

Fuzzy reinforcement learning and its applications to mobile robot navigation

Deng, Chang

2005

Deng, C. (2005). Fuzzy reinforcement learning and its applications to mobile robot navigation. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/4204>

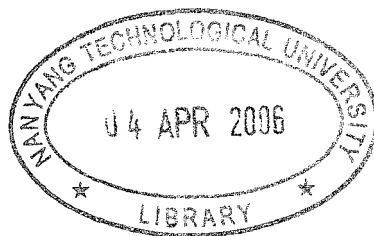
<https://doi.org/10.32657/10356/4204>

Nanyang Technological University

Downloaded on 09 Apr 2024 16:15:00 SGT

**Fuzzy Reinforcement Learning
and
Its Applications to Mobile Robot Navigation**

Deng Chang



School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2005

Acknowledgements

I would like to acknowledge the help rendered by many people who have contributed in one way or another towards the successful completion of this thesis.

First, I would like to thank my supervisor, Associate Professor Er Meng Joo, for giving me the freedom and encouragement to pursue my research. His continuous and valuable guidance throughout the research is the most important factor for the completion of this work.

Next, I would like to thank all my friends for their support and encouragement during the course of my study.

Special thanks should go to my colleagues and technical staff at the Instrumentation & System Engineering Laboratory for their excellent technical support.

I would like to acknowledge the generous support of School of Electrical and Electronic Engineering, Nanyang Technological University for providing me an opportunity to pursue a higher degree with a Research Scholarship, and the use of research facilities.

And lastly, my thanks to my dearest parents. They always give me boundless and unconditional love. My gratitude is beyond words.

Table of Contents

Acknowledgements	i
Table of Contents	ii
List of Figures	v
List of Tables	vi
Summary	vii
Chapter 1 Introduction	1
1.1 Motivation	1
1.1.1 Motivation Based on Fuzzy Systems.....	1
1.1.2 Motivation Based on Reinforcement Learning.....	4
1.1.3 Motivation Based on Robot Learning.....	7
1.2 Major Contributions of Thesis	10
1.3 Outline of Thesis.....	14
Chapter 2 Fuzzy Systems	17
2.1 General Fuzzy Systems.....	17
2.1.1 Fuzzy Set	18
2.1.2 Fuzzy If-Then Rules	20
2.1.3 Fuzzy Inference Systems (FISs)	22
2.1.4 Design Problems in FISs	25
2.2 Learning Paradigms of Fuzzy Systems	27
2.2.1 Supervised Learning	28
2.2.2 Unsupervised Learning.....	31
Chapter 3 Reinforcement Learning	37
3.1 Basic Framework	37
3.1.1 Reinforcement Learning Model.....	38
3.1.2 Markov Decision Processes	41
3.1.3 Learning an Optimal Policy.....	44
3.1.4 Exploration/Exploitation Tradeoff.....	49
3.2 Generalization.....	50

3.2.1	Generalization in States	50
3.2.2	Generalization in States and Actions	52
3.3	Applications in Robotics.....	53
3.3.1	Robot Learning	54
3.3.2	Problems	56
3.3.3	The Khepera Robot	58

Chapter 4 Design of Dynamic Fuzzy Q-Learning (DFQL) Algorithm.....61

4.1	Motivation and Development	61
4.2	Preceding Works.....	65
4.3	Architecture of DFQL.....	67
4.4	DFQL Learning Algorithm.....	72
4.4.1	Generation of Continuous Actions	72
4.4.2	Update of q-values	73
4.4.3	Eligibility Traces	74
4.4.4	ϵ -Completeness Criterion for Rule Generation.....	75
4.4.5	TD Error Criterion for Rule Generation	76
4.4.6	Estimation of Premise Parameters	77
4.4.7	Working Principle.....	80
4.5	Discussions	83
4.6	Experiments	85
4.6.1	Obstacle Avoidance for Khepera Robot	85
4.6.2	Random Policy and Tabular Q-learning.....	87
4.6.3	Neural Q-Learning.....	89
4.6.4	DFQL	90

Chapter 5 Embedding Initial Knowledge in DFQL99

5.1	Efficient Use of Initial Knowledge.....	99
5.1.1	Build-in Bias	100
5.1.2	Initial Knowledge from Fuzzy Rules.....	101
5.2	Experiments	103
5.2.1	Wall Following Task for Khepera Robot	103
5.2.2	Basic Fuzzy Controller	105
5.2.3	Fuzzy Q-Learning with a Fixed Structure	106
5.2.4	DFQL	111
5.2.5	Adaptation in a New Environment.....	112
5.2.6	Continuous-Action Q-Learning	114

Chapter 6 General DFQL with Eligibility Traces	118
6.1 General DFQL	118
6.1.1 Eligibility Traces.....	118
6.1.2 The General DFQL Learning Algorithm.....	120
6.2 Experiments	123
6.2.1 Optimum-Path Task.....	123
6.2.2 Learning Details.....	126
6.2.3 Learning Results	130
6.3 Discussions.....	134
Chapter 7 Conclusions and Future Works	136
7.1 Conclusions.....	136
7.2 Recommendations for Further Research.....	139
7.2.1 The Convergence Property	139
7.2.2 Partially Observable Environments	140
7.2.3 Integrating Planning and Learning.....	141
7.2.4 Multi-Agent Systems	141
Author's Publications	143
Bibliography	145

List of Figures

Figure 2.1. Two common fuzzy membership functions	20
Figure 2.2. Fuzzy inference systems	22
Figure 2.3. TKS-type fuzzy reasoning	25
Figure 3.1: A general model for the reinforcement learning agent	39
Figure 3.2. The miniature mobile robot: Khepera	58
Figure 3.3. The Khepera robot and its working environment	59
Figure 3.4. Position and orientation of sensors on the Khepera	60
Figure 4.1: Structure of fuzzy rule sets of DFQL	67
Figure 4.2. Consequent parts of DFQL	71
Figure 4.3. Flowchart of the DFQL learning algorithm	82
Figure 4.4. Real environment for obstacle avoidance	85
Figure 4.5. Simulation environment for obstacle avoidance	86
Figure 4.6: Distance to the obstacles and local performance indices based on a random exploration policy	93
Figure 4.7: Distance to the obstacles and local performance indices during learning with classical Q-learning	94
Figure 4.8: Distance to the obstacles and local performance indices during learning with Q-KOHON	95
Figure 4.9: Distance to the obstacles and local performance indices during learning with DFQL	96
Figure 4.10. Number of fuzzy rules generated by DFQL during learning	97
Figure 4.11 Distance to the obstacles during learning with (a) the random policy (b) Basic Q-learning (c) Q-KOHON, (d) DFQL	97
Figure 4.12: Discounted cumulative rewards obtained during learning with (a) the random policy (b) Basic Q-learning (c) Q-KOHON (d) DFQL	98
Figure 5.1: Actual environment for wall-following experiments	104
Figure 5.2. Simulation environment for wall-following experiments	104
Figure 5.3: Comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL (b) 81 fuzzy rules based on FQL (c) Basic fuzzy controller with 16 fuzzy rules	108
Figure 5.4: Comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL (b) 81 fuzzy rules based on FQL (c) Basic fuzzy controller with 16 fuzzy rules (d) Fuzzy controller based on DFQL	109
Figure 5.5. Number of fuzzy rules generated by DFQL during learning	110
Figure 5.6. Membership functions after learning at one run	110
Figure 5.7. The new training environment with obstacles	112

Figure 5.8: Performance comparison of DFQL with training directly and retraining in a new environment.....	1 13,114
Figure 5.9: Performance comparison of updating Q-values for Continuous-Action Q-Learning and DFQL.....	117
Figure 6.1 : Architecture of bias component	127
Figure 6.2: Sampling trajectory generated during first and final episodes	130
Figure 6.3: Number of steps taken in each episode when $\lambda = 0.9$ (a) typical single run (b) average of 10 runs..	1 30
Figure 6.4: The total reinforcement received in each episode when $\lambda = 0.9$ (a) typical single run (b) average of 10 runs	131
Figure 6.5: Number of fuzzy rules generated at the end of each episodes $\lambda = 0.9$ (a) typical single run (b) average of 10 runs	131
Figure 6.6: Comparison of number of steps taken during learning when different λ are used (a) $\lambda = 0.0$ (b) $\lambda = 0.5$ (c) $\lambda = 0.9$ (d) $\lambda = 1.0$	133
Figure 6.7: Comparison of the total reinforcement received during learning when different λ are used (a) $\lambda = 0.0$ (b) $\lambda = 0.5$ (c) $\lambda = 0.9$ (d) $\lambda = 1.0$	134

List of Tables

Table 5.1: Basic fuzzy control rules for wall following	105
---	-----

Summary

Fuzzy logic is a mathematical approach to emulate the human way of thinking. It has been shown that fuzzy logic could serve as a powerful methodology for dealing with imprecision and nonlinearity efficiently. However, the conventional way of designing a fuzzy system has been a subjective approach. If the fuzzy system somehow possesses learning abilities, an enormous amount of human efforts would be saved from tuning the system.

Reinforcement learning is concerned with resolving a problem faced by a learner that must learn behavior through trial-and-error interactions with a dynamic environment. For this kind of learning problem, training data give rewards and punishments with respect to the states reached by the learner, but do not provide correct instructions. Q-learning is the most popular and effective model-free algorithm for reinforcement learning. However, it does not address any of the issues involved in generalization over large state and action spaces. Practical learning agents require compact representations to generalize experiences in continuous domains.

In this thesis, a novel algorithm, termed Dynamic Fuzzy Q-Learning (DFQL), is proposed. From the point of view of fuzzy systems, the DFQL method is a learning method capable of generating and tuning fuzzy rules automatically based on simple reinforcement signals. From the point of view of machine learning, the DFQL method is a mechanism of introducing generalization in the state-space and generating continuous actions in reinforcement learning problems. The DFQL generalizes the continuous input space with fuzzy rules and generates continuous-valued actions using fuzzy reasoning. It partitions the input space online dynamically according to both the accommodation boundary and the performance

of learning, which allows us to circumvent the problem of setting up fuzzy rules by hand. The compact fuzzy system considers sufficient rules in the critical state space which requires high resolution and does not include the redundant rules in the unimportant or unvisited state space so that the learning is rapid and optimal.

The if-then fuzzy rules correspond to the initial domain knowledge about the tasks and allow incorporation of bias into the system. Bias increments the safety of the learning process and accelerates the learning process since it focuses on the search process of promising parts of the action space immediately. These biases can eventually be overridden by more detailed and accurate learned knowledge. The premise of rules can be used to generate fuzzy states over the input space and the consequents of rules can be used to generate the initial Q-values so that a greedy policy would select the action suggested by these biases.

The general version of DFQL with an eligibility mechanism leads to faster learning and alleviate the non-Markovian effect in real-life applications. It figures out which actions in the sequence are primarily responsible for the received reward and has the ability of exploration insensitivity, the ability to learn without necessarily following the current policy. This method allows us to obtain a significant learning speedup using the eligibility rate and alleviates the experimentation-sensitive problem at the same time.

Chapter 1

Introduction

1 . Motivation

1.1.1 Motivation Based on Fuzzy Systems

Fuzzy logic is a mathematical approach to model the human way of thinking. It provides a systematic and effective means of capturing the imprecise and nonlinear nature of the real world linguistically. In the literature, there are two kinds of justification for fuzzy theory [144]:

- The real world is too complicated for precise descriptions to be obtained; therefore, fuzziness must be introduced in order to obtain a reasonable model.
- As we move into the information era, human knowledge becomes increasingly important. We need a theory to formulate human knowledge in a systematic manner and put it into engineering systems, together with other information like mathematical models and sensory measurements.

Fuzzy Inference Systems (FISs) are knowledge-based or rule-based systems. The essential part of the FIS is a set of linguistic rules related by the dual concept of fuzzy implication and the compositional rule of inference. Intrinsically, the FIS provides an algorithm, which can convert the linguistic rules based on expert knowledge into some automatic control action. During the last few decades, FISs have emerged as one of the most active and fruitful areas for research in the application of fuzzy theory. FISs have found a variety of applications in numerous fields ranging from industrial process control to medical diagnosis and robotics.

In general, subjective approaches to design a fuzzy system are simple and fast, i.e., they involve neither time-consuming iterative procedures nor a complicated rule-generation mechanism. However, argument of what is the best approach can come from disagreements among experts, from decision rules that are difficult to structure, or due to a great number of variables necessary to solve the control task. If the fuzzy system somehow possesses learning abilities, an enormous amount of human efforts would be saved from tuning the system. These learning methods can be characterized by the information source used for learning and classified with respect to the degree of information of the source. Most of the learning algorithms for fuzzy systems require precise training data sets for various applications. Typically, these learning methods are based on an input-output set of training data, based upon which we have to minimize errors between the teacher's actions and the learner's actions. However, for some real-world applications, precise data for training/learning are usually difficult and expensive, if not impossible, to obtain. For this reason, there has been a growing interest in this kind of learning.

There are several requirements for a learning algorithm to model a FIS effectively.

- Evaluative signals

The learning systems can be used to provide unknown desired outputs based on a suitable evaluation of system performances, which gives rewards and punishments with respect to the states reached by the learner, but does not provide correct actions. For this kind of learning problem, training data are very rough and coarse, and are just "evaluative" as compared with the "instructive" feedback. The learning algorithm should be capable of constructing a FIS based on this simple evaluative scalar signal. As the less informative learning source is needed, the learning method that uses it represents a very flexible tool. In addition to the roughness and non-instructive nature of the information, a more challenging problem the algorithm should be able to deal with is that the signal may only be available at a time long after a sequence of actions have been taken. In other words, prediction capabilities are necessary in this learning system.

- Structure and parameter learning

Although several self-learning FISs have been designed, most of them focus on parameter learning (e.g. adjustment of fuzzy rule parameters). Structure learning (e.g. determination of input space partition, number of membership functions and number of rules) remains a trial-and-error process and it has become a very time-consuming process. It turns out that only adjustment of parameters will not be sufficient in many cases. This reduces the flexibility and numerical processing capability of FISs. The algorithm should deal with not only parameter estimation but also structure identification of a learning FIS. If the premise structure of a fuzzy system is determined by clustering the input via on-line self-organizing learning approach, a more flexible learning scheme can be formed. Furthermore, the

learning method should find meaningful fuzzy terms for each input variable, from which it is possible to interpret acquired knowledge in the form of linguistic rules.

- On-line learning

We are interested in on-line learning, algorithms which are capable of learning the dynamics of a system based on data which arrive one at a time. The main idea is that we do not have to wait for a large batch of data points before training the algorithm. No prescribed training models are needed for on-line learning and the system can learn adaptively from the training data which arrive sequentially.

1.1.2 Motivation Based on Reinforcement Learning

Reinforcement Learning (RL) dates back to the early days of cybernetics and works in Statistics, Psychology, Neuroscience and Computer Science. In the last decade, it has attracted rapidly growing interest in machine learning and artificial intelligence communities. The key idea of RL is that the learner learns through trial-and-error interactions with a dynamic environment. It is learning how to map situations to actions so as to maximize some numerical reward. It should be highlighted that the learner is not told which actions to take, as in other types of machine learning, but instead it discovers which actions yield the most reward by trying them. In the most interesting and challenging case, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards [128].

RL plays an important role in adaptive control. It will certainly help, especially when no explicit teacher signal is available in the environment where an

interacting learner must learn to perform an optimal control action. There are mainly two prevalent approaches to reinforcement learning, namely actor-critic learning and Q-learning. The actor-critic model typically includes two principal components: the critic module and the action module. The critic module generates an estimate of the value function from state vectors and external reinforcement supplied by the environment as inputs. The actor attempts to learn optimal control or decision-making skills. Q-learning is a simple way of learning the action-value function Q that maps state-action pairs to expected returns. The learner attempts an action at a particular state and evaluates its consequence in terms of the immediate reward or penalty it receives and its estimate of the value of the state resulting from the taken action.

We focus on the Q-learning method here since Q-learning is the most popular RL method that directly calculates the optimal action policy without an intermediate cost evaluation step and without the use of a model. It seems to be more difficult to work with actor-critic architectures than Q-learning in practice. It might be very difficult to get the relative learning rates right in actor-critic architectures so that the two components converge together. Furthermore, Q-learning learns the values of all actions, rather than just finding the optimal policy. The main advantage of Q-learning over actor-critic learning is exploration insensitivity, i.e. any action can be carried out at any time and information is gained from this experience.

For these reasons, Q-learning is the most popular and seems to be the most effective model-free algorithm for RL. It does not, however, address any of the issues involved in generalization over large state and/or action spaces. In addition, it may converge quite slowly to a good policy. There are also several requirements for a learning algorithm before it can be used in practice.

- Adaptive generalization

Q-learning with standard tabular states and actions scale poorly. As the number of state and action variables increases, the size of the table used to store Q-values grows exponentially. The large number of states and actions possibilities usually encountered in realistic applications may force us to use some compact representation schemes than a table. The problem of learning in large spaces is addressed through generalization techniques, which allow compact storage of learned information and transfer of knowledge between similar states and actions. Furthermore, it would be desired to employ an online adaptive construction algorithm instead of partitioning the state space evenly prior to learning so as to improve generalization capabilities at the state spaces that are deemed to be important or critical.

- Continuous states and actions

Many real-world control problems require action of a continuous nature in response to continuous state measurements. Most approaches use function approximators to generalize the value function across situations. These works, however, still assume discrete actions and cannot handle continuous-valued actions. In continuous action spaces, the need for generalization over actions is important. It should be possible that actions vary smoothly in response to smooth changes in a state.

- Integration of domain knowledge

The algorithm is used for fast on-line learning and adaptation in real time. Initially, if the learning system knows nothing about the environment, it is forced to act more or less arbitrarily. Integration of domain knowledge to avoid learning from scratch is desired. A way of alleviating the problem of

slow convergence of RL is to use bias from prior knowledge to figure out which part of the action space deserves attention in each situation.

- Eligibility traces

Most of RL methods need to be combined with eligibility traces to obtain more general methods that may learn more efficiently. An eligibility trace is a temporary record of the occurrence of an event. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. The learning algorithm should be able to distribute credit throughout sequences of actions, leading to faster learning and help to alleviate the non-Markovian effect in real applications. It should have the ability of exploration insensitivity, and the ability to learn without necessarily following the current policy.

- Incremental and aggressive learning

The learning algorithm should not be subject to destructive interference or forgetting what it has learned so far but incrementally adapt the model complexity. It should be capable of producing reasonable predictions based on only a few training points.

1.1.3 Motivation Based on Robot Learning

As the robotics field progresses, robots are being employed in increasingly complicated and demanding tasks. To accomplish a given task, a robot collects or receives sensory information concerning its external environment and takes actions within the dynamically changing environment. Both the sensing system and control rules are often dictated by human operators, but ideally the robot should automatically perform the given tasks without assistance from human operators.

Consequently, the robot must be able to perceive the environment, make decisions, represent sensory data, acquire knowledge, and infer rules concerning the environment. The ultimate goal of robotics research is to empower the robots with high autonomous ability to improve their behavior over time, based on their incoming experiences.

Because we are dealing with robotic systems, there are a number of real-world issues that must be addressed [28]. Some of these are:

- Training data

Generally speaking, the robot learning problem is to infer a mapping from sensors to actions given a training sequence of sensory inputs, action outputs, and feedback values. If these sequences are provided by a teacher, the problem corresponds to supervised learning. Here, the robot is being passively guided through the task. A more challenging and interesting situation arises when a robot attempts to learn a task in an unsupervised mode without active guidance of a teacher. It is usually assumed here that the robot can recognize when it is performing the task properly. The robot has to perform the task by executing trial-and-error actions thereby exploring the state space.

- Continuous states and actions

In many real-world tasks for robots, the sensory and action spaces are continuous. These values can be discretized into finite sets if the discretization follows the natural resolution of the devices. However, many quantities are inherently continuous with a fine resolution that leads to many discrete states. Even if they can be discretized meaningfully, it might not be readily apparent how best to do it for a given task. Incorrect

discretizations can limit the final form of the learned control policy, making it impossible to learn the optimal policy. If we discretize coarsely, we risk aggregating states that do not belong together. If we discretize finely, we often end up with an unmanageably huge state or action space. Practical learning robots require compact representations to generalize experiences in continuous domains. Furthermore, actions should vary smoothly in response to smooth changes in a state.

- Incremental learning

A robot has to collect the experience from which it is to learn the task. The data forming the experience is not available offline. The need for efficient exploration dictates that any reasonable learning algorithm must be incremental. Such algorithms should allow the robot to become better at deciding which part of the environment it needs to explore next.

- Initial knowledge

Many learning systems attempt to start learning with no initial knowledge. Although this is appealing, it introduces special problems when working with real robots. Initially, if the learning system knows nothing about the environment, it is forced to act more or less arbitrarily. For example, RL systems attempt to learn the policy by attempting all of the actions in all of the available states in order to rank them in the order of appropriateness. In order to learn a new policy, a large number of time-consuming learning trials are required. On the other hand, critical behavior must be learned with a minimal number of trials, since the robot cannot afford to fail repeatedly. When a real robot is being controlled, a bad choice can damage the environment or the robot itself, possibly causing it to stop functioning. In order for the learned system to be effective, we need to provide some

sort of bias, to give it some idea of how to act initially and how to begin to make progress towards the goal. Systems should have the ability to use previously learned knowledge to speed up the learning process of a new policy.

- Time constraints

The training time available on a real robot is very limited. We are interested in learning on-line while the robot is interacting with the environment. Although computers are continually becoming faster, the amount of computation that we can apply to learning is limited. This is especially important when we are using the learned control policy to control the robot. We must be able to select a suitable control action at an appropriate rate to allow the robot to function safely in the real world.

- Sensor noise

Most cheap-to-build robot sensors are unreliable. Thus, state descriptions computed from such sensors are bound to have inaccuracies in them, and some kind of averaging is required.

