we will define the different types of membership functions, the well-known types of fuzzy inference system, the popular defuzzification methods used and the scaling factors.

Figure 2.1: Fuzzy design methodology.

2.2.1 Membership function

Several types of membership function (MF) are defined with different number of coefficients. Gaussian MF has two coefficients, triangular MF has three and trapezoidal

MF has four coefficients. Both trapezoidal and gaussian MFs have been used extensively especially in real-time implementation and control. The trapezoidal MF depicted in Figure 2.2 is linear but not smooth and is defined as

$$\mu(x) = \begin{cases} 0 & : x < a \\ \frac{x-a}{b-a} & : a \leq x < b \\ 1 & : b \leq x \leq c \\ \frac{x-c}{d-c} & : c < x \leq d \\ 0 & : x > d \end{cases} \quad (2.1)$$

where a , b , c and d are its parameters. The gaussian MF depicted in Figure 2.3 is not linear but smooth and is defined as

$$\mu(x) = \exp\left(-\frac{1}{2}\left(\frac{x-m}{\sigma}\right)^2\right) \quad (2.2)$$

Figure 2.2: Trapezoidal MF.

Figure 2.3: Gaussian MF.

where σ and m are the standard deviation and the mean, respectively.

2.2.2 Fuzzy inference system

The most widely used fuzzy inference systems (FISs) are the Mamdani FIS [54] and the Takagi-Sugeno-Kang FIS (TSK FIS) [55]. Next, we will describe the mechanisms of these types.

Mamdani FIS

The Mamdani FIS consists of the following linguistic rules that describe a mapping from fuzzy sets in the input to fuzzy sets in the output

$$R_l : IF \ x_1 \text{ is } A_1^l \text{ AND } \dots \text{ AND } x_J \text{ is } A_J^l \text{ THEN } y_l \text{ is } C_l, \ w_l = q_l \quad (2.3)$$

where x_j is the j^{th} input variable ($j = 1, 2, \dots, J$), J is the number of input variables, y_l is the output variable of rule R_l ($l = 1, 2, \dots, L$), L is the number of rules, A_j^l and C_l

are fuzzy sets for x_j and y_l respectively, w_l is the weight of rule R_l and $q_l \in [0, 1]$. The contribution of rule R_l to a Mamdani model's output is a fuzzy set whose membership value, $\mu_{C'_l}$, is computed by

$$\mu_{C'_l}(y) = (\mu_{A_1^l} \wedge \mu_{A_2^l} \wedge \cdots \wedge \mu_{A_J^l}) \wedge \mu_{C_l}(y) \quad (2.4)$$

where $\mu_{A_j^l}$ is the membership value of the input variable x_j in rule R_l and μ_{C_l} is the membership value of the output variable y in rule R_l . The output fuzzy set of the model is the aggregation of the outputs from all rules using the maximum operator as follows

$$\mu_C(y) = \max\{\mu_{C'_1}(y), \mu_{C'_2}(y), \dots, \mu_{C'_L}(y)\} \quad (2.5)$$

TSK FIS

Rules that are used in TSK FIS describe a mapping from fuzzy sets in the input to a linear function in the output and have the form

$$R_l : IF \ x_1 \text{ is } A_1^l \text{ AND } \dots \text{ AND } x_J \text{ is } A_J^l \text{ THEN } f_l = K_o^l + \dots + K_J^l x_J \quad (2.6)$$

where K_j^l is the consequent parameter and f_l is the output function of rule R_l . A first order TSK FIS means that the output, f_l , is a linear function of its inputs whereas a zero order TSK FIS means that the output is a constant function.

2.2.3 Defuzzification

The fuzzy output can be defuzzified into a crisp output using one of the defuzzification methods. Center of area method (COA) is a popular defuzzification method used with

Mamdani FIS and is given by

$$u_o = \frac{\sum_{l=1}^L \mu_C(y_l) \cdot y_l}{\sum_{l=1}^L \mu_C(y_l)} \quad (2.7)$$

Weighted average method is used with TSK FIS and is defined as follows

$$f(\bar{x}) = \frac{\sum_{l=1}^L \left(\prod_{j=1}^J \mu^{A_j^l}(x_j) \right) f_l}{\sum_{l=1}^L \left(\prod_{j=1}^J \mu^{A_j^l}(x_j) \right)} \quad (2.8)$$

2.2.4 Scaling factor

It is usually desirable and often necessary to scale, or to normalize, the universe of discourse of the input and the output variables. An input scaling factor transforms a crisp input into a normalized input to keep its value within the universe. An output scaling factor provides a transformation of the defuzzified crisp output from the normalized universe of the controller output into an actual physical output.

Since choosing the value of a scaling factor is a trial-and-error process, it takes much time to get this value which may be good but may not be the best. In Chapter 3, we show how to automatically tune the input and the output scaling factors of a FLC using some type of optimization technique such as genetic algorithms.

2.3 Genetic Algorithms

Genetic algorithms (GAs) are search and optimization techniques that are based on a formalization of natural genetics [56]. A GA searches a multidimensional parameter space to find an optimal solution. A given set of parameters is referred to as a chromosome. The parameters can be either real or binary numbers. In general, the GA is initialized with a number of randomly selected parameter vectors or chromosomes. This set of chromosomes is the initial population. Each chromosome is tested and evaluated based on a fitness function (cost function). The chromosomes are sorted based on the ranking of the fitness functions. One then selects a number of the best chromosomes according to their fitness values to be parents of the next generation of chromosomes. Then, a new set of chromosomes is selected based on reproduction.

In the reproduction process, we generate new chromosomes, which are called children, using two GA operations. The first operation is a crossover, shown in Figure 2.4, in which we choose a pair of parents and select a random point in all of their chromosomes and make a cross replacement from one parent to another. The second operation is a mutation, shown in Figure 2.5, in which a parent is selected and we change one or more of its parameters to get a new child. Now, we have a new population to test again with the fitness function.

Figure 2.4: Crossover operation.

The genetic process is repeated until a termination condition is met. There are

before-mutation	0	1	1	1	0	0	1	1	0	1	0
↓											
after-mutation	0	1	1	0	0	0	1	1	0	1	0

Figure 2.5: Mutation operation.

different conditions to terminate the genetic process such as: (i) the maximum iteration is reached, (ii) a fitness threshold is achieved, (iii) a maximum time is reached and (iv) a combination of the previous conditions. For the work done in this thesis that is based on GAs, we use the maximum iteration condition which is actually based on a threshold condition. The number of GA iterations is determined based on simulations. We tried different numbers of iterations and we determined the number of iterations for which further training would not have any significant improvement in performance.

2.4 Adaptive Network Fuzzy Inference System

Adaptive network fuzzy inference system (ANFIS) is a network-type structure similar to that of a neural network (NN). Figure 2.6 shows the structure of a FIS used in ANFIS model. The structure uses two types of nodes. The first type is an adaptive node (a squared shape) whose output need to be adapted (tuned) and the second type is a fixed node (a circled shape) whose output is a known function of its inputs. In the next subsection we will describe the structure of the ANFIS model mathematically.

Figure 2.6: Structure of a FIS used in ANFIS Model.

2.4.1 Mathematical description

Consider, without loss of generality, that we have two inputs x_1 and x_2 . Each input has three gaussian MFs. The corresponding fuzzy sets for these MFs are A_1 , A_2 and A_3 for the input x_1 and A_4 , A_5 and A_6 for the input x_2 . Their parameters, σ and m , are described using Equation (2.2) to be

$$\mu_{A_i} = \exp \left(-\frac{1}{2} \left(\frac{x - m_i}{\sigma_i} \right)^2 \right) \quad (2.9)$$

where $i = 1, 2, \dots, 6$. These parameters are called the premise parameters. The total number of premise parameters to be tuned = 6 fuzzy sets $\times$ 2 parameters for each fuzzy set = 12. We can put them into a matrix form and denote it by S where

$$S = \begin{pmatrix} \sigma_1 & m_1 \\ \sigma_2 & m_2 \\ \vdots & \vdots \\ \sigma_6 & m_6 \end{pmatrix} \quad (2.10)$$

Each row of S represents the parameters of an input MF. The number of rules we have is $(3 \text{ MFs for each input})^2 \text{ inputs} = 9 \text{ rules}$. We use the zero type TSK FIS described by Equation (2.6) to write down these rules as follow

$$R_1 : IF \ x_1 \text{ is } A_1 \text{ AND } x_2 \text{ is } A_4 \text{ THEN } f_1 = K_1$$

$$R_2 : IF \ x_1 \text{ is } A_1 \text{ AND } x_2 \text{ is } A_5 \text{ THEN } f_2 = K_2$$

$$R_3 : IF \ x_1 \text{ is } A_1 \text{ AND } x_2 \text{ is } A_6 \text{ THEN } f_3 = K_3$$

$$R_4 : IF \ x_1 \text{ is } A_2 \text{ AND } x_2 \text{ is } A_4 \text{ THEN } f_4 = K_4$$

$$R_5 : IF \ x_1 \text{ is } A_2 \text{ AND } x_2 \text{ is } A_5 \text{ THEN } f_5 = K_5$$

$$R_6 : IF \ x_1 \text{ is } A_2 \text{ AND } x_2 \text{ is } A_6 \text{ THEN } f_6 = K_6$$

$$R_7 : IF \ x_1 \text{ is } A_3 \text{ AND } x_2 \text{ is } A_4 \text{ THEN } f_7 = K_7$$

$$R_8 : IF \ x_1 \text{ is } A_3 \text{ AND } x_2 \text{ is } A_5 \text{ THEN } f_8 = K_8$$

$$R_9 : IF \ x_1 \text{ is } A_3 \text{ AND } x_2 \text{ is } A_6 \text{ THEN } f_9 = K_9$$

The previous rules can be represented by a fuzzy decision table as shown in Table 2.1. The consequent parameters to be tuned are K_l where $l = 1, 2, \dots, 9$. Therefore the output has nine parameters to be tuned. The structure of the ANFIS model has five layers. Now we will implement the algorithm in the form of a network.

Table 2.1: Fuzzy decision table.

		x_2		
		A_4	A_5	A_6
x_1	A_1	K_1	K_2	K_3
	A_2	K_4	K_5	K_6
	A_3	K_7	K_8	K_9

In layer 1, all nodes are adaptive. This layer has six outputs denoted by O^1 . The output of each node in layer 1 is the membership value of its input defined by Equation (2.9).

In layer 2, all nodes are fixed. The AND operation (multiplication) between the inputs of each rule is calculated in this layer. This layer has nine outputs denoted by O_l^2 , $l = 1, 2, \dots, 9$. The output of each node in layer 2, known as the firing strength of the rule, ω_l , is calculated as follows

$$O_l^2 = \omega_l = \prod_{i=1}^2 \mu^{A_i^l}(x_i) \quad (2.11)$$

and the outputs of this layer are calculated as follow

$$O_1^2 = \omega_1 = O_1^1 \times O_4^1 = \mu_{A_1}(x_1) \times \mu_{A_4}(x_2)$$

$$O_2^2 = \omega_2 = O_1^1 \times O_5^1 = \mu_{A_1}(x_1) \times \mu_{A_5}(x_2)$$

$$O_3^2 = \omega_3 = O_1^1 \times O_6^1 = \mu_{A_1}(x_1) \times \mu_{A_6}(x_2)$$

$$O_4^2 = \omega_4 = O_2^1 \times O_4^1 = \mu_{A_2}(x_1) \times \mu_{A_4}(x_2)$$

$$O_5^2 = \omega_5 = O_2^1 \times O_5^1 = \mu_{A_2}(x_1) \times \mu_{A_5}(x_2)$$

$$O_6^2 = \omega_6 = O_2^1 \times O_6^1 = \mu_{A_2}(x_1) \times \mu_{A_6}(x_2)$$

$$O_7^2 = \omega_7 = O_3^1 \times O_4^1 = \mu_{A_3}(x_1) \times \mu_{A_4}(x_2)$$

$$O_8^2 = \omega_8 = O_3^1 \times O_5^1 = \mu_{A_3}(x_1) \times \mu_{A_5}(x_2)$$

$$O_9^2 = \omega_9 = O_3^1 \times O_6^1 = \mu_{A_3}(x_1) \times \mu_{A_6}(x_2)$$

In layer 3, all nodes are fixed. This layer has nine outputs denoted by O_l^3 . The output of each node in layer 3 is the normalized firing strength, $\bar{\omega}_l$, which is calculated as follows

$$O_l^3 = \bar{\omega}_l = \frac{O_l^2}{\sum_{l=1}^9 O_l^2} = \frac{\omega_l}{\sum_{l=1}^9 \omega_l} \quad (2.12)$$

Note that layer 2 and layer 3 are fixed layers so we can merge them into one layer.

In layer 4, all nodes are adaptive. The defuzzification process that uses the weighted average method, defined by Equation (2.8), is performed in this layer and the next layer. Layer 4 has nine outputs denoted by O_l^4 . The output of each node is

$$O_l^4 = O_l^3 f_l = \bar{\omega}_l K_l \quad (2.13)$$

Layer 5 is the output layer and has only one fixed node whose output, f , is the sum of all its inputs as follows

$$O^5 = f = \sum_{l=1}^9 O_l^4 = \sum_{l=1}^9 \bar{\omega}_l K_l \quad (2.14)$$

which is the same as that of Equation (2.8). Now, we can extract Equation (2.14) to

be

$$\begin{aligned} f &= \bar{\omega}_1 K_1 + \bar{\omega}_2 K_2 + \cdots + \bar{\omega}_9 K_9 \\ &= a \Theta \end{aligned}$$

where $a = (\bar{\omega}_1 \ \bar{\omega}_2 \ \cdots \ \bar{\omega}_9)$ and

$$\Theta = \begin{pmatrix} K_1 \\ K_2 \\ \vdots \\ K_9 \end{pmatrix} \quad (2.15)$$

For P input/output data we have a linear equation of the form

$$f = A\Theta \quad (2.16)$$

where

$$A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_P \end{pmatrix} = \begin{pmatrix} \bar{\omega}_1^1 & \bar{\omega}_2^1 & \cdots & \bar{\omega}_9^1 \\ \bar{\omega}_1^2 & \bar{\omega}_2^2 & \cdots & \bar{\omega}_9^2 \\ \vdots & \vdots & \ddots & \vdots \\ \bar{\omega}_1^P & \bar{\omega}_2^P & \cdots & \bar{\omega}_9^P \end{pmatrix} \quad (2.17)$$

is a $P \times M$ matrix called the design matrix, M is the number of the unknown parameters (the consequent parameters) and Θ is an $M \times 1$ vector of the unknown parameters (here, $M = 9$). In the next subsection, we will discuss the learning process in the ANFIS model.

2.4.2 Hybrid learning rule

The back-propagation technique (BP) is widely adopted for learning NNs. However, it suffers from the problems of low convergence rate [57–60] and local minimum [59, 60]. Therefore, by exploiting the linearity in the consequent part as derived in Equation (2.16) we can use the hybrid learning rule. The hybrid learning rule is a combination of the BP technique and the least square estimate method (LSE). The learning procedures for the hybrid learning rule is shown in Table 2.2.

Table 2.2: Learning procedures for the hybrid rule.

			Forward pass	Backward pass
Premise parameters (S)	σ_1	m_1	fixed	BP
	σ_2	m_2		
	$\vdots$	$\vdots$		
	σ_6	m_6		
Consequent parameters (Θ)	K_1		LSE	fixed
	K_2			
	$\vdots$			
	K_9			

With the LSE method, we tune the consequent parameters, K , using Equation (2.16) in which the best solution for the least square estimator, Θ , to minimize $\|A\Theta - f\|$ is defined as

$$\hat{\Theta} = (A^T A)^{-1} A^T f \quad (2.18)$$

where $(A^T A)^{-1} A^T$ is the pseudo-inverse of A assuming $(A^T A)$ is nonsingular.

With the BP technique, we tune the premise parameters, σ and m , to minimize

the mean square error (MSE) of the output

$$\varepsilon = \frac{1}{2}(f_d - f)^2 \quad (2.19)$$

where f_d is the desired "target" output and f is the actual output. Here we use the gradient descent approach (GD). The gradient with respect to the parameter σ is defined as

$$\frac{\partial \varepsilon}{\partial \sigma_i} = \frac{\partial \varepsilon}{\partial f} \frac{\partial f}{\partial \sigma_i} \quad (2.20)$$

and the gradient with respect to the parameter m is defined as

$$\frac{\partial \varepsilon}{\partial m_i} = \frac{\partial \varepsilon}{\partial f} \frac{\partial f}{\partial m_i} \quad (2.21)$$

where $i = 1, 2, \dots, 6$ is the total number of fuzzy sets for the inputs. The term $\frac{\partial \varepsilon}{\partial f}$ is calculated from Equation (2.19) as

$$\frac{\partial \varepsilon}{\partial f} = -(f_d - f) \quad (2.22)$$

The terms $\frac{\partial f}{\partial \sigma_i}$ and $\frac{\partial f}{\partial m_i}$ are calculated using the chain rule as

$$\frac{\partial f}{\partial \sigma_i} = \sum_l \frac{\partial f}{\partial \omega_l} \frac{\partial \omega_l}{\partial \sigma_i} \quad (2.23)$$

$$\frac{\partial f}{\partial m_i} = \sum_l \frac{\partial f}{\partial \omega_l} \frac{\partial \omega_l}{\partial m_i} \quad (2.24)$$

The term $\frac{\partial f}{\partial \omega_l}$ is calculated from Equation (2.14) and the terms $\frac{\partial \omega_l}{\partial \sigma_i}$ and $\frac{\partial \omega_l}{\partial m_i}$ are

calculated from both Equation (2.11) and Equation (2.9) so

$$\frac{\partial f}{\partial \sigma_i} = \sum_l \frac{(K_l - f)}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)^2}{(\sigma_i^l)^3} \quad (2.25)$$

$$\frac{\partial f}{\partial m_i} = \sum_l \frac{(K_l - f)}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)}{(\sigma_i^l)^2} \quad (2.26)$$

By substituting from Equation (2.22) and Equation (2.25) into Equation (2.20) and from Equation (2.22) and Equation (2.26) into Equation (2.21) then

$$\frac{\partial \varepsilon}{\partial \sigma_i} = -(f_d - f) \sum_l \frac{(K_l - f)}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)^2}{(\sigma_i^l)^3} \quad (2.27)$$

$$\frac{\partial \varepsilon}{\partial m_i} = -(f_d - f) \sum_l \frac{(K_l - f)}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)}{(\sigma_i^l)^2} \quad (2.28)$$

According to the steepest descent algorithm, we make changes in the values of the parameters σ_i and m_i along the $-ve$ gradient to minimize the error so

$$\Delta \sigma_i = -\eta \frac{\partial \varepsilon}{\partial \sigma_i} \quad (2.29)$$

$$\Delta m_i = -\eta \frac{\partial \varepsilon}{\partial m_i} \quad (2.30)$$

where η is the learning rate, $0 \leq \eta \leq 1$. Then the updating rules for the parameters σ_i and m_i are defined as

$$\sigma_i(t+1) = \sigma_i(t) - \eta \frac{\partial \varepsilon}{\partial \sigma_i} \quad (2.31)$$

$$m_i(t+1) = m_i(t) - \eta \frac{\partial \varepsilon}{\partial m_i} \quad (2.32)$$

By substituting from Equation (2.27) and Equation (2.28) into Equation (2.31) and Equation (2.32), respectively then

$$\sigma_i(t+1) = \sigma_i(t) + \eta(f_d - f) \sum_l \frac{(K_l - f)}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)^2}{(\sigma_i^l)^3} \quad (2.33)$$

$$m_i(t+1) = m_i(t) + \eta(f_d - f) \sum_l \frac{(K_l - f)}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)}{(\sigma_i^l)^2} \quad (2.34)$$

2.5 Reinforcement Learning

Reinforcement learning (RL) is a computational approach to learning through interaction with the environment [29, 30] as shown in Figure 2.7. This figure consists of two blocks, an agent which tries to take actions so as to maximize some cumulative function of rewards and an environment which provides the agent with rewards. The terms agent, environment and action used in RL are equivalent, in the control field, to controller, plant and control signal, respectively.

In a single-agent RL model, the environment is described by a Markov decision

Figure 2.7: Agent-environment interaction in RL.

process. In the next subsection, we will discuss the Markov decision process problem then we will show the way to solve this problem.

2.5.1 Markov decision process problem

The Markov decision process (MDP) relates RL to dynamic programming methods (DP). Considering RL model as a MDP means that all state dynamics are independent of any previous state. A finite MDP is a 4-tuple $\langle S, A, P, R \rangle$ where:

- S is the finite state space.
- A is the finite set of agent actions.
- $P : S \times A \longrightarrow [0, 1]$ is the state transition probability.
- $R : S \times A \longrightarrow r$ is the reward function.

The problem of MDPs is to find a policy, π , for the agent that specifies the action $\pi(s)$ that the agent will choose in state s . The combination between a MDP and a policy to find the suitable action behaves like a Markov chain. A Markov chain is a stochastic process with the Markov property; given the present state, future states are independent of the past states. Suppose that we have a set of states, $S = \{s_1, s_2, \dots, s_n\}$, then the process starts in state s_i and moves to state s_j with a probability P_{ij} which is known as state transition probability. Each move is called a step.

The goal is to find a policy that will maximize a discounted return, R_t , which is defined as

$$R_t = \sum_{k=0}^{T_p} \gamma^k r_{t+k+1} \quad (2.35)$$

where r_{t+1} is the immediate reward, γ , $0 < \gamma \leq 1$, is the discount factor, T_p is the terminal point. Any task can be divided into independent episodes and T_p is the end of an episode. If T_p is finite then the model is called a finite-horizon model. If $T_p \rightarrow \infty$ then the model is called an infinite-horizon discounted model and in this case $\gamma < 1$ to avoid an infinite total rewards.

To be able to maximize the discounted return, we evaluate the performance of state s under policy π by the value function, $V^\pi(s)$, defined as

$$V^\pi(s) = E_\pi(R_t | s_t = s) = E_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} | s_t = s \right) \quad (2.36)$$

where $E_\pi(\cdot)$ is the expected value under policy π . We can also evaluate the performance of action a taken in state s under policy π by the action-value function, $Q^\pi(s, a)$, defined as

$$Q^\pi(s, a) = E_\pi(R_t | s_t = s, a_t = a) = E_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} | s_t = s, a_t = a \right) \quad (2.37)$$

In the learning process, RL is searching for the optimal policy, π^* , by searching for the optimal value function, $V^*(s)$, or the optimal action-value function, $Q^*(s, a)$, where

$$V^*(s) = \max_{\pi} V^\pi(s) \quad (2.38)$$

and

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) \quad (2.39)$$

Then the corresponding optimal policy, $\pi^*(s)$, is calculated as

$$\pi^*(s) = \arg \max_{a \in A(s)} Q^*(s, a) \quad (2.40)$$

One can estimate the optimal value function, V^* and the optimal action-value function, Q^* , recursively using a DP iteration also known as the Bellman equation defined as follows

$$V_{t+1}(s) = \max_{a \in A(s)} \left\{ \bar{r}(s, a) + \gamma \sum_{s' \in S} P_{ss'}(a) V_t(s') \right\} \quad (2.41)$$

and

$$Q_{t+1}(s, a) = \bar{r}(s, a) + \gamma \sum_{s' \in S} P_{ss'}(a) V_t(s') \quad (2.42)$$

If the state transition probability, $P_{ss'}(a)$ and the reward function, $r(s, a)$, are known (i.e. the model is well-defined) then it is easy to calculate the optimal value and optimal action-value functions, however in real applications both $P_{ss'}(a)$ and $r(s, a)$ are not known. The RL methods are used either to directly estimate the optimal value or optimal action-value functions (model-free methods) such as Q-learning and actor-critic learning or to indirectly estimate them by estimating first the state transition probability and the reward function (model-based methods) such as prioritized sweeping [61] and DYNA [62]. In this thesis we will focuss on model-free methods and especially on the Q-learning algorithm that is used to directly estimate the optimal action-value function.

2.5.2 $Q(\lambda)$ -learning

Many methods have been proposed for estimating the optimal value and the optimal action-value functions. The most widely used and well-known method is Q-learning [63, 64]. Q-learning is an off-policy control algorithm. This means that it has the ability to learn without following the current policy. The state and action spaces are discrete and their corresponding value function is stored in a lookup table called a Q-table. To use Q-learning with continuous systems (continuous state and action spaces), one can discretize the state and the action spaces [32, 34, 47, 65-68] or use some type of function approximation such as FISs [37, 69, 70] and NNs [1, 29, 71] or use some type of optimization technique such as GAs [72, 73].

A one-step update rule for Q-learning is defined as

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t)\Delta_t \quad (2.43)$$

where α , $0 < \alpha \leq 1$, is the step-size or the learning rate and Δ_t is known as the temporal difference error (TD-error) and is defined as

$$\Delta_t = r_{t+1}(s_t, a_t) + \gamma V_t(s') - Q_t(s_t, a_t) \quad (2.44)$$

where $V_t(s')$ is the value function of the next state, s' , at step t and is defined as

$$V_t(s') = \max_{a' \in A(s')} Q(s', a') \quad (2.45)$$

A one-step update rule means that it updates the value function according to the immediate reward, r_{t+1} , obtained from the environment. This is a truncated estimation of the return defined in Equation (2.35). The resulting estimation has a low variance

since it is only a one-step but at the same time it is biased for the same reason. If the return defined in Equation (2.35) is used instead then the estimation becomes unbiased but with a high variance. In addition, it is hard to implement because one has to wait until the last step is reached before one can calculate the return. Furthermore, it requires substantial memory to store the returns. To solve this dilemma, eligibility traces are used. To update the value function based on a multi-step update rule then we have to define eligibility traces.

Eligibility traces are used to modify a one-step update rule to be a multi-step update. This means that the agent will take into account not only the immediate reward but also the future rewards. One type of eligibility trace is the replacing trace depicted in Figure 2.8 and defined as follows:

$$e_t(s, a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ \lambda \gamma e_{t-1}(s, a) & \text{otherwise} \end{cases} \quad \forall s \in S, a \in A \quad (2.46)$$

or one can also use the following form [74]

$$e_t(s, a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ 0 & \text{if } s = s_t \text{ and } a \neq a_t \\ \lambda \gamma e_{t-1}(s, a) & \text{otherwise} \end{cases} \quad \forall s \in S, a \in A \quad (2.47)$$

Figure 2.8: Replacing trace.

where λ , ($0 \leq \lambda \leq 1$), is the eligibility decay factor. From Equation (2.46) we see that in each step, when a state-action pair is revisited then its trace is set to 1. In addition, traces of the remaining state-action pairs, that are not revisited, are decayed by $\lambda\gamma$. The value of the parameter λ reflects the number of steps used in the updating rule. When $\lambda = 0$ then it is a one-step update and when $\lambda = 1$ then it is an infinite-step update. Eligibility traces are used to speed up the learning process and hence to make it suitable for on-line applications. Now we modify Equation (2.43) to be a multi-step update rule as follows

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha_t(s_t, a_t)e_t(s, a)\Delta_t \quad (2.48)$$

For the continuous state and action spaces, the eligibility trace is defined as

$$e_t = \gamma\lambda e_{t-1} + \frac{\partial Q_t(s_t, a_t)}{\partial \phi} \quad (2.49)$$

where ϕ is the parameter to be tuned.

2.5.3 Actor-critic learning

Actor-critic learning is another RL method that is directly used to estimate the optimal value function. Unlike Q-learning, actor-critic learning is an on-policy TD control algorithm since it learns the policy that it follows [29]. The structure of actor-critic learning is shown in Figure 2.9 [29]. The policy structure is known as the actor and the estimated value function is known as the critic. To use actor-critic learning with continuous systems one can use some type of function approximation for the critic such as NNs [75] and FIS [76, 77].