1.2 Major Contributions of Thesis

In this thesis, a novel algorithm, termed Dynamic Fuzzy Q-Learning (DFQL), is proposed. From the viewpoint of fuzzy systems, the DFQL method is a learning method capable of tuning a fuzzy system in a very flexible way. From the viewpoint of machine learning, the DFQL method can be regarded as a means of introducing generalization in the state space and generate continuous actions in RL problems. It is implemented on mobile robots so as to learn appropriate navigation

efficiently. The salient characteristics of the DFQL algorithm are summarized as follows:

- Reinforcement information source

The DFQL is based on the Q-learning, the most popular and effective reinforcement learning. The task is described with a reinforcement function, which can be a simple description of success and failure actions. Due to the low informative degree of the information source, the method represents a very flexible tool.

- Self-organizing fuzzy system structure

The DFQL provides an efficient learning way whereby not only the conclusion part of a FIS can be adjusted online, but also the structure of a FIS can be constructed simultaneously. Based on the criteria pertinent to some desired system performance, new fuzzy rules can be generated automatically so as to improve generalization capabilities when necessary.

- Continuous states and actions

In the DFQL, continuous states are handled and continuous actions are generated by fuzzy reasoning. Fuzzy logic introduces generalization in the state space by means of using a vector of fuzzy variables to describe a fuzzy state. The continuous action performed by the learner for a particular state is a weighted sum of the actions elected in the fired rules that describe this state, whose weights are normalized firing strengths of the rules. Since more than one fuzzy state may be visited at the same time, possibly with different degrees, there will be a smooth transition between a state and its neighbors, and, consequently, smooth changes of actions carried out in the different states.

- Incorporating initial knowledge

The if-then fuzzy rules correspond to the initial domain knowledge about the tasks and allow incorporation of bias into the system. Bias accelerates the learning process since it focuses on the search process of promising parts of the action space immediately. These biases can eventually be overridden by more detailed and accurate learned knowledge. Fuzzy rules provide a natural framework of incorporating the bias components for rapid and safe learning. The premise of rules can be used to generate fuzzy states over the input space and the consequents of rules can be used to generate the initial Q-values so that a greedy policy would select the action suggested by these biases.

- Eligibility trace mechanism

The DFQL can be extended to the general version with an eligibility mechanism leading to faster learning, especially from delayed reinforcement. It figures out which actions in the sequence are primarily responsible for the received reward and has the ability of exploration insensitivity, i.e. the ability to learn without necessarily following the current policy. The capability makes it much more appealing for efficient implementation of RL in real-life applications.

- On-line incremental learning

The DFQL is primarily concerned with how to obtain an optimal policy when a model is not known in advance. The learner interacts with its environment directly to obtain the information. No prescribed training models are needed for on-line learning. The DFQL can learn adaptively from the training data set sequentially. The control knowledge is distributively represented in the fuzzy rules. With increasing fuzzy rules

according to the system performance, the learner can incrementally adapt the environment complexity.

- Fast and adaptive learning

The DFQL has a fast learning speed since structure and parameters formulation are done automatically and systematically without partitioning the input space *a priori*. The use of fuzzy theory provides the ability to deal with uncertain and imprecise data in the real world.

The fuzzy rule format makes it easy to implement effective navigation tasks for mobile robots. The interpolation mechanism implemented by fuzzy controllers results in smooth motion of the robot. Thus, compared with the Q-learning method with discrete actions, the DFQL method is superior by virtue of its capability of handling continuous-valued states and actions. Because fuzzy rules can be generated automatically according to system performance, the DFQL is more flexible than fuzzy Q-learning with a fixed structure. A comparative study with the Continuous-Action Q-Learning approach, the only approach which is capable of generating continuous actions by means of Q-learning, also demonstrates the superiority of the DFQL method.

The general version of DFQL with an eligibility mechanism leads to faster learning and alleviate the non-Markovian effect in real-life applications. Simulation studies on searching for optimum paths of the robot demonstrate the efficiency of the method for learning the appropriate policy in multi-step prediction problems. We examine issues pertaining to efficient and general implementation of the DFQL for different eligibility rates for optimizing the sum of rewards. This method allows us to obtain a significant learning speedup using

the eligibility rate and alleviates the experimentation-sensitive problem at the same time, which is superior to the other methods based on the actor-critic learning.

1.3 Outline of Thesis

This thesis is organized in seven chapters each of which is devoted to a particular sub-issue. A summary of the content of each chapter is given here:

- Chapter 1 presents motivations and contributions of the thesis and gives a brief outline of each chapter in the thesis.
- Chapter 2 introduces the foundation of the FIS and a computation framework based on the concepts of fuzzy sets, fuzzy IF-THEN rules and fuzzy reasoning. Intrinsically, the FIS provides an algorithm, which can convert the linguistic rules based on expert knowledge into some automatic control action. In order to circumvent the problem of subjective approaches in designing the FIS, we present the current research on finding automatic methods of self-tuning of FISs. The main issues associated with learning abilities of FISs are parameter estimation and structure identification. We discuss two families of learning methods, namely supervised learning and unsupervised learning, characterized by the information source used for learning.
- Chapter 3 gives an overview of the field of RL, which has only very simple "evaluative" or "critic" information instead of "instructive" information available for learning. We focus on the Q-learning method which is the most popular and arguably the most effective model-free algorithm for RL, learning. Furthermore, the generalization techniques, which allow compact

storage of learned information and transfer of knowledge between similar states and actions are addressed in order to deal with the continuous spaces and actions in practice. Subsequently, we discuss a collection of robotics applications and the major application area in RL. Finally, we provide an overview of the miniature mobile robot (Khepera) used for the experiments described in this thesis.

- Chapter 4 investigates requirements of learning methods of fuzzy systems based on RL and requirements of generalization techniques of Q-learning. Subsequently, we present the development of the proposed DFQL to deal with these requirements. The DFQL architecture and on-line structure and parameter learning algorithm for constructing the DFQL automatically and dynamically are described in details. Finally, experiments performed on the Khepera robot for the obstacle avoidance task are used to verify the efficiency of the proposed DFQL. Compared with the random policy, the Q-learning method and the Q-KOHON method, the DFQL method is superior because of its capability of handling continuous-valued states and actions.
- Chapter 5 describes the natural framework incorporating the initial knowledge to the learning system based on fuzzy rules. We explore the use of reflexes to make learning safer and more efficient. The reflexes correspond to domain knowledge about the task and allow incorporation of bias into the system. Experiments performed on the Khepera robot for the wall following behavior are investigated. A comparative study of the Fuzzy Q-Learning, Continuous-Action Q-Learning and our approach is carried out. All of these methods can handle continuous states and actions and

incorporate initial knowledge for rapid learning and experimental results demonstrate the superiority of the proposed DFQL method.

- Chapter 6 extends the DFQL learning algorithm to the general version with an eligibility mechanism, leading to faster learning and alleviating the experimentation-sensitive problem. We provide a literature review on the eligibility trace mechanism and describe advantages of our design. Next, details of the general DFQL learning algorithm are presented. Subsequently, simulation studies of the general DFQL on optimum-path experiments performed on the Khepera robot demonstrate the efficiency of the method for learning the appropriate policy with a few trials. Finally, we discuss related works based on two prevalent approaches to RL, namely Q-learning and actor-critic learning.
- Chapter 7 concludes this thesis and suggests several promising directions for future research based on the results presented in this thesis. Some theoretical results concerning convergence of the system based on linear architecture with fuzzy basis functions are discussed. Potential algorithms which are used in partially observable environments, based on planning models and deployed for multi-agent systems are suggested.

Chapter 2

Fuzzy Systems

It has been shown that fuzzy-logic-based modeling and control could serve as a powerful methodology for dealing with imprecision and nonlinearity efficiently [47,100]. In this chapter, we begin by introducing the basic concept of Fuzzy Inference Systems (FISs). We then introduce and discuss some issues concerning learning paradigms of fuzzy systems based on different learning methods characterized by the information source used for learning.

2.1 General Fuzzy Systems

Fuzzy logic, first proposed by Lotfi Zadeh in 1965 [150], is primarily concerned with representations of imprecise knowledge which is common in many real-life systems. It facilitates representations of knowledge through the use of fuzzy sets in digital computers. On this basis, fuzzy logic uses logical operators to collate and integrate human knowledge in order to generate some kind of reasoning common in natural intelligence.

An FIS is a computation framework based on the concepts of fuzzy sets, fuzzy if-then rules and fuzzy reasoning. FISs are known by other names such as fuzzy rule-

based systems, fuzzy models or simply fuzzy systems. The essential part of the FIS is a set of linguistic rules related by the dual concept of fuzzy implication and the compositional rule of inference. Intrinsically, the FIS provides an algorithm, which can convert the linguistic rules based on expert knowledge into an automatic control action. Many experiments have shown that FISs yield results far more superior to those obtained by conventional approaches. In particular, the methodology of FISs appears very useful when the processes are too complex for analysis by conventional quantitative techniques or when the available sources of information are interpreted qualitatively, inexactly or uncertainly. Thus, FISs may be viewed as a step towards the approach between conventional precise mathematical paradigms and human-like decisions making [47,100,143,144].

2.1.1 Fuzzy Set

Conventional set theory is based on the premise that an element either belongs to or does not belong to a given set. Fuzzy set theory takes a less rigid view and allows elements to have degrees of membership of a particular set such that elements are not restricted to either being in or out of a set but are allowed to be "somewhat" in. In many cases, this is a more natural approach.

In fuzzy set theory, a precise representation of imprecise knowledge is not enforced since strict limits of a set are not required to be defined; instead, a membership function is defined. A membership function describes the relationship between a variable and the degree of membership of the fuzzy set that corresponds to some specific values of that variable. This degree of membership is defined in terms of a number between 0 and 1 inclusive, where 0 implies total absence of membership, 1 implies complete membership, and any value in between implies partial membership of the fuzzy set. This may be written as $\mu(x) \in [0,1]$ for $x \in U$,

where $\mu(\cdot)$ is the membership function and U is the universe of discourse which defines the total range of interest over which the variable x should be defined.

There are various possible types of fuzzy membership functions and these will each provide a different meaning for the fuzzy values that they quantify. Fuzzy values are sometimes also called linguistic values. We describe two most commonly used membership functions, the triangular and Gaussian membership functions, which represent a very easy way to compute the degree of input variable membership. A triangular membership function is specified by three parameters $\{a, b, c\}$ as follows:

$$\mu(x; a, b, c) = \begin{cases} 0, & x \leq a. \\ \frac{x-a}{b-a}, & a \leq x \leq b. \\ \frac{c-x}{c-b}, & b \leq x \leq c. \\ 0, & c \leq x. \end{cases} \quad (2.1)$$

The parameters $\{a, b, c\}$ with $(a < b < c)$ determine the x coordinates of the three comers of the underlying triangular membership functions. A Gaussian membership function is specified by two parameters $\{c, \sigma\}$

$$\mu(x; c, \sigma) = \exp \left[-\frac{(x-c)^2}{\sigma^2} \right] \quad (2.2)$$

A Gaussian membership function is determined completely by c and σ , where c represents the center of a membership function and σ determines the width of a membership function. The shapes of the membership function are shown in Figure 2.1.

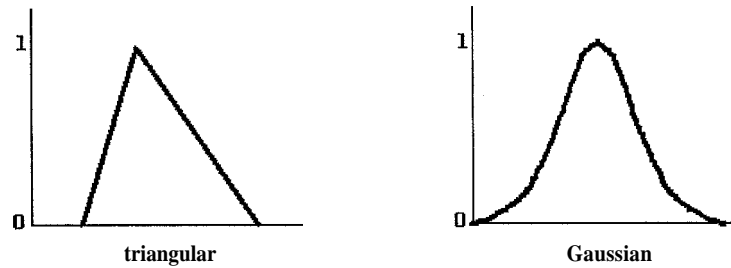


Figure 2.1 Two common fuzzy membership functions

While seemingly imprecise to a human being, fuzzy sets are mathematically precise in that they can be fully represented by exact numbers. They can therefore be seen as a method of combining human and machine knowledge representation together. Given that such a natural method of representing information in a computer exists, information processing methods can be applied to it by the use of FISs.

2.1.2 Fuzzy If-Then Rules

FISs are essentially knowledge-based or rule-based systems, which comprise a collection of rules each of which defines a desired action when a particular combination of fuzzy values occurs. The rules are defined in IF-THEN form as follows:

$$\text{If premise Then consequent} \quad (2.3)$$

Usually, the inputs of the fuzzy system are associated with the premise, and the outputs are associated with the consequent. The basic form of a linguistic rule is

$$R_j: \text{If } (x_i \text{ is } F_i^j) \text{ Then } (y_k \text{ is } G_k^j) \quad (2.4)$$

where x_i , $i = 1, \dots, n$ and y_k , $k = 1, \dots, m$ are input and output linguistic variables respectively, F_i^j , $i = 1, \dots, n$, $j = 1, \dots, l$ and G_k^j , $k = 1, \dots, m$, $j = 1, \dots, l$ are

linguistic variables or labels of fuzzy sets characterized by appropriate membership functions $\mu_i^j(x_i)$ and $\mu_k^j(y_k)$ respectively, and $R_j, j=1,\dots,l$ represents the j th fuzzy rule.

Since the output linguistic variables of a Multi-Input Multi-Output (MIMO) rule are independent, a MIMO FIS can be represented as a collection of Multi-Input Single-Output (MISO) FISs by decomposing the above rule into m sub-rules with $G_k, k=1,\dots,m$ as the single consequent of the k th sub-rule [100,144]. For notational simplicity, we would consider MISO FISs in the rest of the chapter.

Another form of fuzzy IF-THEN rules has fuzzy sets involved only in the premise part. This form of fuzzy IF-THEN rules can be categorized into two models, namely Simplified Model and Takagi-Sugeno-Kan Model.

- Simplified Model (S-model): In S-model, a fuzzy singleton is used for the output [144], i.e.

$$R_j: \text{If } (x_i \text{ is } F_i^j) \text{ Then } y \text{ is } C \quad (2.5)$$

where C is a fuzzy singleton.

- Takagi-Sugeno-Kan Model (TSK-model): Takagi and Sugeno in 1985 [129] proposed the following fuzzy model:

$$R_j: \text{If } (x_i \text{ is } F_i^j) \text{ Then } y \text{ is } g_j(x_1, \dots, x_n) \quad (2.6)$$

The premise of this rule is defined in the same way as that for the rule of the standard fuzzy system. However, the consequents of the rules are different. Instead of a linguistic term with an associated membership function, in the consequent, we use a function that does not have an associated membership function. Usually, $g(x_1, \dots, x_n)$ is a polynomial in

the input variables, but it can be any function as long as it can appropriately describe the output of the model within the fuzzy region specified by the antecedent of the rule. If no input variables are considered, the TSK-model is exactly the same as the S-model. Therefore, the TSK-model can be considered as a special case of the S-model. Experiments show that the TSK-model has advantages like computational efficiency, compatibility with linear, adaptive and optimization techniques and continuity of the output surface.

Both types of fuzzy IF-THEN rules have been extensively used in both modeling and control. Through the use of linguistic labels and membership functions, a fuzzy IF-THEN rule can easily capture the spirit of a "rule of thumb" used by human beings [47]. From another point of view, due to the qualifiers on the premise parts, each fuzzy IF-THEN rule is actually a local description of the system under consideration. On the contrary, conventional approaches of system modeling operate on the entire scope to find a global functional or analytical structure of a nonlinear system.

2.1.3 Fuzzy Inference Systems (FISs)

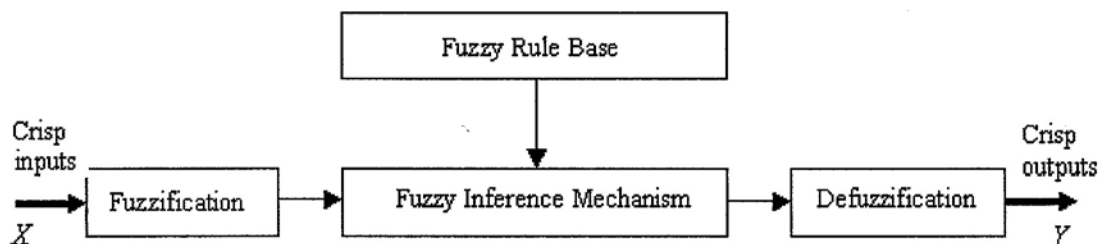


Figure 2.2 Fuzzy inference systems

The basic configuration of FISs is shown in Figure 2.2. An FIS can be defined as a system which transforms or maps one collection of fuzzy or crisp values to another collection of fuzzy or crisp values. This mapping process is performed by four parts:

- Fuzzification - Converts a set of crisp variables into a set of fuzzy variables to enable the application of logical rules.
- Fuzzy Rule Base – Stores a collection of logical IF-THEN rules.
- Fuzzy Inference Mechanism - An algorithm which is used for calculating the extent to which each rule is activated for a given input pattern.
- Defuzzification – Converts a set of fuzzy variables into crisp values in order to enable the output of the FIS to be applied to another non-fuzzy system. If a crisp output is not required, then defuzzification is not necessary.

The steps of fuzzy reasoning, i.e., inference operations upon fuzzy IF-THEN rules, performed by FISs are:

1. Compare the input variables with the membership functions on the premise part to obtain the membership values or compatibility measures of each linguistic label. This step is often called fuzzification.
2. Combine (through a specific T-norm operator, usually multiplication or minimum) the membership values of the premise part to obtain the firing strength of each rule.
3. Generate the qualified consequent (either fuzzy or crisp) of each rule depending on the firing strength.
4. Aggregate the qualified consequent to produce a crisp output. This step is called defuzzification.

Several types of fuzzy reasoning have been proposed in the literature [57,59,75,81,93,101,141]. Depending on the type of fuzzy reasoning, most FISs can be classified into three types [46,47], i.e. Tsukarnoto-type FIS, Mamdani-type FIS and TSK-type FIS. Most of the differences among different type FISs come from the specification of the consequent part and thus the defuzzification schemes are also different. In this thesis, we would use the TSK-type FIS described as follows:

TSK-model fuzzy IF-THEN rules, which are described in Section 2.1.2, can be used to implement FISs [129] and shown in Figure 2.3. The output of each rule is a polynomial in the input variables, and the final crisp output, y is the weighted average of each rule's output, $w = g(x_1, \dots, x_n)$, i.e.

$$y = \frac{\sum_{j=1}^l f^j w^j}{\sum_{j=1}^l f^j} \quad (2.7)$$

where the firing strength f^j is calculated by the T-norm operation, e.g.

- Intersection:

$$f^j = \min[\mu_1^j(x_1) \mu_2^j(x_2) \dots \mu_n^j(x_n)] \quad (2.8)$$

- Algebraic Product:

$$f^j = \prod_{i=1}^n \mu_i^j(x_i) \quad (2.9)$$

One-way to view the TSK-model fuzzy system is that it is a nonlinear interpolator between the mappings that are defined by the functions in the consequents of the rules. When g is a constant, we have a zero-order Sugeno fuzzy model. The output of a zero-order Sugeno model is a smooth function of its input variables as long as the neighboring membership functions in the antecedent have enough overlaps. Since each rule has a crisp output, the overall output is obtained via weighted

average, thus avoiding the time-consuming process of defuzzification required in other fuzzy models.

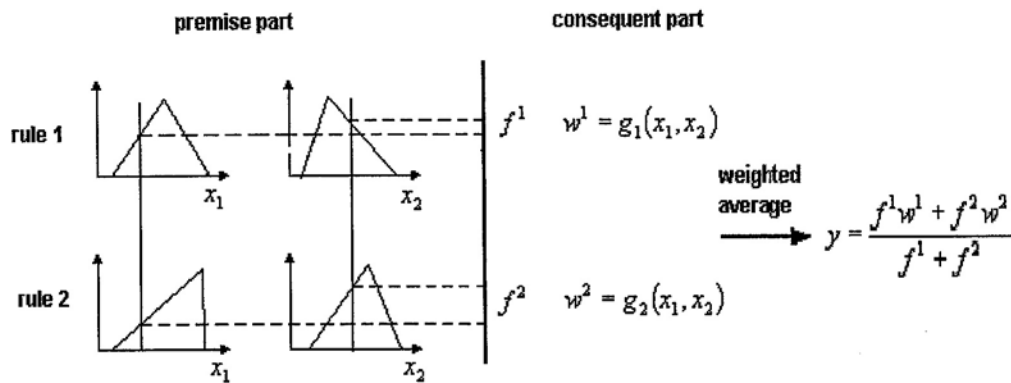


Figure 2.3 TKS-type fuzzy reasoning

2.1.4 Design Problems in FISs

During the last few decades, FISs have emerged as one of the most active and fruitful areas for research in the application of fuzzy set theory. Fuzzy logic has found a variety of applications in various fields ranging from industrial process control to medical diagnosis and robotics [47,100,106,108,114,143,144,150]. Combining multi-valued logic, probability theory and artificial intelligence, FISs are control/decision methodologies that simulate human thinking by incorporating imprecision inherent in all physical systems. From Section 2.1.1, 2.1.2 and 2.1.3, we have a good foundation of how FISs work. The decisions are based on inputs in the form of linguistic variables derived from membership functions. The variables are then matched with the preconditions of linguistic IF-THEN rules, and the response of each rule is obtained through fuzzy implication. To perform a compositional rule of inference, the response of each rule is weighted according to

the confidence or degree of membership of its inputs, and the centroid of responses is calculated to generate an appropriate control signal.

Fuzzy systems that do not require analytical models have demonstrated a number of successful applications. These applications have largely been based on emulating the performance of a skilled human operator in the form of linguistic rules. However, the conventional way of designing a fuzzy system has been a subjective approach. Transferring expert knowledge into a usable knowledge base is time-consuming and nontrivial [59]. Moreover, depending on human introspection and experience may result in some severe problems because, even for human experts, their knowledge is often incomplete and episodic rather than systematic. At present, there is no systematic procedure to determine fuzzy logic rules and membership functions of an FIS. The most straightforward approach is to define membership functions and rules subjectively by studying a human-operated system or an existing controller and then testing the design for a proper output. The membership functions and rules are then adjusted if the design fails the tests. Recently, much research on FISs design has been carried out in [47,100,144]. Unfortunately, the following issues still remain. Hence, bringing learning abilities to FISs may provide a more promising approach.

- Although systematic methods to adjust membership functions and rules are derived in [47,144], structure identification, e.g. determination of input space partition, number of membership functions and number of rules are still difficult to solve.
- Fuzzy systems with high dimensionality often suffer from the problem of curse of dimensionality due to the rapid increase of fuzzy rules [119,120]. Efficient algorithms which relieve this problem and do not increase the complexity of the FISs are highly desired.