A one-step update rule for actor-critic learning is defined as

$$V_{t+1}(s_t) = V_t(s_t) + \alpha_t(s_t)\Delta_t^{AC} \quad (2.50)$$

where the TD-error, Δ_t^{AC} , is different from the TD-error, Δ_t , defined in Equation (2.44). In this case, Δ_t^{AC} is defined as

$$\Delta_t^{AC} = r_{t+1}(s_t) + \gamma V_t(s') - V_t(s_t) \quad (2.51)$$

Figure 2.9: Actor-critic structure.

Replacing eligibility trace for actor-critic learning, $e_t^{AC}(s)$, is defined as

$$e_t^{AC}(s) = \begin{cases} 1 & \text{if } s = s_t \\ \lambda \gamma e_{t-1}^{AC}(s) & \text{otherwise} \end{cases} \quad \forall s \in S \quad (2.52)$$

We then modify Equation (2.50) to be a multi-step update rule as follows

$$V_{t+1}(s) = V_t(s) + \alpha_t(s_t) \Delta_t^{AC} e_t^{AC}(s) \quad (2.53)$$

2.5.4 Action selection

The way to choose an action in RL is a trade off between exploration and exploitation. One way to solve this dilemma is to use the ϵ -greedy action selection method described by

$$a_t = \begin{cases} a^* & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \quad (2.54)$$

where a^* is the greedy action defined as

$$a^* = \arg \max_{a'} Q(s, a') \quad (2.55)$$

Another way to solve the dilemma of exploration/exploitation is to use the Softmax action selection method in which each action is graded by a certain probability according to its estimated value. The most common Softmax method uses Gibbs or Boltzman distribution. The probability to select each action, a_t , given a state, s_t , is

calculated as

$$p(a_t/s_t) = \frac{\exp\left(\frac{Q(s_t, a_t)}{\vartheta}\right)}{\sum_{a'} \exp\left(\frac{Q(s_t, a')}{\vartheta}\right)} \quad (2.56)$$

where ϑ is a positive constant called the computational temperature. $\vartheta \rightarrow \infty$ results in equally probable selections of actions and $\vartheta \rightarrow 0$ means that the Softmax action selection becomes the same as the greedy action selection i.e. the probability of choosing the greedy action will be 1.0.

2.6 Summary

In this chapter we introduced different learning algorithms that we use in this thesis. In the next chapters, we combine two or more of these learning techniques to exploit their benefits and overcome their drawbacks. In Chapter 3, we combine FLC with GAs. In Chapter 4, we combine FLC with adaptive network. In Chapter 6, we combine FLCs and GAs with RL. In Chapter 7, we combine FLC with RL.

Chapter 3

Iterative Genetic Based Fuzzy Logic Controller

3.1 Introduction

In Chapter 2, we introduced different learning algorithms that we use in our thesis. In this chapter we propose a novel hybrid learning technique that combines GAs with FLC. The proposed technique is called *iterative genetic based fuzzy logic controller* (IGBFLC) [78] which is applied to a wall-following mobile robot. The proposed technique is used to tune the input and the output parameters of a Mamdani-type FLC. A two-stage tuning process is proposed. In stage 1, GAs tune the FLC parameters using input/output data pairs obtained from a PD controller "*initial expert*". In stage 2, the PD controller and the FLC tuned in stage 1 operate in parallel to obtain new input/output data pairs which are used by GAs to re-tune the parameters of the FLC. Stage 2 is repeated until the proposed technique has achieved the desired performance. This chapter is organized as follows: in Section 3.2, the dynamics of

a mobile robot following a wall are obtained. Section 3.3 describes the proposed technique. Computer simulations are presented in Section 3.4 and the results are discussed in Section 3.5.

3.2 Model Structure and Control

3.2.1 Wall-following model

We want the mobile robot to follow a wall at a desired distance. The minimum distance to the wall is the perpendicular line from the robot to the wall as shown in Figure 3.1. Equations of motion for a mobile robot are [79]

Figure 3.1: Wall-following model.

$$\begin{aligned}
\dot{x}_m &= V_m \cos(\theta_m) \\
\dot{y}_m &= V_m \sin(\theta_m) \\
\dot{\theta}_m &= \frac{V_m}{L} \tan(u)
\end{aligned} \tag{3.1}$$

where x_m, y_m are the cartesian distances between the robot and the wall, θ_m is the orientation of the robot, V_m is the velocity, L is the robot wheelbase and u is the steering angle of the robot.

3.2.2 PD controller design

Figure 3.2 shows a block diagram of a PD controller system. The input to the PD controller is the distance error from the robot to the wall and is calculated as

$$e = D_d - D_m \tag{3.2}$$

where D_d and D_m are the desired and the actual distances between the robot and the wall, respectively. The output is the control action, u , which is defined as

$$u = k_p e + k_d \dot{e} \tag{3.3}$$

where k_p and k_d are the proportional and the differential gain coefficients, respectively.

Figure 3.2: Block diagram of a PD controller system.

3.2.3 Fuzzy logic controller design

A block diagram of a FLC is shown in Figure 3.3. It has two inputs, the error in distance, e and its derivative, $\dot{e}$ and the output is the control action, u . The FLC is used to control the distance of the mobile robot to the wall.

Figure 3.3: Block diagram of a FLC system.

The FLC used is a Mamdani FIS described in Section 2.2. Each input has five trapezoidal MFs (trapezoidal MF is depicted in Figure 2.2). The output has five trapezoidal MFs. All MFs for the inputs and the output have the following linguistic values: HN, LN, Z, LP and HP where **HN** means **H**igh **N**egative, **LN** means **L**ow **N**egative, **Z** means **Z**ero, **LP** means **L**ow **P**ositive and **HP** means **H**igh **P**ositive. The rules used are of the form of Equation (2.3) which are described as follows

$$R_l : IF \ e \text{ is } A_1^l \text{ AND } \dot{e} \text{ is } A_2^l \text{ THEN } u_l \text{ is } C_l, \ w_l = q_l \quad (3.4)$$

Table 3.1 shows the initial fuzzy decision table for the inputs. The crisp output is calculated from Equation (2.7) as follows

$$u = k_u \frac{\sum_{l=1}^L \mu^{C_l}(u_l) \cdot u_l}{\sum_{l=1}^L \mu^{C_l}(u_l)}$$

Table 3.1: Initial Fuzzy decision table.

		$\dot{e}$				
		HN	LN	Z	LP	HP
e	HN	HN	HN	HN	LN	Z
	LN	HN	HN	LN	Z	LP
	Z	HN	LN	Z	LP	HP
	LP	LN	Z	LP	HP	HP
	HP	Z	LP	HP	HP	HP

where k_u is the output scaling factor.

Due to normalization of the inputs and the output, we need to use scaling factors. The input scaling factors are k_e , $k_{\dot{e}}$ and the output scaling factor is k_u .

3.3 Iterative Genetic Based Fuzzy Logic Controller

Our proposed IGBFLC is shown in Figure 3.4. The learning process passes through two stages. In stage 1, a PD controller is used as an initial expert to collect input/output data pairs. Then a GA is used to tune the input and the output parameters of a FLC using these input/output data pairs as shown in the upper figure. In Stage 2, the PD controller and the tuned FLC operate in parallel as shown in the lower figure. We then collect new input/output data pairs which are used by GAs to re-tune the input and the output parameters of the FLC. In this stage, the output of the PD controller is viewed as an error signal that is to be driven to zero in a what is known as feedback-error learning [80]. Note that in the ideal case, if the output of the PD controller is equal to zero then the FLC will learn the inverse dynamics of the system. Stage 2 is repeated until the FLC has achieved the desired performance. We use the PD controller iteratively as described in Algorithm 3.3.1. Algorithm 3.3.2

describes the learning process in detail. Note that learning in stage 1 and stage 2 is done off-line.

Figure 3.4: The proposed IGBFLC.

Algorithm 3.3.1 : The proposed IGBFLC.

- 1: **Stage 1:**
 - 2: Run the PD controller and obtain input/output data pairs.
 - 3: Tune the input and the output parameters of the FLC using GAs.
 - 4: **Stage 2:**
 - 5: Put the tuned FLC in parallel with the PD controller.
 - 6: Run the system and obtain new input/output data pairs.
 - 7: Re-tune the input and the output parameters of the FLC using GAs.
 - 8: Repeat steps from line 5 to line 7 until the desired performance is achieved.
 - 9: Separate the FLC which is now ready to be used alone.
-

3.3.1 Coding the parameters of a fuzzy logic controller using GAs

In our proposed IGBFLC, we construct three chromosomes. The first chromosome represents the parameters of the input and the output MFs. The second chromosome represents the scaling factors. The third chromosome represents the rules' weights.

The FLC has three variables: two inputs, e and $\dot{e}$ and one output, u . Each variable has five MFs with the linguistic values HN, LN, Z, LP and HP. Each MF has four coefficients so the total number of MF parameters to be tuned is 3 variables $\times$ 5 MFs $\times$ 4 coefficients = 60 parameters. So, the chromosome of a MF has a length of 60 genes. We use decimal numbers for coding these parameters. In order to choose consistent values, we redefine the coefficients of the trapezoidal MF as shown in Figure 3.5 to be

$$a = x_o + p_1;$$

$$b = x_o + p_1 + p_2;$$

$$c = x_o + p_1 + p_2 + p_3;$$

$$d = x_o + p_1 + p_2 + p_3 + p_4.$$

Algorithm 3.3.2 : The learning process.

- 1: **Stage 1:**
 - 2: Run the PD controller and obtain N input/output data pairs.
 - 3: Initial population with size P .
 - 4: **for** each iteration i **do**
 - 5: **for** each population p **do**
 - 6: Construct a FLC from the p^{th} chromosome.
 - 7: **for** each input/output data pair n **do**
 - 8: Evaluate the output, u_{flc}^n , of the FLC.
 - 9: **end for**
 - 10: Calculate the p^{th} fitness function using the MSE
$$MSE = \frac{1}{2N} \sum_{n=1}^N (u_d^n - u_{flc}^n)^2 \quad (3.5)$$
 - 11: **end for**
 - 12: Generate a new population as follows:
 - 13: Sort the entire chromosomes according to their fitness values.
 - 14: Select a portion of the sorted chromosomes as the new parents.
 - 15: Create a new generation for the remaining portion of population by applying crossover and mutation operations.
 - 16: **end for**
 - 17: **Stage 2:**
 - 18: Put the tuned FLC in parallel with the PD controller.
 - 19: Run the system and obtain new N input/output data pairs.
 - 20: Run stage 1 from line 3 to line 16.
 - 21: Run the system using the tuned FLC alone.
 - 22: **if** the tuned FLC achieves the desired performance **then**
 - 23: End the algorithm.
 - 24: **else**
 - 25: Go to line 18.
 - 26: **end if**
-

Figure 3.5: Trapezoidal MF with modified coefficients.

The FLC has three scaling factors: k_e and $k_{\dot{e}}$ for the inputs and k_u for the output. The chromosome of the scaling factors has a length of three genes. We use decimal numbers for coding the scaling factors.

Reducing the number of rules is important for fuzzy control of complex processes with high dimensionality [81]. The proposed IGBFLC has two inputs with five MFs for each input so the total number of rules is $(5)^2 = 25$ rules. Therefore, the chromosome assigned for the weights of the rules has a length of 25 binary genes. When a gene = 0 then the related rule will be omitted from the rule base and a gene of value 1 means that the related rule is added to the rule base.

3.4 Computer Simulation

We use a pentium 4 computer with a 3.2 GHz clock frequency and 1.0 Gigabytes of RAM. We do computer simulation with MATLAB software. We choose the velocity

of the robot $V_m = 0.5$ m/s, the initial orientation $\theta_m = 0.0$, the robot wheelbase $L = 0.1$ m and the desired distance between the robot and the wall $D_d = 1.0$ m.

We tune the proportional and the differential gain coefficients k_p and k_d of the PD controller manually. To choose suitable values for K_p and K_d , we plot the unit step response of the PD controller for different values of K_p and K_d as shown in Figure 3.6. Then, we choose the values of K_p and K_d that have a fast unit step response without overshooting. According to the results shown in Figure 3.6, we choose k_p and k_d of the PD controller to be 1.2 and 1.5, respectively.

Figure 3.6: Unit step response of the PD controller for different values of K_p and K_d .

To choose the number of iterations and the population size, we make combinations of different iteration numbers and different population sizes. Then, we run the GA 100 times and we take the average of the fitness values¹ for each combination over the 100 runs. We sort the averages of the fitness values. Then, we choose the combination at which the improvement in the average fitness value is less than 5%. The iteration

¹The fitness value is calculated from Equation (3.5)

numbers that we test are 10, 15, 30, and 40. The population sizes that we test are 20, 40, 60, and 80. Therefore, we have 16 different combinations. Figure 3.7 shows the average fitness values for the different combinations of the iteration numbers and the population sizes. When the number of iteration = 15 and the population size = 40, the improvement in the average fitness value is 4.4%. Therefore, we choose the number of iterations to be 15 and the population size to be 40.²

The probability of crossover is chosen to be 0.4 therefore, we reproduce 16 chromosomes by crossover. In binary coding, we select a random point, as mentioned before and make the crossover process. In decimal coding, we simply multiply the first chromosome of the selected portion of the old population (20 chromosomes) by a random number r , $0 < r < 1$ and multiply the second chromosome by $(1 - r)$ then we add both of them to generate a new chromosome. We repeat this operation for the next chromosome in the old population until the number of crossover chromosomes is reached. Here, we set $r = 0.5$.

The probability of mutation is chosen to be 0.1 therefore, we reproduce four chromosomes by mutation. For the binary and decimal coding, we generate four new chromosomes randomly to avoid a local minimum for the fitness function.

3.5 Results

In stage 1, we obtain input/output data pairs from the PD controller and use them to tune input and the output parameters of the FLC as we described in Algorithm 3.3.2. We use the PD controller to teach the FLC; the PD controller is thought of as the initial expert. Figure 3.8 shows the error resulting from the PD controller

²Note that this criterion of choosing the iteration number and the population size is applied to all the GAs used in this thesis.

Figure 3.7: Average fitness value for different combinations of the iteration numbers and the population sizes.

compared to the error resulting from the tuned FLC after stage 1. It is clear that the performance of the PD controller is better than that of the FLC tuned in this stage. Therefore, we need to do stage 2. In stage 2, we put the tuned FLC in parallel with

Figure 3.8: Error resulting from the PD controller (solid line) compared to error resulting from the FLC obtained from stage 1 (dotted line).

the PD controller and get new input/output data pairs. We use these data pairs to further tune the FLC. We repeated this iteration twice.

Figure 3.9 shows the error resulting from the PD controller compared to the error resulting from the PD controller in parallel with the FLC obtained from stage 1. Figure 3.10 shows the error resulting from the PD controller compared to the error resulting from the FLC obtained from iteration 1 of stage 2. Figure 3.11 shows the error resulting from the PD controller compared to the error resulting from the PD controller in parallel with the FLC obtained from iteration 1 of stage 2. Figure 3.12 shows the error resulting from the PD controller compared to the error resulting from the FLC obtained from iteration 2 of stage 2. Figure 3.13 shows the error resulting from the PD controller compared to the error resulting from the PD controller in parallel with the FLC obtained from iteration 2 of stage 2. Figure 3.8, Figure 3.10 and Figure 3.12 show us how the performance of the FLC improves from the first to

Figure 3.9: Error resulting from the PD controller (solid line) compared to error resulting from the PD controller parallel with the FLC obtained from stage 1 (dotted line).

the second stage (the two iterations) and the best performance is that of the FLC obtained from iteration 2 of stage 2. Figure 3.14 shows the path of the mobile robot

Figure 3.10: Error resulting from the PD controller (solid line) compared to error resulting from the FLC obtained from iteration 1 of stage 2 (dotted line).

Figure 3.11: Error resulting from the PD controller (solid line) compared to error resulting from the PD controller parallel with the FLC obtained from iteration 1 of stage 2 (dotted line).

to follow a wall using the PD controller and the proposed IGBFLC. Figure 3.12 and Figure 3.14 tell us that the performance of the FLC tuned with the proposed IGBFLC

Figure 3.12: Error resulting from the PD controller (solid line) compared to error resulting from the FLC obtained from iteration 2 of stage 2 (dotted line).

Figure 3.13: Error resulting from the PD controller (solid line) compared to error resulting from the PD controller parallel with the FLC obtained from iteration 2 of stage 2 (dotted line).

is better than that of the PD controller itself.

Tables 3.2–3.4 represent the parameters (a , b , c , and d) of the inputs and the output MFs after tuning. Table 3.5 represents the scaling factors of the inputs and the output after tuning. Table 3.6 represents the fuzzy decision table after the learning process of the FLC. The empty places in Table 3.6 are the omitted rules due to tuning. Here we see that the number of rules is reduced by 44% from 25 rules before tuning to 14 after tuning.

Figure 3.14: Path of the mobile robot to follow a wall using the PD controller (solid line) compared to the IGBFLC (dotted line).

Table 3.2: Tuned MF parameters for the input e .

	a	b	c	d
HN	-1.1277	-0.7744	-0.4998	-0.2049
LN	-0.6312	-0.4160	-0.0660	0.2497
Z	-0.4211	-0.1129	0.1350	0.3521
LP	-0.2460	0.1331	0.4266	0.7748
HP	0.2579	0.6270	0.8409	1.1814

Table 3.3: Tuned MF parameters for the input $\dot{e}$.

	a	b	c	d
HN	-1.1899	-0.9170	-0.5926	-0.3476
LN	-0.7226	-0.4392	-0.1337	0.2220
Z	-0.4226	-0.1821	0.1041	0.3307
LP	-0.2885	0.0385	0.3351	0.6641
HP	0.3777	0.5834	0.8787	1.1502

Table 3.4: Tuned MF parameters for the output u .

	a	b	c	d
HN	-1.1782	-0.8579	-0.5208	-0.1903
LN	-0.7220	-0.4580	-0.0858	0.1457
Z	-0.4592	-0.1918	0.1065	0.3961
LP	-0.1679	0.1698	0.4631	0.6976
HP	0.3242	0.6034	0.8046	1.1057

Table 3.5: Scaling factors.

k_e	$k_{\dot{e}}$	k_u
1.2858	1.6465	1.6206

Table 3.6: Fuzzy decision table after tuning.

		$\dot{e}$				
		HN	LN	Z	LP	HP
e	HN		HN	HN		Z
	LN		HN	LN	Z	
	Z		LN	Z	LP	
	LP				HP	HP
	HP		LP		HP	HP

3.6 Summary

This chapter introduces a novel technique to tune all the parameters of a FLC using GAs. The proposed IGBFLC is used to control a wall-following mobile robot. Two

stages are described for the tuning process. Stage 1 uses a PD controller as the initial expert. Stage 2 uses the PD controller and the FLC in parallel with to continue improving the performance. We tune the coefficients of the input and the output MFs. Also, we tune the input and the output scaling factors. Finally, we reduce the number of rules. The results show that the performance of the FLC tuned by the proposed IGBFLC is better than the performance of the PD controller.

In Chapter 4 we will extend the idea used in this chapter by proposing another learning technique. We will apply this technique to a pursuit-evasion game.

Chapter 4

Iterative Adaptive Network Fuzzy Inference System

4.1 Introduction

In Chapter 3 we use GAs to tune the input and the output parameters of a FLC applied to a wall-following mobile robot. In this chapter we extend (generalize) the technique proposed in Chapter 3. We use another learning algorithm instead of GAs used in Chapter 3 and apply the proposed technique to a different model. In this chapter ANFIS, which is described in Section 2.4, is used to tune the input and the output parameters of a FLC applied to a pursuit-evasion game. The proposed technique is called *iterative adaptive network fuzzy inference system* (IANFIS) [67]. One reason for choosing the pursuit-evasion game is that the time-optimal control strategy is known. Therefore, the minimum time of capture can be calculated to be a reference for our results.

In the proposed IANFIS, a two-stage tuning process is proposed. In stage 1, ANFIS is used to tune the input and the output parameters of FLC using input/output data pairs obtained from a PD controller which is used as an initial expert. In stage 2, the PD controller and the tuned FLC operate in parallel. Then we get new input/output data pairs which are used to re-tune the input and the output parameters of the FLC. Stage 2 is repeated until the FLC has achieved the desired performance. Finally, the performance of the PD controller, ANFIS, GAs, proposed IANFIS and proposed IGBFLC are compared with the classical control strategy.

This chapter is organized as follows: in Section 4.2 the pursuit-evasion model is described. The proposed technique is described in Section 4.3. Section 4.4 represents computer simulation and results are discussed in Section 4.5. Finally, conclusion is pointed out in Section 4.6.

4.2 Model Structure and Control

4.2.1 Pursuit-evasion model

The pursuit-evasion model is shown in Figure 4.1. Equations of motion for the pursuer/evader robot are [82]

$$\begin{aligned}\dot{x}_i &= v_i \cos(\theta_i) \\ \dot{y}_i &= v_i \sin(\theta_i) \\ \dot{\theta}_i &= \frac{v_i}{L_i} \tan(u_i)\end{aligned}\tag{4.1}$$

where "i" is "p" for the pursuer and is "e" for the evader, (x_i, y_i) is the position of the robot, θ_i is the orientation, L_i is the robot wheelbase, u_i is the steering angle.

Figure 4.1: Pursuit-evasion model.

$u_i \in [-u_{imax}, u_{imax}]$ and v_i is the velocity of the robot which is governed by the steering angle, to avoid slips, such that

$$v_i = V_{imax} \cos(u_i) \quad (4.2)$$

where V_{imax} is the maximum velocity of the robot.

Our scenario is to make the pursuer faster than the evader i.e $V_{pmax} > V_{emax}$ but at the same time to make the pursuer less maneuverable than the evader i.e $u_{pmax} < u_{emax}$. The control strategy for the pursuer is to drive the angle difference between the pursuer and the evader, δ , to be zero where the error in angle, δ , as shown in Figure 4.1, is calculated as

$$\delta = \tan^{-1} \left(\frac{y_e - y_p}{x_e - x_p} \right) - \theta_p \quad (4.3)$$

The control strategy for the evader is to maximize the distance between them by two ways [82, 83]:

1. If the distance between the pursuer and the evader is greater than a certain amount, d , then the control strategy for the evader is

$$u_e = \tan^{-1} \left(\frac{y_e - y_p}{x_e - x_p} \right) - \theta_e \quad (4.4)$$

2. If the distance between the pursuer and the evader is smaller than d then the evader exploits its higher maneuverability. Therefore, the control strategy for the evader will be

$$u_e = (\theta_p + \pi) - \theta_e \quad (4.5)$$

This strategy will increase the maneuverability of the evader and makes it more difficult (but not impossible) for the pursuer to catch the evader. We choose this strategy to show the effect of our proposed techniques on the system. The classical control strategy of the pursuer that we compare our results with is defined by

$$u_p = \begin{cases} -u_{p_{max}} & : \quad \delta < -u_{p_{max}} \\ \delta & : \quad -u_{p_{max}} \leq \delta \leq u_{p_{max}} \\ u_{p_{max}} & : \quad \delta > u_{p_{max}} \end{cases} \quad (4.6)$$

where δ is defined by Equation (4.3).

The capture occurs when the distance between the pursuer and the evader is less than a certain number, ℓ . We call this number the capture radius which is defined as

$$\ell = \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} \quad (4.7)$$

4.2.2 PD controller design

We control the pursuer such that it can catch the evader in minimum time. We use a PD controller, as shown in Figure 4.2, as the initial expert. The input to the PD controller is the error in angle, δ and the output is the pursuer steering angle, u_p , which acts as the control action which is defined as

$$u_p = k_p \delta + k_d \dot{\delta} \quad (4.8)$$

where k_p and k_d are the proportional and the differential gain coefficients, respectively.

Figure 4.2: Block diagram of a PD controller system.

4.2.3 Fuzzy logic controller design

A block diagram of a FLC is shown in Figure 4.3. It has two inputs, δ and $\dot{\delta}$ and its output is u_p . The FIS used is a zero-order TSK-FIS, described in Section 2.2. We use five gaussian MFs for the input δ and five gaussian MFs for the input $\dot{\delta}$. The gaussian MF is defined in Equation (2.2). The boundaries of the fuzzy universe of discourse for δ and $\dot{\delta}$ are taken to be the minimum and maximum data values for each input. Here, we have $(5)^2 = 25$ rules and each rule used is of the form of Equation (2.6) and

Figure 4.3: Block diagram of a FLC system.

is described as

$$R_l : IF \delta \text{ is } A_1^l \text{ AND } \dot{\delta} \text{ is } A_2^l \text{ THEN } f_l = K_l$$

Table 4.1 shows the fuzzy decision of the system. The output of the FLC, u_p , is calculated from Equation (2.8) to be

$$u_p = \frac{\sum_{l=1}^{25} \left(\prod_{i=1}^2 \mu^{A_i^l} \right) K_l}{\sum_{l=1}^{25} \left(\prod_{i=1}^2 \mu^{A_i^l} \right)}$$

Table 4.1: Fuzzy decision table.

		$\dot{\delta}$				
		HN	LN	Z	LP	HP
δ	HN	K_1	K_2	K_3	K_4	K_5
	LN	K_6	K_7	K_8	K_9	K_{10}
	Z	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}
	LP	K_{16}	K_{17}	K_{18}	K_{19}	K_{20}
	HP	K_{21}	K_{22}	K_{23}	K_{24}	K_{25}

4.3 Iterative Adaptive Network Fuzzy Inference System

Our proposed IANFIS is shown in Figure 4.4. Learning process passes through two stages. In stage 1, ANFIS is used to tune the input and the output parameters of a FLC using the input/output data pairs obtained from the PD controller shown in the upper figure. In Stage 2, the PD controller and the learned FLC operate in parallel

Figure 4.4: The proposed IANFIS.

as shown in the lower figure. Then we get new input/output data pairs which are used by the ANFIS to re-tune the input and the output parameters of the FLC. In this stage, the output of the PD controller is viewed as an error signal that is to be driven to zero in a what is known as feedback-error learning [80]. Note that in the ideal case, if the output of the PD controller is equal to zero then the FLC will learn the inverse dynamics of the system. Stage 2 is repeated until the FLC has achieved the desired performance. Here, we use the PD controller iteratively as described in Algorithm 4.3.1. Note that learning in stage 1 and stage 2 is done off-line.

Algorithm 4.3.1 : The proposed IANFIS.

- 1: **Stage 1:**
 - 2: Run the PD controller and obtain input/output data pairs.
 - 3: Use ANFIS to tune the input and the output parameters of the FLC as follows:
 - 4: Use the LSE method in the forward pass to tune the output parameters, K , using Equation (2.18).
 - 5: Use the BP technique in the backward pass to tune the input parameters, σ and m , using Equation (2.33) and Equation (2.34).
 - 6: **Stage 2:**
 - 7: Put the PD controller in parallel with the tuned FLC.
 - 8: Run the system and get new input/output data pairs.
 - 9: Use ANFIS to re-tune the input and the output parameters of the FLC.
 - 10: **if** the tuned FLC achieves the desired performance **then**
 - 11: End the algorithm.
 - 12: **else**
 - 13: Go to line 7.
 - 14: **end if**
 - 15: Separate the FLC which is now ready to be used alone.
-

4.3.1 Tuning the parameters of a fuzzy logic controller using ANFIS

In stage 1, we use the ANFIS model with the hybrid learning rule to tune the input and the output parameters of the FLC (see Section 2.4.2 for more details about the ANFIS model). The LSE method is used in the forward pass to tune the 25 parameters of the consequent part according to Equations (2.15)–(2.18). The BP technique is used in the backward pass to tune the 20 parameters of the premise part according to Equations (2.33)–(2.34). Then we start stage 2 in which we put the FLC obtained from stage 1 in parallel with the PD controller and obtain new input/output data pairs. Then we use these new data pairs to re-tune the input and the output parameters of the FLC. The procedures of stage 2 are repeated until the desired performance of the FLC is achieved. Finally, we separate the PD controller and hence the tuned FLC is now ready to be used.