- Fuzzy modeling takes advantage that it is constructed based on both linguistic and numerical information [143]. How to utilize different types of numerical information source seems to be the key to construct a compact fuzzy system.

2.2 Learning Paradigms of Fuzzy Systems

In general, subjective approaches to design fuzzy systems are simple and fast, i.e., they involve neither the-consuming iterative procedures nor complicated rule-generation mechanisms. However, problems arise from disagreements among experts, decision rules that are difficult to structure, or a great number of variables necessary to solve the control task. If the fuzzy system somehow possesses learning abilities, an enormous amount of human efforts would be saved from tuning the system. A fuzzy system with learning abilities, i.e. an adaptive FIS which is equipped with a learning algorithm, where the FIS is constructed from a set of fuzzy IF-THEN rules using fuzzy logic principles, and the learning algorithm adjusts the parameters and the structure of the FIS based on numerical information [143].

The current research trend is to design a fuzzy logic system that has the capability of learning itself. It is expected that the controller perform two tasks: 1) It observes the process environment while issuing appropriate control decisions and 2) It uses the previous decision results for further improvement. The main issues associated with learning abilities of FISs are: 1) Parameter estimation, which involves determining the parameters of premises and consequents, and 2) Structure identification, which concerns partitioning the input space and determining the number of fuzzy rules for a specific performance [119]. We discuss the following two families of learning methods characterized by the information source used for

learning and classified with respect to the degree of information from the source [47].

2.2.1 Supervised Learning

In general, supervised learning implies that the information source used for learning is a direct teacher, which provides, at each time step, the correct control action to be applied by the learner. Typically, these learning methods are based on an input-output set of the training data, on which we have to minimize errors between the teacher's actions and the learner's actions.

At present, the partition of input/output space, the choice of membership functions and fuzzy logic rules from numerical training data are still open issues [119,120]. Recently, attentions have been focused on fuzzy neural networks (FNNs) to acquire fuzzy rules based on the learning ability of Artificial Neural Networks (ANNs) [72]. Functionally, an FIS or an ANN can be described as a function approximator, i.e. they aim to approximate a function $f: \mathcal{R}^n \rightarrow \mathcal{R}^m$ from sample patterns drawn from f . It has been shown by Jang and Sun in 1993 [45] that under some minor restrictions, a simplified class of FISs and Radial Basis Function Neural Networks (RBFNNs) are functionally equivalent. It is interesting to note that the learning algorithms and theorems on representational power for one model can be applied to the other, and vice versa.

RBFNNs, as proposed by Moody and Darken [89] in 1989, are often considered to be a type of ANN that employs local receptive fields to perform function mappings. The RBFNNs performs function approximation by superimposing a set of l RBFs as follows:

$$f_j = \exp \left(- \sum_{i=1}^n \left(\frac{x_i - c_{ij}}{\sigma_{ij}} \right)^2 \right) \quad (2.10)$$

$$y = \frac{\sum_{j=1}^l f_j w_j}{\sum_{j=1}^l f_j} \quad (2.11)$$

where f_j , $j=1,\dots,l$ is the firing strength of the j th receptive field or hidden neuron, x_i , $i=1,\dots,n$ are the input variables, y is the output variable, and c_{ij} , $i=1,\dots,n$, $j=1,\dots,l$, σ_{ij} , $i=1,\dots,n$, $j=1,\dots,l$ and w_j , $j=1,\dots,l$ are free parameters which respectively determine the center, width and height of the hump.

From Eqs. (2.7), (2.9) and (2.11), it is obvious that the functional equivalence between an RBFNN and an FIS can be established if the following conditions are true [45]:

- The number of receptive field units, i.e. hidden neurons is equal to the number of fuzzy if-then rules.
- The output of each fuzzy if-then rule is composed of a constant.
- Membership functions within each rule are chosen as a Gaussian function with the same variance.
- The T-norm operator used to compute each rule's firing strength is multiplication.
- Both the RBF networks and the FIS under consideration use the same method, i.e. either weighted average or weighted sum to derive their overall outputs.

As a result, RBFNNs can be viewed as a mechanism for representing rule-based fuzzy knowledge by using its localized network structure, and performing associated fuzzy reasoning using feedforward computational algorithms.

Integrating the learning abilities of ANNs into FISs is a promising approach because the connectionist structure of ANNs provides powerful learning abilities to FISs. The typical paradigm of FNNs is to build standard neural networks, which are designed to approximate a fuzzy algorithm or a process of fuzzy inference through the structure of neural networks [46,67,72,142]. The main idea is the following: Assuming that some specific membership functions have been defined, we begin with a fixed number of rules by resorting to either the trial-and error method [24,46,113,130] or expert knowledge [63,69]. Next, the parameters are modified by the Back Propagation (BP) learning algorithm [24,63,69,113,130] or hybrid learning algorithm [46]. These methods can readily solve two problems of conventional fuzzy reasoning: 1) Lack of systematic design for membership functions and 2) Lack of adaptability for possible changes in the reasoning environment. These two problems are intrinsically concerned with parameter estimation. Nevertheless, structure identification, such as partitioning the input and output space and determination of number of fuzzy rules, is still time-consuming. The reason is that, as shown in [149], the problem of determining the number of hidden nodes in NNs can be viewed as a choice of the number of fuzzy rules. Different from the aforementioned FNNs, several adaptive paradigms have been presented whereby not only the connection weights can be adjusted but also the structure can be self-adaptive during learning [17,24,27,35,49,113,149]. In [24,113], the FNNS are constructed largely to contain all possible fuzzy rules. After training, a pruning process [113] or fuzzy similarity measure [24] is performed to delete redundant rules for obtaining an optimal fuzzy rule base. In [17], a parsimonious construction algorithm employing linear parameter ANNs was proposed to overcome the curse of dimensionality associated with FNN structure learning. In [27], a hierarchically self-organizing approach, whereby the structure is identified by input-output pairs, is developed. An on-line self-constructing paradigm is proposed in [49]. The premise structure in [49] is

determined by clustering the input via an on-line self-organizing learning approach. A hierarchical on-line self-organizing learning algorithm for dynamic fuzzy neural networks based on RBF neural networks, which are functionally equivalent to TSK fuzzy systems, has been developed in [35,149]. The system starts with no rules. Then, rules can be recruited or deleted dynamically according to their significance to system performance so that not only the parameters can be adjusted but also the structure can be self-adaptive. Above all, all of these learning algorithms belong to the class of supervised learning where the teacher associates the learning system with desired outputs for each given input. Learning involves memorizing the desired outputs by minimizing discrepancies between actual outputs of the system and the desired output.

2.2.2 Unsupervised Learning

Unsupervised learning does not rely on a direct teacher that guides the learning process. It has been shown that if the supervised learning can be used, e.g., when the input/output training data sets are available, it is more efficient than unsupervised learning [7]. However, unsupervised learning systems can be used to provide unknown desired outputs with a suitable evaluation of system performances. In this section, we introduce two design methods that employ evolutionary algorithms and reinforcement learning techniques respectively.

First, we introduce evolutionary learning of fuzzy systems. The information source used for learning is a performance measure, which indicates the quality of a learner on a set of states. This kind of learning method is generally associated with evolutionary algorithms, e.g., genetic algorithms [54,71], evolutionary strategies [29], and Learning Classifier Systems [99]. We shall further narrow our scope by discussing Genetic Algorithms (GA) for fuzzy systems only, although other

approaches can be used similarly. A GA is a parallel global-search technique that emulates the processes of natural evolution including crossover, mutation, and survival of the fittest. GA can, in effect, often seek many local minima and increase the likelihood of finding global minima representing the problem goals [84].

When designing a fuzzy system using GAS, the first important consideration is the representation strategy, that is, how to encode the fuzzy system into chromosomes [115]. Thrift [134] and Hwang and Thompson [42] encode all the rules into a chromosome while fixing the membership functions. Using several critical points to represent each membership function while using all the possible rules, Karr [55] and Karr and Gentry [56] use GAs to evolve these critical points; that is, to adjust the membership functions. Since the membership functions and rule set in a **fuzzy** system are codependent, they should be designed or evolved at the same time. Lee and Takagi [62] encode membership functions and all the rules into a chromosome. They restrict adjacent membership functions to fully overlap and also constrain one membership function to have its center resting at the lower boundaries of the input range. The above-mentioned methods encode all possible rules into a chromosome. There are some drawbacks by doing so [30]: first, the computational efficiency associated with fuzzy logic is lost using a high number of rules and second, robustness diminishes with increasing number of rules. This is especially true when the dimension of the inputs and the number of fuzzy sets for each input variable become great since the number of possible rules increases exponentially with these numbers. In most applications, not all possible rules need to be used; only a portion of the rules are needed. So, only this portion of rules should be encoded into the chromosome and evolved. By doing so, the length of the chromosome will be reduced greatly and, therefore, will be suitable for bigger problems. It is better to encode the number of rules to be included in the rule set

together with rules and/or membership functions into the chromosome to be evolved. There are several ways to do this. Lee and Takagi [61] proposed encoding membership functions and fitness functions in chromosomes. Shimojima *et al.* [116] and Inoue *et al.* [43] defined membership functions for each rule and encoded effectiveness information for each rule and membership functions. Shimojima *et al.* used fitness functions that minimize the number of rules which Inoue *et al.* used a method called "forgetting".

When using GA optimization methods, we can employ a complex fitness function. The genotype representation encodes the problem into a string while the fitness function measures the performance of the system. This means that we can incorporate structure-level information into the objective function [61] and let the GA optimization methods do the entire job: finding the correct number of rules, as well as proper parameters of membership functions in fuzzy systems. This seems too good to be true. However, we should bear in mind that evolutionary algorithms are slow and they could take a tremendous amount of time to obtain a less-than-optimal solution.

Input space partitioning determines the premise part of a fuzzy rule set. For a problem, some parts of pattern space might require fine partition, while other parts require only coarse partition. Therefore, the choice of an appropriate fuzzy partition is important and difficult. One of the flexible input space partitioning methods is based on the GA [121]. The major disadvantage of these methods is that it is very consuming; the computation cost to evaluate a partition result encoded in each individual is very high and many generations are needed to find the final partition. Hence, this scheme is obviously not suitable for online operation. Moreover, the GA-based partitioning methods might not find

meaningful fuzzy terms for each input variable. There could be difficulty accommodating a *priori* knowledge about the target system.

Next, we introduce another kind of learning methods using the information source of critic, which gives rewards and punishments with respect to the state reached by the learner, but does not provide correct actions. These methods, called reinforcement learning methods, consist of active exploration of the state and action spaces to find what action to apply in each state [60].

Reinforcement learning (RL) plays an important role in adaptive control. It is particularly helpful when no explicit teacher signal is available in the environment where an interacting agent must learn to perform an optimal control action. The world informs the agent of a reinforcement signal associated with the control action and the resulting new state. The signal is evaluative rather than instructive. Furthermore, the signal is often delivered infrequently and delayed i.e. it is not available at each time instant. When it is available at a certain moment, it represents the results of a series of control actions probably performed over a lengthy period of time. There are two prevalent approaches to reinforcement learning, namely actor-critic learning [6] and Q-learning [145]. The actor-critic model typically includes two principal components: the critic (evaluation/prediction) module and the action (control) module. The critic generates an estimate of the value function from state vectors and external reinforcement generated by the environment. That is, the critic plays an important role in predicting the evaluation function. The action module attempts to learn optimal control or decision-making skills. Q-learning is a simple way of dealing with incomplete-information Markovian-action problems based on the action-value function Q that maps state-action pairs to expected returns. The learner tries an action at a particular state and evaluates its consequence in terms of the immediate

reward or penalty it receives and its estimate of the state value resulting from the taken action. Actor-critic learning architecture requires two fundamental memory buffers: one for the evaluation function and one for the policy. On the other hand, Q-learning maintains only one: a pair of state and action. Instead, Q-learning requires additional complexity in determining the policy from Q-learning.

The basic idea behind fuzzy RL is to apply fuzzy partitioning to the continuous state-space and to introduce linguistic interpretation. Such averaging over neighboring partitioned subspaces can create generalization abilities [47]. Most of learning methods are based on the idea of actor-critic model. Berenji and Khedkar propose the Generalized Approximate Reasoning for Intelligent Control (GARIC) model [10], which has three components: the action selection network, the action evaluation network and the stochastic action modifier. The action selection network is expressed in a neuro-fuzzy framework. Lin and Lee's Reinforcement Neural-Network-based Fuzzy Logic Control System (RNN-FLCS) [68] consists of a fuzzy controller and a fuzzy predictor. The whole RNN-FLCS is expressed in a neuro-fuzzy framework; both critic and action module share the antecedent parts of the fuzzy rules. In addition to parameter learning, it can perform the structure learning and find the proper fuzzy rules. Lin *et al.* [70] and Chiang *et al.* [26] propose genetic RL algorithms for designing fuzzy systems by exploiting the global optimization capability of GAS in order to overcome the local minima problem in network learning due to the use of the gradient descent learning method. Bruske *et al.* [23] and Rak *et al.* [107] employ actor-critic model to learn fuzzy controllers for autonomous robots. Jouffe's Fuzzy Actor-Critic Learning [48] deals with the conclusion part of fuzzy rules. Kandadai and Tien propose a fuzzy-neural architecture that is capable of automatically generating a fuzzy system for use in hierarchical knowledge-based controllers [53].

On the other hand, the other approaches allow efficient learning of fuzzy systems based on Q-learning since Q-learning is the most popular reinforcement learning method that directly calculates the optimal action policy without an intermediate cost evaluation step and without the use of a model. FISs have strong generalization abilities to deal with continuous inputs and outputs. Moreover, the fuzzy Q-learning method is considered to be a more compact version of the fuzzy actor-critic method. Glorennec and Jouffe consider a collection of fuzzy rules as an agent that produces continuous-valued actions in [37,48]. This approach termed Fuzzy Q-Learning (FQL) produces an action by some rules triggering on the same state-space and cooperating collectively. Similar rule structure and adaptive rewards are used in the simulation of object chase agents [78]. Horiuchi et al. consider a similar algorithm, termed Fuzzy Interpolation-Based Q-Learning and further propose an extended roulette selection method so that continuous-valued actions can be selected stochastically based on the distribution of Q-values [41]. Berenji [11] proposes another version of Q-learning dealing with fuzzy constraints. In this case, we do not have fuzzy rules, but "fuzzy constraints" among the actions that can be done in a given state. These works, however, only adjust the parameters of fuzzy systems online. Structure identification, such as partitioning the input and output space and determination of the number of fuzzy rules are still carried out offline and it is time consuming.

In this thesis, one of the main objectives is to design a novel learning method that is capable of learning the structure and parameters of fuzzy systems automatically and simultaneously using Q-Learning. The following chapters will further investigate this problem and present a thorough discussion on fuzzy system learning by reinforcement methods.

Chapter 3

Reinforcement Learning

The goal of building systems that can adapt to their environments and learn from their experience has attracted researchers from many fields, including computer science, engineering, mathematics, physics, neuroscience, and cognitive science. Reinforcement learning (RL) is a powerful method to solve the problem faced by an agent that must learn through trial-and-error interactions with a dynamic environment [51,109,128]. In this chapter, we begin by presenting the basic framework of RL and then discuss the problem of generalization in large continuous spaces. The last section of this chapter discusses some issues in applying RL to robotics.

3.1 Basic Framework

Basically, RL is concerned with learning through direct experimentation. It does not assume the existence of a teacher that provides training examples on which learning of a task takes place. Instead, experience is the only teacher in RL. The learner acts autonomously on the process and receives reinforcements from its actions. With historical roots on the study of biological conditioned reflexes, RL

attracts the interest of engineers because of its theoretical relevance and potential applications in fields as diverse as operational research and intelligent robotics.

3.1.1 Reinforcement Learning Model

RL is concerned with solving a problem faced by an agent that must learn through trial-and-error interactions with a dynamic environment. We are particularly interested in a learning system which is composed of two subjects, namely the learning agent (or simply the learner) and a dynamic environment. In the standard RL model, a learner is connected to its environment via perceptions and actions. On each step of the interaction, the learner receives as its input, x which shows some indication of the current state, s , of the environment. The learner then selects an action, a , to generate an appropriate output. The action changes the state of the environment, and the value of this state transition is communicated to the learner through a scalar reinforcement signal, r . Those reinforcement signals encode information about how well the learner is performing the required task, and are usually associated with a dramatic condition-such as the accomplishment of a subtask (reward) or complete failure (punishment). The ultimate goal of the learner is to optimize its behavior based on some performance measure (usually maximization of some long-run measure of reinforcement). In order to do that, the learner must learn a policy π , which describes the associations between observed states and chosen actions that lead to rewards or punishments. In other words, it must learn how to assign credit to past actions and states by correctly estimating costs associated with these events.

Referring to Figure 3.1, the accumulation of experience that guides the behavior (action policy) is represented by a cost estimator whose parameters are learned as new experiences are carried out by the learner. The learner is also equipped with

sensors that define how observations about the external environment are made. These observations can be combined with past observations or input to a state estimator which defines an internal state that represents the agent's belief about the real state of the process. The cost estimator then maps these internal states and presented reinforcements to associated costs, which are basically expectations about how good or bad these states are, given the experience obtained so far. Finally, these costs guide the action policy. A prior built-in knowledge may affect the behavior of the learner either directly, altering the action policy or indirectly, influencing the cost estimator or sensors.

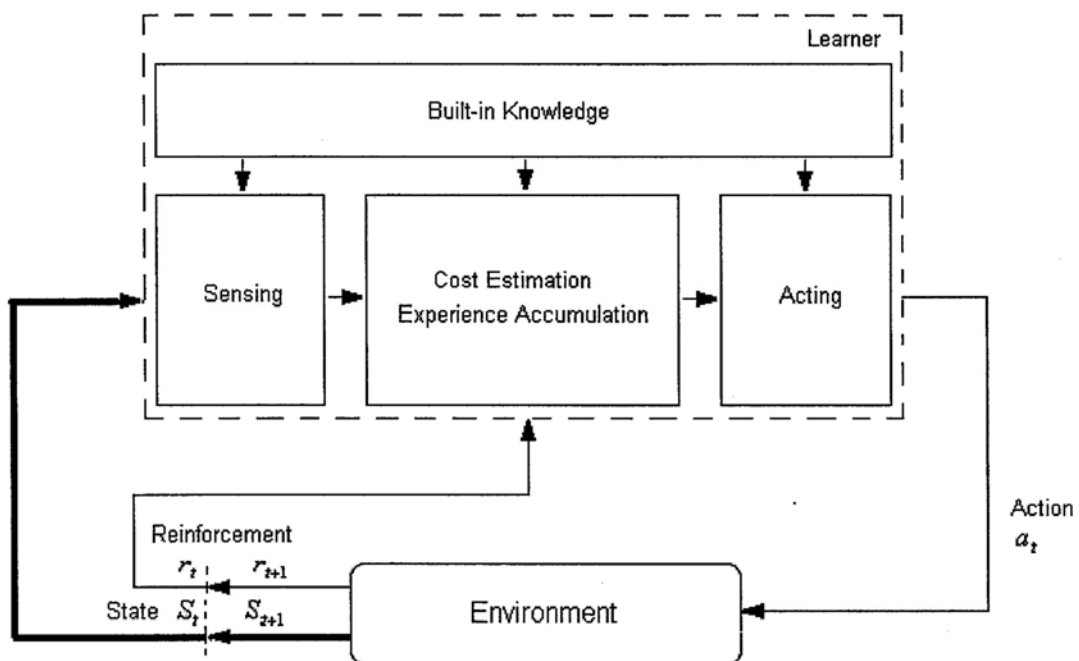


Figure 3.1 A general model for the reinforcement learning agent

The experience accumulation and action taking process is represented by the following sequence:

1. The learner makes an observation and perceives any reinforcement signal provided by the environment.

2. The learner takes an action based on the former experience associated with the current observation and reinforcement.
3. The learner makes a new observation and updates its cumulated experience.

In order to control policies, we must be able to evaluate them with respect to each other. In RL, the cost estimator is some function of the rewards received by the learner. There are three models that have been the subject of the majority of work in this area. The finite-horizon model is the easiest to understand. The idea is the following: At a given moment in time, the learner should optimize its expected reward for the next k steps, which is given by

$$V = E\left(\sum_{t=0}^k r_t\right) \quad (3.1)$$

It needs not worry about what will happen after that. In this and subsequent expressions, r represents the scalar reward received t steps into the future. The finite-horizon model is not always appropriate since in many cases, we may not know the precise length of the agent's life in advance. In the average-reward model, the learner is supposed to take actions that optimize its long-run average reward, which is given by

$$V = \lim_{k \rightarrow \infty} E\left(\frac{1}{k} \sum_{t=0}^k r_t\right) \quad (3.2)$$

The infinite-horizon discounted model takes the long-run reward of the agent into account, but rewards that are received in the future are geometrically discounted according to the discount factor, $0 \leq \gamma \leq 1$ is follows:

$$V = E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \quad (3.3)$$

If we set the discount factor to be zero, when we obtain the one-step greedy policy, i.e. the best action is the one that gives the greatest immediate reward. Values greater than zero reflect how much we are concerned with actions that happen

further in the future. The average-reward model can be seen as the limiting case of the infinite-horizon discounted model as the discount factor approaches to 1. The infinite-horizon discounted model has received wide attention mostly because the theoretical aspects are better understood.

RL differs from the more widely studied problem of supervised learning in several ways. The most important difference is that there is no presentation of input/output pairs. Instead, after choosing an action, the learner is told the immediate reward and the subsequent state, but is not told which action would have been in its best long-term interest. It is necessary for the learner to gather useful experience about the possible system states, actions, transitions and rewards actively to act optimally. Another difference from supervised learning is that on-line performance is important; the evaluation of the system is often concurrent with learning.

3.1.2 Markov Decision Processes

RL problems are typically cast as Markov Decision Processes (MDPs), which are widely used to model controlled dynamical systems in control theory, operations research and artificial intelligence. An MDP consists of

- A set of states S ,
- A set of actions A ,
- A reward function $R : S \times A \rightarrow \mathbb{R}$, and
- A state transition function $F : S \times A \rightarrow \Pi(S)$, where a member of $\Pi(S)$ is a probability distribution over the set S . We write $P(s, a, s')$ for the probability of making a transition from state s to state s' using action a .