4.4 Computer Simulation

The pursuer starts motion from position $(x_p, y_p) = (0, 0)$ with initial orientation $\theta_p = 0.0$ rad and the robot wheelbase $L_p = 1.0$ m. The maximum velocity of the pursuer $V_{pmax} = 1.0$ m/s and the steering angle $u_p \in [-\frac{\pi}{6}, \frac{\pi}{6}]$ rad. The evader starts motion from several positions $(x_e, y_e) = (3, 3), (-2, 4), (2, 5)$ and $(-1, 5)$ to cover most of situations, with initial orientation $\theta_e = 0.0$ rad and the robot wheelbase $L_e = 1.0$ m. The maximum velocity of the evader $V_{emax} = 0.5$ m/s which is half that of the pursuer (i.e. the evader is slower than the pursuer) and the control input $u_e \in [-\frac{\pi}{3}, \frac{\pi}{3}]$ rad which is twice that of the pursuer (i.e. the evader is more maneuverable than the pursuer). The capture radius is $\ell = 0.10$ m.

We tune the proportional and the differential gain coefficients k_p and k_d of the PD controller manually. To choose suitable values for K_p and K_d , we plot the unit step response of the PD controller for different values of K_p and K_d as shown in Figure 4.5. Then, we choose the values of K_p and K_d that have a fast unit step response without overshooting. According to the results shown in Figure 4.5, we choose k_p and k_d of the PD controller to be 0.99 and 0.02, respectively.

Figure 4.5: Unit step response of the PD controller for different values of K_p and K_d .

The FLC has five gaussian MFs for each input with a total of ten MFs for the inputs. Each MF has two parameters, σ and m , with a total of $10 \times 2 = 20$ parameters for the premise part. The consequent part has 25 parameters to be tuned. Therefore, the total number of parameters to be tuned for the FLC is $20 + 25 = 45$ parameters. Table 4.2 represents the initial fuzzy decision table for the inputs, δ and $\dot{\delta}$. Regarding the proposed IGBFLC used in this simulation, the values of the GA parameters are shown in Table 4.3.¹

¹These values are chosen according to the criterion described in Section 3.4.

Table 4.2: Initial fuzzy decision table.

		δ				
		HN	LN	Z	LP	HP
δ	HN	-0.8	-0.6	-0.4	-0.2	0.0
	LN	-0.6	-0.4	-0.2	0.0	0.2
	Z	-0.4	-0.2	0.0	0.2	0.4
	LP	-0.2	0.0	0.2	0.4	0.6
	HP	0.0	0.2	0.4	0.6	0.8

Table 4.3: Values of GA parameters.

No. of iterations	Population size	Crossover probability	Mutation probability
100	80	0.15	0.10

4.5 Results

In stage 1, we run the simulation of the PD controller for different initial positions of the evader to cover most situations and get input/output data pairs. The maximum and minimum values of the input/output data pairs are set to be the upper and the lower boundaries of the inputs and the output MFs. We then use these input/output data pairs to tune the input and the output parameters of the FLC. In stage 2, we put the tuned FLC obtained from stage 1 in parallel with the PD controller and get new input/output data pairs. We use these data pairs to re-tune the input and the output parameters of the FLC. We repeated this iteration once.

Figure 4.6 and Table 4.4 show the input and the output parameters of the FLC after tuning using the proposed IANFIS, respectively. Figure 4.7 and Table 4.5 show the input and output parameters of the FLC after tuning using the proposed IGBFLC, respectively.

Table 4.6 shows the capture time using the classical control strategy, the PD

Figure 4.6: Tuned MFs for the inputs using the proposed IANFIS.

Table 4.4: Fuzzy decision table after tuning using the proposed IANFIS.

		$\dot{\delta}$				
		HN	LN	Z	LP	HP
δ	HN	-0.6442	0.5204	3.3461	0.6967	0.6198
	LN	1.0458	-0.2798	-6.7812	0.3023	0.2690
	Z	0.5954	0.3267	0.0487	0.0058	0.0051
	LP	0.2834	1.2875	0.0017	0.0	0.0
	HP	0.8922	-0.6527	-0.0021	0.0	0.0

Figure 4.7: Tuned MFs for the inputs using the proposed IGBFLC.

controller, ANFIS, GAs, the proposed IANFIS and the proposed IGBFLC. From Table 4.6 we can see that the proposed IANFIS and the proposed IGBFLC have better performance than ANFIS, GAs and the PD controller. Their results are the

Table 4.5: Fuzzy decision table after tuning using the proposed IGBFLC.

		$\dot{\delta}$				
		HN	LN	Z	LP	HP
δ	HN	-0.9279	-0.8289	-0.6959	0.2914	0.4784
	LN	-0.0816	-0.0867	-0.1114	-0.0681	0.2129
	Z	-0.1767	0.0169	0.5664	-0.0474	0.6246
	LP	0.6362	0.1725	0.7642	0.4957	0.8390
	HP	0.6552	0.5013	0.8510	0.6737	0.8532

same and approach the results of the classical control strategy. To check again the validity of our proposed techniques we select positions for the evader different from those used in the learning process. The results are shown in Table 4.7. These results agree with those shown in Table 4.6.

Table 4.6: Capture time, in seconds, using the different techniques.

	Evader position			
	(3,3)	(-2,4)	(2,5)	(-1,5)
Classical control strategy	7.0	8.7	9.2	9.3
PD controller	18.8	21.5	21.3	22.0
ANFIS	19.1	21.7	21.6	22.0
GAs	19.2	23.0	21.7	22.2
Proposed IANFIS	7.5	10.1	10.0	10.5
Proposed IGBFLC	7.5	10.1	10.0	10.5

From the results shown in this section we can also see that the performance of the proposed IGBFLC and the proposed IANFIS are the same although the tuned FLCs for the both techniques are different (as shown in Figures 4.6–4.7 and Tables 4.4–4.5). Therefore we can conclude that we can obtain more than one solution for the system.

Table 4.7: Capture time, in seconds, using different initial positions for the evader.

	Evader position			
	(1,6)	(-3,5)	(4,-2)	(7,1)
Classical control strategy	10.7	11.5	7.6	13.0
PD controller	23.1	24.3	19.2	24.4
Proposed IANFIS	11.6	12.9	7.9	13.1
Proposed IGBFLC	11.6	12.9	7.9	13.1

4.6 Summary

From the work done in this chapter we can conclude that we can enhance the performance of FLC by using the proposed IANFIS and the proposed IGBFLC. We can see that the proposed IANFIS and the proposed IGBFLC have better performances than ANFIS, GAs and the PD controller itself. Another advantage of the proposed techniques is that there is no need to have or calculate the classical control strategy for the pursuer. By using the proposed techniques the time of capture is reduced and it is extremely near to that of the classical control strategy.

However, the proposed techniques introduced in Chapter 3 and Chapter 4 are used only if an expert or training data is available. In many practical cases the model is totally or partially unknown and it is difficult or expensive and in some cases impossible to get training data. In Chapter 6 and Chapter 7 we will solve this problem by proposing two novel techniques that do not need a priori knowledge about the controlled model. These techniques are based on RL and especially on the $Q(\lambda)$ -learning algorithm. Therefore, in Chapter 5 we will introduce a convergence proof for the model-building Q-learning algorithm with eligibility traces then in Chapter 6 and Chapter 7 the proposed techniques are discussed.

Chapter 5

On the Convergence of Model-building Q-learning with Eligibility Traces

5.1 Introduction

Due to the importance of RL and its usage in many applications, many authors are interested in discussing the convergence analysis of temporal difference (TD) learning and Q-learning algorithms as methods for solving RL problems. TD learning is a class of incremental learning used to solve learning-to-predict problems [84] whereas Q-learning may be viewed as a direct adaptive optimal control algorithm [85].

Sutton [84] proved that a one-step TD learning, $TD(0)$, converges to the optimal estimate if it is repeatedly presented with a finite training set. In addition, he relates TD learning to supervised learning and shows that TD learning is cheaper to implement and takes less time and less memory storage. Watkins [63] proved that

Q-learning converges with probability one when all the state-action pairs are visited infinitely and the action-values are represented discretely. Watkins and Dayan [64] proved the convergence of a restricted version of [63] since it does not permit updates from more than one iteration. Dayan [86] uses the analysis of Watkins [63] to extend the convergence proof of Sutton [84] for TD(0) to multi-step TD learning, TD(λ), for general λ . All the previous work depends on the relation between RL methods and the theory of dynamic programming (DP) [87] to prove convergence. However, the above cited convergence proofs have some complexities and propose some assumptions that may restrict the proofs.

Some other authors built their convergence proofs on the theory of stochastic approximation [88]. The stochastic approximation algorithm is simply defined as a stochastic difference equation with a small step size that tries to find minimum/maximum of unknown functions that are estimated through noisy observations. Dayan [89] proved that TD(λ) with a linear representation converges with probability one. Tsitsiklis [90] provided a proof of convergence for the Q-learning. Jaakkola et al. [91] proved convergence for the Q-learning and the TD(λ). Szepesvári and Littman [92] introduced a generalized Markov decision process (MDP) for RL that is applied for a two-player zero-sum game. Then the convergence of many RL algorithms are proved for the generalized MDP. Jiang et al. [93] showed that under certain continuity assumptions the Q-learning algorithm converges to the optimal solution obtained with the continuous DP algorithm. Singh et al. [94] provided a proof for SARSA(0), a one-step on-policy RL method, by slightly extending the works of Jaakkola et al. [91] and Szepesvári and Littman [92]. Dar and Mansour [95] derived convergence rates for the Q-learning and showed that there is a relationship between the learning rates and the rate of convergence. In their proof, Jaakkola et al. [91] assumed that the

Q-values are bounded. Later, Gosavi [96] proved the boundedness of the Q-values for the Q-learning based on simple induction.

In this chapter,, we extend the proof of boundedness of the Q-learning algorithm showed by Gosavi [96] to the more general model-building Q-learning algorithm with eligibility traces. In addition, we present a convergence proof for the model-building Q-learning algorithm with eligibility traces supported by the proof of boundedness.

The reason for choosing Q-learning with eligibility traces is that it outperforms the one-step Q-learning [97]. To the best of our knowledge, all the work done in Q-learning with eligibility traces does not have a theoretical basis and the work done in this chapter is the first to show the theoretical convergence of the model-building Q-learning algorithm with eligibility traces.

5.2 Model-building Q-learning with Eligibility Traces

Previously, model-based algorithms were built by knowing the transition probability in advance. Gosavi [98, pp. 253–259] presented model-building algorithms in which the transition probability model is built throughout the algorithm runtime. Hence, a new definition of the one-step update rule for Q-learning is proposed by Gosavi [98, pp. 257–258], as

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t(s_t, a_t)) Q_t(s_t, a_t) + \alpha_t(s_t, a_t) \left[\bar{r}(s_t, a_t) + \gamma \sum_j P_{s_t j}(a_t) \max_{b \in A(j)} Q_t(j, b) \right] \quad (5.1)$$

Now we modify Equation (5.1) by using eligibility traces to be as follows

$$Q_{t+1}(s, a) = (1 - \alpha_t(s_t, a_t)e_t(s, a))Q_t(s, a) + \alpha_t(s_t, a_t)e_t(s, a) \left[\bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_t(j, b) \right] \quad (5.2)$$

In our convergence proof we will use an equivalent form of Equation (2.46) which is defined as

$$e_t(s, a) = (\gamma\lambda)^{\chi_t(s, a)\tau(s, a)} \quad (5.3)$$

where $\chi_t(s, a)$ is the eligibility indicator of the state-action pair, (s, a) , which is defined as

$$\chi_t(s, a) = \begin{cases} 0 & \text{if } s = s_t \text{ and } a = a_t; \\ 1 & \text{otherwise.} \end{cases} \quad (5.4)$$

and $\tau(s, a)$, $\tau(s, a) \leq t$, counts the number of steps since the last visit to the state-action pair, (s, a) . If $\tau(s, a) = t$ then the corresponding state-action pair was never visited. Next, we will describe the model-building Q-learning algorithm with eligibility traces using a step-by-step format.

Step 1. Set $\forall s, s' \in S$ and $a \in A(s)$,

$Q(s, a) \leftarrow 0$, $e(s, a) \leftarrow 1$, $\bar{r}(s, a) \leftarrow 0$, $V(s, a) \leftarrow 1$, and $W(s, a, s') \leftarrow 0$ where $V(s, a)$ and $W(s, a, s')$ are counters defined as follows: when action a is selected in state s , then $V(s, a)$ will be incremented by 1 and when state s' is visited as a result of performing action a , then $W(s, a, s')$ will be incremented by 1.

for $t = 1$ to t_{max} **do**

Step 2. For the current state, s_t , select an action, a_t , using one of the action selection

methods (such as the ϵ -greedy method).

Step 3. Perform the action, a_t , obtain a next state, s' , and get the immediate reward, $r(s_t, a_t, s')$.

Step 4. Update the expected reward, $\bar{r}(s, a)$, $\forall (s, a)$ as follows:

Step 4a. For the current state-action pair (s_t, a_t) :

$$\bar{r}(s_t, a_t) \leftarrow \bar{r}(s_t, a_t) + [r(s_t, a_t, s') - \bar{r}(s_t, a_t)]/V(s_t, a_t)$$

Step 4b. Otherwise:

$$\bar{r}(s, a) \leftarrow \bar{r}(s, a)$$

Step 5. Increment the counter $W(s_t, a_t, s')$ by 1.

Step 6. Calculate the transition probability, $P_{ss'}(a)$, $\forall (s, a)$ as follows:

$$P_{ss'}(a) \leftarrow W(s, a, s')/V(s, a)$$

Step 7. Increment the counter $V(s_t, a_t)$ by 1.

Step 8. Update the eligibility traces, $e(s, a)$, $\forall (s, a)$ as follows:

Step 8a. For the current state-action pair, (s_t, a_t) , $e(s_t, a_t) \leftarrow 1$.

Step 8b. Otherwise: $e(s, a) \leftarrow \gamma \lambda e(s, a)$.

Step 9. Update $Q(s, a)$ $\forall (s, a)$ using Equation (5.2).

Step 10. Increment t by 1.

Step 11. If $t < t_{max}$ **then**, set $s_t \leftarrow s'$ and go to step 2, **else**, stop.

In the next section, we introduce the boundedness proof for the Q-value of the Q-learning algorithm with eligibility traces.

5.3 Boundedness Proof

In the next theorem, we prove the boundedness of $Q_t(s, a)$ as we need it in proving the convergence of Q-learning with eligibility traces in Theorem 5.3. Gosavi [96] proved the boundedness of the Q-values in the Q-learning algorithm. Here, we prove the boundedness in the Q-learning algorithm with eligibility traces. Our proof, as in [96], is based on induction.

Theorem 5.1 *In the model-building Q-learning algorithm with eligibility traces for discounted-reward Markov decision process and with the replacing eligibility trace, $e_t(s, a)$, defined by*

$$e_t(s, a) = (\gamma\lambda)^{\chi_t(s, a)\tau(s, a)}$$

the action value, $Q_t(s, a)$, remains bounded for every state-action pair $(s, a) \in (S, A(s))$ and for all t .

Proof We assume that for every $(s, a) \in (S, A(s))$ and for any $t \in (1, 2, \dots)$,

$$|Q_t(s, a)| \leq M(1 + \gamma + \gamma^2 + \dots + \gamma^t) \quad (5.5)$$

and if $t \rightarrow \infty$, then

$$\limsup_{t \rightarrow \infty} |Q_t(s, a)| \leq M \frac{1}{1 - \gamma} \quad \forall (s, a) \in (S, A(s)) \quad (5.6)$$

where M is a positive finite number defined as

$$M = \max\{\bar{r}_{max}, \max_{s \in S, a \in A(s)} Q_o(s, a)\}, \quad (5.7)$$

where $Q_o(s, a)$ is the initial action value which is finite for every $(s, a) \in (S, A(s))$

and the maximum expected reward, $\bar{r}_{max}$, is defined as

$$\bar{r}_{max} \equiv \max_{s \in S, a \in A(s)} |\bar{r}(s, a)| \quad (5.8)$$

Since the rewards, $r(s, a)$, are finite then $\bar{r}_{max}$ is finite and hence M is also finite. The boundedness proof is based on the assumption in Equation (5.5) therefore, to prove the boundedness of $Q_t(s, a)$, we have to prove the assumption in Equation (5.5). The update rule for the t^{th} iteration of the Q-learning algorithm with eligibility traces is

$$\begin{aligned} Q_t(s, a) = & (1 - \alpha_{t-1}e_{t-1})Q_{t-1}(s, a) \\ & + \alpha_{t-1}e_{t-1} \left[\bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_{t-1}(j, b) \right] \end{aligned} \quad (5.9)$$

For $t = 1$ and for every $(s, a) \in (S, A(s))$, we have

$$|Q_1(s, a)| \leq (1 - \alpha_o e_o) |Q_o(s, a)| + \alpha_o e_o \left| \bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_o(j, b) \right|$$

Then from Equation (5.7) and Equation (5.8), we get

$$\begin{aligned} |Q_1(s, a)| & \leq (1 - \alpha_o e_o) M + \alpha_o e_o M + \alpha_o e_o \gamma \sum_j P_{sj}(a) M \\ & = (1 - \alpha_o e_o) M + \alpha_o e_o M + \alpha_o e_o \gamma M \sum_j P_{sj}(a) \\ & = (1 - \alpha_o e_o) M + \alpha_o e_o M + \alpha_o e_o \gamma M \\ & = M + \alpha_o e_o \gamma M \\ & = M(1 + \alpha_o e_o \gamma) \end{aligned}$$

Since α_o and $e_o \leq 1$, then

$$|Q_1(s, a)| \leq M(1 + \gamma)$$

Therefore, the assumption in Equation (5.5) is proved for $t = 1$. Now we assume that the assumption in Equation (5.5) is true for $t = n$, i.e. for every $(s, a) \in (S, A(s))$

$$|Q_n(s, a)| \leq M(1 + \gamma + \gamma^2 + \dots + \gamma^n) \quad (5.10)$$

then we prove it for $t = n + 1$. The update rule for the $(n + 1)^{th}$ iteration of the Q-learning algorithm with eligibility traces is

$$Q_{n+1}(s, a) = (1 - \alpha_n e_n) Q_n(s, a) + \alpha_n e_n \left[\bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_n(j, b) \right]$$

Then we have

$$|Q_{n+1}(s, a)| \leq (1 - \alpha_n e_n) |Q_n(s, a)| + \alpha_n e_n \left| \bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_n(j, b) \right|$$

By substituting for Q_n from Equation (5.10) and for e_n from Equation (5.3) then

$$\begin{aligned} |Q_{n+1}(s, a)| &\leq (1 - \alpha_n (\lambda \gamma)^{x_n \tau}) M(1 + \gamma + \gamma^2 + \dots + \gamma^n) + \alpha_n (\lambda \gamma)^{x_n \tau} M \\ &\quad + \alpha_n (\lambda \gamma)^{x_n \tau} \gamma \sum_j P_{sj}(a) M(1 + \gamma + \gamma^2 + \dots + \gamma^n) \\ &= (1 - \alpha_n (\lambda \gamma)^{x_n \tau}) M(1 + \gamma + \gamma^2 + \dots + \gamma^n) + \alpha_n (\lambda \gamma)^{x_n \tau} M \\ &\quad + \alpha_n (\lambda \gamma)^{x_n \tau} \gamma M(1 + \gamma + \gamma^2 + \dots + \gamma^n) \sum_j P_{sj}(a) \\ &= (1 - \alpha_n (\lambda \gamma)^{x_n \tau}) M(1 + \gamma + \gamma^2 + \dots + \gamma^n) + \alpha_n (\lambda \gamma)^{x_n \tau} M \\ &\quad + \alpha_n (\lambda \gamma)^{x_n \tau} \gamma M(1 + \gamma + \gamma^2 + \dots + \gamma^n) \end{aligned}$$

$$\begin{aligned}
&= M(1 + \gamma + \gamma^2 + \dots + \gamma^n) - \alpha_n(\lambda\gamma)^{x_n\tau} M(1 + \gamma + \gamma^2 + \dots + \gamma^n) \\
&\quad + \alpha_n(\lambda\gamma)^{x_n\tau} M + \alpha_n(\lambda\gamma)^{x_n\tau} M(\gamma + \gamma^2 + \dots + \gamma^{n+1}) \\
&= M(1 + \gamma + \gamma^2 + \dots + \gamma^n) - \alpha_n(\lambda\gamma)^{x_n\tau} M(\gamma + \gamma^2 + \dots + \gamma^n) \\
&\quad + \alpha_n(\lambda\gamma)^{x_n\tau} M(\gamma + \gamma^2 + \dots + \gamma^{n+1}) \\
&= M(1 + \gamma + \gamma^2 + \dots + \gamma^n) + \alpha_n(\lambda\gamma)^{x_n\tau} M\gamma^{n+1} \\
&= M(1 + \gamma + \gamma^2 + \dots + \gamma^n + \alpha_n(\lambda\gamma)^{x_n\tau} \gamma^{n+1})
\end{aligned}$$

Since α_n and $(\lambda\gamma)^{x_n\tau} \leq 1$, then

$$|Q_{n+1}(s, a)| \leq M(1 + \gamma + \gamma^2 + \dots + \gamma^n + \gamma^{n+1})$$

Therefore, the assumption in Equation (5.5) is proved for $t = n + 1$. $\square$

5.4 Convergence Proof

The convergence proof of the model-building Q-learning algorithm with eligibility traces is based on the fact that the Q-learning with eligibility traces algorithm can be considered as a stochastic process to which techniques of stochastic approximation are applicable. To prove the convergence of the Q-learning with eligibility traces algorithm, we present first the theorem of a random iterative process introduced by Jaakkola et al. [91], then we introduce the convergence proof of the model-building Q-learning algorithm with eligibility traces.

Theorem 5.2 [91] *A random iterative process*

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \beta_t(x)F_t(x) \quad (5.11)$$

converges to zero with probability 1 under the following assumptions:

- 1) $x \in X$ where X is a finite set.
- 2) $\sum_{t=0}^{\infty} \alpha_t(x) = \infty, \sum_{t=0}^{\infty} \alpha_t^2(x) < \infty, \sum_{t=0}^{\infty} \beta_t(x) = \infty, \sum_{t=0}^{\infty} \beta_t^2(x) < \infty,$
and $E\{\beta_t(x)|P_t\} \leq E\{\alpha_t(x)|P_t\}$ uniformly over x with probability 1.
- 3) $\|E\{F_t(x)|P_t, \beta_t\} - \gamma \Delta_t\|_W \leq \gamma \|\Delta_t\|_W$, where $\gamma \in (0, 1)$.
- 4) $\text{Var}\{F_t(x)|P_t, \beta_t\} \leq C(1 + \|\Delta_t\|_W)^2$, where C is some constant.

Here $P_t = \{X_t, X_{t-1}, \dots, F_{t-1}, \dots, \alpha_{t-1}, \dots, \beta_{t-1}, \dots\}$ stands for the past at step t . The terms $F_t(x)$, $\alpha_t(x)$, and $\beta_t(x)$ are allowed to depend on the past and $\alpha_t(x)$ and $\beta_t(x)$ are assumed to be nonnegative and mutually independent given P_t . The notation $\|\cdot\|_W$ refers to some weighted maximum norm where $\|\cdot\|_W = \max_x |\cdot|/W(x)$. The proof of Theorem 5.2 is described in [91].

In the next theorem, we prove the convergence of the model-building Q-learning algorithm with eligibility traces by relating the algorithm to the converging stochastic process defined by Theorem 5.2.

Theorem 5.3 *The model-building Q-learning algorithm with eligibility traces given by*

$$Q_{t+1}(s, a) = (1 - \alpha_t(s_t, a_t)e_t(s, a))Q_t(s, a) + \alpha_t(s_t, a_t)e_t(s, a) \left[\bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_t(j, b) \right] \quad (5.12)$$

with the replacing eligibility trace, $e_t(s, a)$, defined by

$$e_t(s, a) = (\gamma \lambda)^{\chi_t(s, a) \tau(s, a)}$$

converges to the optimal action-value, $Q^*(s, a)$, if:

1) The state and action spaces are finite.

2) $\sum_{t=0}^{\infty} \alpha_t(s_t, a_t) = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2(s_t, a_t) < \infty$.

3) $\text{Var}\{r(s, a)\}$ is finite.

Proof By substituting for $e_t(s, a)$ from Equation (5.3) into Equation (5.12)

$$\begin{aligned} Q_{t+1}(s, a) = & (1 - \alpha_t(s_t, a_t)(\gamma\lambda)^{\chi_t(s,a)\tau(s,a)})Q_t(s, a) \\ & + \alpha_t(s_t, a_t)(\gamma\lambda)^{\chi_t(s,a)\tau(s,a)} \left[\bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_b Q_t(j, b) \right] \end{aligned} \quad (5.13)$$

By subtracting the optimal value function, $Q^*(s, a)$, from both sides of Equation (5.12), we get

$$\begin{aligned} Q_{t+1}(s, a) - Q^*(s, a) = & (1 - \alpha_t(s_t, a_t)(\gamma\lambda)^{\chi_t(s,a)\tau(s,a)})(Q_t(s, a) - Q^*(s, a)) \\ & + \alpha_t(s_t, a_t)(\gamma\lambda)^{\chi_t(s,a)\tau(s,a)} \left[\bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_t(j, b) - Q^*(s, a) \right] \end{aligned} \quad (5.14)$$

Let's define $\Delta_t(s, a) = Q_t(s, a) - Q^*(s, a)$. Theorem 5.2 proves that $\Delta_t(x)$ converges to 0 with probability 1. Therefore, we want to prove that our case can follow Theorem 5.2 then we can prove that $Q_t(s, a) - Q^*(s, a)$ converges to 0 with probability 1 i.e. $Q_t(s, a)$ converges to $Q^*(s, a)$ with probability 1. By substituting for $\Delta_t(s, a)$ in

Equation (5.14) then

$$\begin{aligned} \Delta_{t+1}(s, a) = & (1 - \alpha_t(s_t, a_t)(\gamma\lambda)^{\chi_t(s, a)\tau(s, a)})\Delta_t(s, a) \\ & + \alpha_t(s_t, a_t)(\gamma\lambda)^{\chi_t(s, a)\tau(s, a)} \left[\bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_t(j, b) - Q^*(s, a) \right] \end{aligned} \quad (5.15)$$

The algorithm in Equation (5.15) can be seen to have the form of the process of Equation (5.11) in Theorem 5.2 with $\alpha'_t(s, a) = \alpha_t(s_t, a_t)(\gamma\lambda)^{\chi_t(s, a)\tau(s, a)}$, $\beta_t(s, a) = \alpha'_t(s, a)$, and $F_t(s, a) = \bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_t(j, b) - Q^*(s, a)$. We can see $F_t(s, a)$ as an error in which $Q^*(s, a)$ is the desired value and the actual (estimated) value is $\bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_t(j, b)$. We want to show that $\alpha'_t(s, a)$ and $F_t(s, a)$ satisfy the conditions of Theorem 5.2. For $\alpha'_t(s, a)$, we can identify $(\gamma\lambda)^{\chi_t(s, a)\tau(s, a)} = C(s, a)$ where $C(s, a) \leq 1$ is a constant for the state-action pair (s, a) . Therefore, $\alpha'_t(s, a) = C(s, a)\alpha_t(s_t, a_t)$ which satisfies the conditions of Theorem 5.2.

We show that $F_t(s, a)$ has the required properties defined in Theorem 5.2 items 3 and 4 by showing that it is a contraction mapping¹ with respect to some maximum norm. This is done by relating $F_t(s, a)$ to the DP value iteration operator for the same Markov chain. More specifically, $\forall s \in S$ and $a \in A(s)$,

$$F_t(s, a) = \bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q_t(j, b) - Q^*(s, a) \quad (5.16)$$

The optimal action-value function, $Q^*(s, a)$ is defined using the Bellman optimality

¹ $f(x) : \mathcal{R} \rightarrow \mathcal{R}$ is a contraction mapping if for some $\zeta \in (0, 1)$

$$\rho(f(x_1), f(x_2)) \leq \zeta \rho(x_1, x_2), \forall x_1, x_2 \in \mathcal{R}$$

where $\rho(c, d) \equiv |c - d|$

equation as follows

$$Q^*(s, a) = \bar{r}(s, a) + \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} Q^*(j, a) \quad (5.17)$$

The expectation of $F_t(s, a)$ is defined as

$$E\{F_t(s, a)\} = \sum_j P_{sj}(a) F_t(s, a) \quad (5.18)$$

In the limit, for stochastic cases, $\bar{r}$ and P can be assumed to be equal to the actual average rewards and transitions (see Gosavi [98, pp. 214–215] for more detail. Based on this and substituting from Equation (5.17) into Equation (5.16) and then substituting the results into Equation (5.18), we get

$$E\{F_t(s, a)\} = \sum_j P_{sj}(a) \left[\gamma \sum_j P_{sj}(a) \left[\max_{b \in A(j)} Q_t(j, b) - \max_{b \in A(j)} Q^*(j, b) \right] \right]$$

Since the outer summation is over a constant, it will equal the constant therefore.

$$E\{F_t(s, a)\} = \gamma \sum_j P_{sj}(a) \left[\max_{b \in A(j)} Q_t(j, b) - \max_{b \in A(j)} Q^*(j, b) \right]$$

Then, we can write that $\forall s \in S$ and $a \in A(s)$,

$$\begin{aligned} |E\{F_t(s, a)\}| &= \gamma \left| \sum_j P_{sj}(a) \left[\max_{b \in A(j)} Q_t(j, b) - \max_{b \in A(j)} Q^*(j, b) \right] \right| \\ &\leq \gamma \sum_j P_{sj}(a) \left| \max_{b \in A(j)} Q_t(j, b) - \max_{b \in A(j)} Q^*(j, b) \right| \\ &\leq \gamma \sum_j P_{sj}(a) \max_{b \in A(j)} |Q_t(j, b) - Q^*(j, b)| \end{aligned}$$

$$\begin{aligned}
&\leq \gamma \sum_j P_{sj}(a) \max_{j \in S, b \in A(j)} |Q_t(j, b) - Q^*(j, b)| \\
&= \gamma \sum_j P_{sj}(a) \|Q_t(j, b) - Q^*(j, b)\|_\infty \\
&= \gamma \|Q_t(j, b) - Q^*(j, b)\|_\infty \sum_j P_{sj}(a) \\
&= \gamma \|\Delta_t(s, a)\|_\infty (1)
\end{aligned}$$

Therefore, $\forall s \in S$ and $a \in A(s)$,

$$|E\{F_t(s, a)\}| \leq \gamma \|\Delta_t(s, a)\|_\infty \quad (5.19)$$

Since the above inequality holds for all values of $s \in S$ and $a \in A(s)$, then it is also hold for the values that maximize the left hand side in Equation (5.19). Therefore,

$$\|E\{F_t(s, a)\}\|_\infty < \gamma \|\Delta_t(s, a)\|_\infty \quad (5.20)$$

Therefore, assumption 3 in Theorem 5.2 is met.

The variance of $F_t(s, a)$ is within the bounds of Theorem 5.2 because it depends on $Q_t(s, a)$ which, based on Theorem 5.1, is bounded and the variance of $r(s, a)$ is bounded. Hence, Theorem 5.3 is completely proved. $\square$

Note that a special case of the analysis presented in Theorem 5.3 is the model-free one-step update version defined by Equation (2.43). In this equation, the learning rate $\alpha_t(s_t, a_t)$ is used instead of $\alpha_t(s_t, a_t)e_t(s, a)$ that is used in Equation (5.2). Note that in this one-step model-free algorithm, we only update the Q-factor of the current state-action pair, $Q(s_t, a_t)$. Therefore, the convergence analysis that is presented in Theorem 5.3 can be easily applied to the model-free one-step update case with

eligibility traces that also has not been analyzed in the literature.

5.5 Summary

In this work, we extend the proof of boundedness of Q-learning as shown by Gosavi [96] to the more general model-building Q-learning with eligibility traces case. In addition, we present a convergence proof for the model-building Q-learning algorithm with eligibility traces supported by the proof of boundedness for Q-learning with eligibility traces. Although the convergence of Q-learning is proved by many authors, all the work done in Q-learning with eligibility traces does not have a theoretical basis and the work done in this chapter is the first to show the theoretical convergence of the model-building Q-learning algorithm with eligibility traces.

In Chapter 6 we propose a new hybrid learning technique based on the $Q(\lambda)$ -learning algorithm. The proposed technique overcomes the drawback found in the techniques proposed in Chapter 3 and Chapter 4 in that it does not need an expert or training data to tune the input/output parameters of FLC.

Chapter 6

Q(λ)-Learning Based Genetic Fuzzy Logic Controller

6.1 Introduction

In Chapter 3 and Chapter 4 we used supervised learning-based techniques (IGBFLC and IANFIS) to tune the input and the output parameters of FLC. These techniques require a training data set or an expert (a PD controller in our case) to collect data pairs needed for the learning process. In this chapter, we assume that the training data set is not available and we do not even have a simplistic PD controller strategy. In this case we can not use the proposed supervised learning-based techniques.

In this chapter, we propose a novel technique based on Q(λ)-learning and GAs to tune the input and the output parameters of FLC automatically. The proposed technique is called *Q(λ)-learning based genetic fuzzy logic controller* (QLBGFLC) [67, 68]. The proposed technique is applied to three versions of pursuit-evasion differential games. We start with a simple pursuit-evasion game in which only the pursuer

self-learns its control strategy on-line while the evader plays a simple known strategy. In the second game, we make the evader play an intelligently known strategy by exploiting the advantage of its higher maneuverability during the game. Finally, we make both the pursuer and the evader self-learn their control strategies on-line by the interaction with each other. We compare the proposed technique with the classical control strategy, the $Q(\lambda)$ -learning algorithm only and the reward-based GA.

This chapter is organized as follows: in Section 6.2 the problem statement is discussed. In Section 6.3, the pursuit-evasion model is described. The proposed technique is described in Section 6.4. Section 6.5 discusses computer simulation and results. Finally, conclusion is presented in Section 6.6.

6.2 Problem Statement

The problem assigned in this chapter is that we assume that the pursuer/evader does not know its control strategy. It is not told which actions to take so as to be able to optimize its control strategy. The learning goal is to make the pursuer/evader able to self-learn its control strategy. It should do that on-line by interaction with the evader/pursuer.

From several learning techniques we choose RL and especially the $Q(\lambda)$ -learning method. The reason for choosing $Q(\lambda)$ -learning is that it outperforms both the one-step Q -learning and the actor-critic learning [97]. RL methods learn without a teacher, without anybody telling them how to solve the problem. RL is related to problems where the learning agent does not know what it must do so it is the most appropriate learning technique for our problem.

However, the controlled model used is a pursuit-evasion differential game i.e. the

state and the action spaces are continuous. At the same time, $Q(\lambda)$ -learning is designed only for discrete state and action spaces. To design and optimize a controller with continuous states and actions is not a simple task for classical RL methods. Therefore, using RL alone has the limitation in that it is too hard to visit all the state-action pairs. We try to cover most of the state-action space but we can not cover all the space. In addition, there are hidden states that are not taken into consideration due to the discretization process. Hence RL alone can not find the optimal strategy.

Since we want to use RL in the robotics domain which is a continuous domain, then we need to use some type of function approximation such as FIS to generalize the discrete state-action space into a continuous state-action space. Therefore, from the RL point of view, a FIS is used as a function approximation to compensate for the limitation in RL and from the FIS point of view, RL is used to tune the input and the output parameters of the fuzzy system. In this case, the FIS is used as an adaptive controller whose parameters are tuned on-line by RL. Therefore, combining RL and FIS has two objectives; to compensate for the limitation in RL and to tune the parameters of the FLC.

The proposed QLBGFLC combines $Q(\lambda)$ -learning with GAs and FLC to solve the limitations in both RL and FLC. The proposed technique is used in the case that the state and the action spaces can be discretized in such a way that make the resulting state and action spaces have acceptable dimensions. This can be done, as we will see in our case, if the state and the action values are bounded. Note that in Chapter 7 we propose a novel technique that is used for controlled models in which this case can not be satisfied.

6.3 Pursuit-Evasion Model

The pursuit-evasion model is shown in Figure 6.1. This model is different from the model described in Section 4.2. The model described in Section 4.2 takes into account the maneuverability of the evader therefore, the velocity of each robot, v_i , is governed by the steering angle as defined by Equation (4.2). For the model described here, we use a simple strategy for the evader so each robot uses a constant velocity, V_i .¹ Equations of motion for the pursuer and the evader robots are [79]

$$\begin{aligned}\dot{x}_i &= V_i \cos(\theta_i) \\ \dot{y}_i &= V_i \sin(\theta_i) \\ \dot{\theta}_i &= \frac{V_i}{L_i} \tan(u_i)\end{aligned}\tag{6.1}$$

where "i" is "p" for the pursuer and is "e" for the evader, (x_i, y_i) is the position of the robot, V_i is the velocity, θ_i is the orientation, L_i is the robot wheelbase and u_i is the steering angle where $u_i \in [-u_{i_{max}}, u_{i_{max}}]$. The minimum turning radius is calculated as

$$Rd_{i_{min}} = \frac{L_i}{\tan(u_{i_{max}})}\tag{6.2}$$

Our scenario is to make the pursuer faster than the evader ($V_p > V_e$) but at the same time to make it less maneuverable than the evader ($u_{p_{max}} < u_{e_{max}}$). Note that in this game, the evader will only make use of its higher maneuverability at the beginning of the game. The classical control strategies that we compare our results

¹Later, in Section 6.5.3, we will make use of the maneuverability of the evader and as a result, we will use the model described in Section 4.2.

Figure 6.1: Pursuit-evasion model.

with are defined by

$$u_i = \begin{cases} -u_{imax} & : \quad \delta_i < -u_{imax} \\ \delta_i & : \quad -u_{imax} \leq \delta_i \leq u_{imax} \\ u_{imax} & : \quad \delta_i > u_{imax} \end{cases} \quad (6.3)$$

where

$$\delta_i = \tan^{-1} \left(\frac{y_e - y_p}{x_e - x_p} \right) - \theta_i \quad (6.4)$$

where "i" is "p" for the pursuer and is "e" for the evader. The capture occurs when the distance between the pursuer and the evader is less than the capture radius, ℓ , which is defined by Equation (4.7).

6.4 $Q(\lambda)$ -learning Based Genetic Fuzzy Logic Controller

The proposed QLBGFLC combines $Q(\lambda)$ -learning with GAs to tune the input and the output parameters of the FLC. In the proposed QLBGFLC, the learning process is performed sequentially in which $Q(\lambda)$ -learning is used with the discretized state and action values to estimate the optimal action-value function, $Q^*(s, a)$, which is then used by GAs to tune the input and the output parameters of the FLC which is used at the same time to generalize the discrete state and action values over the continuous state and action spaces. The learning process passes through two phases as shown in Figure 6.2. Now, we describe the FLC used in the proposed technique then we will discuss the learning in the two phases.

6.4.1 Fuzzy logic controller design

A block diagram of a FLC system is shown in Figure 6.3. The FLC has two inputs, the error in the pursuer angle, δ_p , defined in Equation (6.4) and its derivative, $\dot{\delta}_p$ and the output is the steering angle, u_p . Each input to the FLC has three gaussian MFs that are described by Equation (2.2). For the rules, we modify Equation (2.6) to be

$$R_l : IF \delta_p \text{ is } A_1^l \text{ AND } \dot{\delta}_p \text{ is } A_2^l \text{ THEN } f_l = K_l \quad (6.5)$$

Figure 6.2: The proposed QLBGFLC technique.

where $l = 1, 2, \dots, 9$. The crisp output, u_p , is calculated using Equation (2.8) as follows

$$u_p = \frac{\sum_{l=1}^9 \left(\prod_{j=1}^2 \mu^{A_j^l}(x_j) \right) K_l}{\sum_{l=1}^9 \left(\prod_{j=1}^2 \mu^{A_j^l}(x_j) \right)} \quad (6.6)$$

Figure 6.3: Block diagram of a FLC system.

6.4.2 Learning in phase 1

In phase 1, $Q(\lambda)$ -learning is used to obtain a suitable estimation for the optimal strategy of the pursuer. The state, s , consists of the error in angle of the pursuer, δ_p and its derivative, $\dot{\delta}_p$ and the action, a , is the steering angle of the pursuer, u_p . The states, $(\delta_p, \dot{\delta}_p)$ and their corresponding greedy actions, a^* , are then stored in a lookup table.

Building the discrete state and action spaces

To build the state space, we discretize the ranges of the inputs, δ_p and $\dot{\delta}_p$, by 0.2. The ranges of δ_p and $\dot{\delta}_p$ are set to be from -1.0 to 1.0 so the discretized values for δ_p and $\dot{\delta}_p$ will be $-1.0, -0.8, -0.6, \dots, 0.0, \dots, 0.8, 1.0$. There are 11 discretized values for δ_p and 11 discretized values for $\dot{\delta}_p$. These values are combined to form $11 \times 11 = 121$ states.

To build the action space, we discretize the range of the action, u_p , by 0.1. The range of the action is set to be from $-u_{pmax}$ to u_{pmax} i.e. from -0.5 to 0.5 so the discretized values for u_p will be $-0.5, -0.4, -0.3, \dots, 0.0, \dots, 0.4, 0.5$. There are 11 actions and the dimension of the Q-table will be 121-by-11.

Constructing the reward function

How to choose the reward function is very important in RL because the agent depends on the reward function in updating its value function. The reward function differs from one system to another according to the desired task. In our case, we want the pursuer to catch the evader in minimum time. In other words, we want the pursuer to decrease the distance to the evader at each time step. The distance between the pursuer and the evader at time t is calculated as follows

$$D(t) = \sqrt{(x_e(t) - x_p(t))^2 + (y_e(t) - y_p(t))^2} \quad (6.7)$$

The difference between two successive distances, $\Delta D(t)$, is calculated as

$$\Delta D(t) = D(t) - D(t+1) \quad (6.8)$$

A positive value of $\Delta D(t)$ means that the pursuer approaches the evader. The maximum value of $\Delta D(t)$ is defined as

$$\Delta D_{max} = V_{rmax}T \quad (6.9)$$

where V_{rmax} is the maximum relative velocity of the pursuer with respect to the evader ($V_{rmax} = V_p + V_e$) and T is the sampling time. So, we choose the reward, r , to be

$$r_{t+1} = \frac{\Delta D(t)}{\Delta D_{max}} \quad (6.10)$$

The learning process in phase 1 is described in Algorithm 6.4.1.

Algorithm 6.4.1 Phase 1: $Q(\lambda)$ -learning.

-
- 1: Discretize the state space, S and the action space, A .
 - 2: Initialize $Q(s, a) = 0.0 \forall s \in S, a \in A$.
 - 3: Initialize $e(s, a) = 0.0 \forall s \in S, a \in A$.
 - 4: **for** each episode **do**
 - 5: Initialize $(x_p, y_p) = (0, 0)$.
 - 6: Initialize (x_e, y_e) randomly.
 - 7: Compute $s_t = (\delta_p, \dot{\delta}_p)$ according to Equation (6.4).
 - 8: Select a_t using Equation (2.54).
 - 9: **for** each step **do**
 - 10: Receive r_{t+1} according to Equation (6.10).
 - 11: Observe s_{t+1} .
 - 12: Select a_{t+1} using Equation (2.54) or Equation (2.56) according to the action-selection method used.
 - 13: Calculate e_{t+1} using Equation (2.47).
 - 14: Update $Q(s_t, a_t)$ according to Equation (2.48).
 - 15: **end for**
 - 16: **end for**
 - 17: $Q \leftarrow Q^*$.
 - 18: Assign a greedy action, a^* , to each state, s using Equation (2.55).
 - 19: Store the state-action pairs in a lookup table.
-

6.4.3 Learning in phase 2

Phase 2 consists of two stages. In stage 1, the state-action pairs stored in the lookup table are used as the training data to tune the input and the output parameters of the FLC using GAs. Stage 1 is an off-line tuning. In this stage, the fitness function used is the MSE defined by Equation (2.19) as follows

$$MSE = \frac{1}{2N} \sum_{n=1}^N (a^{*n} - u_{flc}^n)^2 \quad (6.11)$$

where N is the number of input/output data pairs which is equivalent to the number of states, a^{*n} is the n^{th} greedy action obtained from phase 1 and u_{flc} is the output of the FLC. The GA in this stage is used as supervised learning so the results of this stage will not be better than that of phase 1 so we need to perform stage 2.

In stage 2, we run the pursuit-evasion game with the tuned FLC. Then, GA is used to fine tune the input and the output parameters of the FLC on-line during the interaction with the evader. In this stage, the capture time which the pursuer wants to minimize is used as the fitness function. In this case, the GA is used as a reward-based learning technique with a priori knowledge obtained from stage 1.

Coding a FLC into a chromosome

In phase 2, we use a GA to tune the input and the output parameters of FLC. Now, we will describe the coding of these parameters using the GA. Note that the coding process of the FLC is the same for all the different GAs used in this chapter so we will describe it in general. The FLC to be tuned has two inputs, the error, δ_p and its derivative, $\dot{\delta}_p$. Each input variable has three MFs with a total of six MFs. We use the Gaussian MF defined by Equation (2.2) which has two parameters to be tuned. These parameters are the standard deviation, σ and the mean, m . The total number of input parameters to be tuned is 12 parameters. The rules being used is defined by Equation (6.5) that has a parameter, K , to be tuned with a total number of 9 parameters to be tuned for the output part.

A FLC will be coded into a chromosome with length $12+9 = 21$ genes as shown in Figure 6.4. We use real numbers in the coding process. The population consists of a set of chromosomes, P , (coded FLCs). In the reproduction process, we generate new chromosomes by using two GA operations. The first operation is crossover in which

we choose a pair of parents and select a random gene, g , between 1 and 20 and make a cross replacement from one parent to another as shown in Figure 6.5. The second operation is mutation in which we generate a chromosome randomly to avoid a local minimum/maximum for the fitness function. Now, we have a new population to test again with the fitness function. The genetic process is repeated until the termination condition is met (see Section 2.3). The learning process in phase 2 with its two stages is described in Algorithm 6.4.2 and Algorithm 6.4.3.

Figure 6.4: A FLC coded into a chromosome.

Figure 6.5: Crossover process in GAs.

Algorithm 6.4.2 Phase 2 Stage 1: GA learning.

- 1: Get the state-action pairs from the lookup table.
 - 2: Initialize the population with a set of chromosomes, P , randomly.
 - 3: **for** each iteration **do**
 - 4: **for** each chromosome **do**
 - 5: Construct a FLC.
 - 6: **for** each state s **do**
 - 7: Calculate the FLC output, u_{flc} , using Equation (6.6).
 - 8: **end for**
 - 9: Calculate the fitness value using Equation (6.11).
 - 10: **end for**
 - 11: Sort the entire chromosomes according to their fitness values.
 - 12: Build the new population as follows:
 - 13: Select a portion of the sorted chromosomes as the new parents.
 - 14: Create a set of new chromosomes for the remaining portion of the new population using the crossover and the mutation operations.
 - 15: **end for**
-

6.4.4 Reward-based genetic algorithm

For comparative purpose, we will also implement a general reward-based GA. The reward-based GA will be initialized with randomly chosen FLC parameters (chromosomes). The GA adjust the parameters to maximize the closing distance given by Equation (6.10). Therefore, Equation (6.10) acts as the fitness function for the reward-based GA learning. The learning process in the reward-based GA is described in Algorithm 6.4.4

Algorithm 6.4.3 Phase 2 Stage 2: GA learning.

-
- 1: Initialize the population with a set of chromosomes, P , obtained from the tuned FLC obtained from stage 1.
 - 2: **for** each iteration **do**
 - 3: Initialize (x_e, y_e) randomly.
 - 4: Initialize $(x_p, y_p) = (0, 0)$.
 - 5: Calculate $s_t = (\delta_p, \dot{\delta}_p)$ according to Equation (6.4).
 - 6: **for** each chromosome **do**
 - 7: Construct a FLC.
 - 8: **for** each step **do**
 - 9: Calculate the FLC output, u_p , using Equation (6.6).
 - 10: Observe s_{t+1} .
 - 11: **end for**
 - 12: Observe the fitness value which is chosen to be the capture time that the pursuer wants to minimize.
 - 13: **end for**
 - 14: Sort the entire chromosomes according to their fitness values.
 - 15: Build the new population as follows:
 - 16: Select a portion of the sorted chromosomes as the new parents.
 - 17: Create a set of new chromosomes for the remaining portion of the new population using the crossover and the mutation operations.
 - 18: **end for**
-

6.5 Computer Simulation and Results

We use a core 2 duo with a 2.0 GHz clock frequency and 4.0 Gigabytes of RAM. We do computer simulation with MATLAB software. $Q(\lambda)$ -learning and GAs have many parameters to be set a priori therefore we tested computer simulation for different parameter values and different parameter value combinations and chose the values that give the best performance. The initial position of the evader is randomly chosen from a set of 64 different positions in the space.

Algorithm 6.4.4 Reward-based GA.

-
- 1: Initialize the population with a set of chromosomes, P , randomly.
 - 2: **for** each iteration **do**
 - 3: Initialize (x_e, y_e) randomly.
 - 4: Initialize $(x_p, y_p) = (0, 0)$.
 - 5: Calculate $s_t = (\delta_p, \dot{\delta}_p)$ according to Equation (6.4).
 - 6: **for** each chromosome **do**
 - 7: Construct a FLC.
 - 8: **for** each step **do**
 - 9: Calculate the FLC output, u_p , using Equation (6.6).
 - 10: Observe s_{t+1}
 - 11: **end for**
 - 12: Observe the fitness value defined by Equation (6.10).
 - 13: **end for**
 - 14: Sort the entire chromosomes according to their fitness values.
 - 15: Build the new population as follows:
 - 16: Select a portion of the sorted chromosomes as the new parents.
 - 17: Create a new set of chromosomes for the remaining portion of the new population using the crossover and the mutation operations.
 - 18: **end for**
-

One of the factors that is taken into consideration when we evaluate a learning technique is learning time. Learning time is an important factor especially in on-line learning. Since $Q(\lambda)$ -learning techniques and reward-based GAs are performed on-line therefor, we will define learning time in detail to be clearly understood.

6.5.1 Learning time

Before we define the learning time of a simulation, we first define some parameters of $Q(\lambda)$ -learning algorithms and some parameters of GAs that affect the learning time.

For a $Q(\lambda)$ -learning algorithm, the number of episodes (games) is N and the number of steps (plays) is M_Q . Let's also define the sampling time to be T_s and the capture time of a game to T_c . The capture time, T_c , can be determined as

$$T_c = m T_s \quad (6.12)$$

where m is a certain time step at which the pursuer captures the evader. Note that if a game ends and no capture occurs, then $m = M_Q$. Hence, the learning time of the $Q(\lambda)$ -learning algorithm, LT_Q , is calculated as follows

$$\begin{aligned} LT_Q &= k \sum_{n=1}^N T_c^{(n)} \\ &= k \sum_{n=1}^N m_Q^{(n)} T_s \\ &= k T_s \sum_{n=1}^N m_Q^{(n)} \end{aligned} \quad (6.13)$$

where k , $0 < k \leq 1$, is a simulation factor that depends on the speed of the computer processor by which calculations have been carried out. Note that, for real-time applications, $k = 1$.

For a reward-based GA, the number of iterations is I , the population size is P and the number of steps is M_G . Hence, the learning time of the GA, LT_G , is calculated as follows

$$\begin{aligned} LT_G &= k \sum_{i=1}^I \sum_{p=1}^P T_c^{(ip)} \\ &= k \sum_{i=1}^I \sum_{p=1}^P m_G^{(ip)} T_s \end{aligned}$$

$$= k T_s \sum_{i=1}^I \sum_{p=1}^P m_G^{(ip)} \quad (6.14)$$

The learning time for the proposed QLBGFLC which is a combination of the $Q(\lambda)$ -learning algorithm and the GA is calculated as follows

$$\begin{aligned} LT &= LT_Q + LT_G \\ &= k T_s \left[\sum_{n=1}^N m_Q^{(n)} + \sum_{i=1}^I \sum_{p=1}^P m_G^{(ip)} \right] \end{aligned} \quad (6.15)$$

In $Q(\lambda)$ -learning algorithms, the learning time is defined as the time needed to complete the learning process when a specific number of episodes and a specific number of steps are used. In reward-based GAs, the learning time is defined as the time needed to complete the learning process when a specific number of iterations, a specific population size and a specific number of steps are used. Note that learning time is considered as an important factor in on-line applications.

6.5.2 Evader using a simple control strategy

The pursuer starts motion from the position $(0, 0)$ with an initial orientation $\theta_p = 0.0$ and with a constant velocity $V_p = 2.0$ m/s. The robot wheelbase $L_p = 0.3$ m and the steering angle $u_p \in [-0.5, 0.5]$.

The evader starts motion from a random position at the beginning of each episode with an initial orientation $\theta_e = 0.0$ and with a constant velocity $V_e = 1.0$ m/s which is half that of the pursuer (i.e. the evader is slower than the pursuer). The robot wheelbase $L_e = 0.3$ m and the steering angle $u_e \in [-1.0, 1.0]$ which is twice that of the pursuer (i.e. the evader is more maneuverable than the pursuer). The capture

radius, ℓ , is set to 0.10 m and the sampling time, T_s , is set to 0.1 seconds.

To choose the number of steps (plays) in each episode (game), we fix the number of episodes to 200 then we run the computer simulation 100 times for different numbers of steps. We test the numbers of steps 600, 1200, 1800, 2400, 3000, 3600, 4200, 4800, 5400, 6000, and 6600. Then, we calculate the average capture time of the 100 runs for each choice of the number of steps. To assign a performance measure for each choice, we calculate the mean of the resulting average capture time for each choice as shown in Figure 6.6. We also calculate the average learning time for each choice of the number of steps as shown in Figure 6.7. We found that the mean value of the average capture time decreases as the number of steps increases but there is no significant decrease in the mean value of the average capture time after 6000 steps as we see in Figure 6.6. We also found that the average learning time decreases as the number of steps increases as shown in Figure 6.7. That means the pursuer could learn faster as the number of steps increases and can catch the evader before the maximum number of steps in a game is reached. But the average learning time no longer decreases after 6000 steps as we see in Figure 6.7. Therefore, we choose the number of steps to be 6000.

To choose the number of episodes, we fix the number of steps to 6000 then we run the computer simulation 100 times for different numbers of episodes. We test the numbers of episodes 50, 100, 150, 200, 300, 400, 500, 600, 800, 1000, and 1200. Then, we calculate the average capture time of the 100 runs for each choice of the numbers of episodes. To assign a performance measure for each choice, we calculate the mean of the resulting average capture time for each choice as shown in Figure 6.8. We also calculate the average learning time for each choice of the numbers of episodes as shown in Figure 6.9. We found that the average learning time increases

Figure 6.6: Mean values of the average capture times for different numbers of steps.