The state transition function probabilistically specifies the next state of the environment as a function of its current state and the learner's action. The reward

function specifies expected instantaneous reward as a function of the current state and action. The model is Markov if the state transitions are independent of any previous environment states or agent actions [9,12,105].

Given a perfect model of the environment as an MDP, we can use Dynamic Programming (DP) techniques to determine the optimal policy. Classical DP algorithms are of limited utility in RL both because of their assumption of a perfect model and the great computational expense, but they still serve as the foundation and inspiration for the learning algorithms to follow. We restrict our attention mainly to finding optimal policies for the infinite-horizon discounted model, but most of these algorithms have analogs for the finite-horizon and average-case models as well. We rely on the result that for the infinite-horizon discounted model, there exists an optimal deterministic stationary policy [9].

The optimal value of a state is the expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy. Using π as a complete decision policy, it is written as

$$V^*(s) = \max_{\pi} E \left(\sum_{t=0}^{\infty} \gamma^t r_t \right) \quad (3.4)$$

This optimal value function is unique and can be defined as the solution to the simultaneous equations

$$V^*(s) = \max_a \left(R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') V^*(s') \right), \quad \forall s \in S \quad (3.5)$$

which states that the value of a state s is the expected instantaneous reward plus the expected discounted value of the next state, using the best available action. Given the optimal value function, we can specify the optimal policy as

$$\pi^*(s) = \arg \max_a \left(R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') V^*(s') \right) \quad (3.6)$$

There are two main classes of well-established methods for finding out optimal policies in MDPs: Value Iteration and Policy Iteration [9,12]. The value iteration method is determined by a simple iterative algorithm that can be shown to converge to the correct V^* value. The gist of the method is as follows:

```

initialize  $V(s)$  arbitrarily
loop until policy good enough
  loop for  $s \in S$ 
    loop for  $a \in A$ 
       $Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') V(s')$ 
     $V(s) = \max_a Q(s, a)$ 
  end loop
end loop

```

Value iteration is very flexible. The assignments to V need not be done in strict order as shown above, but instead can occur asynchronously in parallel provided that the value of every state gets updated infinitely often on an infinite run. The computational complexity of the value-iteration algorithm with full backups, per iteration, is quadratic in the number of states and linear in the number of actions.

The policy iteration algorithm manipulates the policy directly rather than finding it indirectly via the optimal value function. It operates as follows:

```

choose an arbitrary policy  $\pi'$ 
loop
   $\pi = \pi'$ 
  compute the value function of policy  $\pi$ :
    solve the linear equations
       $V_\pi = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s, \pi(s), s') V_\pi(s')$ 
  improve the policy at each state:
     $\pi'(s) = \arg \max_a (R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') V_\pi(s'))$ 
until  $\pi = \pi'$ 

```

The value function of a policy is simply the expected infinite discounted reward that will be gained, at each state, by executing that policy. It can be determined by solving a set of linear equations. Once we know the value of each state under the current policy, we consider whether the value could be improved by changing the first action taken. If it can, we change the policy to take the new action whenever it is in that situation. This step is guaranteed to strictly improve the performance of the policy. When no improvements are possible, then the policy is guaranteed to be optimal.

3.1.3 Learning an Optimal Policy

We use DP techniques for obtaining an optimal policy for an MDP assuming that we already have a model. The model consists of knowledge of the state transition probability function $P(s,a,s')$ and the reinforcement function $R(s,a)$. RL is primarily concerned with how to obtain the optimal policy when such a model is not known in advance. The agent must interact with its environment directly to obtain information which, by means of an appropriate algorithm, can be processed to produce an optimal policy. Here we examine some online, model-free algorithms that attempt to obtain the optimal policy. For more details of other methods computing optimal policies by learning models, see [51,1281].

The biggest problem facing an RL agent is temporal credit assignment. We use insights from value iteration to adjust the estimated value of a state based on the immediate reward and the estimated value of the next state. This class of algorithms is known as Temporal Difference (TD) learning methods [123]. We will consider two different TD learning strategies for the discounted infinite-horizon model.

- Adaptive Heuristic Critic

The Adaptive Heuristic Critic (AHC) algorithm of [6] is an adaptive version of policy iteration in which the value-function computation is no longer implemented by solving a set of linear equations, but instead computed by an algorithm called $TD(0)$. It has a separate memory structure to explicitly represent the policy independent of the value function. The policy structure is known as the actor because it is used to select actions, and the estimated value function is known as the critic because it criticizes the actions made by the actor. The critic must learn about and criticize whatever policy is currently being followed by the actor. We can see the analogy with modified policy iteration if we imagine these components working in alternation. The policy implemented by actor is fixed and the critic learns the value function V_π for that policy. Now, we fix the critic and let the actor learn a new policy π' that maximizes the new value function and so on. In most implementations, however, both components operate simultaneously.

We define $\langle s, a, r, s' \rangle$ to be an experience 4-tuple summarizing a single transition in the environment. Here, s is the agent's state before the transition, a is its choice of action, r is the instantaneous reward it receives, and s' is its resulting state. The value of a policy is learned using Sutton's $TD(0)$ algorithm [123] which uses the following update rule

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s)) \quad (3.7)$$

Whenever a state s is visited, its estimated value is updated to a value closer to $r + \gamma V(s')$, since r is the instantaneous reward received and $V(s')$ is the estimated value of the next actual state. This is analogous to the sample-backup rule from value iteration; the only difference is that the

sample is drawn from the real world rather than by simulating a known model. The key idea is that $r + \gamma V(s')$ is a sample value of $V(s)$ and it is more likely to be correct because it incorporates the real r .

The TD(0) rule as presented above is really an instance of a more general class of algorithms called $TD(\lambda)$, with $\lambda = 0$. TD(0) looks only one step ahead when adjusting value estimates. Although it will eventually arrive at the correct answer, it can take quite a while to do so. The general $TD(\lambda)$ rule is similar to the TD(0) rule given above in that we have

$$V(x) = V(x) + \alpha (r + \gamma V(s') - V(s)) e(x) \quad (3.8)$$

but it is applied to every state according to its eligibility $e(x)$, rather than just to the immediately previous state, s . The eligibility of a state s is the degree to which it has been visited in the recent past. When a reinforcement signal is received, it is used to update all the states that have been recently visited, according to their eligibility. We can update the eligibility online as follows:

$$e(s) = \begin{cases} \gamma \lambda e(s) + 1 & \text{if } s = \text{current state} \\ \gamma \lambda e(s) & \text{otherwise} \end{cases} \quad (3.9)$$

It is computationally more expensive to execute the general $TD(\lambda)$, though it often converges considerably faster for large λ .

- Q-learning

The work of the two components of AHC can be accomplished in a unified manner by Watkins' Q-learning algorithm [145,146]. Q-learning is typically easier to implement. In order to understand Q-learning, we have to develop some additional notations. Let $Q^*(s,a)$ be the expected discounted reinforcement of taking action a in state s . Continuing by

choosing actions optimally and noting that $V^*(s)$ is the value of s assuming the best action is taken initially, the term $V^*(s) = \max_a Q^*(s, a)$. $Q^*(s, a)$ can be written recursively as follows:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s' | s, a) \max_{a'} Q^*(s', a') \quad (3.10)$$

Note also that since $V^*(s) = \max_a Q^*(s, a)$, we have $\pi^*(s) = \arg \max_a Q^*(s, a)$ as an optimal policy. Because the Q function makes the action explicit, we can estimate the Q values online using a method essentially the same as $TD(0)$. We can also use them to define the policy because an action can be chosen by simply taking the one with the maximum Q value for the current state.

The Q -learning rule is

$$Q(s, a) = Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \quad (3.11)$$

where $\langle s, a, r, s' \rangle$ is an experience 4-tuple as described earlier. If each action is executed in each state an infinite number of times on an infinite run and the well-known assumption in stochastic approximation theory given below is valid:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=1}^{\infty} \alpha_t^2 < \infty \quad (3.12)$$

the Q values will converge with the probability of 1 to Q^* [32,145,146]. The conditions guarantee that the learning rate α is large enough and diminishes to zero at a suitable rate. Although learning rates that meet these conditions are often used in theoretical work, they are seldom used in applications and empirical research because sequences of learning rates that meet these conditions often converge very slowly and are not suitable for non-stationary scenarios.

Q-learning can also be extended to update states that occur more than one step previously, as in $TD(\lambda)$, which is discussed in Chapter 6.

An interesting variation for Q-learning is the SARSA algorithm [110,127], which is similar to Q-learning in that it attempts to learn the state-action value function, $Q^*(s, a)$. The main difference between SARSA and Q-learning, however, is in the incremental update function. SARSA takes a 5-tuple, (s, a, r, s', a') , of experience, rather than the 4-tuple that Q-learning uses. The additional element, a' , is the action taken from the resulting state, s' , according to the current control policy. This removes the maximization from the update rule, which becomes

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)) \quad (3.13)$$

Moreover, it allows us to consider a $SARSA(\lambda)$ algorithm, very similar to causal $TD(\lambda)$.

AHC architectures seem to be more difficult to work with than Q-learning in practice. In addition, Q-learning is exploration insensitive; this feature is discussed in Chapter 6. Furthermore, it is the extension of autonomous learning concept to optimal control, in the sense that it is the simplest technique that directly calculates the optimal action policy without an intermediate cost evaluation step and without the use of a model. For these reasons, Q-learning is the most popular and seems to be the most effective model-free algorithm for RL learning. It does not, however, address any of the issues involved in generalization over large state and/or action spaces. In addition, it may converge quite slowly to a good policy.

3.1.4 Exploration/Exploitation Tradeoff

One of the necessary conditions under which RL algorithms can find an optimal action policy is the complete exploration of the state space, normally infeasible in practical situations. When control and learning are both at stake, the learning agent must try to find a balance between the exploration of alternatives to a given policy and the exploitation of this policy as a mechanism for assessing its associated costs. In other words, it must realize that trying unknown alternatives can be risky, but keeping the same policy infinitely will never lead to improvement. Thrun [135] has surveyed a variety of these techniques, which, in addition to ensuring sufficient exploratory behavior, exploit accumulated knowledge.

The strategy that always selects the action with the highest estimated payoff corresponds to pure exploitation. Unfortunately, pure exploitation applied from the beginning of learning will not work in general. Typical suggestions to overcome these difficulties include choosing random actions occasionally and exploiting actions at other times, or selecting actions that minimize some kind of artificially biased values, where the bias is such that bias values of less often visited state-action pairs become smaller. The most popular of these which is called the ε -greedy strategy is to take the action with the best estimated expected reward by default, but with a probability of ε and select an action at random. Some versions of this strategy start with a large value of ε to encourage initial exploration, which is slowly decreased. An objection to the simple strategy is that when it experiments with a non-greedy action, it is no more likely to try a promising alternative than a clearly hopeless alternative. A slightly more sophisticated strategy is Boltzmann exploration, where the probability of choosing action a in state s is given by

$$\Pr(a | s) = \frac{\exp(Q(s, a)/T)}{\sum_{a' \in A} \exp(Q(s, a')/T)} \quad (3.14)$$

The temperature parameter T can be reduced over time to reduce exploration.

3.2 Generalization

All the previous discussions have tacitly assumed that it is possible to enumerate the state and action spaces and store tables of values over them. Except in very small environments, this means impractical memory requirements. It also makes inefficient use of experience. In a large smooth state space, we generally expect similar states to have similar values and similar optimal actions. Surely, therefore, there should be some more compact representation than a table. The problem of learning in large spaces is addressed through generalization techniques, which allow compact storage of learned information and transfer of knowledge between similar states and actions.

3.2.1 Generalization in States

Mahadevan and Connell propose a generalization solution on RL applied to real robotic navigation [80], which is based on the fact that similar sensed states must have similar values. They define a weighted Hamming distance for the calculation of this similarity based on a previously assessed relative importance of sensors. However, in this case, states are still represented in a lookup table. In the case of a large continuous state space, this representation is intractable. This problem is known as the curse of dimensionality. Generally speaking, it is necessary to use function approximation schemes of [18] due to an extremely large number of states that makes implementation of the state space by a lookup table is impossible. The generalization issue in RL can be seen as a structural credit assignment problem, which decides how the different aspects of an input affect the value of the output.

Most of the methods that allow RL techniques to be applied in large state spaces are modeled on value iteration and Q-learning. Typically, a function approximator is used to represent the value function by mapping a state description to a value. Various approaches using neural networks techniques have been reported to work on various problem domains [5,148]. Lin [65] use back propagation networks for Q-learning. Tesauro [131] use back propagation for learning the value function in backgammon. Boyan and Moore [18] use local memory-based methods in conjunction with value iteration. Mitchell and Thrun [88] use explanation-based neural networks for robot control. Touzet describes a Q-learning system based on Kohonen's self-organizing map [138]. Actions are taken by choosing the node which most closely matches the state and the maximum possible value. Unfortunately the actions are always piecewise constant. The Cerebellar Model Articulation Controller (CMAC) [112,117,127,133,145] is another algorithm that has been proven to be popular for value-function approximation work. The CMAC is a function approximation system which features spatial locality. It is a compromise between a lookup table and a weight-based approximator. It can generalize between similar states, but it involves discretization, making it impossible that actions vary smoothly in response to smooth changes in a state. The other algorithms based on local averaging have been suggested in the context of RL [19,98]. Locally weighted regression [4] can be used as the basis of value-function approximation scheme. There has been some work on using variable-resolution discretization methods. Variable Resolution Dynamic Programming [90] begins with a coarse discretization of the state space, on which standard dynamic programming techniques can be applied. This discretization is refined in parts of the state space that are deemed to be "important". Moore's Parti-Game algorithm [92] also starts with a coarse discretization of the state space. The algorithm assumes the availability of a local controller for the system that can be used to make transitions from one discretized cell to another. Another related

approach is proposed by Munos [94,95] that again begins with a coarse discretization of the state space and selectively refines this discretization. A cell is split (or not), depending on its influence on other cells, as determined by the dynamics of the system and the discount factor. However, these approaches assume that local controllers or the model of the system dynamics are known.

Though a large number of successful applications of RL based on function approximation on various problem domains have been reported in [5,28,31,65,88,111,112,118,131-133,137,138,148,151], there is always some kind of ad hoc adaptation that includes the use of nonlinear architecture or auxiliary mechanisms for value estimation, with which theoretical proofs of convergence are not concerned. These algorithms with nonlinear architecture lead to improved performance. Unfortunately, it is very hard to quantify or analyze the performance of these techniques. Boyan and Moore [18] give some examples of value functions errors growing arbitrarily large when generalization is used with value iteration. Sutton [127] shows how modified versions of Boyan and Moore's examples can converge successfully. Tsitsiklis and Roy [140] provide a methodological foundation of a few different ways that compact representations can be combined to form the basis of a rational approach to difficult control problems. Bertsekas and Tsitsiklis [13] provide an excellent survey of the state-of-art in the area of value function approximation. However, whether general principles, ideally supported by theory, can help us understand when value function approximation will succeed is still an open question. More careful research is needed in the future.

3.2.2 Generalization in States and Actions

Most approaches use function approximators to generalize the value function across situations. These works, however, still assume discrete actions and cannot

handle continuous-valued actions. In continuous action spaces, the need for generalization over actions is important. It should be possible that actions vary smoothly in response to smooth changes in a state. Santamaria *et al.* [112] go further and consider continuous action spaces, but their approach cannot actually generate continuous actions except when exploring randomly a small fraction of the time. In the other words, this approach does not yield truly continuous-action policies. The continuous-action Q-learning approach, which is the only approach restricted to the generation of continuous actions by means of Q-learning, is proposed in [87]. On the other hand, fuzzy logic can be used to facilitate generalization in the state space and to generate continuous actions in RL [48]. The FIS learner has a continuous perception of the state space, and based on a strategy for policies, it can trigger continuous actions. This proposed strategy consists of inferring the global policy (relative to states) from local policies associated with each rule of the learner. In this thesis, we discuss a novel Q-learning method that can handle continuous states and continuous actions based on the contribution of fuzzy logic.

3.3 Applications in Robotics

The study of RL agents used to have a strong biological motivation [125], but in the last few years the enthusiasm switched towards engineering applications. One of the most impressive applications of RL to date is that by Gerry Tesauro to the game of backgammon [131,132]. Crites and Barto [31] study the application of RL to the elevator dispatching. Zhang and Dietterich [151] use back propagation and TD to learn good strategies for job-shop scheduling. Singh and Bertsekas [118] apply RL to the dynamic channel allocation problem in the operation of cellular telephone systems. Tong and Brown [137] solve the call admission control and routing problem in multimedia networks via RL. Next, we discuss a variety of

robotics applications, which is one of the major successful application areas in RL [28,111].

3.3.1 Robot Learning

Many intelligent methodologies have often been used for robotic systems in areas that are critical and dangerous for human beings [2]. To accomplish a given task, a robot collects or receives sensory information concerning its external environment and takes actions within the dynamically changing environment. Furthermore, the intelligent robot should automatically generate its motion for performing the task. Brooks [20] proposes a sub-sumption architecture, and later behavior-based artificial intelligence for robotics [21]. This kind of behavior-based artificial intelligence stresses the importance of direct interactions between a robot and the environment. RL is employed in situations where a representative training set is not available and the agent must itself acquire this knowledge through interactions with its environment. Therefore, autonomous learning in robotics is a natural application area for RL. Barto [8] distinguishes firstly, non-associative RL tasks, where the learning system receives only evaluative input; secondly, associative RL tasks, where a controller aims to maximize the immediate reward at each step; and thirdly, adaptive sequential decision tasks where the maximization of long term performance may entail foregoing immediate favorable rewards. Since the first type has been studied based on genetic algorithms, we regard it as outside the scope of this thesis. Some interesting examples can be found in [34,97]. Most of the works discussed here are of the second or third type of RL. We prefer RL to genetic algorithms for the purpose of the evaluating the system online concurrent with learning.

Maes and Brooks [79] describe a six-legged robot that learns to sequence its gait by associating immediate positive and negative rewards with action preconditions. Mahadevan and Connell [80] develop RL strategies to train a real robot performing a box-pushing task based on decompositions of the task. In contrast, Kalmar *et al.* [52] use *RL* in an adaptive switching strategy for subtasks. Lin [65,66] uses RL with a neural network to learn navigation tasks. Asada *et al.* [3] uses discretization of the state space, based on domain knowledge, to learn offensive strategies for robot soccer. Meeden *et al.* apply complementary reinforcement back propagation to the temporally extended problems of obstacle avoidance. Thrun [136] describes a hybrid approach towards enabling a mobile robot to exploit previously learned knowledge by applying it to multiple tasks. Gullapalli *et al.* [38] develop the skills of the peg-in-hole insertion task and the ball-balancing task via RL. Millan [85,86] reports an approach towards navigation in an unknown indoor environment based on a mobile robot, which is able to optimize the total reinforcement it receives as it progresses towards the goal. Hailu [40] gives similar results in a similar task. Mataric [82] describes a robotics experiment with an unthinkable high dimensional state space, containing many degrees of freedom. Bonarini [15] presents some approaches based on evolutionary RL algorithms which are capable of evolving real-time fuzzy models that control behaviors and proposes an approach towards designing of reinforcement functions [16]. Boada *et al.* [14] present an RL algorithm which allows a robot to learn simple skills and obtain the complex skill approach which combines the previously learned ones. Gaussier *et al.* [36] conclude some limitations of reinforcement approaches and suggest how to bypass them.

3.3.2 Problems

RL is currently perhaps the most popular methodology for various types of learning. However, there are still some difficulties that must be overcome in order to implement a useful learning system on a real robot [22].

- Lack of initial knowledge

Many learning systems attempt to learn by starting with no initial knowledge. Although this is appealing, it introduces special problems when working with real robots. Initially, if the learning system knows nothing about the environment, it is forced to act more or less arbitrarily. RL systems attempt to learn the policy by attempting all the actions in all the available states in order to rank them in the order of appropriateness. In order to learn a new policy, large numbers of time-consuming learning trials are required. On the other hand, critical behavior must be learned with a minimal number of trials, since the robot cannot afford to fail too frequently. When controlling a real robot, a bad choice can result in the damage of the environment or the robot itself, possibly causing it to stop functioning. In order for the learning system to be effective, we need to provide some sort of bias, to give it some ideas of how to act initially and how to begin to make a progress towards the goal. Systems should have the ability of using previously learned knowledge to speed up the learning of a new policy.

- Continuous states and actions

In many real-world scenarios, sensory and action spaces are continuous. These values can be discretized into finite sets if the discretization follows the natural resolution of the devices. However, many quantities are

inherently continuous with a fine resolution that leads to many discrete states. Even if they can be discretized meaningfully, it might not be readily apparent how best to do it for a given task. Incorrect discretizations can limit the final form of the learned control policy, making it impossible to learn the optimal policy. If we discretize coarsely, we risk aggregating states that do not belong together. If we discretize finely, we often end up with an unmanageably huge state or action space. Practical learning robots require compact representations capable of generalizing experiences in continuous domains. Furthermore, actions should vary smoothly in response to smooth changes in a state.

- Lack of training data

Since we are generating data by interacting with the real world, the rate at which we get new training points is limited. Robot sensors often have an inherent maximum sampling rate. Sensors which sample extremely quickly will simply generate many training points that are almost identical. We are interested in learning on-line, while the robot is interacting with the world. This means that we cannot wait until we have a large batch of training examples before we begin learning. Our learning system must learn aggressively and rapidly. It must also be able to use whatever data points it has efficiently, extracting as much information from them as possible, and generalizing between similar observations when appropriate.

- Sensor noise

Finally, RL depends on the ability to perceive the unique state of the robot in order to map it to the appropriate action. Sensor noise and error increase state uncertainties, which further slow down the learning process.

In spite of its weaknesses, RL appears to be a promising direction for learning with real robots, in particular because it uses direct information from the real world to improve the robot's performance. In this thesis, we discuss a new learning paradigm which offers some possible solutions to these problems.

3.3.3 The Khepera Robot

The robot employed in the experiments described in this thesis is a miniature mobile robot called Khepera [50] shown in Figure 3.2. It is cylindrical in shape, measuring 55mm in diameter and 30 mm in height. Its weight is only 70 g and its small size allows experiments to be performed in a small work area. The robot is supported by two lateral wheels that can rotate in both directions and two rigid pivots in the front and in the back.

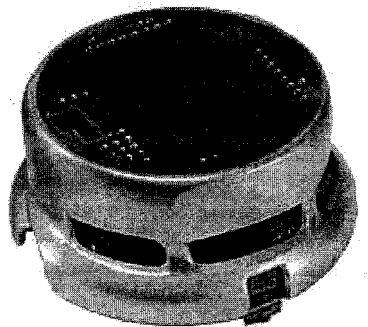


Figure 3.2 The miniature mobile robot: Khepera

Khepera can be remotely controlled by a computer through a serial link depicted in Figure 3.3. The serial connection provides electrical power and supports fast data communication between the robot and the computer. The control system of the robot can run on the computer that reads in sensory data and gives motor commands in real time while the robot moves on a nearby environment.

Alternatively, one can download the code of the control system on the processor of the robot and then disconnect the cable.

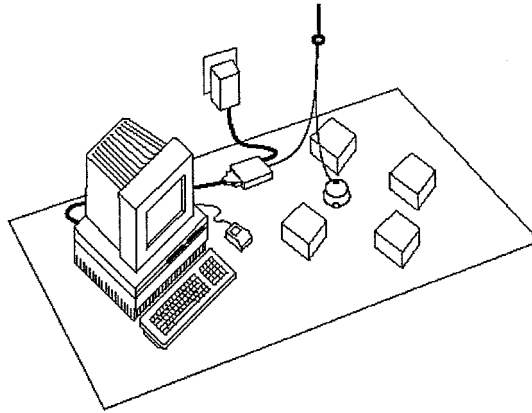


Figure 3.3 The Khepera robot and its working environment

The basic configuration of Khepera is composed of the CPU and the sensory/motor boards. The CPU board encloses the robot's micro-controller, system and user memory, an A/D converter for the acquisition of analog signals coming from the sensory/motor board, and an RS232 serial line miniature connector that can be used for data transmission and power supply from an external computer. The micro-controller includes all the features needed for easy interfacing with memories, with I/O ports and with external interrupts.

The sensory/motor board includes two DC motors coupled with incremental sensors, eight analogues denoted by infra-red(IR) proximity sensor (S_0, \dots, S_7) in Figure 3.4 and an on-board power supply. Each IR sensor is composed of an emitter and an independent receiver. The dedicated electronic interface uses multipliers, sample/holds and operational amplifiers. This allows absolute ambient light and estimation, by reflection, of the relative position of an object to the robot to be measured. This estimation gives, in fact, information about the distance between the robot and the obstacle. The sensor readings are integer values in the

range of $[0, 1023]$. A sensor value of 1023 indicates that the robot is very close to the object, and a sensor value of 0 indicates that the robot does not receive any reflection of the IR signal.

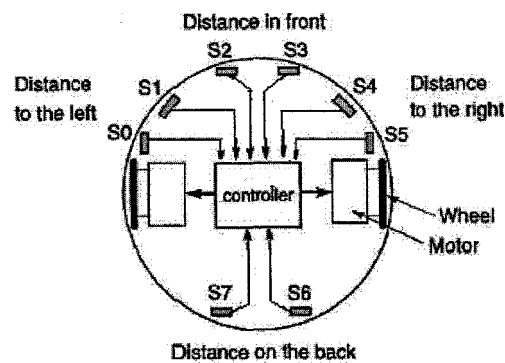


Figure 3.4 Position and orientation of sensors on the Khepera

In addition to the real robot, we also use the simulation version of the Khepera [96] for carrying out a systematic comparison of the different approaches we are interested in. The program simulates Kheperas connected to the computer via a serial link in a very realistic way in the Matlab environment. Simulated Kheperas are controlled in the same way as real, physical Kheperas.

Chapter 4

Design of Dynamic Fuzzy Q-Learning (DFQL) Algorithm

In Chapter 2, we describe the fuzzy logic systems, and the learning methods of fuzzy systems based on Q-learning. However, all the algorithms described only adjust the parameters of fuzzy systems and do not involve structure identification. In Chapter 3, we introduce RL and some algorithms for generalizing experiences. However, most of these works assume discrete actions. In order to cope with these problems, we propose Dynamic Fuzzy Q-Learning (DFQL) in this chapter. Detailed descriptions of the DFQL architecture, on-line structure and parameter learning algorithm and modeling method are presented. In order to demonstrate the efficiency of proposed algorithms, the proposed algorithm is applied to obstacle avoidance behavior of the Khepera robot.

4.1 Motivation and Development

Two main research tracks influence our work. The first one uses the concept of fuzzy logic, and the second one uses that of machine learning. From the first point

of view, DFQL is the learning method used to tune a fuzzy system in a very flexible way whereas in the second point of view, DFQL is a means of introducing generalization in the state space and generating continuous actions in RL problems.

FISs, which are numerical model-free estimators and dynamical systems, are a popular research topic due to the following reasons: 1) The rule structure of an FIS makes it easy to incorporate human knowledge of the target system directly into the fuzzy modeling process; 2) When numerical information of a target system is available, mathematical modeling methods can be used for fuzzy modeling. Several requirements for a learning algorithm must be fulfilled for appropriate modeling of an FIS.

- Evaluative signals

For the learning problem, training data are very rough and coarse, and are just "evaluative" in contrast with the "instructive" feedback in supervised learning. The learning algorithm should be capable of constructing an FIS based on this simple evaluative scalar signal. In addition to the roughness and non-instructive nature of the information, a more challenging problem the algorithm should be able to deal with is that the signal may only be available at a time long after a sequence of actions have occurred. In other words, prediction capabilities are necessary in this learning system.

- Structure and parameter learning

The algorithm should deal with not only parameter estimation but also structure identification of a learning FIS. Structure identification of fuzzy system is equivalent to partitioning the input space. The spirit of FIS resembles that of "divide and conquer"- the antecedent of a fuzzy rule defines a local fuzzy region, while the consequent describes the behavior

within the region via various constituents. If the premise structure of a fuzzy system is determined by clustering the input via an on-line self-organizing learning approach, a more flexible learning scheme can be formed. Furthermore, the learning method should find meaningful fuzzy terms for each input variable, by which it is possible to interpret the acquired knowledge in the form of linguistic rules.

- On-line learning

We are interested in on-line learning, so the algorithm must be capable of learning one data point at a time. We should not have to wait until we have a large batch of data points before training the algorithm. It precludes the use of learning algorithms that take a long time to learn such as GA. GA is a flexible input space partitioning learning method, however, it is very time consuming and unsuitable for on-line learning.

Q-learning is the most popular RL method that can be used to learn a mapping from state-action pairs to long-term expected values. Some forms of generalization are used to extend Q-learning to deal with large continuous state and action spaces. Several requirements for a learning algorithm for practical use are:

- Adaptive Generalization

Q-learning with discretised states and actions scale poorly. As the number of state and action variables increase, the size of the table used to store Q-values grows exponentially. In a large, smooth state space, we generally expect compact representations to be able to generalize experience in continuous domains. Furthermore, it would be desired to improve generalization capabilities at state spaces that are deemed to be "important".

- Continuous states and actions

Many real-world control problems require actions of a continuous nature, in response to continuous state measurements. It should be possible that actions vary smoothly in response to smooth changes in a state. Some problems that we may wish to address, such as high performance control of mobile robots, cannot be adequately carried out with coarse coded inputs and outputs. Motor commands need to vary smoothly and accurately in response to continuous changes in a state.

- Integration of domain knowledge

The algorithm is used for fast on-line learning so as to adapt in real time. Initially, if the learning system knows nothing about the environment, it is forced to act more or less arbitrarily. Integration of domain knowledge to avoid learning from scratch is highly desired.

- Incremental and aggressive learning

The learning algorithm should not be subject to destructive interference or forget what it has learned so far but incrementally adapt the model complexity. It should be capable of producing reasonable predictions based on only a few training points.

In order to cope with these requirements, a novel DFQL learning algorithm is proposed. It is an automatic method capable of self-tuning an FIS based only on reinforcement signals. The DFQL provides an efficient learning method whereby not only the conclusion part of an FIS can be adjusted online, but also the structure of an FIS can be constructed simultaneously. Based on the criterion of system performance, new fuzzy rules can be generated automatically. Continuous states

are handled and continuous actions are generated by fuzzy reasoning. Based only on the reinforcement signals, the proposed method consists of assigning a quality to each local action available in the learner's fuzzy rules. The obtained local policies are then used to produce a global policy that allows us to solve the problem. Prior knowledge can be embedded into the fuzzy rules, which can reduce the training time significantly.

4.2 Preceding Works

As foreshadowed in Chapter 2, the idea and implementation of dynamic partition of the input spaces have been proposed in several previous works in the family of supervised learning algorithms.

A kind of sequential learning algorithms based on RBF are presented in [64,74,104,152,153] in order to overcome the drawback that the number of hidden units is fixed *a priori* in the classical approach to RBF network implementation. The neural network, called a Minimal Resource Allocation Network (MRAN), starts with no hidden units and grows by allocating new hidden units based on the novelty in the observations which arrive sequentially. When input-output data are received during training, the decision to generate a new hidden unit depends on the distance and error conditions. Furthermore, the MRAN learning algorithm combines the growth criterion of the resource allocation network with a pruning strategy based on the relative contribution of each hidden unit to the overall network output. The resulting network leads toward a minimal topology for the resource allocation network.

As mentioned in Chapter 2, it has been shown that a simplified class of FISs and RBF Networks are functionally equivalent [45]. Therefore, the same idea of

MRAN can be implemented in the fuzzy systems. Though fuzzy systems could serve as a very powerful tool for system modeling and control, partitioning the input space and determining an appropriate number of rules in a fuzzy system are still open issues. In order to cope with this problem, a learning algorithm for dynamic fuzzy neural networks (DFNN) based on extended RBF neural networks has been developed in [35,149]. The DFNN learning algorithm is an online supervised structure and parameter learning algorithm for constructing the fuzzy system automatically and dynamically. Fuzzy rules can be recruited or deleted dynamically according to their significance to the system's performance so that not only parameters can be adjusted, but also the structure can be self-adaptive. Given the supervised training data, the algorithm firstly decides whether or not to generate fuzzy rules based on two proposed criteria, system errors and ε - completeness of fuzzy rules. Subsequently, the algorithm will decide whether there are redundant rules to be deleted based on the error reduction ratio [143].

The methods of MRAN and DFNN provide the idea of dynamic partitioning of the input spaces, though both these methods are classified as supervised learning. In order to deal with the requirements of parameter and structure learning in FIS and adaptive generalization in Q-learning, the idea of automatic generation of new fuzzy rules is used in DFQL, which is derived from the concepts of MRAN and DFNN. The FIS we used in DFQL is functionally equivalent to RBF networks. The incremental on-line learning scheme of DFQL is closely related to sequential learning in MRAN and DFNN. Incremental growth of the DFQL is accomplished by generating fuzzy rules when the regions are not sufficiently covered by the rules or the system performance is unsatisfactory. The way of estimating premise parameters of new rules is realized by the same mechanism introduced in DFNN.

Next, we present details of the architecture and algorithm of DFQL. After that, we discuss the main differences between the DFQL and the supervised learning algorithms including MRAN and DFNN.

4.3 Architecture of DFQL

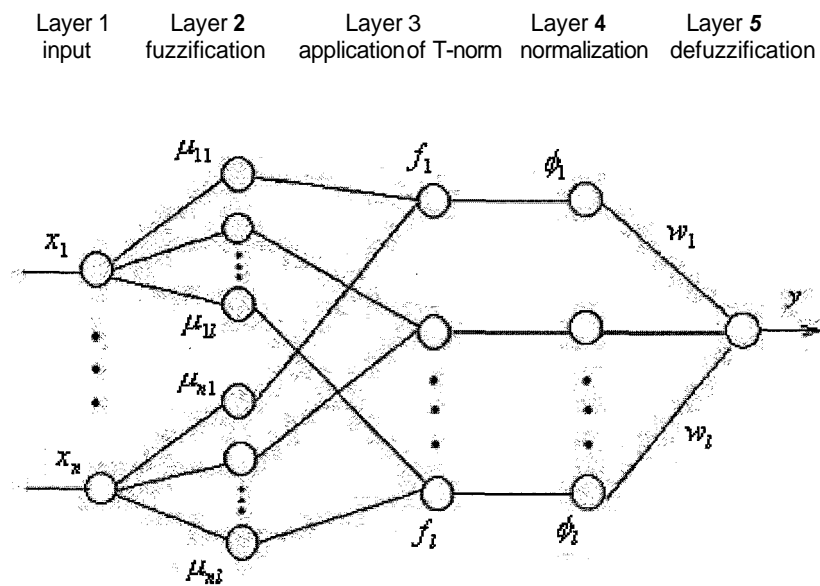


Figure 4.1 Structure of fuzzy rule sets of DFQL

DFQL is an extension of the original Q-Learning method into a fuzzy environment. State-space coding is realized by the input variable fuzzy sets. A state described by a vector of fuzzy variables is called a fuzzystate. A learner may partially visit a fuzzy state, in the sense that real-valued descriptions of the state of the system may be matched by a fuzzy state description with a degree less than one. Since more than one fuzzystate may be visited at the same time, possibly with different degrees, we have a smooth transition between a state and its neighbors, and, consequently, smooth changes of actions done in different states. Both the actions and the Q-function are represented by an FIS whose fuzzy logic

rules can be self-constructed based on the system performance. The structure of the fuzzy rule sets of DFQL is shown in Figure 4.1.

We describe an FIS based on the TSK-type structure, which has a total of five layers. In Section 2.2.1, we point out the equivalence between RBF networks and the TSK-type FIS. Similarly, the FIS we used is functionally equivalent to Ellipsoidal Basis Function (EBF) networks. Nodes in layer one are input nodes which represent input linguistic variables. Layer five is the output layer. Nodes in layer two act as membership functions which represent the terms of the respective linguistic variables. Each node in layer three is a rule node which represents one fuzzy rule. Nodes in layer four normalize the outputs of layer three. Thus, all the layer-four nodes form a fuzzy rule set. Layer four links define the consequents of the rule nodes. In the following context, we will indicate signal propagation and the basic function in each layer of the DFQL.

- Layer one transmits values of the input linguistic variable x_i , $i = 1, \dots, n$ to the next layer directly.
- Layer two performs membership functions to the input variables. The membership function is chosen as a Gaussian function of the following form:

$$\mu_{ij}(x_i) = \exp\left[-\frac{(x_i - c_{ij})^2}{\sigma_{ij}^2}\right] \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, l \quad (4.1)$$

where μ_{ij} is the j th membership function of x_i , c_{ij} and σ_{ij} are the center and width of the j th Gaussian membership function of x_i respectively.

- Layer three is a rule layer. The number of nodes in this layer indicates the number of fuzzy rules. If the T-norm operator used to compute each rule's firing strength is multiplication, the output of the j th rule R_j ($j=1,2,\dots,l$) in layer 3 is given by

$$f_j(x_1, x_2, \dots, x_n) = \exp \left[- \sum_{i=1}^n \frac{(x_i - c_{ij})^2}{\sigma_{ij}^2} \right] \quad j = 1, 2, \dots, l \quad (4.2)$$

- Normalization takes place in layer 4 and we have

$$\phi_j = \frac{f_j}{\sum_{i=1}^l f_i} \quad j = 1, 2, \dots, l \quad (4.3)$$

- Layer five nodes define output variables. If defuzzification is performed in layer 5 using the center-of-gravity method, the output variable as a weighted summation of incoming signals, is given by

$$y = \sum_{j=1}^l \phi_j w_j \quad (4.4)$$

where y is the value of an output variable and w_j is the consequent parameter of the j th rule which is defined as a real-valued constant.

The firing strength of each rule shown in Eq. (4.2) can be regarded as a function of regularized Mahalanobis distance (M-distance), i.e.

$$f_j = \exp(-md^2(j)) \quad (4.5)$$

where

$$md(j) = \sqrt{(X - C_j)^T \sum_j^{-1} (X - C_j)} \quad (4.6)$$

is the M-distance, $X = [x_1 \cdots x_n]^T \in \mathfrak{R}^n$, $C_j = [c_{1j}, c_{2j} \cdots c_{nj}]^T \in \mathfrak{R}^n$ and Σ_j^{-1} is defined as follows:

$$\Sigma_j^{-1} = \begin{bmatrix} \frac{1}{\sigma_{1j}^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_{2j}^2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{\sigma_{nj}^2} \end{bmatrix} \quad j = 1, 2, \dots, l \quad (4.7)$$

Thus, the input variable fuzzy sets are used to represent appropriate high-dimensional continuous sensory spaces. The fuzzy rule identification is equivalent to determination of the DFQL structure. The number and position of the input fuzzy labels can be set using a priori knowledge of the user. Generally speaking, if we do not have any knowledge about the system, identical membership functions whose domains can cover the region of the input space evenly are chosen, and for every possible combination of input fuzzy variables, a fuzzy rule has to be considered. However, the number of rules increases exponentially with increase in the number of input variables. As a consequence, the FIS often includes many redundant or improper membership functions and fuzzy rules. This leads us to develop a learning algorithm that is capable of automatically determining the fuzzy rules online.

In the DFQL approach, each rule R_i has m possible discrete actions $A = \{a_1, \dots, a_m\}$ and it memorizes the parameter vector q associated with each of these actions. These q values are then used to select actions so as to maximize the discounted sum of reward obtained while achieving the task. We build the FIS with competing actions for each rule R_i as follows:

$$\begin{aligned}
 R_i: \text{ If } X \text{ is } S_i \text{ then } & a_1 \text{ with } q(S_i, a_1) \\
 & \text{or} \quad a_2 \text{ with } q(S_i, a_2) \\
 & \dots\dots\dots \\
 & \text{or} \quad a_m \text{ with } q(S_i, a_m)
 \end{aligned}$$

where X is the vector of input variables and S_i are labels of fuzzy sets that describe the a fuzzy state of the i th rule. Figure 4.2 shows the consequent parts of DFQL. The continuous action performed by the learner for a particular state is a weighted sum of the actions elected in the fired rules that describe this state, whose weights are normalized firing strengths vector of the rules, ϕ . Subsequently, the TD method updates the Q-values of the elected actions according to their contributions.

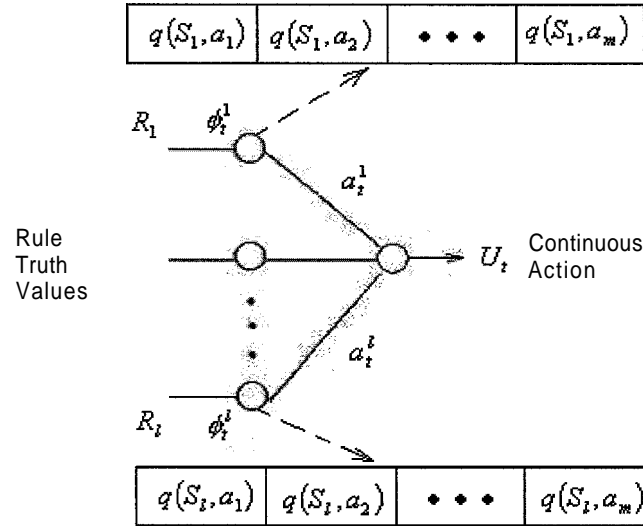


Figure 4.2 Consequent parts of DFQL

4.4 DFQL Learning Algorithm

This section proposes an on-line structure and parameter learning algorithm for constructing the DFQL automatically and dynamically. The details of the DFQL algorithm are presented as follows. After a brief description of generation of continuous actions, the mechanism of updating Q-values is introduced. Next, self-organizing features of the FIS based on the ε -completeness of fuzzy rules and the TD error criteria are elaborated. Finally, the flowchart of the algorithm and an overview of the one-time-step global working procedure are presented.

4.4.1 Generation of Continuous Actions

The generation of continuous actions depends upon a discrete number of actions of every fuzzy rule and the vector of firing strengths of fuzzy rules. In order to explore the set of possible actions and acquire experiences through the reinforcement signals, the actions in every rule are selected using the exploration-exploitation strategy that is described in Section 3.1.4. Here, we use π to denote the exploration/exploitation policy employed to select a local action a from possible discrete actions vector A , as follows:

$$a_\pi = \pi_{a \in A}(q(S, a)) \quad (4.8)$$

At time step t , the input state is X_t . Assume that l fuzzy rules have been generated and the normalized firing strength vector of rules is ϕ_t . Each rule R_i has m possible discrete actions A . Local actions selected from A compete with each other based on their q-values, while the winning local action a^i of every rule cooperates

to produce the global action based on the rule's normalized firing strength, ϕ . The global action is given by

$$U_t(X_t) = \sum_{i=1}^l \pi_A(q_t) \phi_t^i = \sum_{i=1}^l a_t^i \phi_t^i \quad (4.9)$$

where a_t^i is the selected action of rule R_i at time step t .

4.4.2 Update of q-values

As in DFQL, we define a function Q , which gives the action quality with respect to states. Q -values are also obtained by the FIS outputs, which are inferred from the quality of local discrete actions that constitute the global continuous action. Under the same assumptions used for generation of continuous actions, the Q function is given by

$$Q_t(X_t, U_t) = \sum_{i=1}^l q_t(S_i, a_t^i) \phi_t^i \quad (4.10)$$

where U_t is the global action, a_t^i is the selected action of rule, R_i at time step t and q_t is the q -value associated with the fuzzy state, S_i and action, a_t^i .

Based on TD learning, the Q -values corresponding to the rule optimal actions which are defined as follows:

$$V_t(X_t) = \sum_{i=1}^l \left(\max_{a \in A} q_t(S_i, a) \right) \phi_t^i \quad (4.11)$$

are used to estimate the TD error which is defined by

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma V_t(X_{t+1}) - Q_t(X_t, U_t) \quad (4.12)$$

where r_{t+1} is the reinforcement signal received at time $t+1$ and γ is the discount factor used to determine the present value of future rewards. Note that we have to estimate this error term only with quantities available at time step $t+1$.