Figure 6.7: Average learning time for different numbers of steps.

as the number of episodes increases. Also, we can see that after 200 episodes, there is a small decrease in the mean value of the average capture time with a big increase in the average learning time. In other words, if we choose the number of episodes to be 200 then the mean value of the average capture time is 28.8 seconds and the average learning time is 7.8 seconds and if we choose the number of episodes to be

Figure 6.8: Mean values of the average capture times for different numbers of episodes.

300, then the mean value of the average capture time will be 28.5 seconds and the average learning time will be 11.6 seconds i.e. with 300 episodes, the mean value of the average capture time decreases by 1.04% (which is a low percentage) at the cost of increasing the average learning time by 48.7% (which is a comparatively high percentage). As a result, we choose the number of episodes to be 200.²

The duration of a game (an episode) = number of steps $\times$ sampling time. The game (episode) ends when this duration time passed without capturing or when the capture occurs before the end of this time. We choose $\gamma = 0.5$ and $\lambda = 0.3$. We make the learning rate, α , decrease with each episode such that

$$\alpha = \left(\frac{1.0}{i} \right)^{0.7} \quad (6.16)$$

where i is the current episode. We use the ϵ -greedy action selection method defined

²Note that we use this criterion for setting the parameters of the $Q(\lambda)$ -learning algorithms that are used in this thesis.

Figure 6.9: Average learning time for different numbers of episodes.

by Equation (2.54) in which we make the value of ϵ decrease with each episode such that

$$\epsilon = \frac{0.1}{i} \quad (6.17)$$

The position of the evader, (x_e, y_e) , is chosen randomly at the beginning of each episode to cover most of the states. Table 6.1 shows the values of the GAs parameters used in phase 2 stage 1 and phase 2 stage 2.³

To validate the proposed QLBGFLC, we compare its results with the results of the classical control strategy, the $Q(\lambda)$ -learning algorithm only and the reward-based GA. The classical control strategies of the pursuer and the evader are defined by Equations (6.3)–(6.4). The parameters of the $Q(\lambda)$ -learning algorithm only have the following values: the number of episodes is set to 1000, the number of steps in each episode is 6000, $\gamma = 0.5$ and $\lambda = 0.3$. The learning rate, α , and the value of ϵ are defined by Equation (6.16) and Equation (6.17), respectively. Note that in the proposed

³These values are chosen according to the criterion described in Section 3.4.

Table 6.1: Values of GAs parameters.

	Phase 2	
	Stage 1	Stage 2
Number of iterations	800	200
Population size	40	10
Number of steps	—	300
Crossover probability	0.2	0.2
Mutation probability	0.1	0.1
Fitness function	MSE defined by Equation (6.11)	capture time
Fitness function objective	minimize	minimize

QLBGFLC, we have shown how the number of episodes is set to 200. However, as the proposed QLBGFLC is compared to the $Q(\lambda)$ -learning algorithm only, we should consider to obtain the best performance of the $Q(\lambda)$ -learning algorithm only. In Figure 6.8, we can see that there is no improvement in the performance of the $Q(\lambda)$ -learning algorithm after 1000 episodes. Hence, in case of the $Q(\lambda)$ -learning algorithm only, we set the number of episodes to 1000.

The parameters of the reward-based GA are chosen as follows: the number of iterations = 1000, the population size = 40, the number of steps = 300, the probability of crossover = 0.2 and the probability of mutation = 0.1. Note that in phase 2 stage 2 of the proposed QLBGFLC, we already have a learned FLC (obtained from phase 2 stage 1) and in this stage, we just fine tune its parameters. By contrast, in the reward-based GA, the parameters of the FLC are unknown, thus being randomly initialized. Thereby, this random initialization will cause the pursuer not be able or take a longer time to catch the evader for some iterations. As a result, the learning time will increase.⁴

⁴Note that the learning time depends on the capture time as shown in Equation (6.14).

Figure 6.10 and Table 6.2 show the input and the output parameters of the FLC after tuning using the proposed QLBGFLC, respectively where "N", "Z" and "P" are referred to the linguistic values "Negative", "Zero" and "Positive". Table 6.3 shows the capture times for different initial positions of the evader using the classical control strategy, the $Q(\lambda)$ -learning algorithm only, the reward-based GA and the proposed QLBGFLC. Also, the learning times for the different learning techniques are shown in this table. From Table 6.3 we can see that although the $Q(\lambda)$ -learning algorithm only has the minimum learning time, it is not enough to get the desired performance in comparison with the classical control strategy. It is also noticed that although the performance of the reward-based GA approaches that of the classical control strategy, it has the maximum learning time. We can also see that the performance of the proposed QLBGFLC approaches that of the classical control strategy. In addition,

Figure 6.10: MFs for the inputs after tuning using the proposed QLBGFLC.

Table 6.2: Fuzzy decision table after tuning using the proposed QLBGFLC.

		$\dot{\delta}_p$		
		N	Z	P
δ_p	N	-1.0927	-0.4378	-0.7388
	Z	-0.5315	-0.2145	0.0827
	P	0.9100	0.1965	0.0259

Table 6.3: Capture time, in seconds, for different evader initial positions and learning time, in seconds, for the different techniques.

	Evader initial position				Learning time
	(-6,7)	(-7,-7)	(2,4)	(3,-8)	
Classical control strategy	9.6	10.4	4.5	8.5	—
Q(λ)-learning only	12.6	15.6	8.5	11.9	32.0
Reward-based GA	9.7	10.5	4.5	8.6	460.8
Proposed QLBGFLC	9.9	10.5	4.6	8.7	47.8

its learning time is only about 10.4% of the learning time of the reward-based GA.

Note that in the previous simulation the evader takes advantages of its maneuverability at the beginning of the game. The evader does not use the advantage of its maneuverability later in the game. In the next section we will do simulations in which the evader will take advantage of its maneuverability.

6.5.3 Evader using the advantage of higher maneuverability

In this section, we will increase the complexity of the model by using another version of the game. This time, we will make use of the advantage of the maneuverability of the evader during the game. Equations of motion for the pursuer and the evader are defined in Section 4.2 by Equations (4.1) – (4.2). The strategy of the evader is defined by Equations (4.4) – (4.5). The distance d defined in this strategy is calculated as

$$d = \frac{L_p}{\tan(u_{p_{max}})} \quad (6.18)$$

where d is the same as the minimum turning radius, $Rd_{p_{min}}$, defined by Equation (6.2). Equation (6.18) and Equation (4.5) mean that if the pursuer approach the evader and the distance between them is equal to the minimum turning radius of the

pursuer then the evader makes use of its higher maneuverability by turning to take the opposite direction of the pursuer.

The pursuer starts motion from the position $(0,0)$ with an initial orientation $\theta_p = 0.0$ and with a maximum velocity $V_{p_{max}} = 2.0$ m/s. The robot wheelbase $L_p = 0.3$ m and the steering angle $u_p \in [-0.5, 0.5]$. The evader starts motion from a random position at the beginning of each episode with an initial orientation $\theta_e = 0.0$ and with a maximum velocity $V_{e_{max}} = 1.0$ m/s which is half that of the pursuer (i.e. the evader is slower than the pursuer). The robot wheelbase $L_e = 0.3$ m and the steering angle $u_e \in [-1.0, 1.0]$ which is twice the steering angle of the pursuer (i.e. the evader is more maneuverable than the pursuer). The capture radius, ℓ , is set to 0.05 m and the sampling time is set to 0.1 seconds.

We choose the number of episodes to be 300 (not 200) as the evader uses an intelligent control strategy. Therefore, the pursuer needs more episodes to learn. We choose the number of steps in each episode to be 6000, $\gamma = 0.5$ and $\lambda = 0.3$. The learning rate, α and the value of ϵ are defined by Equation (6.16) and Equation (6.17), respectively. The position of the evader, (x_e, y_e) , is chosen randomly at the beginning of each episode to cover most of the states. Table 6.4 shows the values of GAs parameters used in phase 2 stage 1 and stage 2.

Figure 6.11 show the paths of the pursuer and the evader when the proposed QLBGFLC is used. From Figure 6.11 we can see that the evader can first escape from the pursuer by turning quickly when the distance between them is equal to the minimum turning radius of the pursuer. However, the pursuer can modify his path using its learned control strategy and can successfully catch the evader.

Table 6.4: Values of GAs parameters.

	Phase 2	
	Stage 1	Stage 2
Number of iterations	800	200
Population size	80	10
Number of steps	—	600
Crossover probability	0.2	0.2
Mutation probability	0.1	0.1
Fitness function	MSE defined by Equation (6.11)	capture time
Fitness function objective	minimize	minimize

6.5.4 Learning in a multi-robot system

In this section, we will increase the complexity of the model by making both the pursuer and the evader self-learn their control strategies simultaneously. The difficulty in the learning process is that each robot will try to find its optimal control strategy based on the control strategy of the other robot which, at the same time, is still learning. In addition, we assume that there is no information to share between the robots (such as rewards, states and actions).

In Section 6.4.2, we built the discrete state and action spaces for the Q-table of the pursuer. Now, we will build the state and action spaces for the Q-table of the evader. The state space of the evader, S , consists of the error in angle, δ_e and its derivative, $\dot{\delta}_e$. To build the state space, we discretize the ranges of the inputs, δ_e and $\dot{\delta}_e$, by 0.2. The ranges of δ_e and $\dot{\delta}_e$ are set to be from -1.0 to 1.0 so, the discretized values for δ_e and $\dot{\delta}_e$ will be $-1.0, -0.8, -0.6, \dots, 0.0, \dots, 0.8, 1.0$. There are 11 discretized values for δ_e and 11 discretized values for $\dot{\delta}_e$. These values are combined to form $11 \times 11 = 121$ states.

To build the action space, we discretize the range of the action, u_e , by 0.1. The

Figure 6.11: Paths of the pursuer ($\circ \circ \circ$) and the evader ($+++$) using the proposed QLBGFLC.

range of the action is set to be from $-u_{e_{max}}$ to $u_{e_{max}}$ i.e. from -1.0 to 1.0 so, the discretized values for u_e will be $-1.0, -0.9, -0.8, \dots, 0.0, \dots, 0.9, 1.0$. There are 21 actions and the dimension of the Q-table for the evader will be 121-by-21.

The parameters of the pursuit-evasion model, the $Q(\lambda)$ -learning algorithm and the GAs are the same as those defined in Section 6.5.2 except for the value of ϵ in the $Q(\lambda)$ -learning algorithm which is increased to do more exploration during the learning process. The reason is that as we said at the beginning of this section that each robot will try to explore its optimal control strategy based on the control strategy of the other robot which, at the same time, is still learning. Therefore, an action that is chosen as a greedy action at the beginning of a game may not be a greedy action after some learning. We make ϵ decrease with each episode such that

$$\epsilon = \frac{0.5}{i} \quad (6.19)$$

where i is the current episode. Note that the ability to explore an action using

Equation (6.19) is greater than the ability to explore an action using Equation (6.17).

Figure 6.12 and Table 6.5 show the input and the output parameters of the FLC for the pursuer using the proposed QLBGFLC. Figure 6.13 and Table 6.6 show the input and the output parameters of the FLC for the evader using the proposed QLBGFLC. The results shown in Figure 6.12 and Table 6.5 are different from those shown in Figure 6.13 and Table 6.6 because the pursuer and the evader have different capabilities (i.e. different velocities and different steering angles).

Figure 6.12: MFs for the inputs of the pursuer after tuning using the proposed QLBGFLC.

Table 6.5: Fuzzy decision table for the pursuer after tuning using the proposed QLBGFLC.

		$\dot{\delta}_p$		
		N	Z	P
δ_p	N	-0.4677	-0.2400	-0.7332
	Z	-0.8044	-1.2307	-0.2618
	P	0.8208	0.1499	0.9347

To check the performance of the different techniques we can not use the capture time as a measure, as we did in Section 6.5.2, because in this game both the pursuer and the evader are learning so we may find capture times that are smaller than those

Figure 6.13: MFs for the inputs of the evader after tuning using the proposed QLBGFLC.

Table 6.6: Fuzzy decision table for the evader after tuning using the proposed QLBGFLC.

		$\dot{\delta}_e$		
		N	Z	P
δ_e	N	-1.1479	-0.0022	-0.2797
	Z	-0.0529	-0.9777	-0.1257
	P	0.4332	0.4061	0.7059

corresponding to the classical control strategies. Of course that does not mean that the performance of the learned strategies is better than that of the classical control strategies but it means that the evader does not learn well and as a result it is captured in a shorter time. Therefore, the measure that we use is the paths of both the pursuer and the evader instead of the capture time. Figures 7.7–7.10 show the paths of the pursuer and the evader using the $Q(\lambda)$ -learning algorithm, the reward-based GA and the proposed QLBGFLC compared to the classical control strategies of the pursuer and the evader. We can see that the best performance is that of the proposed QLBGFLC. We can also see that the performance of the reward-based GA diminishes as a result of increasing the complexity of the system by making both the pursuer and the evader learn their control strategies simultaneously.

Figure 6.14: Paths of the pursuer and the evader (solid line) using the $Q(\lambda)$ -learning only compared to the classical control strategies (dotted line).

Figure 6.15: Paths of the pursuer and the evader (solid line) using the reward-based GA compared to the classical control strategies (dotted line).

Table 6.7 shows the learning time for the $Q(\lambda)$ -learning algorithm only, the reward-based GA and the proposed QLBGFLC. Table 6.7 shows that the proposed QLBGFLC has the minimum learning time. Finally, we can conclude that the proposed

Figure 6.16: Paths of the pursuer and the evader (solid line) using the proposed QLBGFLC compared to the classical control strategies (dotted line).

QLBGFLC has the best performance and the minimum learning time among all the other techniques.

Table 6.7: Learning time, in seconds, for the different techniques.

	Learning time
Q(λ)-learning only	62.8
Reward-based GA	602.5
Proposed QLBGFLC	54.7

6.6 Summary

In this chapter we proposed a novel technique to tune the input and the output parameters of a FLC. The proposed technique is applied to a pursuit-evasion game in which we assume that the pursuer does not know its control strategy. However it can self-learn its control strategy by interaction with the evader. Then we make use

of the advantage of maneuverability of the evader during the game. Finally, we apply the proposed technique to a pursuit-evasion game in which both the pursuer and the evader do not know their control strategies. Computer simulation and the results show that the proposed QLBGFLC technique outperforms all the other techniques in performance when compared with the classical control strategy and in the learning time which is also an important factor especially in on-line applications.

The limitation of the proposed technique is that we assume that the state and the action spaces can be discretized in such a way that make the resulting state and action spaces have acceptable dimensions. In Chapter 7, we propose a novel technique that overcomes this limitation.

Chapter 7

Q(λ)-Learning Fuzzy Inference System

7.1 Introduction

In Chapter 6, we use the proposed QLBGFLC to tune the input and the output parameters of a FLC without any a priori knowledge about the controlled model. The proposed QLBGFLC can be considered as an "*indirect*" technique of using function approximation. First, the state and the action spaces are discretized and a Q(λ)-learning algorithm is used to obtain an estimate of the desired training data set. Then, the training data set is used to generalize the discrete state-action space over the continuous space using a FLC supported by a GA as an optimization technique that is used to tune the input and the output parameters of the FLC.

As mentioned in Chapter 6, the proposed QLBGFLC is used if the state and the action spaces can be relatively coarsely discretized such that the state and the action spaces are not prohibitively large. However, if this condition can not be satisfied

then we need to search for another technique that "*directly*" makes use of function approximation i.e. directly deals with continuous state and action spaces without the need to do the discretization process first.

A number of articles used FIS as a function approximation with Q-learning [37, 69, 70]. However, these works have the following disadvantages: (i) the action space is considered to be discrete and (ii) only the output parameters of the FIS are tuned. In this chapter, we propose a novel technique called *Q(λ)-learning fuzzy inference system* (QLFIS) that is used to design a self-learning FLC [99]. The proposed technique is used "*directly*" with the continuous state and action spaces to tune the input and the output parameters of FLCs. The learning process in the proposed QLFIS is performed simultaneously in which a FIS is used as a function approximation to estimate the optimal action-value function, $Q^*(s, a)$, of the Q(λ)-learning algorithm in the continuous state and action spaces while the Q(λ)-learning algorithm is used to tune the input and the output parameters of both the FIS "the function approximation" and the FLC "the controller".

The proposed technique is applied to three versions of pursuit-evasion differential games. As in Chapter 6, we start with a simple pursuit-evasion game in which only the pursuer self-learns its control strategy on-line while the evader plays a simple classical control strategy. In the second game, we make the evader play an intelligent control strategy by exploiting the advantage of its higher maneuverability during the game. Finally, we make both the pursuer and the evader self-learn their control strategies on-line by the interaction with each other. We compare the proposed technique with the classical control strategy, the Q(λ)-learning algorithm only and the technique proposed by Dai et al. [1]. We will also compare our results to those obtained in Chapter 6 as we will use the same pursuit-evasion models.

This chapter is organized as follows: in Section 7.2 the proposed technique is described. Section 7.3 discusses the computer simulation and the results.¹

7.2 $Q(\lambda)$ -learning Fuzzy Inference System

The FIS used in this chapter has the same structure as the one shown in Figure 2.6. The FIS is used as a function approximation for $Q(\lambda)$ -learning to generalize the discrete state and action spaces into continuous state and action spaces. At the same time, $Q(\lambda)$ -learning is used to tune the input and the output parameters of the FIS and the FLC. The structure of the proposed QLFIS is shown in Figure 7.1 which is a modified version of the proposed techniques used by Dai et al. [1] (Q-learning with NN as a function approximation) and Givigi et al. [53] (actor-critic learning with FIS as a function approximation).

Figure 7.1: The proposed QLFIS technique.

The difference between the proposed QLFIS and the technique proposed by Givigi

¹Note that in this chapter we will use the same pursuit-evasion models as those that are used in Chapter 6.

et al. [53] is that in [53], a FIS is used to approximate the value function, $V(s)$, but in the proposed QLFIS, the FIS is used to approximate the action-value function, $Q(s, a)$. In addition, only the output parameters of the FIS and the FLC are tuned in [53] while in this work the input and the output parameters of the FIS and the FLC are tuned. The reason for choosing Q-learning in our work is that it outperforms the actor-critic learning [97]. The main advantage of Q-learning over actor-critic learning is exploration insensitivity since Q-learning is an off-policy algorithm whereas actor-critic learning is an on-policy algorithm (see Section 2.5).

The difference between the proposed QLFIS and the technique proposed by Dai et al. [1] is that in [1], a NN is used as a function approximation but here we use a FIS as a function approximation. There are some advantages of using a FIS rather than a NN such as: (i) linguistic fuzzy rules can be obtained from human experts [77] and (ii) the ability to represent fuzzy and uncertain knowledge [100]. In addition, our results show that the proposed QLFIS outperforms the technique proposed by Dai et al. [1] in both the performance and the learning time.

Now we will derive the adaptation laws for the input and the output parameters of the FIS and the FLC. The adaptation laws will be derived only once and are applied to both the FIS and the FLC.² Our objective is to minimize the TD-error, Δ_t , defined by Equation (2.44). By using the MSE we can formulate the error as

$$E = \frac{1}{2} \Delta_t^2 \quad (7.1)$$

We use the gradient descent approach and according to the steepest descent algorithm,

²Note that the derivation of the adaptation laws are taken from Section 2.4 as we use the same FIS structure that is discussed there.

we make a change along the $-ve$ gradient to minimize the error so,

$$\phi(t+1) = \phi(t) - \eta \frac{\partial E}{\partial \phi} \quad (7.2)$$

where η is the learning rate and ϕ is a vector of the input and the output parameters of the FIS to be tuned where

$$\phi = \begin{pmatrix} \sigma \\ m \\ K \end{pmatrix} \quad (7.3)$$

By differentiating Equation (7.1) with respect to the vector ϕ , we get

$$\frac{\partial E}{\partial \phi} = \Delta_t \frac{\partial \Delta_t}{\partial \phi} \quad (7.4)$$

Then differentiate Equation (2.44) with respect to $Q_t(s_t, a_t)$ and substitute in Equation (7.4) using the chain rule, we get

$$\frac{\partial E}{\partial \phi} = -\Delta_t \frac{\partial Q_t(s_t, u_t)}{\partial \phi} \quad (7.5)$$

Substituting in Equation (7.2), we get

$$\phi(t+1) = \phi(t) + \eta \Delta_t \frac{\partial Q_t(s_t, u_t)}{\partial \phi} \quad (7.6)$$

We can obtain $\frac{\partial Q_t(s_t, u_t)}{\partial \phi}$ for the output parameter, K_l , by differentiating Equation (2.14) with respect to K_l , where f in Equation (2.14) is substituted by $Q_t(s_t, u_t)$ for

the FIS and is substituted by u for the FLC, as follows

$$\frac{\partial Q_t(s_t, u_t)}{\partial K_l} = \bar{\omega}_l \quad (7.7)$$

Then we can obtain $\partial Q_t(s_t, u_t)/\partial \phi$ for the input parameters, σ_i^l and m_i^l , based on the chain rule,

$$\frac{\partial Q_t(s_t, u_t)}{\partial \sigma_i^l} = \frac{\partial Q_t(s_t, u_t)}{\partial \omega_l} \frac{\partial \omega_l}{\partial \sigma_i^l} \quad (7.8)$$

$$\frac{\partial Q_t(s_t, u_t)}{\partial m_i^l} = \frac{\partial Q_t(s_t, u_t)}{\partial \omega_l} \frac{\partial \omega_l}{\partial m_i^l} \quad (7.9)$$

The term $\frac{\partial Q_t(s_t, u_t)}{\partial \omega_l}$ is calculated from Equation (2.14) and Equation (2.12). The terms $\frac{\partial \omega_l}{\partial \sigma_i^l}$ and $\frac{\partial \omega_l}{\partial m_i^l}$ are calculated from Equation (2.11) and Equation (2.2) so

$$\frac{\partial Q_t(s_t, u_t)}{\partial \sigma_i^l} = \frac{(K_l - Q_t(s_t, u_t))}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)^2}{(\sigma_i^l)^3} \quad (7.10)$$

$$\frac{\partial Q_t(s_t, u_t)}{\partial m_i^l} = \frac{(K_l - Q_t(s_t, u_t))}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)}{(\sigma_i^l)^2} \quad (7.11)$$

Substituting from Equation (7.7), Equation (7.10) and Equation (7.11) in Equation (2.49) and modifying Equation (7.6) to use eligibility trace, the update law for the FIS parameters becomes

$$\phi_Q(t+1) = \phi_Q(t) + \eta \Delta_t e_t \quad (7.12)$$

Substituting from Equation (7.3) into Equation (7.12) then we get

$$\begin{pmatrix} \sigma_i^l(t+1) \\ m_i^l(t+1) \\ K_i^l(t+1) \end{pmatrix}_Q = \begin{pmatrix} \sigma_i^l(t) \\ m_i^l(t) \\ K_i^l(t) \end{pmatrix}_Q + \eta \Delta_t \left\{ \gamma \lambda e_{t-1} + \begin{pmatrix} \frac{(K_l - Q_t(s_t, u_t))}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)^2}{(\sigma_i^l)^3} \\ \frac{(K_l - Q_t(s_t, u_t))}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)}{(\sigma_i^l)^2} \\ \sum_l \bar{\omega}_l \end{pmatrix}_Q \right\} \quad (7.13)$$

The update law in Equation (7.6) is applied also to the FLC by replacing $Q_t(s_t, u_t)$ with the output of the FLC, u . In addition and as shown from Figure 7.1, a random Gaussian noise, $n(0, \sigma_n)$, with zero mean and standard deviation σ_n is added to the output of the FLC in order to solve the exploration/exploitation dilemma such as using the ϵ -greedy exploration method in the discrete state and action spaces. Then the update law for the input and the output parameters of the FLC is defined by

$$\phi_u(t+1) = \phi_u(t) + \xi \Delta_t \frac{\partial u}{\partial \phi_u} \left(\frac{u_c - u}{\sigma_n} \right) \quad (7.14)$$

where u_c is the output of the random Gaussian noise generator and ξ is the learning rate for the input and the output parameters of the FLC. The term $\frac{\partial u}{\partial \phi_u}$ can be calculated by replacing $Q_t(s_t, u_t)$ with the output of the FLC, u , in Equation (7.7), Equation (7.10) and Equation (7.11) then substituting into Equation (7.14) we get

$$\begin{pmatrix} \sigma_i^l(t+1) \\ m_i^l(t+1) \\ K_i^l(t+1) \end{pmatrix}_u = \begin{pmatrix} \sigma_i^l(t) \\ m_i^l(t) \\ K_i^l(t) \end{pmatrix}_u + \xi \Delta_t \left(\frac{u_c - u}{\sigma_n} \right) \begin{pmatrix} \frac{(K_l - u)}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)^2}{(\sigma_i^l)^3} \\ \frac{(K_l - u)}{\sum_l \omega_l} \omega_l \frac{2(x_i - m_i^l)}{(\sigma_i^l)^2} \\ \sum_l \bar{\omega}_l \end{pmatrix}_u \quad (7.15)$$

7.3 Computer Simulation and Results

We use a core 2 duo with a 2.0 GHz clock frequency and 4.0 Gigabytes of RAM. We do computer simulation with MATLAB software. $Q(\lambda)$ -learning has many parameters to be set a priori therefore we tested computer simulation for different parameter values and different parameter value combinations and chose the values that give the best performance. The initial position of the evader is randomly chosen from a set of 64 different positions in the space.

7.3.1 Evader using a simple control strategy

The pursuer starts motion from the position $(0, 0)$ with an initial orientation $\theta_p = 0.0$ and with a constant velocity $V_p = 2.0$ m/s. The robot wheelbase $L_p = 0.3$ m and the steering angle $u_p \in [-0.5, 0.5]$. The evader starts motion from a random position for each episode with an initial orientation $\theta_e = 0.0$ and with a constant velocity $V_e = 1.0$ m/s which is half that of the pursuer (i.e. the evader is slower than the pursuer). The robot wheelbase $L_e = 0.3$ m and the steering angle $u_e \in [-1.0, 1.0]$ which is twice that of the pursuer (i.e the evader is more maneuverable than the pursuer). The capture radius, ℓ , is set to 0.10 m and the sampling time is set to 0.1 seconds.

For the proposed QLFIS, we choose the number of episodes (games) to be 1000. the number of steps (plays) in each episode is 600, $\gamma = 0.95$ and $\lambda = 0.9$. We make the learning rate for the FIS, η , decrease with each episode such that

$$\eta = 0.1 - 0.09 \left(\frac{i}{\text{Max. Episodes}} \right) \quad (7.16)$$

We also make the learning rate for the FLC, ξ , decrease with each episode such that

$$\xi = 0.01 - 0.009 \left(\frac{i}{\text{Max. Episodes}} \right) \quad (7.17)$$

where i is the current episode. Note that the value of η is 10 times the value of ξ i.e. the FIS converges faster than the FLC to avoid instability in tuning the parameters of the FLC. The initial input and output parameters of the FLC are shown in Figure 7.2 and Table 7.1, respectively. Note that, if the pursuer uses a FLC with these initial parameters, the pursuer will not catch the evader. The FIS has the same initial parameters as the FLC. We choose $\sigma_n = 0.08$. The duration of a game equals to *the number of steps $\times$ sampling time* = 60 seconds. The game ends when 60 seconds passed without capturing or when the capture occurs before the end of this time. The learning process in the proposed QLFIS is described in detail in Algorithm 7.3.1.

Figure 7.2: Initial MFs of the FLC inputs.