This TD error can be used to evaluate the action just selected. If the TD error is positive, it suggests that the quality of this action should be strengthened for future use, whereas if the TD error is negative, it suggests that the quality should be weakened. The learning rule is given by

$$q_{t+1}(S_i, a_t^i) = q_t(S_i, a_t^i) + \alpha \tilde{\epsilon}_{t+1} \phi_t^i \quad i = 1, 2, \dots, l \quad (4.13)$$

where α is the learning rate.

4.4.3 Eligibility Traces

In order to speed up learning, eligibility traces are used to memorize previously visited rule-action pairs, weighted by their proximity to time step t . The trace value indicates how rule-action pairs are eligible for learning. Thus, it not only permits tuning of parameters used at time step t , but also those involved in past steps. Here, we introduce the basic method without complex implementation, similar to [48]. The more efficient and faster learning method using eligibility traces for complicated tasks is discussed in Chapter 6.

Let $Tr_t(S_i, a_i)$ be the trace associated with discrete action a_i of rule R_i at time step t

$$Tr_t(S_i, a_i) = \begin{cases} \gamma \lambda Tr_{t-1}(S_i, a_i) + \phi_t^i & \text{if } a_i = a_t^i \\ \gamma \lambda Tr_{t-1}(S_i, a_i) & \text{otherwise} \end{cases} \quad (4.14)$$

where the eligibility rate λ is used to weight time steps.

The parameter updating law given by Eq. (4.13) becomes, for all rules and actions,

$$q_{t+1}(S_i, a_j) = q_t(S_i, a_j) + \alpha \tilde{\epsilon}_{t+1} Tr_t(S_i, a_j) \quad i=1, 2, \dots, l, j=1, 2, \dots, m \quad (4.15)$$

and the traces are updated between action computation and its application.

4.4.4 ε -Completeness Criterion for Rule Generation

Due to the highly complex and nonlinear characteristic of the problem space, uniform distribution of the fuzzy sets is usually not optimal. If a fuzzy partition is too coarse, the performance may be poor. If a fuzzy partition is too fine, many fuzzy IF-THEN rules cannot be generated because of the lack of training patterns in the corresponding fuzzy subspaces. For a problem, some parts of pattern space might require fine partition, while other parts require only coarse partition. Therefore, the choice of an appropriate fuzzy partition, i.e. structure identification of FIS is important and difficult. In this section and the next one, we propose two criteria, namely the ε -completeness and TD error criteria, for generating fuzzy rules automatically, which allow us to circumvent the problem of creating the structure of an FIS by hand.

Definition 4.1: ε -Completeness of Fuzzy Rules [59]:

For any input in the operating range, there exists at least one fuzzy rule so that the match degree (or firing strength) is no less than ε .

Remark: In fuzzy applications, the minimum value of ε is usually selected as $\varepsilon = 0.5$.

From the viewpoint of fuzzy rules, a fuzzy rule is a local representation over a region defined in the input space. If a new pattern satisfies ε -completeness, the DFQL will not generate a new rule but accommodate the new sample by updating the parameters of existing rules. According to ε -completeness, when an input vector $\mathbf{X} \in \mathbb{R}^n$ enters the system, we calculate the M-distance $md(j)$ between the observation \mathbf{X} and centers C_j ($j = 1, 2, \dots, l$) of existing EBF units according to Eqs. (4.6) and (4.7). Next, find

$$\mathbf{J} = \arg \min_{1 \leq j \leq l} (\text{md}(j)) \quad (4.16)$$

If

$$\text{md}_{\min} = \text{md}(\mathbf{J}) > k_d \quad (4.17)$$

where k_d is a pre-specified threshold and is chosen as follows

$$k_d = \sqrt{\ln(1/\varepsilon)} \quad (4.18)$$

then we have

$$f(\mathbf{J}) < \exp(-k_d^2) = \exp(-(\sqrt{\ln(1/\varepsilon)})^2) = \varepsilon \quad (4.19)$$

This implies that the existing system is not satisfied with ε -completeness and a new rule should be considered.

4.4.5 TD Error Criterion for Rule Generation

It is not sufficient to consider ε -completeness of fuzzy rules as the criterion of rule generation only. New rules need to be generated in regions of the input fuzzy subspace where the approximation performance of the DFQL is unsatisfactory. We introduce a separate performance index, ξ^i , for each fuzzy subspace which enables the discovery of "problematic" regions in the input space. If the performance index indicates that the situation is wrongly classified, a new fuzzy rule is created at the location of the input situation. This index can be attested by any method which captures critical areas that require high resolution. Here, we present a general method based on TD errors. The performance index is updated as follows:

$$\xi_{t+1}^i = [(K - \phi_t^i)\xi_t^i + \phi_t^i(\tilde{\varepsilon}_{t+1})^2]/K \quad K > 0 \quad (4.20)$$

Using the squared TD error as the criterion, the normalized rule firing strength ϕ^i determines how much the fuzzy rule R_i affects the TD error. It should be noted that Eq. (4.20) acts as a digital low-pass filter. In this way, TD errors in the past are

gradually "forgotten" as time passes, but are never completely lost. The more recent the TD error is received, the more it affects the value of ξ . The initial value of ξ is set to zero. The parameter K controls the overall behavior of ξ . A small value of K makes ξ adapt very rapidly and a large value makes ξ more stable in a noisy environment. Thus, if ξ^i is bigger than a certain threshold, k_e , further segmentations should be considered for this fuzzy subspace at least.

4.4.6 Estimation of Premise Parameters

Combining the ε -completeness criterion and the TD error criterion together, we obtain the following procedure of generating a new rule: When an input vector $X \in \mathbb{R}^n$ enters the system, we calculate the M-distance $md(j)$ between the observation X and centers C_j ($j=1,2,\dots,l$) of existing EBF units. Next, find

$$J = \arg \min_{1 \leq j \leq l} (md(j))$$

If

$$md_{\min} = md(J) > k_d$$

where k_d is a ε -completeness threshold, this implies that maybe the existing system is not satisfied with ε -completeness and a new rule should be considered.

Otherwise, if

$$\xi^J > k_e \quad (4.21)$$

where k_e is a TD error criterion threshold, this fuzzy rule R_J does not satisfy the TD error criterion and a new rule should be considered.

Once a new rule is considered, the next step is to assign centers and widths of the corresponding membership functions. A new rule will be formed when the input

pattern $X \in \mathfrak{R}^n$ enters the system according to two criteria of rules generation. Next, the incoming multidimensional input vector X is projected to the corresponding one-dimensional membership function for each input variable i ($i=1, \dots, n$). Assume that l membership functions have been generated in the i th input variable and the Euclidean distance ed_i between the data x_i and the boundary set Φ_i is computed as follows:

$$ed_i(j) = |x_i - \Phi_i(j)| \quad j=1, 2, \dots, l+2 \quad (4.22)$$

where $\Phi_i \in \{x_{i\min}, c_{i1}, c_{i2}, \dots, c_{il}, x_{i\max}\}$, we find

$$j_k = \arg \min_{j=1, 2, \dots, l+2} (ed_i(j)) \quad (4.23)$$

If

$$ed_i(j_k) \leq k_{mf} \quad (4.24)$$

where k_{mf} is a predefined constant that controls the similarity of neighboring membership functions, x_i is deemed completely represented by the existing fuzzy set $E_{ijk}(c_{ijk}, \sigma_{jik})$ without generating a new membership function. Usually, k_{mf} is selected between 0.1 and 0.3 for normalized input variables. Decreasing k_{mf} indicates that higher similarity between two membership functions is allowed. Otherwise, if

$$ed_i(j_k) > k_{mf}$$

a new Gaussian membership function is allocated whose center is

$$c_{i(l+1)} = x_i \quad (4.25)$$

and the widths of membership functions in the i th input variable are adjusted as follows:

$$\sigma_{ik} = \frac{\max\{|c_{ik} - c_{i(k-1)}|, |c_{i(k+1)} - c_{ik}|\}}{\sqrt{\ln(1/\varepsilon)}} \quad k=1, \dots, l+1 \quad (4.26)$$

where $c_{i(k-1)}$ and $c_{i(k+1)}$ are the two centers of adjacent membership functions of the middle membership function whose center is c_{ik} . Note that only the new membership function and its neighboring membership functions need to be adjusted. The main result concerning adjusting membership functions to satisfy ε -completeness of fuzzy rules in each one-dimensional input variable can be summarized in the following statement.

Statement 4.1: Let $N = [x_{\min}, x_{\max}] = [a, b]$ be the universe of one-dimensional input x . If each fuzzy set $E_k = \{(x, \mu_k(x)) | x \in N\}$ ($k = 1, \dots, m$) is represented as a Gaussian membership function constructed by the preceding c_k and σ_k , the fuzzy sets E_k satisfy ε -completeness of fuzzy rules, i.e., for all $x \in N$, there exists $k \in \{1, \dots, m\}$, such that $\mu_k(x) \geq \varepsilon$ [149].

We can explain this statement under several different cases:

(a) If there exists only one fuzzy set, i.e. $m = 1$, the membership function can be generated as follows. If $|x_1 - a| \geq k_{mf}$ and $|b - x_1| \geq k_{mf}$, and assuming that

$|x_1 - a| \geq |b - x_1|$, we have $\mu_1(x) = \exp\left(-\frac{(x - c_1)^2}{\sigma_1^2}\right)$ where $c_1 = x_1$ and

$$\sigma_1 = \frac{|x_1 - a|}{\sqrt{\ln(1/\varepsilon)}} \quad \text{For any } x \in [a, b], \quad \text{we have}$$

$$\mu_1(x) \geq \mu_1(a) = \exp\left(-\left(\ln \frac{1}{\varepsilon}\right) \left(\frac{a - x_1}{x_1 - a}\right)^2\right) = \varepsilon. \quad \text{If } |x_1 - a| \leq k_{mf}, \quad \text{we have}$$

$\mu_1(x) = \exp\left(-\frac{(x-c_1)^2}{\sigma_1^2}\right)$ where $c_1 = a$ and $\sigma_1 = \frac{|b-a|}{\sqrt{\ln(1/\varepsilon)}}$. For any $x \in [a, b]$, we

have $\mu_1(x) \geq \mu_1(b) = \exp\left(-\left(\ln \frac{1}{\varepsilon}\right)\left(\frac{b-a}{b-a}\right)^2\right) = \varepsilon$.

(b) If there exists $k \in \{1, \dots, m\}$, such that $|c_k - c_{k-1}| \geq |c_k - c_{k+1}|$, σ_k is chosen as

$\sigma_k = \frac{|c_k - c_{k-1}|}{\sqrt{\ln(1/\varepsilon)}}$ For any $x \in (c_{k-1}, c_{k+1})$ we have

$\mu_k(x) \geq \mu_k(c_{k-1}) = \exp\left(-\left(\ln \frac{1}{\varepsilon}\right)\left(\frac{c_{k-1} - c_k}{c_k - c_{k-1}}\right)^2\right) = \varepsilon$.

We can obtain the same result for other cases.

4.4.7 Working Principle

The flowchart of the algorithm is depicted in Figure 4.3. In order to make the working principle easy to understand, a one-time-step global execution procedure of DFQL is presented. The details of computing the TD error and tuning q-values are described in steps a to d. Next, the procedure of tuning the structure of FIS elaborated on the flowchart is given in step e. This is then followed by taking action and estimating the Q-value in steps f to g, together with the updating eligibility trace in step h. Let $t+1$ be the current time step and we assume that the learner has performed action U_t and has received a reinforcement signal r_{t+1} . The steps are summarized here:

- a. Check the ε -completeness and TD error criteria according to the current state, X_{t+1} . If a new fuzzy rule need to be generated, tune the structure of

FIS and initialize the parameter vector q according to the algorithm described in Section 4.3.6;

- b. Approximate the optimal evaluation function corresponding to the current state and FIS by using the optimal local action quality defined at time step t i.e. $V_t(X_{t+1})$ as defined in Eq. (4.11);
- c. Compute the TD error $\tilde{\epsilon}_{t+1}$ using Eq. (4.12);
- d. Tune the parameter vector q according to Eq. (4.15) based on current eligibility trace;
- e. Tune parameter ξ with Eq. (4.20) being used as a TD error criterion;
- f. Elect local actions based on the new vector q_{t+1} and compute the global action $U_{t+1}(X_{t+1})$ according to the new FIS governed by Eq. (4.9);
- g. Estimate the new evaluation function for the current state with the new vector q_{t+1} and the actions effectively elect Eq. (4.10). Note that $Q_{t+1}(X_{t+1}, U_{t+1})$ will be used for error computation at the next time step;
- h. Update the eligibility trace according to Eq. (4.14), which will be used in parameter updating at the next time step. Eligibility trace values need to be reset to zeros at the end of each episode.

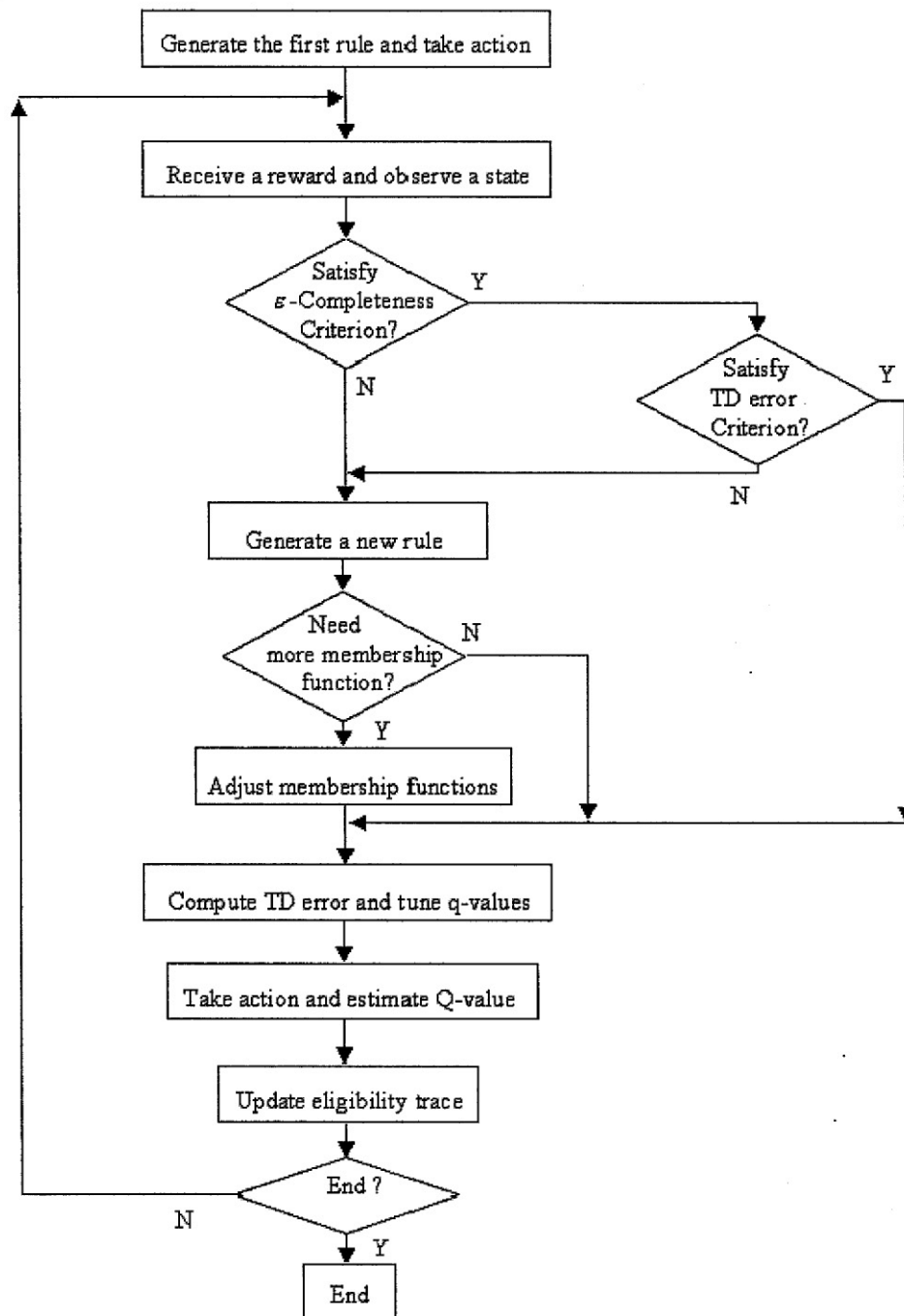


Figure 4.3 Flowchart of the DFQL learning algorithm

4.5 Discussions

The main differences between the DFQL and the supervised learning algorithms including MRAN and DFNN are as follows:

First of all, MRAN and DFNN are both supervised learning where the input-output training data guide the learning process, while DFQL extends these earlier works to reinforcement learning which is based on the simple evaluative scalar signal. Generally speaking, a robot attempts to learn a task in an unsupervised mode without a teacher since it is difficult to find the direct training data. The robot has to perform the task by executing trial-and-error actions through evaluative reinforcement signals. A salient point about DFQL is that the consequent parts of DFQL are based on Q-Learning and the TD method updates these Q-values.

Second, the two criteria for generating new rules are slightly different. New rules are generated as they are needed to better cover the sensory space or when the approximation performance of DFQL is unsatisfactory. Since DFQL is based on Q-Learning, the performance is evaluated according to TD errors instead of the output error in supervised learning.

Third, MRAN removes the units based on their relative contributions and DFNN deletes the rules based on the error reduction ratio, while DFQL does not. MRAN and DFNN are suitable for sequential learning. There may be a large batch of training data and the significance of the observations may be subsequently reduced. On the contrary, DFQL is suitable for robotics systems which require learning to be incremental and rapid. Critical behavior must be learned with a minimal number of trials. For robot navigation applications, deletion would be hazardous since they will not perceive all sensory situations repeatedly after a

fixed number of steps. The learning algorithm should not forget what it has learned so far but incrementally adapt the model complexity. In the future works, pruning might be considered in other applications if a fuzzy rule is not active for a period.

Fourth, DFQL provides a method of exploiting a prior knowledge which is not the same in different supervised learning algorithms. If the learning systems are not able to exploit the prior knowledge in reinforcement learning, it is almost certain that the learning will not be effective. The robot either collides with obstacles that terminate the learning process or explores aimlessly without ever reaching the goal that can take unacceptably long time to discover interesting parts of the space. A way of overcoming this problem is to use prior knowledge as bias to figure out which action deserves attention in each situation. The bias components can be incorporated in the framework of fuzzy rules based on prior knowledge. The premise of rules can be used to generate fuzzy states over the fuzzy input space and the consequents of rules can be used to generate the initial q values, which are called bias. The details are discussed in Chapter 5.

Fifth, the supervised learning provides, at each time step, the desired outputs during learning. On the contrary, the reinforcement learning only gives the quality of actions. In addition to the roughness and non-instructive nature of the information, a more challenging problem is that the signal may only be available at a time long after a sequence of actions have been taken. In other words, prediction capabilities are necessary in this learning system. DFQL with eligibility traces obtain an efficient method which is able to distribute credit throughout sequences of actions, leading to faster and more effective learning in real applications. The details are discussed in Chapter 6.

4.6 Experiments

4.6.1 Obstacle Avoidance for Khepera Robot

In this section, we describe experiments performed on the Khepera robot. The task Khepera has to perform is to navigate freely. Two behaviors are involved. One, which is of higher priority, concerns moving forward. The second concerns avoiding obstacles. The first behavior which involves moving forward when nothing is detected by the sensors is simple and is of no interest here. The second behavior involves knowing how much to turn and in which direction so as to avoid the obstacles. The environment used for implementation and simulation studies is shown in Figure 4.4 and Figure 4.5 respectively. It is a 25cmx35cm arena with lighted shaped walls. Obstacles with different shape and form are introduced at different sections of the maze. Obstacles are put in the maze at the beginning of the experiment. We use the simulated robot to find appropriate parameters and make a systematic comparison of different approaches. These methods are also implemented on the real robot and the results presented as follows are based on experimental data of the real robot.

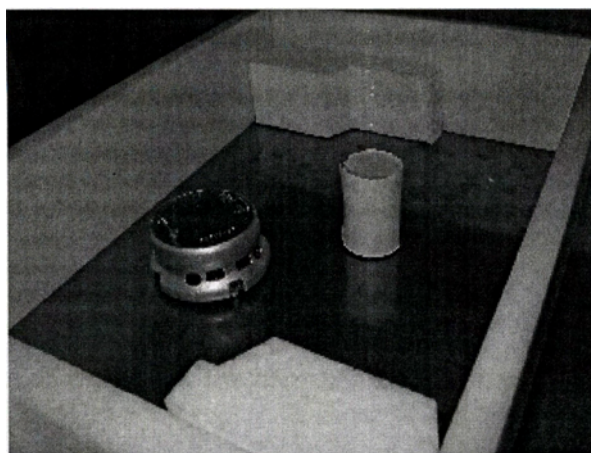


Figure 4.4 Real environment for obstacle avoidance

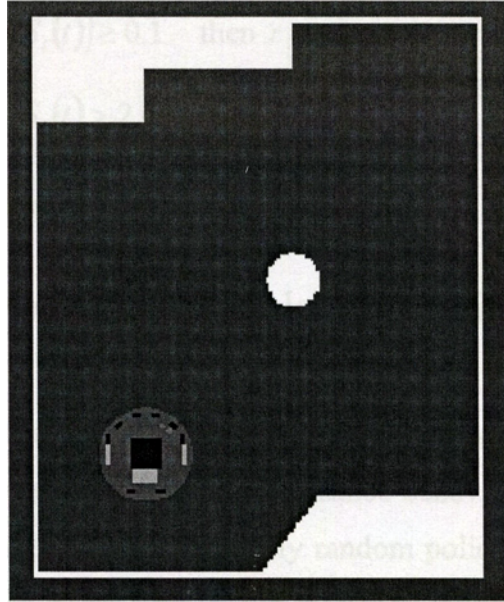


Figure 4.5 Simulation environment for obstacle avoidance

In this experiment, the six sensors on the front of the robot are used for obstacle detection. These sensor values are normalized within the interval $[0, 1]$. A normalized sensor value of 1 indicates that the robot is very close to obstacles, and 0 indicates that the robot is away from obstacles. The robot receives reinforcement signals during the learning phase. For learning obstacle-avoidance behavior, we compare past and present sensor values. The robot will avoid the obstacles when the present sum of the six front sensor values is smaller than the last one and the difference is greater than 0.1 between them. A collision occurs when the sum of the six front sensor values is greater than 2.0. The collision is inferred from the sensor performance. It does not necessarily mean that the robot has touched the wall. The reinforcement signals are as follows:

Let $S_i(t)$ be the sensor value of the sensor i at time t and r be the reinforcement signal:

$$\begin{aligned}
&\text{If } \left(\sum_{i=0}^5 S_i(t-1) - \sum_{i=0}^5 S_i(t) \right) \geq 0.1 \quad \text{then } r = +1; \\
&\quad \text{else if } \sum_{i=0}^5 S_i(t) > 2.0 \quad \text{then } r = -1; \\
&\quad \text{else} \quad \quad \quad r = 0.
\end{aligned}$$

4.6.2 Random Policy and Tabular Q-learning

In order to make a comparison, a look-up table implementation of Q-learning is used with the robot to generate obstacle avoidance behavior. And the experiments are given along with those for a completely random policy. For practical reasons, each sensor value is coded as 1 bit. If the measured value is below the threshold value of 0.2, the sensor bit value is 0, otherwise it is 1. Therefore, the total number of possible situations is restricted to $2^6 = 64$. The total number of actions is reduced to three different speeds per motor, so the total number of possible actions is 9.

Experimental results are presented in two ways: an indicator of the effectiveness of obstacle avoidance $I(t)$ and a local performance index $L(t)$, where t is the number of robot moves executed by the avoidance behavior module from the beginning of the experiment. The distance to the obstacles measures the correspondence of the robot's behavior with the target behavior. When the robot encounters obstacles, the sum of the six front sensor values is memorized as $D(t)$. Note that the higher the sensor value, the shorter the distance is to the obstacles. The indicator $I(t)$ is defined as $\sum_{i=1}^t D(i)/t$. Two local performance indices measuring the effectiveness of the learning process are defined as $L^+(t) = R^+(t)/t$ and $L^-(t) = R^-(t)/t$ respectively, where $R^+(t)$ is the number of moves that have received positive reinforcement signals from the beginning of the experiment and $R^-(t)$ is the

number of moves that have received negative reinforcement signals from the beginning of the experiment. We use 3000 learning iterations and the results are averaged over ten runs. Note that the iteration steps are increased only when the robot detects obstacles and executes avoidance behaviors. The results for a completely random exploration policy are given in Figure 4.6.

The basic Q-learning algorithms of [145] store Q values, i.e. the expected reinforcement values associated with each situation-action pair are organized in a look-up table and updated according to Eq. (3.11). In the experiments, the learning rate α and the discount factor γ are set to 0.5 and 0.9 respectively. The exploration function uses a ε -greedy strategy described in Section 3.1.4 with randomly decreasing proportionate with the number of iterations. The results for basic Q-learning are shown in Figure 4.7. After learning, the synthesized behavior is improved but not perfect. The distance to the obstacles measured during learning is better than pure random behaviors, however, there are still negative reinforcements experienced. Any difficulty in the use of basic Q-learning is the result of its standard tabular formulation. An exhaustive exploration of all situation-action pairs is impossible due to time constraint. Therefore, there are an incredibly small number of explored situation-action pairs versus unknown situation-action pairs. Several refinements have been proposed in order to speed up learning. Madadevan and Connell [80] use the Hamming distance to generalize between similar situations; the same authors also use clusters to generalize across similar situation-action sets. Sutton [126] proposes the Dyna-Q model in which situation-action pairs are randomly carried out to speed up propagation of the Q values through time. However, those methods use the same look-up table as in basic Q-learning implementation and they are subject to a memory requirement for storing all possible situation-action utility values. In practice, there are many situation-action pairs that are never visited and it is pointless to store these utility

values. The most important problem that RL methods based on tabular formulation faces is the limitation of generalization. In this experiment, a unique bit is used to code each sensor value. This is certainly not precise enough. However, it is impossible to use more bits per sensor due to the curse of dimension described in Section 3.2. In order to cope with continuous states and actions, it is necessary to use efficient generalization processes which revolve around the use of experienced situation-action pairs to induce ways of dealing with new unknown situations and actions.

4.6.3 Neural Q-Learning

Neural networks is a kind of approach suitable for generalization of RL methods. Numerous authors [6,65,117,131,138] have proposed a neural implementation of RL, which seems to offer better generalization. The memory size required by the system to store the knowledge is defined, a priori, by the number of connections in the network. In [138], an efficient method named Q-KOHON, which is based on the Kohonen self-organizing map (SOM) [58], is proposed. The self-organizing map is distinguished by the development of a significant spatial organization of the layer. Following the implementation of [138], there are 16 neurons in the map. During the learning phase, the neurons of the SOM approximate the probability density function of the inputs. The inputs are situation, action and the associated Q value. The number of neurons is equal to the number of stored associations. The best action selected in a world situation is given by the neuron that has the minimal distance to the input situation and to a Q value of +1. The learning algorithm updates the Q value weight using Eq. (3.11) and the situation and action weights. The neuron corresponding to the situation and the action effectively performed is selected by finding the minimal distance to the situation and action vectors but nothing concerning the Q value. Together with the selected neuron, the four

neighbors are also updated. The learning coefficient is 0.9 for the selected neuron and 0.5 for the neighborhood. The results are shown in Figure 4.8. The Q-KOHON implementation requires much less memory and learns faster than basic Q-learning. The neural generalization process, through the continuity of the input space, allows us to speed up the Q-learning method. The distance to the obstacles measured during learning is improved and negative reinforcements received reduce distinctly during learning.

However, Q-KOHON and most approaches using neural networks to generalize situations still assume piecewise constant actions. These approaches cannot generate continuous actions and cannot obtain optimal performance. It should be possible that actions vary smoothly in response to smooth changes in a state. The Q-KOHON implementation is not sufficient to solve an application. Q-KOHON is capable of learning behaviors with no prior knowledge; however, more effective learning solutions would be obtained if the initial experience can be incorporated during learning. Although Q-KOHON solves the structural credit assignment problem described in Section 3.2, its efficiency is strictly limited to short sequences of actions. In this experiment, the synthesized behavior is a reactive behavior. It does not integrate sequences of actions. A solution would be to change the reinforcement function so as to take into account sequences of actions.

4.6.4 DFQL

The DFQL approach uses fuzzy rules to introduce generalization in the state space and generate continuous actions. The parameter values are the same as those used in Q-learning. The other parameters for rule generation are: ε -completeness, $\varepsilon = 0.5$; similarity of membership function, $k_{mf} = 0.3$; TD error factor, $K = 50$

and TD error criterion, $k_e = 1$. The eligibility traces are not considered for a fair comparison. The results are shown in Figure 4.9. At the beginning, performances of the robot based on DFQL are worse than other methods due to exploration. However, the performance is improved rapidly and is much better than that of others subsequently and less negative reinforcements received than all other implementations.

The DFQL method achieves good performance very rapidly because the DFQL displays localized generalization in the state space and it updates the Q values with the local actions involved in the selection of the global action according to their contributions at the same time. Furthermore, the optimal action for every possible situation is most likely continuous. Unlike the piecewise constant outputs of Q-KOHON, the DFQL generates the continuous actions by continuous state perception so that actions can vary smoothly with smooth changes in a state. We will present comparisons between the DFQL and other methods that can also handle continuous actions in the following chapters. The number of fuzzy rules is generated automatically based on the criteria of ε -completeness and the TD errors during learning and is shown in Figure 4.10. The compact structure of fuzzy systems is obtained online, which does not include redundant or improper membership functions and fuzzy rules. The number of rules does not increase exponentially with increase in the number of input variables. Detailed comparisons between adaptive structure and fixed structure of fuzzy systems are discussed in Chapter 5. A clear comparison of all the algorithms based on the distance to the obstacles during learning is shown in the Figure 4.11. In order to show the advantage of generalization, Figure 4.12 makes a performance comparison during the learning phase with the random policy, Q-learning, Q-KOHON and the DFQL based on the discounted cumulative reward. This discounted cumulative reward

$\frac{t - R^-(t)}{t}$ corresponds to a measure of the cumulated performance of the robot. It is the number of good actions performed that receive non-negative reinforcements per the total number of moves.

We introduce the basic features of the DFQL algorithm above and more details of the learning abilities are discussed in the following chapters. Furthermore, we present a mechanism using fuzzy rules to incorporate initial knowledge for rapid learning in Chapter 5 and we describe a more general method combined with eligibility traces, the basic mechanism for temporal credit assignment, in Chapter 6.

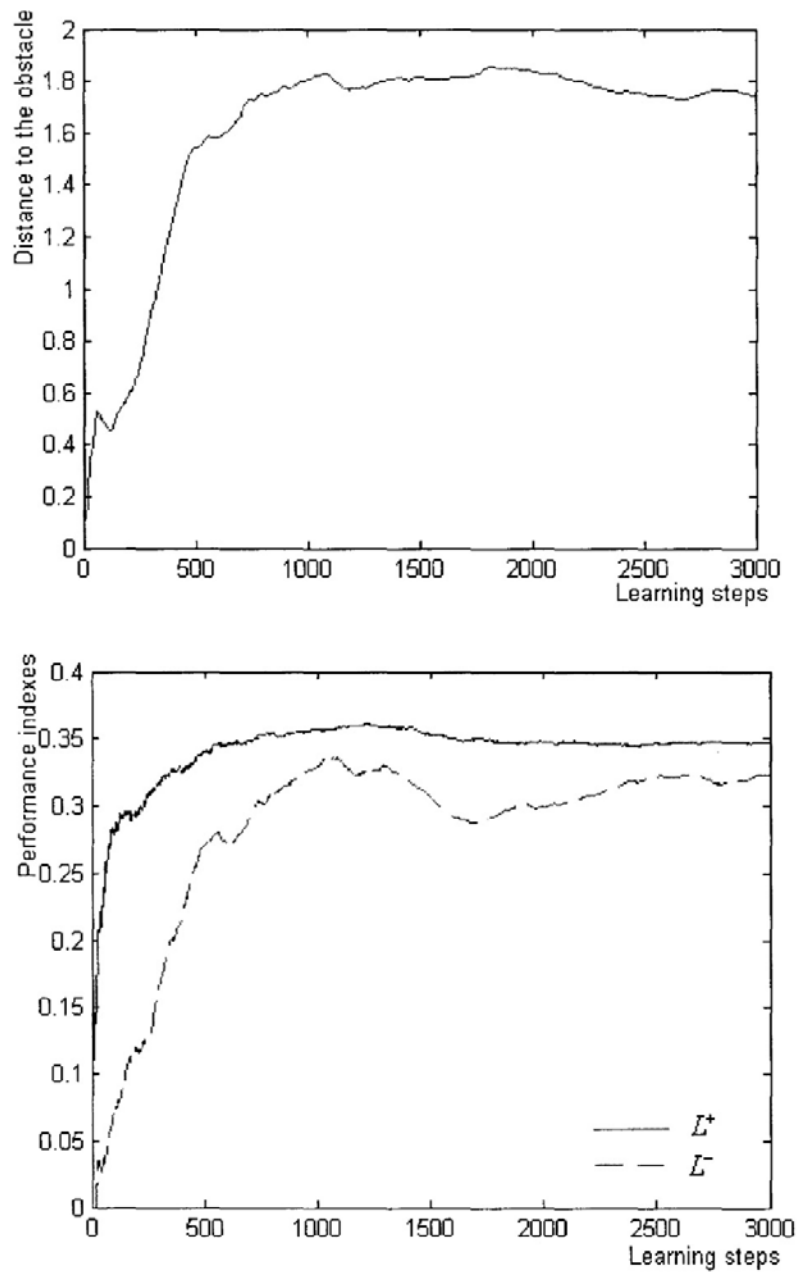


Figure 4.6 Distance to the obstacles and local performance indices based on a random exploration policy

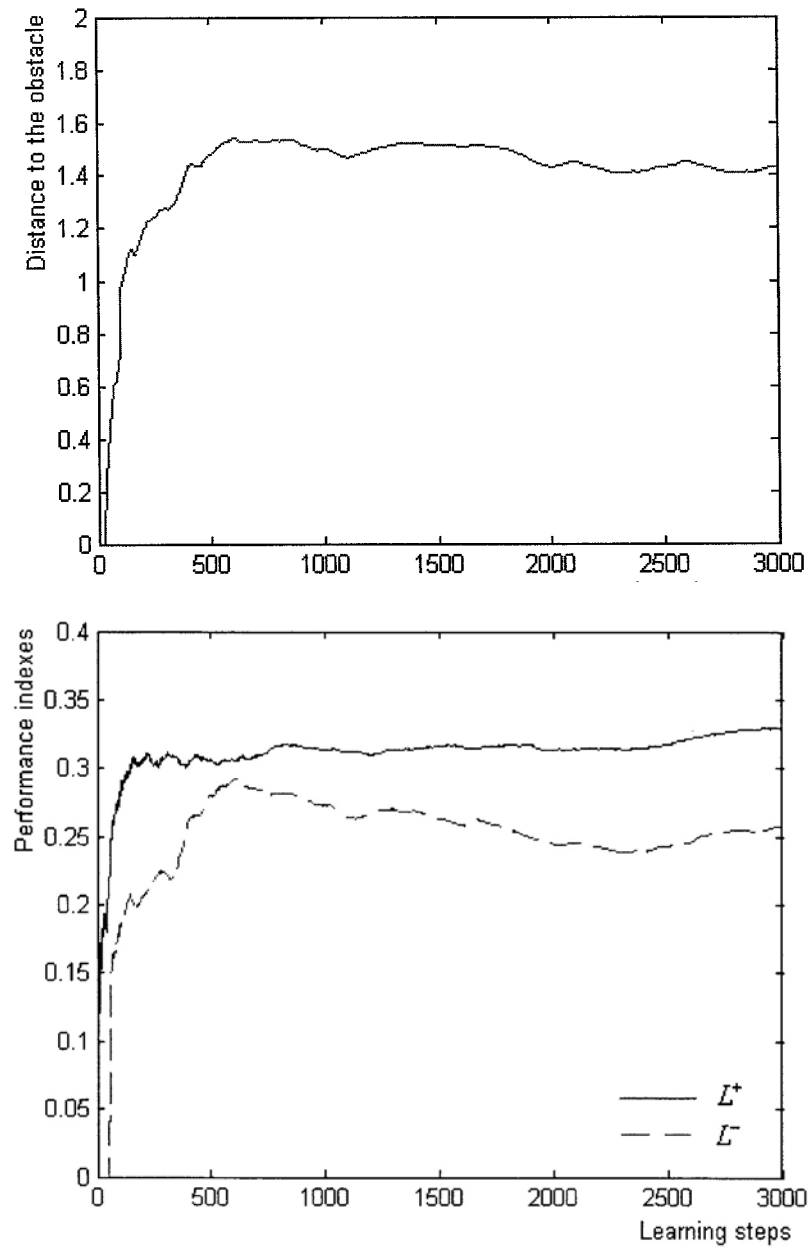


Figure 4.7 Distance to the obstacles and local performance indices during learning with basic Q-learning

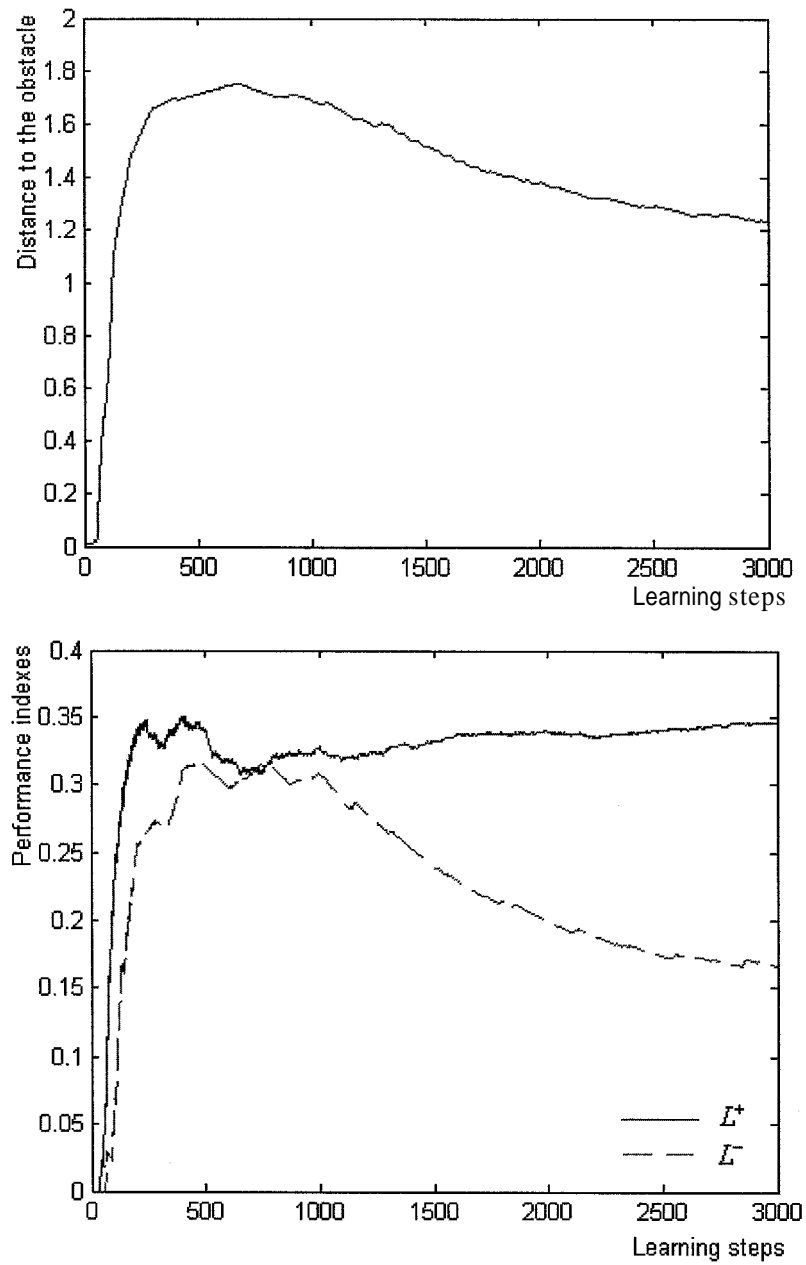


Figure 4.8 Distance to the obstacles and local performance indices during learning with Q-KOHON

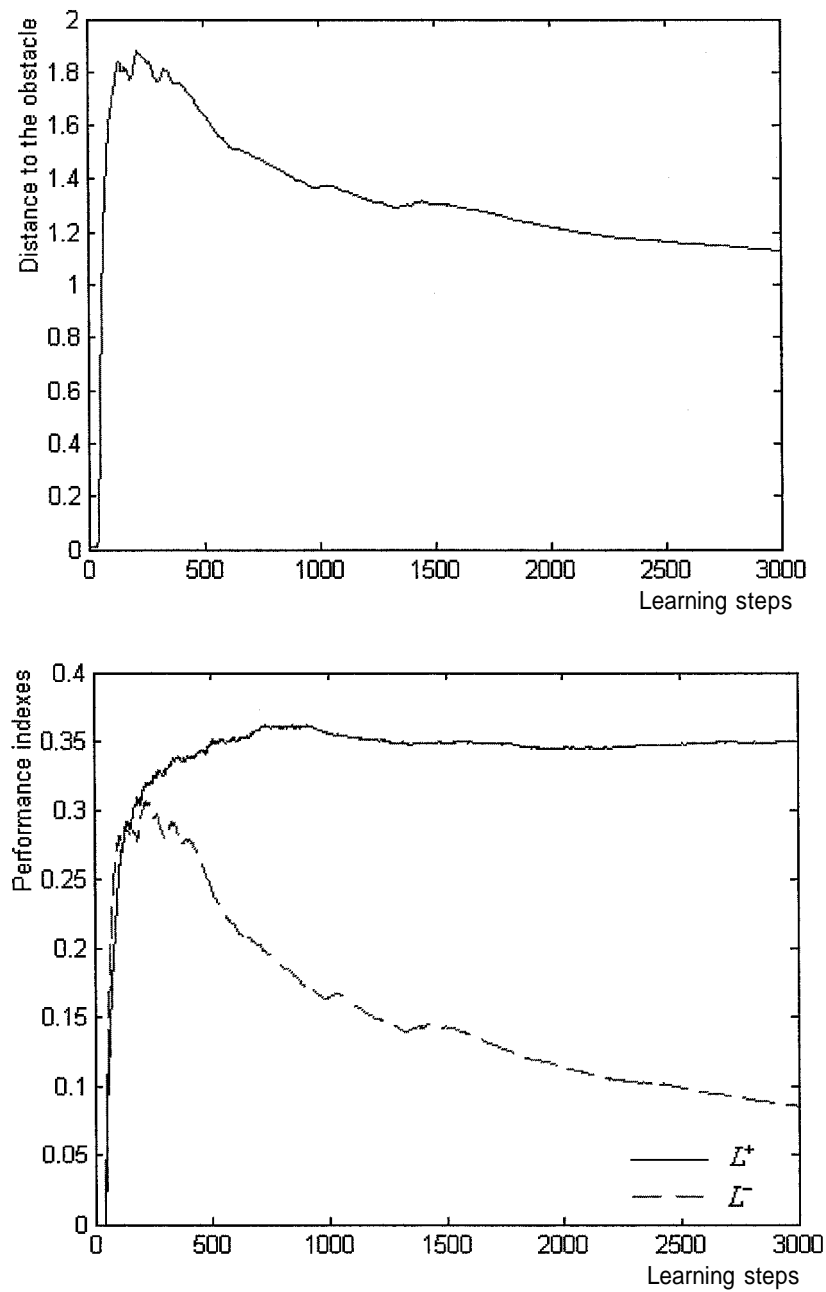


Figure 4.9 Distance to the obstacles and local performance indices during learning with DFQL