To validate the proposed QLFIS, we compare its results with the results of the classical control strategy, the technique proposed by Dai et al. [1] and the techniques introduced in Chapter 6 (including the $Q(\lambda)$ -learning algorithm only, the reward-based GA and the proposed QLBGFLC).

Table 7.1: Initial fuzzy decision table.

		δ_p		
		N	Z	P
δ_p	N	-0.50	-0.25	0.0
	Z	-0.25	0.0	0.25
	P	0.0	0.25	0.50

For the technique proposed by Dai et al. [1], we choose the same values for the parameters of the NN. The NN has a three-layer structure with 7-21-1 nodes. The RL parameters and the initial values of the input and the output parameters of the FLC are all chosen to be the same as those chosen in the proposed QLFIS. We choose $\sigma_n = \frac{0.1}{i}$ which is decreasing each episode where i is the current episode.

Figure 7.3 and Table 7.2 show the input and the output parameters of the FLC after tuning using the proposed QLFIS, respectively. Table 7.3 shows the capture times for different initial positions of the evader using the classical control strategy of the pursuer, the $Q(\lambda)$ -learning algorithm only, the proposed QLFIS, the technique proposed by Dai et al. [1], the reward-based GA and the proposed QLBGFLC. In addition, the learning times for the different techniques are also shown in this table. From Table 7.3 we can see that the proposed QLFIS outperforms the technique proposed by Dai et al. [1] in both performance and learning time and both of them have better performance than using the $Q(\lambda)$ -learning algorithm only. We can also see that the proposed QLBGFLC still has the best performance when compared with all the other techniques.

Note that for the comparative purposes, we set the number of episodes and the number of steps of the $Q(\lambda)$ -learning algorithm that is used in the technique proposed by Dai et al. [1] the same as those of the $Q(\lambda)$ -learning algorithm that is used in the

Algorithm 7.3.1 Learning in the proposed QLFIS.

-
- 1: Initialize the input and the output parameters of the FLC as shown in Figure 7.2 and Table 7.1, respectively.
 - 2: Initialize the input parameters of the FIS with the same values as those of the FLC and initialize the output parameters of the FIS to zeros. Note that the output of the FIS is a Q-value.
 - 3: Initialize the eligibility traces, e , for the input and the output parameters of the FIS to zeros.
 - 4: Set $\gamma \leftarrow 0.95$, $\lambda \leftarrow 0.9$ and $\sigma_n \leftarrow 0.08$.
 - 5: **for** each episode (game) **do**
 - 6: Calculate η from Equation (7.16) and calculate ξ from Equation (7.17).
 - 7: Initialize the position of the pursuer, (x_p, y_p) to $(0, 0)$.
 - 8: Initialize the position of the evader, (x_e, y_e) , randomly.
 - 9: Calculate the initial state, $s = (\delta, \dot{\delta})$, from Equation (6.4).
 - 10: Calculate the output of the FLC, u , using the weighted average defuzzification method from Equation (2.8).
 - 11: **for** each step (play) **do**
 - 12: Calculate the output $u_c = u + n(0, \sigma_n)$.
 - 13: Calculate the output of the FIS, $Q(s, u)$, using the weighted average defuzzification method from Equation (2.8).
 - 14: Run the game for the current step and observe the next state s_{t+1} .
 - 15: Get the reward, r , from Equation (6.10).
 - 16: Calculate the Q-value of the next state, $Q(s_{t+1}, u')$, which is the output of the FIS, from Equation (2.8).
 - 17: Calculate the TD-error, Δ_t , from Equation (2.44).
 - 18: Calculate the gradient for the input and the output parameters of the FLC and the FIS from Equation (7.7), Equation (7.10) and Equation (7.11).
 - 19: Update the eligibility traces, e_t , for the input and the output parameters of the FIS from Equation (2.49).
 - 20: Update the input and the output parameters of the FIS from Equation (7.13).
 - 21: Update the input and the output parameters of the FLC from Equation (7.15).
 - 22: Set $s_t \leftarrow s_{t+1}$ and $u \leftarrow u'$.
 - 23: **end for**
 - 24: **end for**
-

Figure 7.3: MFs for the inputs after tuning using the proposed QLFIS.

Table 7.2: Fuzzy decision table after tuning using the proposed QLFIS.

		$\dot{\delta}_p$		
		N	Z	P
δ_p	N	-0.9526	-0.4411	-0.4300
	Z	-0.3962	-0.0247	0.3680
	P	0.4114	0.3905	0.9309

proposed QLFIS. The reason for that is to evaluate the performance and the learning time of these techniques under the same conditions. In other words, we can enhance the performance of the technique proposed by Dai et al. [1] by increasing the number of episodes and/or the number of steps. However, this enhancement will be at the cost of increasing the learning time.

7.3.2 Evader using the advantage of higher maneuverability

Now we will use another version of the game. This time, we will make use of the advantage of the maneuverability of the evader during the game. Equations of motion for the pursuer and the evader are defined in Section 4.2 by Equations (4.1)–(4.2). The strategy of the evader is defined by Equations (4.4)–(4.5). The pursuer starts

Table 7.3: Capture time, in seconds, for different evader initial positions and learning time, in seconds, for the different techniques.

	Evader initial position				Learning time
	(-6,7)	(-7,-7)	(2,4)	(3,-8)	
Classical control strategy	9.6	10.4	4.5	8.5	—
Q(λ)-learning only	12.6	15.6	8.5	11.9	32.0
Dai et al. [1]	10.9	12.9	4.7	9.1	258.6
Proposed QLFIS	10.0	10.7	4.6	8.8	65.2
Reward-based GA	9.7	10.5	4.5	8.6	460.8
Proposed QLBGFLC	9.9	10.5	4.6	8.7	47.8

motion from the position $(0,0)$ with an initial orientation $\theta_p = 0.0$ and with a maximum velocity $V_{p_{max}} = 2.0$ m/s. The robot wheelbase $L_p = 0.3$ m and the steering angle $u_p \in [-0.5, 0.5]$. The evader starts motion from a random position at the beginning of each episode with an initial orientation $\theta_e = 0.0$ and with a maximum velocity $V_{e_{max}} = 1.0$ m/s which is half that of the pursuer (i.e. the evader is slower than the pursuer). The robot wheelbase $L_e = 0.3$ m and the steering angle $u_e \in [-1.0, 1.0]$ which is twice the steering angle of the pursuer (i.e. the evader is more maneuverable than the pursuer). The capture radius, ℓ , is set to 0.05 m and the sampling time is set to 0.1 seconds. We also use the same parameters of the proposed QLFIS defined in Section 7.3.1.

Figure 7.4 shows the paths of the pursuer and the evader when the proposed QLFIS is used. From this figure we can see that the evader can first escape from the pursuer by turning quickly when the distance between them is equal to the minimum turning radius of the pursuer. However, the pursuer can modify his path using its learned control strategy and can successfully catch the evader.

7.3.3 Learning in a multi-robot system

Now, we will increase the complexity of the model by making both the pursuer and the evader self-learn their control strategies simultaneously. In addition, we assume that there is no information to share between the robots (such as rewards, states and actions). The parameters of the pursuit-evasion model, the parameters of the technique proposed by Dai et al. [1] and the parameters of the proposed QLFIS are the same as those defined in Section 7.3.1.

Figure 7.5 and Table 7.4 show the input and the output parameters of the FLC for the pursuer using the proposed QLFIS, respectively. Figure 7.6 and Table 7.5 show the input and the output parameters of the FLC for the evader using the proposed QLFIS, respectively.

To check the performance of the different techniques we can not use the capture time as a measure, as we did in Section 7.3.1, because in this game both the pursuer and the evader are learning so we may find capture times that are smaller than

Figure 7.4: Paths of the pursuer (ooo) and the evader (+++) using the proposed QLFIS.

those corresponding to the classical control strategies. Of course that does not mean that the performance of the learned strategies is better than the performance of the classical control strategies but it means that the evader does not learn well and as a result it is captured in a shorter time. Therefore, the measure that we use is the paths of both the pursuer and the evader instead of the capture time.

Figure 7.5: MFs for the inputs of the pursuer after tuning using the proposed QLFIS.

Table 7.4: Fuzzy decision table for the pursuer after tuning using the proposed QLFIS.

		$\dot{\delta}_p$		
		N	Z	P
δ_p	N	-1.2990	-0.6134	-0.4064
	Z	-0.3726	-0.0097	0.3147
	P	0.3223	0.5763	0.9906

Figures 7.7–7.11 show the paths of the pursuer and the evader using the $Q(\lambda)$ -learning algorithm only, the reward-based GA, the technique proposed by Dai et al. [1], the proposed QLBGFLC, and the proposed QLFIS, respectively, compared to the classical control strategies of the pursuer and the evader. We can see that the proposed QLFIS has better performance than the technique proposed by Dai et al.

Figure 7.6: MFs for the inputs of the evader after tuning using the proposed QLFIS.

Table 7.5: Fuzzy decision table for the evader after tuning using the proposed QLFIS.

		$\hat{\delta}_e$		
		N	Z	P
δ_e	N	-1.4827	-0.4760	-0.0184
	Z	-0.5365	-0.0373	0.5500
	P	-0.0084	0.4747	1.1182

[1]. We can also see that both the proposed QLBGFLC and the proposed QLFIS have almost the same performance that outperforms the performance of the other techniques when they are compared to the classical control strategies.

Table 7.6 shows the learning time for the $Q(\lambda)$ -learning algorithm only, the reward-based GA, the technique proposed by Dai et al. [1], the proposed QLBGFLC, and the proposed QLFIS. From Table 7.6, we can see that the proposed QLFIS still outperforms the technique proposed by Dai et al. [1] in both performance and learning time. We can also see that the proposed QLBGFLC has the minimum learning time. Finally, we can conclude that the proposed QLBGFLC has the best performance and the best learning time among all the other techniques.

Figure 7.7: Paths of the pursuer and the evader (solid line) using the $Q(\lambda)$ -learning only compared to the classical control strategies (dotted line).

Figure 7.8: Paths of the pursuer and the evader (solid line) using the reward-based GA compared to the classical control strategies (dotted line).

7.4 Summary

In this chapter we proposed a novel technique to tune the input and the output parameters of a FLC automatically. The proposed QLFIS combines RL with FIS

Figure 7.9: Paths of the pursuer and the evader (solid line) using the technique proposed by Dai et al. [1] compared to the classical control strategies (dotted line).

Figure 7.10: Paths of the pursuer and the evader (solid line) using the proposed QLBGFLC compared to the classical control strategies (dotted line).

as a function approximation to generalize the state and the action spaces to the continuous case. First, we apply the proposed technique to a simple pursuit-evasion

Figure 7.11: Paths of the pursuer and the evader (solid line) using the proposed QLFIS compared to the classical control strategies (dotted line).

Table 7.6: Learning time, in seconds, for the different techniques.

	Learning time
Q(λ)-learning only	62.8
Dai et al. [1]	137.0
Reward-based GA	602.5
Proposed QLBGFLC	54.7
Proposed QLFIS	110.3

game in which the pursuer does not know its control strategy and the evader plays a classical control strategy. The pursuer can self-learn its control strategy by interaction with the evader. Second, we apply the proposed technique to a pursuit-evasion game in which we make use of the higher maneuverability of the evader during the game. Finally, we apply the proposed technique to a pursuit-evasion game in which both the pursuer and the evader do not know their control strategies. Computer simulation and the results show that the proposed QLFIS outperforms the compared technique used

by Dai et al. [1] in performance and in the learning time which is also an important factor especially in on-line applications. We also show that the proposed QLBGFLC introduced in Chapter 6 still outperforms all the other techniques that it is compared with.

In Chapter 8 we will move on to the multi-pursuit-evasion game in which more than one pursuer and more than one evader are interacting and are learning and adapting their control strategies.

Chapter 8

Learning in Multi-Player Pursuit-Evasion Games

8.1 Introduction

In this chapter, two different $Q(\lambda)$ -learning algorithms are introduced. These algorithms are the minimax $Q(\lambda)$ -learning algorithm, which is one type of team learning algorithm [101] and the independent $Q(\lambda)$ -learning (IQL) algorithm as one type of independent learning algorithm. A comparison between these algorithms is discussed to show the advantages and disadvantages of each algorithm. The reason for introducing and comparing these algorithms is the need to use a suitable algorithm since we will extend the single-robot model used in Chapter 6 to the multi-robot case. We then introduce learning in a two-pursuer one-evader game. We show that if the pursuers can get more information about the evader, then they can learn to catch the evader faster. Finally, we present learning in the n -pursuer n -evader game.

This chapter is organized as follows: Section 8.2 discusses the link between RL

and the multi-robot systems. In Section 8.3 a comparison between the minimax $Q(\lambda)$ -learning algorithm and the IQL algorithm is introduced. In Section 8.4, learning in a two-pursuer one-evader game is described. Finally, learning in n -pursuer n -evader games is showed in Section 8.5.

8.2 Multi-Robot Reinforcement Learning

In general, combining RL with multi-agent systems (MASs) is known as multi-agent reinforcement learning (MARL) [102]. MARL has some benefits and some challenges comparing with a single-agent reinforcement learning.

- **Benefits of MARL:**

1. *Learning time:* It is reduced due to parallel computations.
2. *Experience acquirement:* Agent with similar tasks can share their experience and hence accelerate the process of learning.
3. *Goal achievement:* When one or more agents fail to achieve their tasks, the remaining agents can do some of these tasks.

- **Challenges in MARL:**

1. *Dimensionality:* The state and the action spaces will increase exponentially.
2. *Rewards:* The state and the action spaces will be correlated and can not be maximized independently.
3. *Learning process:* The environment will be non-stationary (dynamic) since agents are learning simultaneously.

4. *Actions*: An agent, when executing an action, must take into account other agents in the system.
5. *Exploration/exploitation dilemma*: The model will be more complex since the agent will explore not only the environment but also the other agents.
6. *Communications*: There must be some type of communication between agents to share similar tasks, know their actions and observe their rewards.

When we talk about MARL, we have to talk about stochastic game (SG) which is an extension from MDP in a single-agent RL model to a Markov game (MG) in MARL. SG is a tuple $\langle G, S, \{A_i\}_{i \in G}, P, R_{i \in G} \rangle$ where:

- $G = 1, 2, \dots, n$ is the set of agents.
- S is the finite state space.
- A_i is the finite set of actions for agent i .
- $P : S \times A \times S \longrightarrow [0, 1]$ is the state transition probability.
- $R_i : S \times A \times S \longrightarrow \mathbb{R}$ is the reward function for agent i .

There has been some research on MARL in multi-robot systems (MRSs) [103, 104]. A MRS is a group of robots that coexist and interact with one another. Therefore, It can do tasks that are difficult to be done by a single-robot system. Interactions in MRS can be cooperative, competitive, or mixed. In fully cooperative interactions, $R_1 = R_2 = \dots = R_n$ (general sum game) whereas, in fully competitive interactions, $R_1 + R_2 + \dots + R_n = 0$ (zero sum game). Note that the pursuit-evasion models used in Section 6.5.4 and Section 7.3.3 are considered as simple SGs. In these models,

two players are learning their strategies with opposite goals and the game is fully competitive (i.e. a zero sum game in which $r_p + r_e = 0$).

In MRS, the complexity and uncertainty in the environment will increase. The environment will be dynamic. As the number of robots increases, the state and the action spaces will increase and then we will face a problem known as *curse of dimensionality*. In addition, the action that the robot takes must depend not only on the current state but also on the actions taken by the other robots in the system. This type of action is called *joint action*.

8.3 Minimax $Q(\lambda)$ -learning Compared to Independent $Q(\lambda)$ -learning

In this section we compare between the minimax- $Q(\lambda)$ -learning algorithm and the independent $Q(\lambda)$ -learning algorithm (IQL). Both algorithms are applied to a multi-robot system. The multi-robot system is a pursuit-evasion game in which both the pursuer and the evader do not know their control strategies. They self-learn their control strategies on-line by interaction with each other. The pursuit-evasion game used is the same as the one that is described in Section 6.3.

The minimax $Q(\lambda)$ -learning algorithm has only one Q -table for both the pursuer and the evader. The state-action value, $Q(s, a, o)$, is calculated by

$$Q_{t+1}(s, a, o) = Q_t(s, a, o) + \alpha_t e_t(s, a, o) \left(r_{t+1} + \gamma \max_{a' \in A} \min_{o' \in O} Q_t(s', a', o') - Q_t(s, a, o) \right) \quad (8.1)$$

where s is the current state which consists of the error in angle of the pursuer, δ_p and the error in angle of the evader, δ_e , s' is the next state, a is the action of the

pursuer (which is the steering angle of the pursuer, u_p), o is the action of the evader (which is the steering angle of the evader, u_e), r_{t+1} is the immediate reward defined by Equation (6.10), α_t is the learning rate, e is the eligibility trace and γ is the discount factor.

The IQL algorithm has two Q-tables, one for the pursuer and one for the evader (as we did in Section 6.5.4). The state-action value of the pursuer, $Q^p(s, a)$, is defined as

$$Q_{t+1}^p(s, a) = Q_t^p(s, a) + \alpha_t^p e_t^p(s, a) \left(r_{t+1}^p + \gamma \max_{a' \in A} Q_t^p(s', a') - Q_t^p(s, a) \right) \quad (8.2)$$

and the state-action value of the evader, $Q^e(s, o)$, is defined as

$$Q_{t+1}^e(s, o) = Q_t^e(s, o) + \alpha_t^e e_t^e(s, o) \left(r_{t+1}^e + \gamma \max_{o' \in O} Q_t^e(s', o') - Q_t^e(s, o) \right) \quad (8.3)$$

where r_{t+1}^p and r_{t+1}^e are the immediate rewards of the pursuer and the evader, respectively. Note that in this game $r_{t+1}^e = -r_{t+1}^p$.

The pursuer starts motion from position $(x_p, y_p) = (0, 0)$, $\theta_p = 0.0$, $V_p = 1.0$ m/s, $u_p \in [-0.5, 0.5]$ and $L_p = 0.3$ m. The evader starts motion from a random position (x_e, y_e) , $\theta_e = 0.0$, $V_e = 0.5$ m/s, $u_e \in [-1.0, 1.0]$ and $L_e = 0.3$ m. The capture radius, $\ell = 0.1$ m.

In the minimax $Q(\lambda)$ -learning algorithm, the state space, S , consists of δ_p and δ_e where $\delta_p \in [-1.0, 1.0]$ and $\delta_e \in [-1.0, 1.0]$. The ranges of δ_p and δ_e are discretized by 0.2 to get the values $-1.0, -0.8, \dots, 0.0, \dots, 0.8, 1.0$. These values are then combined and the state, s , takes the values $(-1.0, -1.0), (-1.0, -0.8), \dots, (0.0, 0.0), \dots, (0.8, 1.0), (1.0, 1.0)$. The state space, S , has $(11)^2 = 121$ states. Note that if the state space, S , consists of $\delta_p, \dot{\delta}_p, \delta_e$ and $\dot{\delta}_e$ then the number of states will be

$(11)^4 = 14,641$ which is prohibitively large. Therefore and for simplicity, the state space, S , consists only of δ_p and δ_e .

The action space of the pursuer, A , is discretized by 0.1 and the action, a , takes the values $-0.5, -0.4, \dots, 0.0, \dots, 0.4, 0.5$. The action space of the evader, O , is discretized by 0.1 and the action, o , takes the values $-1.0, -0.8, \dots, 0.0, \dots, 0.8, 1.0$. Therefore, we have 11 actions for the pursuer and 21 actions for the evader. The actions of the pursuer and the actions of the evader are combined to form $11 \times 21 = 231$ joint actions and the size of the Q-table will be 121-by-231.

The parameters of the minimax $Q(\lambda)$ -learning algorithm are: the number of episodes (games) = 200, the number of steps (plays) in each episode = 6000, $\gamma = 0.5$, $\lambda = 0.3$. The value of the learning rate, α , is set so as to decrease from episode to episode as follows

$$\alpha = \left(\frac{1.0}{i} \right)^{0.7} \quad (8.4)$$

where i is the current episode. We use the ϵ -greedy action selection method defined by Equation (2.54) in which the value of ϵ is set so as to decrease from episode to episode as follows

$$\epsilon = \frac{1.0}{i} \quad (8.5)$$

In the IQL algorithm, the state space of the pursuer, S_p , consists of δ_p and $\dot{\delta}_p$ where $\delta_p \in [-1.0, 1.0]$ and $\dot{\delta}_p \in [-1.0, 1.0]$. The ranges of δ_p and $\dot{\delta}_p$ are discretized by 0.2 to get the values $-1.0, -0.8, \dots, 0.0, \dots, 0.8, 1.0$. These values are then combined and the state, s_p , takes the values $(-1.0, -1.0), (-1.0, -0.8), \dots, (0.0, 0.0), \dots, (0.8, 1.0), (1.0, 1.0)$. The state space of the evader, S_e , consists of δ_e and $\dot{\delta}_e$ where $\delta_e \in [-1.0, 1.0]$ and $\dot{\delta}_e \in [-1.0, 1.0]$. The ranges of δ_e and $\dot{\delta}_e$ are discretized by 0.2 to get the values $-1.0, -0.8, \dots, 0.0, \dots, 0.8, 1.0$. These values are then combined and the state, s_e ,

takes the values $(-1.0, -1.0), (-1.0, -0.8), \dots, (0.0, 0.0), \dots, (0.8, 1.0), (1.0, 1.0)$. The state space of the pursuer, S_p , has 121 states and the state space of the evader, S_e , has 121 states.

The action space of the pursuer, A , is discretized by 0.1 and the action, a , takes the values $-0.5, -0.4, \dots, 0.0, \dots, 0.4, 0.5$ therefore, the action space of the pursuer, A , has 11 actions. The pursuer has a Q-table, Q_p , of size 121-by-11. The action space of the evader, O , is discretized by 0.1 and the action, o , takes the values $-1.0, -0.8, \dots, 0.0, \dots, 0.8, 1.0$ therefore, the action space of the evader, O , has 21 actions. The evader has a Q-table, Q_e , of size 121-by-21.

For comparative purposes, the parameters of the IQL algorithm are chosen the same as those of the minimax $Q(\lambda)$ -learning algorithm except for the value of ϵ that is chosen as the one defined by Equation (8.6) as follows

$$\epsilon = \frac{0.5}{i} \quad (8.6)$$

where i is the current episode. Note that the value of ϵ using the minimax $Q(\lambda)$ -learning algorithm that is defined by Equation (8.5) is greater than the value of ϵ using the IQL algorithm that is defined by Equation (8.6). The reason is that in the minimax $Q(\lambda)$ -learning algorithm, we need to explore between 231 actions for each state while in the IQL algorithm, we need only to explore between 11 actions for each state of the pursuer and 21 actions for each state of the evader. Therefore, the ability to explore an action using Equation (8.5) is greater than the ability to explore an action using Equation (8.6).

We run the simulation of the minimax $Q(\lambda)$ -learning algorithm and the IQL algorithm 100 times then we compute the average of the capture time and the average of

the total reward gained in a game over the 100 runs. Now, we will make a comparison between the two learning algorithms from several points of view. Before we do this comparison, we will make a small analysis regarding the effect of the Q-table size on the learning process.

In the simulation process, we have 200 episodes and 6000 steps in each episode with a total of $200 \times 6000 = 1,200,000$ steps in a simulation. The size of the Q-table in the minimax $Q(\lambda)$ -learning algorithm is 121-by-231 i.e. the Q-table has $121 \times 231 = 27,951$ state-action pairs. In the IQL algorithm, the Q-table of the pursuer is of size 121-by-11 i.e. the Q-table of the pursuer has $121 \times 11 = 1,331$ state-action pairs. In addition, the Q-table of the evader is of size 121-by-21 i.e. the Q-table of the evader has $121 \times 21 = 2,541$ state-action pairs. By looking to these numbers, we can see that by using the minimax $Q(\lambda)$ -learning algorithm, each state-action pair is visited $\frac{1,200,000}{27,951} = 43$ times on average and by using the IQL algorithm, each state-action pair in the Q-table of the pursuer is visited $\frac{1,200,000}{1,331} = 902$ times on average and each state-action pair in the Q-table of the evader is visited $\frac{1,200,000}{2,541} = 472$ times on average.

From this analysis, we can conclude that a state-action pair in the IQL algorithm is visited more often than a state-action pair in the minimax $Q(\lambda)$ -learning algorithm considering that both algorithms have the same number of episodes and the same number of steps (a state-action pair in the IQL algorithm is visited at least 11 times more than a state-action pair in the minimax $Q(\lambda)$ -learning algorithm).¹ This of course will affect the learning process as we will see in the following comparison.

¹To increase the number of state-action visits in the minimax $Q(\lambda)$ -learning algorithm, one should increase the number of episodes and/or the number of steps. However, increasing the number of episodes and/or the number of steps will be at the cost of increasing the learning time (see Equation (6.13)).

1. **Learning process:**

(a) *Learning time:*

- The minimax $Q(\lambda)$ -learning algorithm: The learning process takes about 2.6 min. on average.
- The IQL algorithm: The learning process takes about 0.5 min. on average (the learning time of the minimax $Q(\lambda)$ -learning algorithm $\simeq 5.2$ times the learning time of the IQL algorithm when both the learning algorithms use the same number of episodes and the same number of steps).

(b) *Performance of the learning process:* Figure 8.1 shows the average of the capture time (over 100 runs of the simulation) using the minimax $Q(\lambda)$ -learning algorithm and the IQL algorithm. Figure 8.2 shows the average total reward gained in a game (over 100 runs of the simulation) using the minimax $Q(\lambda)$ -algorithm and the IQL algorithm. By comparing the results in Figures 8.1–8.2, we can see that the performance of the IQL algorithm is better than that of the minimax $Q(\lambda)$ -learning algorithm.

2. **Q-table structure:**

- The minimax $Q(\lambda)$ -learning algorithm has only one Q-table for all the learners. Therefore, it can be considered as a single-agent RL with its Markovian properties and theoretical basis [101]. However, the state space of this Q-table consists of a combination of the state spaces of all the learners (in our case 14,641 states if the state space consists of δ_p , $\dot{\delta}_p$, δ_e and $\dot{\delta}_e$ and 121 states if the state space consists only of δ_p and δ_e just for

simplification purpose). Also, the action space of this Q-table consists of a combination of the action spaces of all the learners (231 actions in our case). As a result, the curse of dimensionality problem will appear.

- By contrast, the IQL algorithm has a number of Q-tables equal to the number of learners (in our case we have two Q-tables; one for the pursuer and one for the evader). Therefore, it violates the assumption of the Markovian property of the environment and hence lacks the theoretical basis. However, using a Q-table for each learner reduces the dimension of the state-action spaces (121-by-11 for the pursuer's Q-table and 121-by-21 for the evader's Q-table) i.e. the curse of dimensionality problem will disappear.

3. Communication demands:

- The minimax $Q(\lambda)$ -learning algorithm has only one Q-table and therefore contains the joint states and the joint actions. As a result, the learners should communicate to share their states and their actions with each other (which is not realistic).
- In the IQL algorithm, the learners do not need to communicate or share their states and actions as each learner does not take the other learners into account and acts as if they do not exist. In other words, each learner considers the other learners as part of the environment.

It is obvious that the minimax $Q(\lambda)$ -learning algorithm and team learning algorithms, in general, have better performance than IQL algorithms. That is simply because the IQL algorithm takes into consideration only the state and the action of the learner considering the other learners as part of the environment. By contrast,

Figure 8.1: Average capture time using the IQL algorithm (solid line) compared to the minimax $Q(\lambda)$ -learning algorithm (dotted line).

Figure 8.2: Average total reward using the IQL algorithm (solid line) compared to the minimax $Q(\lambda)$ -learning algorithm (dotted line).

the minimax $Q(\lambda)$ -learning algorithm takes into consideration not only the state and the action of the learner but also the states and the actions of the other learners in the system. However, in our case, we show that the IQL algorithm has better performance than the minimax $Q(\lambda)$ -learning algorithm. That is because in our case a

state-action pair using the IQL algorithm is visited more often than using the minimax $Q(\lambda)$ - algorithm (about 11 more times). In other words, if we assume that the state-action pair using the minimax $Q(\lambda)$ - algorithm is visited as equal times as the IQL algorithm then the minimax $Q(\lambda)$ -learning algorithm should have better performance than the IQL algorithm. However, it is hard and in some cases it is impossible to do that because of the curse of dimensionality problem.

From the previous discussion and the simulations done, we can conclude that although the IQL algorithm does not have a theoretical basis, it is simpler and more realistic than the minimax $Q(\lambda)$ -learning algorithm and the team learning algorithms in general.

8.4 Learning in Two-Pursuer One-Evader Game

In this section we will extend our work from learning in a single pursuit-evasion game to leaning in a two-pursuer one-evader game. The two-pursuer one-evader model is shown in Figure 8.3. In this game, both the pursuers and the evader are self-learning their control strategies on-line by interaction between each other. We use the same equations of motion for the pursuers and the evader robots that are defined by Equation (6.1). We assume that the only information that both of the pursuers know is the instant position of the evader. Also, we assume that the only information that the evader knows is the instant positions of both of the pursuers. In other words, we assume that there is no information to share between the robots (such as rewards, states and actions). The input state to the FLC of the pursuer p_i , $i = 1, 2$, consists of the error in angle, δ_{p_i} , and its derivative, $\dot{\delta}_{p_i}$, where δ_{p_i} is the angle difference between the velocity direction of the pursuer p_i and the line-of-sight (LoS) vector, $\overrightarrow{P_i E}$, to the

evader. The error in angle, δ_{p_i} is calculated as follows

$$\delta_{p_i} = \arctan\left(\frac{y_e - y_{p_i}}{x_e - x_{p_i}}\right) - \theta_{p_i} \quad (8.7)$$

The output action from the FLC, a_{p_i} , of the pursuer p_i is its steering angle, u_{p_i} .

Figure 8.3: Two-Pursuer one-evader model.

The input state to the FLC of the evader consists of δ_e and $\dot{\delta}_e$ where δ_e is defined as the angle difference between the velocity direction of the evader and the direction of the evader. However, the direction of the evader can be defined by several cases as we have two pursuers in the game. The first case in which the direction of the

evader is defined as the LoS vector between the nearest pursuer and the evader. The direction of the evader, $\overrightarrow{EP}_{dir}$, if this case is used, is defined as follows

$$\overrightarrow{EP}_{dir} = \frac{\overrightarrow{P_i E}}{\|\overrightarrow{P_i E}\|} \quad (8.8)$$

where $i = 1$ if pursuer 1 is closer to the evader and $i = 2$ if pursuer 2 is closer to the evader. However, this case neglects the existence of the other pursuer (which may also be near to the evader) and the evader may take a path that makes it easy for the other pursuer to catch it. Therefore, we do not use this case in our work.

The second case in which the evader takes into consideration both of the pursuers. The direction of the evader, $\overrightarrow{EP}_{dir}$, if this case is used, is defined as follows

$$\overrightarrow{EP}_{dir} = \frac{\overrightarrow{P_1 E} + \overrightarrow{P_2 E}}{\|\overrightarrow{P_1 E} + \overrightarrow{P_2 E}\|} \quad (8.9)$$

However, if one of the pursuers is close to the evader while the other pursuer is far away from the evader so the evader should not give both of the pursuers the same importance. In this case, the evader should take into consideration the nearest pursuer more than the far pursuer. Therefore, we do not use this case in our work.

As a result of the previous discussion, we will give a general definition for the direction of the evader that mixes between the two cases that are previously described. In other words, the direction that the evader should take will take into consideration not only both of the pursuers but also the positions of both of them. This means that the evader will take a direction, $\overrightarrow{EP}_{dir}$, that depends on the positions of both of the pursuers. This direction is described as follows

$$\overrightarrow{EP}_{dir} = (x_{dir}, y_{dir})$$

$$= \frac{W \overrightarrow{P_1 E} + \frac{1}{W} \overrightarrow{P_2 E}}{\|W \overrightarrow{P_1 E} + \frac{1}{W} \overrightarrow{P_2 E}\|} \quad (8.10)$$

where W , $W > 0$, is defined as

$$W = \frac{\|\overrightarrow{P_2 E}\|}{\|\overrightarrow{P_1 E}\|} \quad (8.11)$$

Equation (8.10) shows that the evader takes into consideration the directions of both of the pursuers but with a weighting factor, W , that depends on the distances of the two pursuers to the evader.

As defined by Equations (8.10)–(8.11), if we imagine that $W \rightarrow \infty$ or $W \rightarrow 0$ (we mean that one pursuer is too close to the evader while the other pursuer is too far from the evader), then the evader will only take into consideration one of the pursuers (this is the first case that is previously described which is defined by Equation (8.8)). If $W = 1$ (i.e. both of the pursuers are at the same distance to the evader), then the evader will give both of the pursuers the same care (this is the second case that is previously described which is defined by Equation (8.9)). If $W > 1$ (i.e. pursuer 1 is closer to the evader than pursuer 2), then the evader will give more attention to pursuer 1 according to the value of W and vice versa if $W < 1$. Now, we can define δ_e as follows

$$\delta_e = \arctan \left(\frac{y_{dir}}{x_{dir}} \right) - \theta_e \quad (8.12)$$

The output action from the FLC of the evader is its steering angle, u_e .

8.4.1 Computer simulation and results

In this game, we use the proposed QLBGFLC described in Section 6.4 for both the pursuers and for the evader. In the computer simulation, the evader starts motion from the position $(0, 0)$ with an initial orientation $\theta_e = 0.0$, $V_e = 0.3$ m/s, $L_e = 0.2$ m and $u_e \in [-0.8, 0.8]$. The pursuers start motion from random positions for each episode with initial orientations $\theta_{p_1} = \theta_{p_2} = 0.0$, $V_{p_1} = V_{p_2} = 0.5$ m/s which are greater than that of the evader (i.e. both of the pursuers are faster than the evader), $L_{p_1} = L_{p_2} = 0.2$ m and $u_{p_1}, u_{p_2} \in [-0.5, 0.5]$ which are smaller than that of the evader (i.e. both of the pursuers are less maneuverable than the evader). The capture radius $\ell = 0.10$ m.

We choose the number of episodes (games) to be 2000, the number of steps (plays) in each episode is 6000, $\gamma = 0.5$ and $\lambda = 0.3$. These values are chosen the same as those that are chosen in Section 6.5.2 except for the number of episodes. Here, we choose the number of episodes to be 2000 instead of 200 since we have three robots that are learning simultaneously. Therefore, we need more episodes for the learning process. We make the learning rate, α , decrease with each episode such that

$$\alpha = \left(\frac{1.0}{i} \right)^{0.7} \quad (8.13)$$

where i is the current episode. We use the ϵ -greedy action selection method defined by Equation (2.54) and we make the value of ϵ decrease with each episode such that

$$\epsilon = \frac{1.0}{i} \quad (8.14)$$

To build the state spaces for the two pursuers, we discretize the ranges of the

inputs, δ_{p_i} and $\dot{\delta}_{p_i}$, by 0.2. The ranges of δ_{p_i} and $\dot{\delta}_{p_i}$ are set to be from -1.0 to 1.0 so the discretized values for δ_{p_i} and $\dot{\delta}_{p_i}$ will be -1.0, -0.8, -0.6, ..., 0.8, 1.0. There are 11 discretized values for δ_{p_i} and 11 discretized values for $\dot{\delta}_{p_i}$. These values are combined to form $11 \times 11 = 121$ states for each pursuer. The state space of the evader is identical to those of the pursuers since the ranges of δ_e and $\dot{\delta}_e$ are set also to be from -1.0 to 1.0.

To build the action space of each pursuer, we discretize the range of the action, u_{p_i} , by 0.1. The range of the action is set to be from $-u_{p_{i_{max}}}$ to $u_{p_{i_{max}}}$ i.e. from -0.5 to 0.5 so the discretized values in the action space are -0.5, -0.4, -0.3, ..., 0.4, 0.5. There are 11 actions and the size of the Q-table of each pursuer will be 121-by-11. To build the action space of the evader, we discretize the range of the action, u_e , by 0.1. The range of the action is set to be from $-u_{e_{max}}$ to $u_{e_{max}}$ i.e. from -0.8 to 0.8 so the discretized values in the action space are -0.8, -0.7, -0.6, ..., 0.7, 0.8. There are 17 actions and the size of the Q-table of the evader will be 121-by-17.

Note that if we want to use the minimax $Q(\lambda)$ -learning algorithm then we will have only one Q-table. However, the number of states $= (121)_{p_1} \times (121)_{p_2} \times (121)_e = 1,771,561$ and the number of joint actions $= (11)_{p_1} \times (11)_{p_2} \times (17)_e = 2,057$. The dimension of this Q-table will be 1,771,561-by-2,057 i.e. this Q-table will have 3,644,100,977 state-action pairs. This number of state-action pairs is considered as a huge number that is impossible to deal with. On the other hand, the IQL algorithm has $(121 \times 11)_{Q_{p_1}} + (121 \times 11)_{Q_{p_2}} + (121 \times 17)_{Q_e} = 4719$ state-action pairs in all the Q-tables of the two pursuers and the evader which is a relatively small number.

Now, we will construct the reward function that is based on the distance between the pursuers and the evader. The distance between each pursuer, p_i , $i = 1, 2$ and the

evader at time t is calculated as follows

$$D_{p_i}(t) = \sqrt{(x_e(t) - x_{p_i}(t))^2 + (y_e(t) - y_{p_i}(t))^2} \quad (8.15)$$

The difference between two successive distances, $\Delta D_{p_i}(t)$, is calculated as

$$\Delta D_{p_i}(t) = D_{p_i}(t) - D_{p_i}(t+1) \quad (8.16)$$

The maximum value of $\Delta D_{p_i}(t)$ is defined as

$$\Delta D_{p_{imax}} = V_{p_{imax}} T \quad (8.17)$$

So, we choose the immediate reward, r_{t+1} , for each pursuer and for the evader to be

$$r_{t+1} = \begin{cases} \frac{\Delta D_{p_i}(t)}{\Delta D_{p_{imax}}} & \text{for pursuer } p_i \\ -\sum_{i=1}^2 \frac{\Delta D_{p_i}(t)}{\Delta D_{p_{imax}}} & \text{for the evader} \end{cases} \quad (8.18)$$

The values of the GAs parameters used in phase 2 (with its two stages) are shown in Table 8.1.²

To execute our computer simulation, we use a core 2 duo computer with a 2.0 GHz clock frequency and 4.0 Gigabytes of RAM. We do the computer simulation in MATLAB.

Figure 8.4 shows the paths of the pursuers and the evader using the proposed QL-BGFLC. The evader starts moving from position $(0, 0)$ and both of the two pursuers start moving from different initial positions; also, the capture time for each game is

²These values are chosen according to the criterion described in Section 3.4.

Table 8.1: Values of GAs parameters.

	Phase 2	
	Stage 1	Stage 2
Number of iterations	1000	500
Population size	100	5
Number of steps	—	600
Crossover probability	0.2	0.4
Mutation probability	0.3	0.2
Fitness function	MSE defined by Equation (6.11)	capture time
Fitness function objective	minimize	minimize for pursuers & maximize for evader

shown in this figure. From Figure 8.4 we can see that both of the pursuers and the evader can successfully learn their control strategies using the proposed QLBGFLC. However and as shown in Figure 8.4, both of the pursuers may collide or at least may follow the same path at some time (especially when both of them start moving from adjacent positions as shown in Figure 8.4(f)). In this case, both of the pursuers may lose the advantage of the multi pursuers case.

In the previous pursuit-evasion game, the pursuers try to capture the evader by following its instant position. Although both of the pursuers only need to know the instant position of the evader to perform this task, there are several disadvantages to using this model. These disadvantages are:

1. The pursuers may collide with each other.
2. If collision does not occur then they may take the same path at some time acting as a single pursuer and lose the effect of using multi pursuers.
3. To avoid the disadvantages in points 1 and 2, some type of communication

between the pursuers must exist and the joint actions must be considered in the learning process. However, this will make the system more complex.

Next, we assume that the pursuers can get more information about the evader. This will help in solving these disadvantages.

Figure 8.4: Paths of the pursuers and the evader after learning. The initial position of the evader is (0,0) and the pursuers start at different initial positions. The capture time for each game is also shown.

8.4.2 Pursuers have more information about the evader

The pursuit-evasion model is shown in Figure 8.5. The pursuers, instead of following

Figure 8.5: Two-pursuer one-evader model with more information.

the instant position of the evader, try to predict the future position of the evader where capture will occur and then move directly to that position. The idea of a pursuer predicting the capture position of an evader and moving directly to that position is described in [105–108]. In their work, Li and Cruz [105], Wei et al. [106, 107] and Wanga et al. [108] try to find a strategy for a group of pursuers to be able to catch a faster evader. Here, we use this idea from the learning point of view. Our objective is

to make the pursuers able to self-learn strategies by which, they can catch the evader in minimum time and at the same time to reduce the possibility of collision between the pursuers. To do that, the pursuers need to know more information about the evader. They need to know not only the instant position of the evader but also the value and the direction of its velocity vector.

The input state to the FLC of the evader consists of the error in angle, δ_e , defined by Equation (8.12) and its derivative, $\dot{\delta}_e$. The output action from the FLC of the evader is its steering angle, u_e . Now we want to define the error in angle, δ_p , and its derivative, $\dot{\delta}_p$, which are the input state to the FLC of the pursuer p_i . Before doing that, we will calculate the magnitude and direction of the the modified LoS, $\overrightarrow{P_i E'}$, that the pursuer p_i should follow instead of following the direct LoS, $\overrightarrow{P_i E}$. Figure 8.6 helps us in calculating $\overrightarrow{P_i E'}$.

Figure 8.6: Description of the angles α and β .

First, we want to define the angle β and the angle α . As shown in Figure 8.6,

the angle β_i , $i = 1, 2$, is the angle between the direction of the evader, $\overrightarrow{EP}$, and the LoS vector, $\overrightarrow{P_iE}$, from the pursuer p_i to the evader. One can calculate β by using the *dot-product* operation, $\odot$, as follows

$$\begin{aligned}\overrightarrow{P_iE} \odot \overrightarrow{EP} &= -\overrightarrow{EP_i} \odot \overrightarrow{EP} \\ &= -\|\overrightarrow{P_iE}\| \|\overrightarrow{EP}\| \cos(\beta_i)\end{aligned}$$

Therefore,

$$\beta_i = \arccos \left(-\frac{\overrightarrow{P_iE} \cdot \overrightarrow{EP}}{\|\overrightarrow{P_iE}\| \|\overrightarrow{EP}\|} \right) \quad (8.19)$$

where $i = 1, 2$. The angle α_i is the angle between the LoS vector, $\overrightarrow{P_iE}$ and the modified LoS vector, $\overrightarrow{P_iE'}$. One can calculate α_i from the triangle properties as follows [105–108]

$$\frac{V_e}{\sin(\alpha_i)} = \frac{V_{p_i}}{\sin(\beta_i)}$$

Therefore,

$$\alpha_i = \arcsin \left(\left(\frac{V_e}{V_{p_i}} \right) \sin(\beta_i) \right) \quad (8.20)$$

From trigonometry, the magnitude of $\overrightarrow{EP_i}$ can be calculated as

$$\|\overrightarrow{EP_i}\| = \|\overrightarrow{P_iE}\| \frac{\sin \alpha_i}{\sin(\alpha_i + \beta_i)} \quad (8.21)$$

Therefore

$$\overrightarrow{EP_i} = \|\overrightarrow{EP_i}\| (\overrightarrow{EP})_{dir} \quad (8.22)$$

Finally

$$\begin{aligned}\overrightarrow{P_i E'} &= (x', y') \\ &= \overrightarrow{P_i E} + \overrightarrow{EP_i}\end{aligned}\tag{8.23}$$

Here, the error in angle, δ_{p_i} , is defined to be the angle difference between the velocity direction of the pursuer and the modified LoS vector, $\overrightarrow{P_i E'}$, as shown in Figure 8.5 and is calculated as follows

$$\delta_{p_i} = \arctan\left(\frac{y'}{x'}\right) - \theta_{p_i}\tag{8.24}$$

The output action of the FLC of the pursuer p_i is its steering angle, u_{p_i} .

In the computer simulation, we use the proposed QLBGFLC described in Section 6.4. The proposed QLBGFLC has the same parameters described in Section 8.4.1. Figure 8.7 shows the paths of the pursuers and the evader using the proposed QLBGFLC. The evader starts moving from position $(0, 0)$ and both of the two pursuers start moving from different initial positions; also, the capture time for each game is shown in this figure. We choose the same initial positions of both of the pursuers as those that are chosen in Section 8.4.1 to compare the results. From Figure 8.7 we can see that both of the pursuers and the evader can successfully learn their control strategies using the proposed QLBGFLC. We can also see that the capture time of the game is reduced. In addition, the possibility of collision between the pursuers is reduced even if both of the pursuers start moving from adjacent positions as shown in Figure 8.7(f). The pursuit-evasion model with more information about the evader has the following advantages:

1. The possibility of collision between both of the pursuers is reduced.
2. Consequently, the pursuers will take different paths and then will have a greater advantage in catching the evader.
3. As a result, it will take less time for the pursuers to catch the evader (as shown in Figure 8.7).

Figure 8.7: Paths of the pursuers and the evader after learning when the pursuers get more information about the evader. The initial position of the evader is (0.0) and the pursuers start at different initial positions. The capture time for each game is also shown.

4. There is no need to use any type of communication between the pursuers where the pursuers can perform the task independently (but effectively as we mention in points 2 and 3). This will lead to a simple system design.

8.5 Learning in n -Pursuer n -Evader Games

In the previous section, we show learning in the two-pursuer one-evader game. In this section, we will introduce learning in the n -pursuer n -evader game. Our objective in this section is to implement a decomposition process to decompose the n -pursuer n -evader game into n one-pursuer one-evader models [109].

In the decomposition process, we build a new Q-table, denoted by Q_{CF} , which is responsible for learning the coordination between the pursuers and the evaders so that each pursuer is assigned to only one evader. After executing the decomposition process, the proposed QLBGFLC introduced in Chapter 6 is used to perform the learning process in the resulting n one-pursuer one-evader models.

In the n -pursuer n -evader game, we assume that each player only knows the instantaneous position of the other players but at the same time none of them knows its control strategy nor the control strategy of the other players. We also assume that there is no information to share between the robots (such as rewards, states and actions). In addition, none of the pursuers know the evader that it should be coupled with. Therefore, the learning process consists of two stages. In stage one, the decomposition process is executed based on the coordination that is learned using Q_{CF} i.e. in this stage, the best coupling between the pursuers and the evaders is learned. Then in stage two, the pursuers and their coupled evaders have to self-learn their control strategies on-line by interaction with each other.

In this section we extend the work done in Chapter 6 from learning in a single pursuit-evasion game to learning in a multi-pursuit-evasion game. We will do the simulation for $n = 2$ and $n = 3$ then we can generalize the work for $n > 3$. To test the performance of the proposed technique, it is compared with the classical control strategy of a single pursuit-evasion game.

8.5.1 Building the Q-tables

The state is the error in angle between the pursuer and its coupled evader, δ , defined by Equation (6.4) and its derivative, $\dot{\delta}$. The range of this angle and its derivative are set from -1.0 to 1.0 . The state space is discretized by 0.2 to be $(-1.0, -1.0)$, $(-1.0, -0.8)$, $\dots$, $(0.0, 0.0)$, $\dots$, $(1.0, 0.8)$, $(1.0, 1.0)$ with a total of 121 states. The action is the control strategy of the robot, u . The action space of each robot is discretized by 0.1 as shown in Table 8.2 and Table 8.3 for $n = 2$ and $n = 3$, respectively. The dimensions of the different Q-tables are shown in Table 8.4.

Table 8.2: Discretized actions ($n = 2$).

	u_{max}	u	No. of actions
P_1	0.5	-0.5, -0.4, ..., 0, ..., 0.4, 0.5	11
P_2	0.6	-0.6, -0.5, ..., 0, ..., 0.5, 0.6	13
E_1	0.8	-0.8, -0.7, ..., 0, ..., 0.7, 0.8	17
E_2	1.0	-1.0, -0.9, ..., 0, ..., 0.9, 1.0	21

We build a new Q-table, Q_{CF} , to learn the best coupling among the robots (pursuers with evaders). In this work we have two cases. In the first case, $n = 2$ in which we have two possible coupling. Therefore, the dimension of $\{Q_{CF}\}_{n=2}$ is 1×2 where the first value is assigned to the first coupling choice whereas the second value is assigned to the second coupling choice. After learning has finished, we choose the

Table 8.3: Discretized actions ($n = 3$).

	u_{max}	u	No. of actions
P_1	0.4	-0.4, -0.3, ..., 0, ..., 0.3, 0.4	9
P_2	0.5	-0.5, -0.4, ..., 0, ..., 0.4, 0.5	11
P_3	0.6	-0.6, -0.5, ..., 0, ..., 0.5, 0.6	13
E_1	1.0	-1.0, -0.9, ..., 0, ..., 0.9, 1.0	21
E_2	0.6	-0.6, -0.5, ..., 0, ..., 0.5, 0.6	13
E_3	0.8	-0.8, -0.7, ..., 0, ..., 0.7, 0.8	17

Table 8.4: Dimensions of the different Q-tables.

	Q_{p_1}	Q_{p_2}	Q_{p_3}	Q_{e_1}	Q_{e_2}	Q_{e_3}
$n = 2$	121×11	121×13	—	121×17	121×21	...
$n = 3$	121×9	121×11	121×13	121×21	121×13	121×17

best coupling which is assigned to the maximum value in Q_{CF} . In the second case, $n = 3$ in which we have six possible coupling choices. Therefore, the dimension of $\{Q_{CF}\}_{n=3}$ is 1×6 . In general, for n -pursuer n -evader game, we have $n!$ possible coupling choices and the dimension of Q_{CF} is $1 \times |n!|$. Table 8.5 shows the possible coupling choices for $n = 2$ and $n = 3$.

8.5.2 Constructing the reward function

The distance between each pursuer and the corresponding evader, which is coupled with, at time t is calculated as follows

$$D(t) = \sqrt{(x_e(t) - x_p(t))^2 + (y_e(t) - y_p(t))^2} \quad (8.25)$$

Table 8.5: All possible coupling choices.

	Choice	Coupling
$n = 2$	First	P_1 with E_1 and P_2 with E_2
	Second	P_1 with E_2 and P_2 with E_1
$n = 3$	First	P_1 with E_1 , P_2 with E_2 and P_3 with E_3
	Second	P_1 with E_1 , P_2 with E_3 and P_3 with E_2
	Third	P_1 with E_2 , P_2 with E_1 and P_3 with E_3
	Fourth	P_1 with E_2 , P_2 with E_3 and P_3 with E_1
	Fifth	P_1 with E_3 , P_2 with E_1 and P_3 with E_2
	Sixth	P_1 with E_3 , P_2 with E_2 and P_3 with E_1

The difference between two successive distances, $\Delta D(t)$, is calculated as

$$\Delta D(t) = D(t) - D(t + 1) \quad (8.26)$$

The maximum value of $\Delta D(t)$ is defined as

$$\Delta D_{max} = V_{rmax} T \quad (8.27)$$

So, we choose the reward, r , to be

$$r_{t+1} = \begin{cases} \frac{\Delta D(t)}{\Delta D_{max}}, & \text{for the pursuer;} \\ -\frac{\Delta D(t)}{\Delta D_{max}}, & \text{for the evader.} \end{cases} \quad (8.28)$$

The updating of Q_{CF} is different from the updating of the other Q -tables in that the coupling occurs at the beginning of each episode (game) and of course it can not be changed during the episode. Therefore, Q_{CF} is updated each episode not each step (play). Accordingly, the reward function, r_{CF} , is different where we use the total

reward of an episode as a parameter in the reward function instead of the immediate reward of a step. The total reward of an episode, r_{total} , is calculated as follows

$$r_{total} = \sum_{i=1}^I \sum_{j=p_1}^{p_n} r_j^i \quad (8.29)$$

where r is the immediate reward which is defined by Equation (8.28), I is the step at which the episode ends and n is the number of pursuers/evaders. We want Q_{CF} to learn the best coupling and at the same time the coupling choice affects the number of steps. In other words, if a wrong coupling is selected then the capture time will increase and as a result, the number of steps will also increase. Therefore, we calculate the reward function, r_{CF} , as follows

$$r_{CF} = \frac{r_{total}}{I} \quad (8.30)$$

Note that as I decreases (i.e. the capture time decreases) then r_{CF} will increase and vice versa.

8.5.3 Computer simulation and results

We use the same pursuit-evasion model described in Section 6.3. First, we will show results for $n = 2$. The parameters of the mobile robots for $n = 2$ are shown in Table 8.6. The parameters of the $Q(\lambda)$ -learning for $n = 2$ are shown in Table 8.7 where m is the current episode and ϑ and ϑ_{CF} are the computational temperatures defined in Equation (2.56) (here, we use the softmax action selection method instead of the ϵ -greedy method).³ The parameters of the GAs used in phase 2, stage 1 and stage 2,

³These values are chosen according to the criterion described in Section 6.5.2.

for $n = 2$ are shown in Table 8.8.⁴

Table 8.6: Parameters of mobile robots ($n = 2$).

	P_1	P_2	E_1	E_2
V [m/s]	1.6	1.2	1.0	0.6
u_{max} [rad]	0.5	0.6	0.8	1.0
L [m]	0.3	0.3	0.3	0.3

Table 8.7: Parameters of $Q(\lambda)$ -learning ($n = 2$).

Episodes	Steps	γ	λ	T	α	ϑ	ϑ_{CF}
200	600	0.3	0.8	0.1	$(\frac{1.0}{m})^{0.7}$	$\frac{1.0}{m}$	$\frac{1.0}{m}$

Table 8.8: Values of GAs parameters ($n = 2$).

	Phase 2	
	Stage 1	Stage 2
Number of iterations	1000	100
Population size	80	10
Number of steps	—	300
Crossover probability	0.2	0.4
Mutation probability	0.1	0.2

Table 8.9: The state-action values in Q_{CF} ($n = 2$).

Coupling choice	First	Second
Q-value	0.4036	0.0588

Table 8.9 shows the values in the Q_{CF} -table for $n = 2$. From Table 8.9 we can see that the best coupling is the first choice which is P_1 with E_1 and P_2 with E_2 .

⁴These values are chosen according to the criterion described in Section 3.4.

Figure 8.8 shows the paths of the pursuers and the evaders after the learning process for $n = 2$. The capture time is 9.6 seconds. When we use the classical control strategies for the single pursuit-evasion game, the capture time is 9.6 seconds which is the same as our results which means that the pursuers and the evaders can successfully learn their classical control strategies using the proposed QLBGFLC. In addition, the coordination between the pursuers and the evaders is successfully learned. Figure 8.9 shows the paths of the pursuers and the evaders if the wrong coupling is chosen (i.e. the second choice). The capture time is 20.3 seconds which is longer than choosing the right coupling.

Figure 8.8: Paths of the pursuers and the evaders after learning for $n = 2$ compared with the classical control strategies for the single pursuit-evasion game.

Now, we will show results for $n = 3$. The parameters of the mobile robots for $n = 3$ are shown in Table 8.10. The parameters of the $Q(\lambda)$ -learning for $n = 3$ are shown in Table 8.11 where m is the current episode.⁵ The parameters of the GAs

⁵These values are chosen according to the criterion described in Section 6.5.2.

Figure 8.9: Paths of the pursuers and the evaders after learning when the wrong coupling is chosen for $n = 2$.

used in phase 2, stage 1 and stage 2, for $n = 3$ are shown in Table 8.12.⁶

Table 8.10: Parameters of mobile robots ($n = 3$).

	P_1	P_2	P_3	E_1	E_2	E_3
V [m/s]	2.0	1.6	1.2	0.6	1.4	1.0
u_{max} [rad]	0.4	0.5	0.6	1.0	0.6	0.8
L [m]	0.3	0.3	0.3	0.3	0.3	0.3

Table 8.11: Parameters of $Q(\lambda)$ -learning ($n = 3$).

Episodes	Steps	γ	λ	T	α	ϑ	ϑ_{CF}
200	600	0.3	0.9	0.1	$(\frac{1.0}{m})^{0.7}$	$\frac{1.0}{m}$	$\frac{20.0}{m}$

Table 8.13 shows the values in the Q_{CF} -table for $n = 3$. From Table 8.13 we can see that the best coupling is the fourth choice which is P_1 with E_2 , P_2 with E_3

⁶These values are chosen according to the criterion described in Section 3.4.

Table 8.12: Values of GAs parameters ($n = 3$).

	Phase 2	
	Stage 1	Stage 2
Number of iterations	1000	100
Population size	80	10
Number of steps	—	300
Crossover probability	0.2	0.2
Mutation probability	0.1	0.1

and P_3 with E_1 . Figure 8.10 shows the paths of the pursuers and the evaders after the learning process for $n = 3$. The capture time is 10.0 seconds. When we use the classical control strategies for the single pursuit-evasion game, the capture time is 9.9 seconds which means that all the pursuers and all the evaders can successfully learn their control strategies using the proposed QLBGFLC. In addition, the coordination between the pursuers and the evaders is successfully learned. Table 8.14 shows the capture times for all possible choices of coupling. The results in Table 8.14 agree with those of Table 8.13 which shows the Q-values of the different coupling. Note that for $n = 3$, if the second or the fifth choice is chosen for coupling, then the game has no solution because in both choices, pursuer 3 is coupled with evader 2 (see Table 8.5) which is faster ($V_{e_2} = 1.4 > V_{p_3} = 1.2$) and that agrees with the results shown in Table 8.13 and Table 8.14.

Table 8.13: The state-action values in Q_{CF} ($n = 3$).

Coupling choice	First	Second	Third	Fourth	Fifth	Sixth
Q-value	0.1413	0.0229	0.2846	0.7967	0.0185	0.3732

Figure 8.10: Paths of the pursuers and the evaders after learning for $n = 3$ compared with the classical control strategies for the single pursuit-evasion game.

Table 8.14: The capture times of all possible coupling for $n = 3$.

Choice	First	Second	Third	Fourth	Fifth	Sixth
Capture time	17.1	No	17.1	10.0	No	13.8

8.6 Summary

In this chapter we introduced a comparison between the minimax $Q(\lambda)$ -learning algorithm and the IQL algorithm. We show that the IQL algorithm has many advantages over the minimax $Q(\lambda)$ -learning algorithm. We also show that in some cases, it is impossible to use minimax $Q(\lambda)$ -learning. We implement scenarios for the two-pursuer one-evader game in which both of the pursuers and the evader are learning their control strategies. We show that as the pursuers know more information about the evader, they can learn to catch the evader faster. Then we extend the work done in Chapter 6 from learning in a single pursuit-evasion game to learning in a multiple

pursuit-evasion game. To do that, we build a new Q-table that is responsible for learning the coupling process (coordination) between the pursuers and the evaders. We did the simulation for $n = 2$ and $n = 3$. The results show that the pursuers are successful in being able to choose the right coupling. In addition, the pursuers and the evaders are also able to learn their control strategies.

Chapter 9

Conclusion, Contributions and Publications

9.1 Conclusion

In our work we illustrate different learning techniques to tune the input and the output parameters of fuzzy logic controllers. The proposed techniques are applied to different robotic applications. The proposed techniques are based on two different machine learning algorithms. The first algorithm is supervised learning that needs input/output data or an expert to learn from. In this thesis we chose a PD controller as an expert. The proposed technique (IGBFLC) is applied to a wall-following mobile robot. Then, we examine this idea using different learning technique and different application. We use a gradient-based technique instead of GAs to form the proposed IANFIS. The proposed technique is applied to a pursuit evasion game instead of the wall-following problem. With these two applications, we show that the proposed techniques outperform the expert itself (which is the PD controller in our case).

The second machine learning algorithm is reinforcement learning. Based on reinforcement learning and especially on the $Q(\lambda)$ -learning algorithm, we proposed a technique (QLBGFLC) that overcomes the limitation of the proposed supervised learning techniques. The limitation is that the proposed supervised learning techniques need an expert or an input/output data set. The proposed technique is applied to different pursuit-evasion games and is compared to different learning techniques. Computer simulation shows that the proposed technique outperforms all the other techniques when compared to the classical control strategy. In addition and before we apply this technique, we introduced a convergence proof for the model-building Q -learning algorithm with eligibility traces.

However, the proposed technique is applicable only for continuous domains that can be discretized such that the dimensions of the resulting discrete domains are not prohibitively large. For continuous domains that can not be discretized with acceptable dimensions we proposed another technique (QLFIS). This technique makes use of fuzzy system as a function approximation in addition to its main job as a controller. This technique is also applied to different pursuit-evasion games. Computer simulation shows that the proposed technique outperforms all the other techniques when compared with the classical control strategy.

We also made a comparison between two different $Q(\lambda)$ -learning algorithms that are based on two different concepts for the learning process in multi-robot systems. These algorithms are minimax $Q(\lambda)$ -learning and independent $Q(\lambda)$ -learning. We show that independent $Q(\lambda)$ -learning is the appropriate algorithm for our work. We also show that in some cases it is impossible to use minimax $Q(\lambda)$ -learning. Then, we implement scenarios for learning in two-pursuer one-evader game in which both of the pursuers and the evader self-learn their control strategies. We also show that

as the pursuers know more information about the evader, they can learn to catch the evader faster. Finally, we extend our work for the n -pursuer n -evader games by constructing a new Q-table that is responsible for learning the coupling between the pursuers and the evaders.

9.2 Thesis Contributions

To reiterate from the introduction chapter, the contributions can be itemized as follows:

1. A novel technique is proposed to combine fuzzy logic controllers with genetic algorithms that we called an *iterative genetic based fuzzy logic controller* (IG-BFLC).
2. We also use the same idea of this technique to combine fuzzy inference systems with gradient-based techniques resulting in an *iterative adaptive network fuzzy inference system* (IANFIS).
3. A proof of convergence for the model-building Q-learning algorithm with eligibility traces is presented.
4. A novel technique is proposed to combine fuzzy logic controllers and genetic algorithms with reinforcement learning methods that we called a $Q(\lambda)$ -learning based genetic fuzzy logic controller (QLBGFLC).
5. A novel technique is proposed to combine fuzzy logic controllers with $Q(\lambda)$ -learning that we called a $Q(\lambda)$ -learning fuzzy inference system (QLFIS).

6. We apply the techniques proposed in this thesis to different robotic problems such as wall-following applications and different pursuit-evasion games.
7. We investigate a number of different scenarios and approaches for the pursuit-evasion game. We introduce learning in different pursuit-evasion games with pursuers and evaders that have different capabilities (heterogeneous robots). The idea of individual robots adapting their behavior based on their capabilities is new to the literature (see Section 6.5.3 and Section 7.3.2).
8. We solve the curse of dimensionality problem (large state and action spaces) using the proposed QLBGFLC and the proposed QLFIS (see Section 6.4, Section 7.2 and Section 8.4.1).
9. We reduce the communication complexity by using independent learning techniques in combination with the proposed QLBGFLC technique. By using independent learning, the robots do not need to share information such as rewards, states and actions between each other (see Section 8.4.1 and Section 6.5.4, Section 7.3.3, Section 8.4.1 and Section 8.5).
10. A decentralized learning approach is proposed for learning in the n -pursuer n -evader game. In this approach, the robots can successfully learn coordination between each other.

9.3 Publications

The work done in this thesis produced the following publications:

1. S. F. Desouky and H. M. Schwartz. Genetic based fuzzy logic controller for a wall-following mobile robot. In *ACC'09: Proceedings of the 2009 conference*

-
- on American Control Conference*, pages 3555–3560, St. Louis, MO, Jun. 2009. IEEE Press.
2. S. F. Desouky and H. M. Schwartz. Different hybrid intelligent systems applied for the pursuit-evasion game. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 2677–2682, Oct. 2009.
 3. S. F. Desouky and H. M. Schwartz. A novel technique to design a fuzzy logic controller using $Q(\lambda)$ -learning and genetic algorithms in the pursuit-evasion game. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 2683–2689, San Antonio, TX, Oct. 2009.
 4. S. F. Desouky and H. M. Schwartz. A novel hybrid learning technique applied to a self-learning multi-robot system. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 2690–2697, San Antonio, TX, Oct. 2009.
 5. S. F. Desouky and H. M. Schwartz. Learning in n-Pursuer n-Evader Differential Games. Accepted in *2010 IEEE International Conference on Systems, Man and Cybernetics*, Istanbul, Turkey, Oct. 2010.
 6. S. F. Desouky and H. M. Schwartz. $Q(\lambda)$ -learning Fuzzy Logic Controller For a Multi-robot System. Accepted in *2010 IEEE International Conference on Systems, Man and Cybernetics*, Istanbul, Turkey, Oct. 2010.

References

- [1] 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*, 6(3):285–293, Sep. 2005.
- [2] S. Micera, A. M. Sabatini, and P. Dario. Adaptive fuzzy control of electrically stimulated muscles for arm movements. *Medical and biological engineering and computing*, 37(6):680–685, Jan. 1999.
- [3] F. Daldaban, N. Ustkoyuncu, and K. Guney. Phase inductance estimation for switched reluctance motor using adaptive neuro-fuzzy inference system. *Energy Conversion and Management*, 47:485–493, 2006.
- [4] S. Labiod and T. M. Guerra. Adaptive fuzzy control of a class of SISO nonaffine nonlinear systems. *Fuzzy Sets and Systems*, 158(10):1126–1137, 2007.
- [5] H. K. Lam and F. H. F. Leung. Fuzzy controller with stability and performance rules for nonlinear systems. *Fuzzy Sets and Systems*, 158:147–163, 2007.
- [6] H. Hagnas, V. Callaghan, and M. Colley. Learning and adaptation of an intelligent mobile robot navigator operating in unstructured environment based on a novel online fuzzy-genetic system. *Fuzzy Sets and Systems*, 141(1):107–160, Jan. 2004.

- [7] M. Mucientes, D. L. Moreno, A. Bugarin, and S. Barro. Design of a fuzzy controller in mobile robotics using genetic algorithms. *Applied Soft Computing*, 7(2):540–546, 2007.
- [8] J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.
- [9] C. Karr. Genetic algorithms for fuzzy controllers. *AI Expert*, 6(2):26–33, 1991.
- [10] L. X. Wang. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1414–1427, Nov./Dec. 1992.
- [11] F. Herrera, M. Lozano, and J. L. Verdegay. Tuning fuzzy logic controllers by genetic algorithms. *International Journal of Approximate Reasoning*, 12:299–315, 1995.
- [12] A. Lekova, L. Mikhailov, D. Boyadjiev, and A. Nabout. Redundant fuzzy rules exclusion by genetic algorithms. *Fuzzy Sets and Systems*, 100:235–243, 1998.
- [13] L. Magdalena and F. Monasterio. Learning gait patterns for the fuzzy synthesis of biped walk. In *Proceedings of NAFIPS/IFIS/NASA-94*, pages 248–250, San Antonio, Texas, Dec. 1994.
- [14] L. Magdalena and F. Monasterio. Evolutionary based learning applied to fuzzy controller. In *Proceedings of Fourth IEEE International Conference on Fuzzy Systems and Second International Fuzzy Engineering Symposium*, pages 1111–1118, Yokohama, Mar. 1995.
- [15] D. A. Linkens and H. O. Nyongesa. Genetic algorithms for fuzzy control, part

-
- 1: Offline system development and application. In *IEE Proceedings-Control Theory and Applications*, volume 142, pages 161–176, May 1995.
- [16] T. C. Chin and X. M. Qi. Genetic algorithms for learning the rule base of fuzzy logic controller. *Fuzzy Sets and Systems*, 97:1–7, 1998.
- [17] N. Kubota, T. Morioka, F. Kojima, and T. Fukuda. Learning of mobile robots using perception-based genetic algorithm. *Measurement*, 29(3):237–248, 2001.
- [18] L. Ming, G. Zailin, and Y. Shuzi. Mobile robot fuzzy control optimization using genetic algorithm. *Artificial Intelligence in Engineering*, 10(4):293–298, 1996.
- [19] K. H. Wu, C. H. Chen, and J. D. Lee. Genetic-based adaptive fuzzy controller for robot path planning. In *Proceedings of the fifth IEEE International Conference*, volume 3, pages 1687–1692, New Orleans, LA, Sep. 1996.
- [20] G. L. C. M. de Abreu and J. F. Ribeiro. Optimal design of fuzzy controllers using evolutionary genetic algorithms. In *Fourth International Conference on knowledge-Based Intelligent Engineering Systems and Allied Technologies*, Brighton, UK, Aug./Sep. 2000.
- [21] S. F. Desouky. Applied fuzzy mathematics in control and automation. Master's thesis, Military Technical College, Cairo, Egypt, Dec. 2001.
- [22] Y. C. Chiou and L. W. Lan. Genetic fuzzy logic controller: An iterative evolution algorithm with new encoding method. *Fuzzy Sets and Systems*, 152(3): 617–635, Jun. 2005.
- [23] M. Mucientes, D. L. Moreno, A. Bugarin, and S. Barro. Evolutionary learning of a fuzzy controller for wall-following behavior in mobile robotics. *Soft*

-
- Computing - A Fusion of Foundations, Methodologies and Applications*, 10(10): 881–889, 2006.
- [24] J. S. R. Jang. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, May/Jun. 1993.
- [25] E. Entchev and L. Yang. Application of adaptive neuro-fuzzy inference system techniques and artificial neural networks to predict solid oxide fuel cell performance in residential microgeneration installation. *Journal of Power Sources*, 170:122–129, 2007.
- [26] X. J. Wu, X. J. Zhu, G. Y. Cao, and H. Y. Tu. Nonlinear modeling of a SOFC stack based on ANFIS identification. *Simulation Modelling Practice and Theory*, 16(4):399–409, Apr. 2008.
- [27] E. R. Nucci, R. G. Silva, V. R. Souza, R. L. C. Giordano, R. C. Giordano, and A. J. G. Cruz. Comparing the performance of multilayer perceptrons networks and neuro-fuzzy systems for on-line inference of bacillus megaterium cellular concentrations. *Bioprocess and biosystems engineering*, 30:429–438, 2007.
- [28] I. T. K. Güney. Computation of association probabilities for single target tracking with the use of adaptive neuro-fuzzy inference system. *Turkish Journal of Electrical Engineering and Computer Sciences*, 13(1):105–118. 2005.
- [29] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

-
- [30] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
 - [31] C. K. Tham. Reinforcement learning of multiple tasks using a hierarchical CMAC architecture. *Robotics and Autonomous Systems*, 15(4):247–274, 1995.
 - [32] B. Bhanu, P. Leang, C. Cowden, Y. Lin, and M. Patterson. Real-time robot learning. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 491–498, Seoul, Korea, May 2001.
 - [33] W. D. Smart and L. P. Kaelbling. Effective reinforcement learning for mobile robots. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3404–3410, Washington, DC, May 2002.
 - [34] K. Maček, I. Petrović, and N. Perić. A reinforcement learning approach to obstacle avoidance of mobile robots. In *7th International Workshop on Advanced Motion Control*, pages 462–466, Maribor, Slovenia, 2002.
 - [35] T. Kondo and K. Ito. A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robots control. *Robotics and Autonomous Systems*, 46(2):111–124, Feb. 2004.
 - [36] D. A. Gutnisky and B. S. Zanutto. Learning obstacle avoidance with an operant behavior model. *Artificial Life*, 10(1):65–81, 2004.
 - [37] Y. Duan and X. Hexu. Fuzzy reinforcement learning and its application in robot navigation. In *International Conference on Machine Learning and Cybernetics*, volume 2, pages 899–904, Guangzhou, Aug. 2005. IEEE.

-
- [38] M. Rodríguez, R. Iglesias, C. V. Regueiro, J. Corra, and S. Barro. Autonomous and fast robot learning through motivation. *Robotics and Autonomous Systems*, 55(9):735–740, Sep. 2007.
- [39] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3–4):257–277, 1992.
- [40] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems*, 8: 1017–1023, 1996.
- [41] H. Benbrahim. *Biped dynamic walking using reinforcement learning*. PhD thesis. Durham, NH, 1996.
- [42] H. Benbrahim and J. Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3–4):283–302, 1997.
- [43] J. Nie and S. Haykin. A dynamic channel assignment policy through Q-learning. *IEEE Transactions on Neural Networks*, 10(6):1443–1455, 1999.
- [44] P. Stone and R. S. Sutton. Scaling reinforcement learning toward robocup soccer. In *The Eighteenth International Conference on Machine Learning*, pages 537–544, San Francisco, CA, 2001.
- [45] C. Ye, N. H. C. Yung, and D. Wang. A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, 33(1):17–27. Feb. 2003.

-
- [46] M. J. Er and C. Deng. Obstacle avoidance of a mobile robot using hybrid learning approach. *IEEE Transactions on Industrial Electronics*, 52(3):898–905, Jun. 2005.
- [47] H. Xiao, L. Liao, and F. Zhou. Mobile robot path planning based on Q-ANN. In *IEEE International Conference on Automation and Logistics*, pages 2650–2654, Jinan, China, Aug. 2007.
- [48] J. V. Neuman and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press., Princeton, second edition, 1947.
- [49] R. Isaacs. *Differential Games*. John Wiley and Sons, 1965.
- [50] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18(5):662–669, Oct. 2002.
- [51] D. Li, G. Chen, C. Kwan, and M. Chang. A hierarchical approach to multi-player pursuit-evasion differential games. In *44th IEEE Conference on Decision and Control, and the European Control Conference 2005*, pages 5674–5679, Seville, Spain, Dec. 2005.
- [52] Y. Ishiwaka, T. Satob, and Y. Kakazu. An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning. *Robotics and Autonomous Systems*, 43(4):245–256, 2003.
- [53] S. N. Givigi, H. M. Schwartz, and X. Lu. A reinforcement learning adaptive fuzzy controller for differential games. *Journal of Intelligent and Robotic Systems*, 2009. DOI: 10.1007/s10846-009-9380-4.

-
- [54] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1): 1–13, 1975.
- [55] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modelling and control. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(1):116–132, Jan. 1985.
- [56] D. E. Goldberg. *Genetic Algorithms for Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, 1989.
- [57] Y. He, Y. Zhang, and L. Xiang. Study of application model on BP neural network optimized by fuzzy clustering. In *MICAI*, pages 712–720, 2005.
- [58] M. Negnevitsky. Multi-layer neural networks with improved learning algorithms. In *Digital Imaging Computing: Techniques and Applications, DICTA*, pages 1–6, Cairns, Australia, Dec. 2005.
- [59] L. Zhao and F. Y. Wang. Design for self-organizing fuzzy neural networks using a novel hybrid learning algorithm. In *IEEE Congress on Evolutionary Computation, CEC*, pages 2972–2979, Singapore, Sep. 2007.
- [60] L. Zhao and F. Y. Wang. The design of self-organizing fuzzy neural networks based on GA-ECPSO and MBP. In *IEEE International Conference on Systems, Man, and Cybernetics, ISIC*, pages 1618–1623, Montreal, QC, Canada, Oct. 2007.
- [61] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

-
- [62] R. S. Sutton. Integrated architecture for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, Austin, TX, 1990.
- [63] C. J. C. H. Watkins. *Learning From Delayed Rewards*. PhD thesis, Cambridge University, 1989.
- [64] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.
- [65] T. M. Marin and T. Duckett. Fast reinforcement learning for vision-guided mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 4170–4175, Barcelona, Spain, Apr. 2005.
- [66] Y. Yang, Y. Tian, and H. Mei. Cooperative Q-learning based on blackboard architecture. In *International Conference on Computational Intelligence and Security Workshops*, pages 224–227, Harbin, China, Dec. 2007.
- [67] S. F. Desouky and H. M. Schwartz. A novel technique to design a fuzzy logic controller using $Q(\lambda)$ -learning and genetic algorithms in the pursuit-evasion game. In *2009 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2683–2689, San Antonio, TX, Oct. 2009.
- [68] S. F. Desouky and H. M. Schwartz. A novel hybrid learning technique applied to a self-learning multi-robot system. In *2009 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2690–2697, San Antonio, TX, Oct. 2009.
- [69] L. Jouffe. Fuzzy inference system learning by reinforcement methods. *IEEE*

-
- Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 28(3):338–355, Aug. 1998.
- [70] A. Waldock and B. Carse. Fuzzy q-learning with an adaptive representation. In *2008 IEEE 16th International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 720–725, Piscataway, NJ, 2008.
- [71] V. Stephan, K. Debes, and H. M. Gross. A new control scheme for combustion processes using reinforcement learning based on neural networks. *International Journal of Computational Intelligence and Applications*, 1(2):121–36, Jun. 2001.
- [72] X. W. Yan, Z. D. Deng, and Z. Q. Sun. Genetic Takagi-Sugeno fuzzy reinforcement learning. In *IEEE International Symposium on Intelligent Control*, pages 67–72, Mexico City, Mexico, Sep. 2001.
- [73] D. Gu and H. Hu. Accuracy based fuzzy Q-learning for robot behaviours. In *International Conference on Fuzzy Systems*, volume 3, pages 1455–1460, Budapest, Hungary, Jul. 2004.
- [74] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3):123–158, Jan./Mar. 1996.
- [75] C. G. Li, M. Wang, Z. J. Huang, and Z. F. Zhang. An actor-critic reinforcement learning algorithm based on adaptive RBF network. In *2009 Eighth International Conference on Machine Learning and Cybernetics (ICMLC)*, pages 984–988, Baoding, Hebei, China, Jul. 2009.
- [76] W. M. V. Buijtenen, G. Schram, R. Babuška, and H. Verbruggen. Adaptive

-
- fuzzy control of satellite attitude by reinforcement learning. *IEEE Transactions on Fuzzy Systems*, 6(2):185–194, May 1998.
- [77] L. Jouffe. Actor-critic learning based on fuzzy inference system. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 339–344, Beijing, China, Oct. 1996.
- [78] S. F. Desouky and H. M. Schwartz. Genetic based fuzzy logic controller for a wall-following mobile robot. In *2009 American Control Conference (ACC-09)*, pages 3555–3560, St. Louis, MO, Jun. 2009. IEEE.
- [79] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
- [80] Z. Hamavand and H. M. Schwartz. Trajectory control of robotic manipulators by using a feedback-error-learning neural network. *Robotica*, 13(5):449–459, 1995.
- [81] N. Xiong and L. Litz. Reduction of fuzzy control rules by means of premise learning – method and case study. *Fuzzy Sets and Systems*, 132(2):217–231, Dec. 2002.
- [82] S. H. Lim, T. Furukawa, G. Dissanayake, and H. F. D. Whyte. A time-optimal control strategy for pursuit-evasion games problems. In *International Conference on Robotics and Automation*, New Orleans, LA, Apr. 2004.
- [83] S. F. Desouky and H. M. Schwartz. Different hybrid intelligent systems applied for the pursuit-evasion game. In *2009 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2677–2682, Oct. 2009.

- [84] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.
- [85] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, Apr. 1992.
- [86] P. Dayan. The convergence of TD(λ) for general λ . *Machine Learning*, 8(3–4):341–362, May 1992.
- [87] R. Bellman. *Dynamic programming*. Princeton University Press, Princeton, NJ, 1957.
- [88] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.
- [89] P. Dayan. TD(λ) converges with probability 1. *Machine learning*, 14(3):295–301, Mar. 1994.
- [90] J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202, Sep. 1994.
- [91] T. Jaakkola, M. I. Jordan, and P. S. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, Nov. 1994.
- [92] C. Szepesvári and M. L. Littman. Generalized markov decision processes: Dynamic-programming and reinforcement-learning algorithms. Technical report, CS-96-11, Brown University, Providence, RI, 1996.

-
- [93] G. Jiang, H. Q. Gao, C. P. Wu, and G. Cybenko. Convergence analysis of discretization procedure in Q-learning. In *Proceedings of the 14th World Congress. International Federation of Automatic Control*, volume 18, pages 281–286, Beijing, China, Jul. 1999.
- [94] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, Mar. 2000.
- [95] E. E. Dar and Y. Mansour. Learning rates for Q-learning. *Journal of Machine Learning Research*, 5:1–25, Dec. 2003.
- [96] A. Gosavi. Boundedness of iterates in Q-learning. *Systems & control letters*, 55(4):347–349, 2006.
- [97] J. Peng and R. J. Williams. Incremental multi-step Q-learning. *Machine Learning*, 22(1–3):283–290, Jan. 1996.
- [98] A. Gosavi. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.
- [99] S. F. Desouky and H. M. Schwartz. $Q(\lambda)$ -learning fuzzy logic controller for a multi-robot system. In *2010 IEEE International Conference on Systems, Man, and Cybernetics*, Istanbul, Turkey, Oct. 2010.
- [100] X. S. Wang, Y. H. Cheng, and J. Q. Yi. A fuzzy actor-critic reinforcement learning network. *Information Sciences*, 177:3764–3781, 2007.

-
- [101] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.
- [102] L. Buşoniu, R. Babuška, and B. de Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, Mar. 2008.
- [103] Y. Wang and C. W. de Silva. Extend single-agent reinforcement learning approach to a multi-robot cooperative task in an unknown dynamic environment. In *2006 International Joint Conference on Neural Networks*, pages 4999–5005, Vancouver, BC, Canada, Jul. 2006.
- [104] X. Cheng, J. Shen, H. Liu, and G. Gu. Multi-robot cooperation based on hierarchical reinforcement learning. In *Computational Science ICCS 2007*, volume 4489/2007, pages 90–97. Springer Berlin / Heidelberg, Jul. 2007.
- [105] D. Li and J. B. Jr. Cruz. Better cooperative control with limited look-ahead. In *2009 American Control Conference (ACC-09)*, pages 3555–3560, St. Louis, MO, Jun. 2009. IEEE.
- [106] M. Wei, G. Chen, J. B. Jr. Cruz, L. Hayes, and M. Chang. A decentralized approach to pursuer-evader games with multiple superior evaders. In *2006 IEEE Intelligent Transportation Systems Conference*, Toronto, ON., Canada, Sep. 2006.
- [107] M. Wei, G. Chen, J. B. Jr. Cruz, L. S. Hayes, M. Chang, and E. Blasch. A decentralized approach to pursuer-evader games with multiple superior evaders in noisy environments. In *2007 IEEE Aerospace Conference*, pages 1–10, Big Sky, MT, Mar. 2007.

- [108] X. Wang, J. B. Jr. Cruz, G. Chen, K. Pham, and E. Blasch. Formation control in multi-player pursuit evasion game with superior evaders. *Proceedings of the SPIE - The International Society for Optical Engineering*, 6578:657811–1, Apr. 2007.
- [109] S. F. Desouky and H. M. Schwartz. Learning in n-pursuer n-evader differential games. In *2010 IEEE International Conference on Systems, Man, and Cybernetics*, Istanbul, Turkey, Oct. 2010.