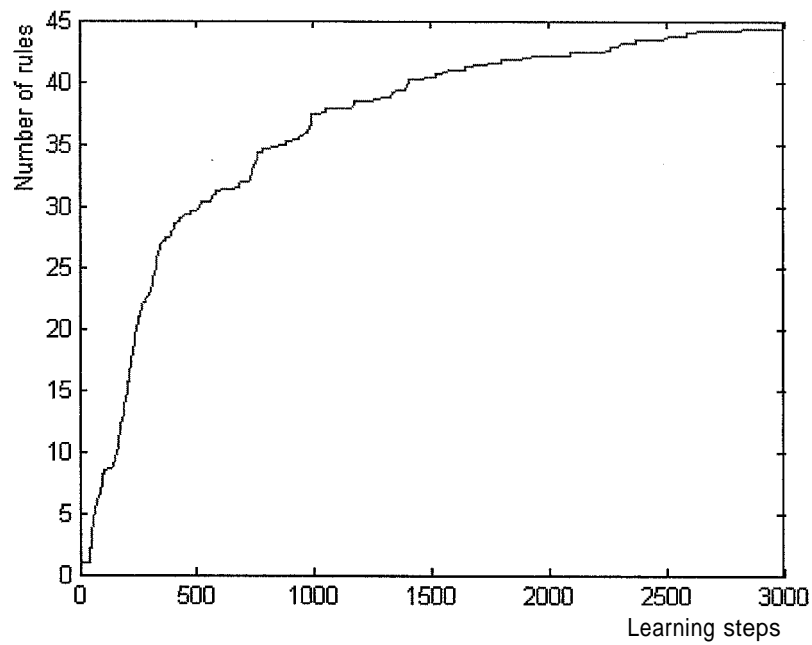


Figure 4.10 Number of fuzzy rules generated by DFQL during learning

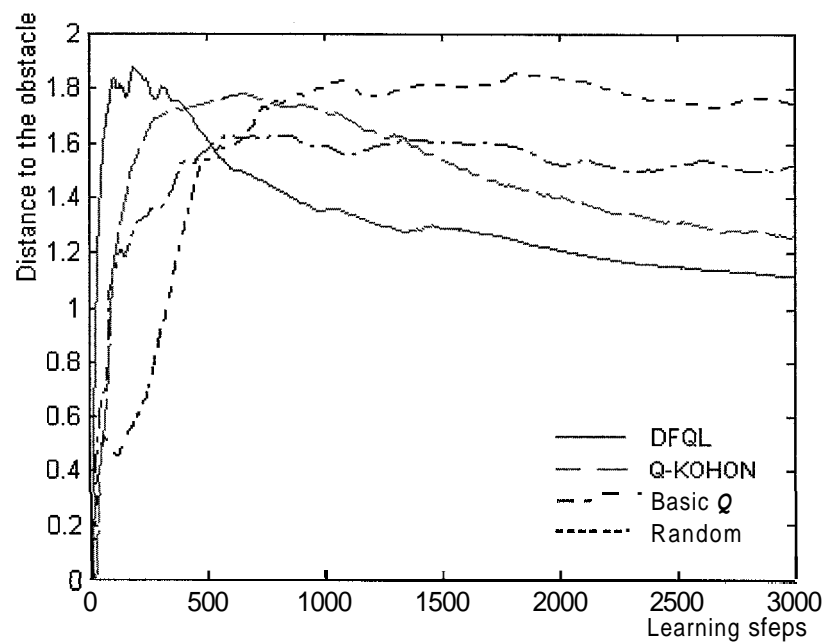


Figure 4.11 Distance to the obstacles during learning with (a) the random policy, (b) Basic Q-learning, (c) Q-KOHON, (d) DFQL

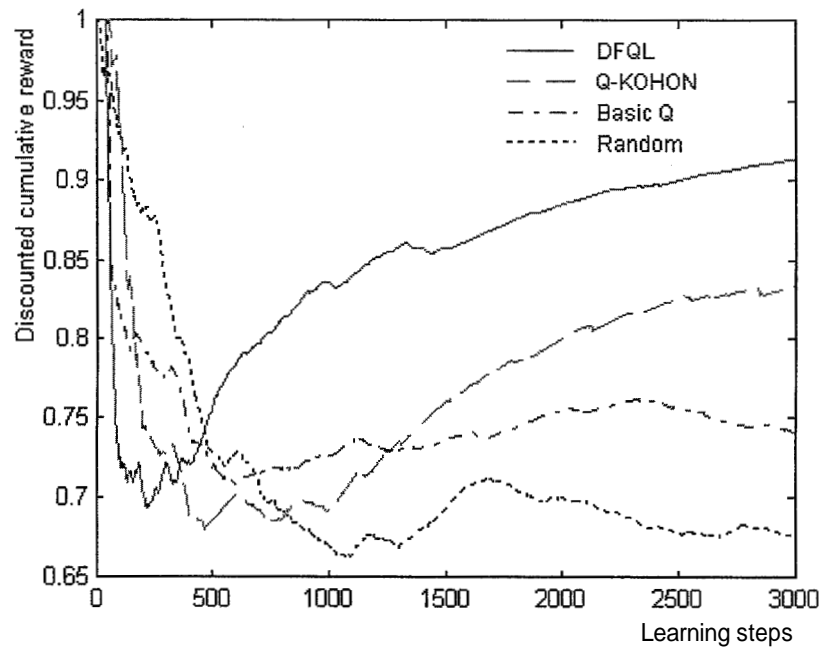


Figure 4.12 Discounted cumulative rewards obtained during learning with (a) the random policy, (b) Basic Q-learning, (c) Q-KOHON, (d) DFQL

Chapter 5

Embedding Initial Knowledge in DFQL

In Chapter 4, we describe the DFQL, an algorithm for generating a fuzzy system based on Q-learning. The key idea of the DFQL algorithm is that the system can start with no fuzzy rules and fuzzy rules can be recruited automatically according to the system performance. However, if we can incorporate initial knowledge to the learning system, especially in the early stages, we can greatly increase the speed of learning. In this chapter, we introduce a natural framework of incorporating initial knowledge by fuzzy rules. Subsequently, the wall-following behavior of the Khepera robot is investigated in experiments. A comparative study of the Jouffe's fuzzy Q-learning [48], Millan's continuous-action Q-learning [87] and our approach is carried out. All these methods can handle continuous states and actions and incorporate initial knowledge for rapid learning.

5.1 Efficient Use of Initial Knowledge

Q-learning is capable of learning the optimal value function with no prior knowledge of the problem domain, given sufficient experience of the world.

However, it is difficult for a robot to gain "sufficient" knowledge in reality. We are fundamentally limited in the amount of data we can generate, especially if we want to perform learning on-line, particularly when we are dealing with real-world problems. Moreover, we are fundamentally limited by the number of training runs that we can realistically hope to perform on a robot.

If we start with no knowledge of the world, we are essentially forced to act arbitrarily. If we are taking arbitrary actions, this amounts to a random walk through the state space, which is unlikely to reach the goal state in a reasonable time. If we can bias the learning system, especially in the early stages of learning, so that it is more likely to find the "interesting" parts of the state space, we can greatly increase the speed of learning.

5.1.1 Build-in Bias

Because it is very difficult to solve arbitrary problems in the general case, we must use generalization and begin to incorporate bias that will leverage on the learning process. One problem that prevents learners from learning anything is that they have a hard time even finding the interesting parts of the space. As a result, they wander around at random never even getting near the goal. Millan [85-87] explores the use of reflexes to make robot learning safer and more efficient. The reflexes correspond to domain knowledge about the task and allow the incorporation of bias into the system. Bias suggests actions for situations that otherwise would be time consuming to learn. These biases can eventually be overridden by more detailed and accurate learned knowledge. There are two ways in which the learner can efficiently learn from bias.

- Safer learning

Bias makes the learner operational from the very beginning and increments the safety of the learning process. The design problem initially is to provide the learner with basic behavior, which ensure its immediate safety. Once these basic aims have been achieved, more sophisticated skills can be added.

- Guessing where to search

Bias accelerates the learning process since it focuses the search process on promising parts of the action space immediately. The bias initializes the system in such a way that greedy policies are immediately operational even if far from optimal. The learner need only explore actions around the best ones currently known.

5.1.2 Initial Knowledge from Fuzzy Rules

Fuzzy rules provide a natural framework of incorporating the bias components for RL [39]. On one hand, fuzzy systems are multi-input-single-output mappings from a real-valued vector to a real-valued scalar and precise mathematical formulas of these mappings can be obtained. On the other hand, fuzzy systems are knowledge-based systems constructed from human knowledge in the form of fuzzy IF-THEN rules. An important contribution of fuzzy systems theory is that it provides a systematic procedure for transforming from a knowledge base to some nonlinear mapping. We use knowledge-based linguistic information to construct an initial fuzzy system, and then adjust the parameters of the initial fuzzy logic system based on numerical information.

The if-then fuzzy rules corresponding to the domain knowledge pertaining to the tasks can be incorporated into the DFQL design. The premise of rules can be used to generate EBF units over the fuzzy input space and the consequents of rules can be used to generate the initial Q-values, which are called bias. Bias suggests actions for situations that otherwise would be time consuming to learn. Thus, bias accelerates the learning process since it focuses on the search process on promising parts of the action space immediately. The parameter vector of a rule, q associated with discrete actions is initialized so that a greedy policy would select the action a suggested by this rule. Similar to [87], the main idea of the method employed in this chapter is that the Q-value of the selected discrete action a is initialized to a fixed value k_q , while all other values are given random values according to a uniform distribution in $[0, k_q/2]$.

The DFQL is an automatic method capable of self-tuning FIS, i.e., generating fuzzy rules according to system performance. The generated basic rules initialized from prior knowledge keep the learner safe and direct it in the right direction during the early stages of learning. The basic fuzzy rules, deduced from a human driver's intuitive experience, can yield an action that is feasible but far from optimal. It is almost impossible or difficult to find optimal fuzzy rules through a trial-and-error approach where a great number of variables are involved in the control task. In view of this, RL is added to tune the fuzzy rules online, which are eventually overwritten and improved by more accurate learned actions. Because the basic fuzzy rules are used as starting points, it is possible to determine optimal parameters without too many iterations and the robot can be operated safely even during learning.

5.2 Experiments

5.2.1 Wall Following Task for Khepera Robot

In this section, the DFQL approach has been applied to the Khepera robot for the wall following task. The aim of the experiment is to design a simple controller for wall following. In order to simplify the problem, we only consider robots moving in clockwise direction at a constant speed. Thus, we only need to deal with four input variables, which are the values of sensor S_i ($i = 0, \dots, 3$). All these sensor values can be normalized within the interval $[0, 1]$. The output of the controller is the steering angle of the robot. In order for the robot to follow a wall, it must move in a straight line as much as possible while staying between a maximum distance, d_+ , and a minimum distance, d_- , from that wall. The value of sensor S_0 , d can be regarded as the distance to the wall being followed. The robot receives a reward after performing every action U . The reward function depends on this action and the next situation:

$$r = \begin{cases} 0.1, & \text{if } (d_- < d < d_+) \text{ and } (U \in [-8^\circ, +8^\circ]) \\ -3.0, & \text{if } (d \leq d_-) \text{ or } (d_+ \leq d) \\ 0.0, & \text{otherwise} \end{cases} \quad (5.1)$$

If an action brings the robot outside the range of $[d_-, d_+]$, the robot will stop, move back inside the region, and receive a punishment. In these experiments, $d_+ = 0.85$ and $d_- = 0.15$.

The training environment with lighted shaped walls used for a real robot and simulation studies are shown in Figure 5.1 and Figure 5.2 respectively. The performance of the different approaches is evaluated at every episode of 1000

control steps according to two criteria, namely *failures* which correspond to the total number of steps the robot has left the "lane" and *reward* which is accumulated. In order to compare the different approaches systematically and find appropriate parameters, we implement these methods on both simulation and the real robot. The experimental results described as follows are based on the real robot.

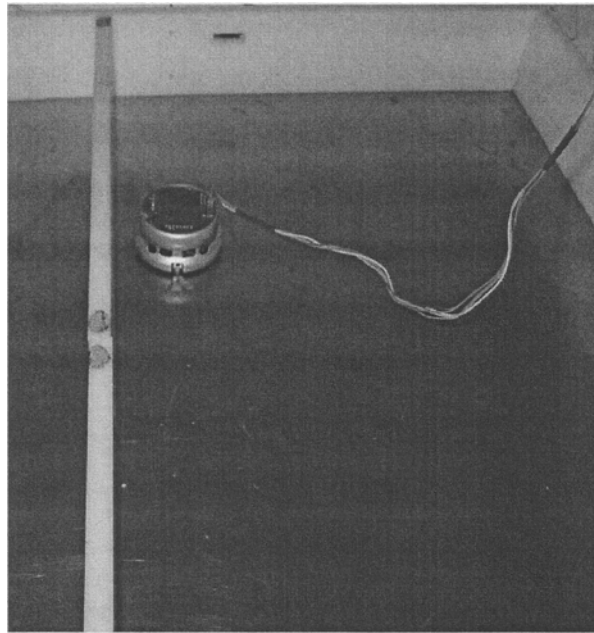


Figure 5.1 Actual environment for wall-following experiments

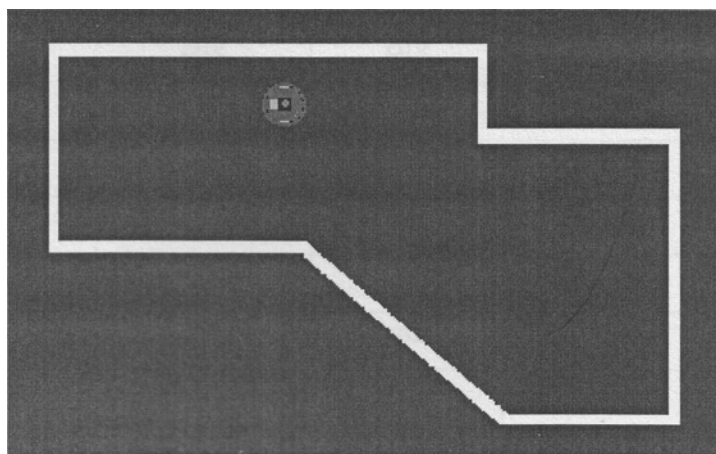


Figure 5.2 Simulation environment for wall-following experiments

5.2.2 Basic Fuzzy Controller

First, we design the fuzzy controller based on intuitive experiences. For each input linguistic variable, we define two linguistic values: Small and Big, whose membership functions cover the region of the input space evenly with the value of ε -Completeness set to 0.5. This means that there are 16 (2^4) fuzzy rules. Through trial and error, we can obtain the 16 fuzzy rules as a special case of the TSK fuzzy controller, whose consequents are constant, as follows:

Table 5.1 Basic fuzzy control rules for wall following

Rule	S_0	S_1	S_2	S_3	Steering angle
1	Small	Small	Small	Small	30
2	Small	Small	Small	Big	30
3	Small	Small	Big	Small	30
4	Small	Small	Big	Big	15
5	Small	Big	Small	Small	30
6	Small	Big	Small	Big	15
7	Small	Big	Big	Small	15
8	Small	Big	Big	Big	15
9	Big	Small	Small	Small	30
10	Big	Small	Small	Big	15
11	Big	Small	Big	Small	0
12	Big	Small	Big	Big	0
13	Big	Big	Small	Small	30
14	Big	Big	Small	Big	-15
15	Big	Big	Big	Small	-15
16	Big	Big	Big	Big	-15

If the robot only uses the basic fuzzy controller, it can actually follow the wall, but along inefficient trajectories. When only the basic fuzzy controller is used, the robot encounters 63 failures, and -130.7 of reward per episode on average. Certainly, we can provide finer partitioning of the input space, or tune the parameters of the membership functions and consequents so as to obtain better performances. However, the number of rules will increase exponentially with increase in the number of input variables. Furthermore, tuning consequents of rules is time consuming because of the risk of creating conflicts among the rules. It is almost impossible or impractical to design an optimal fuzzy controller by hand due to a great number of variables involved. Similar to the idea of [87], we incorporate RL into fuzzy controller design and the basic fuzzy rules designed from intuitive experiences are used as a starting point for learning. As a consequence, it overcomes some limitations of basic RL where an extremely long learning time is needed and unaccepted behavior may be generated during learning.

5.2.3 Fuzzy Q-Learning with a Fixed Structure

Next, we consider the FQL approach of [48] that has a fixed structure of fuzzy rule sets. A total of 16 fuzzy rules, same as the basic fuzzy controller, are used. However, the consequents of the rules can be adjusted based on the fuzzy Q-learning. Here, we simply use the undirected exploration method employed in [48] to select a local action a from possible discrete action vector A , as follows:

$$a_{\pi} = \pi_{a \in A}(q(S, a)) = \arg \max_{a \in A} (q(S, a) + \eta(S, a)) \quad (5.2)$$

The term of exploration η stems from a vector of random values, ψ (exponential distribution) scaled up or down to take into account the range of q values as follows:

$$s_f = \begin{cases} 1, & \text{if } \max_a(q(S,a)) = \min_a(q(S,a)) \\ s_p \frac{(\max_a(q(S,a)) - \min_a(q(S,a)))}{\max(\psi)} & \text{otherwise} \end{cases} \quad (5.3)$$

$$\eta = s_f \psi$$

where s_p is the noise size, with respect to the range of qualities, and s_f is the corresponding scaling factor. Decreasing the s_p factor implies reducing the exploration. We choose an exploration rate s_p of 0.001 in the experiments.

The set of discrete actions is given by $A = [-30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30]$. The initial q-value, $k_q = 3.0$ is chosen according to the method described in Section 5.1. The other parameters in the learning algorithm are: Discounted factor, $\gamma = 0.95$; Trace-decay factor, $\lambda = 0.7$; TD learning rate, $\alpha = 0.05$. The controller with 81 (3^4) fuzzy rules whose membership functions satisfy the 0.5 ε -completeness is also considered. Average performances of the two controllers during 40 episodes over 10 runs are shown in Figure 5.3

At the very beginning, performances of the two controllers based on the FQL are worse than that of the basic fuzzy controller due to the exploration feature of RL. The robot has to explore different actions in order to ensure that better actions are selected in the future. However, the performance of the robot is improved gradually and is much better than that of the basic fuzzy controller. To assess the effectiveness of finer partitioning of the input space, we compare the performances of the FQL using 16 rules and 81 rules. The speed of learning 81 rules is slower than that of 16 rules because a lot more parameters need to be explored. However, asymptotic performances of these two methods are almost the same. It is impractical to partition the input space further due to the curse of dimensionality.

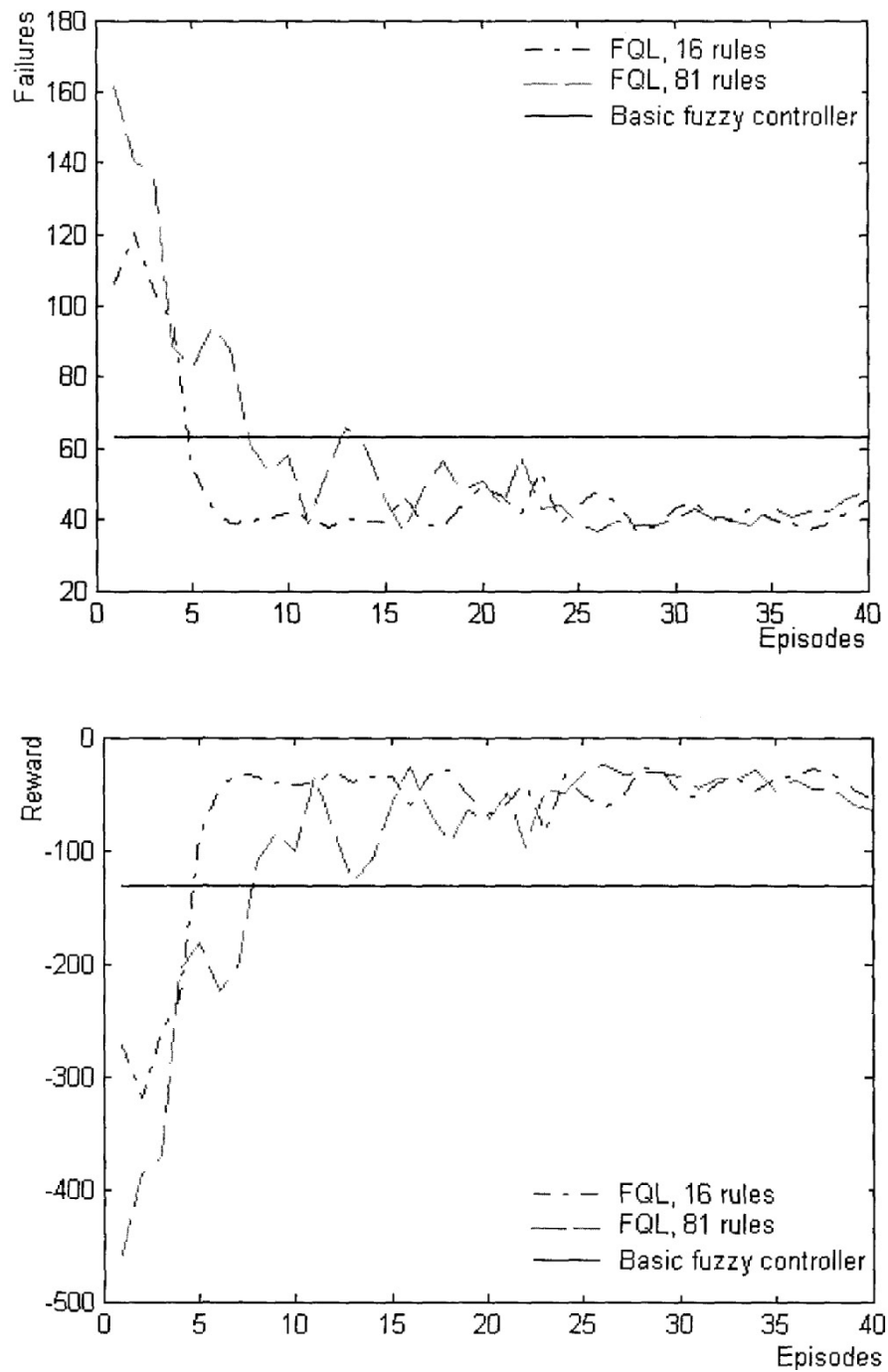


Figure 5.3 Comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL, (b) 81 fuzzy rules based on FQL, (c) Basic fuzzy controller with 16 fuzzy rules

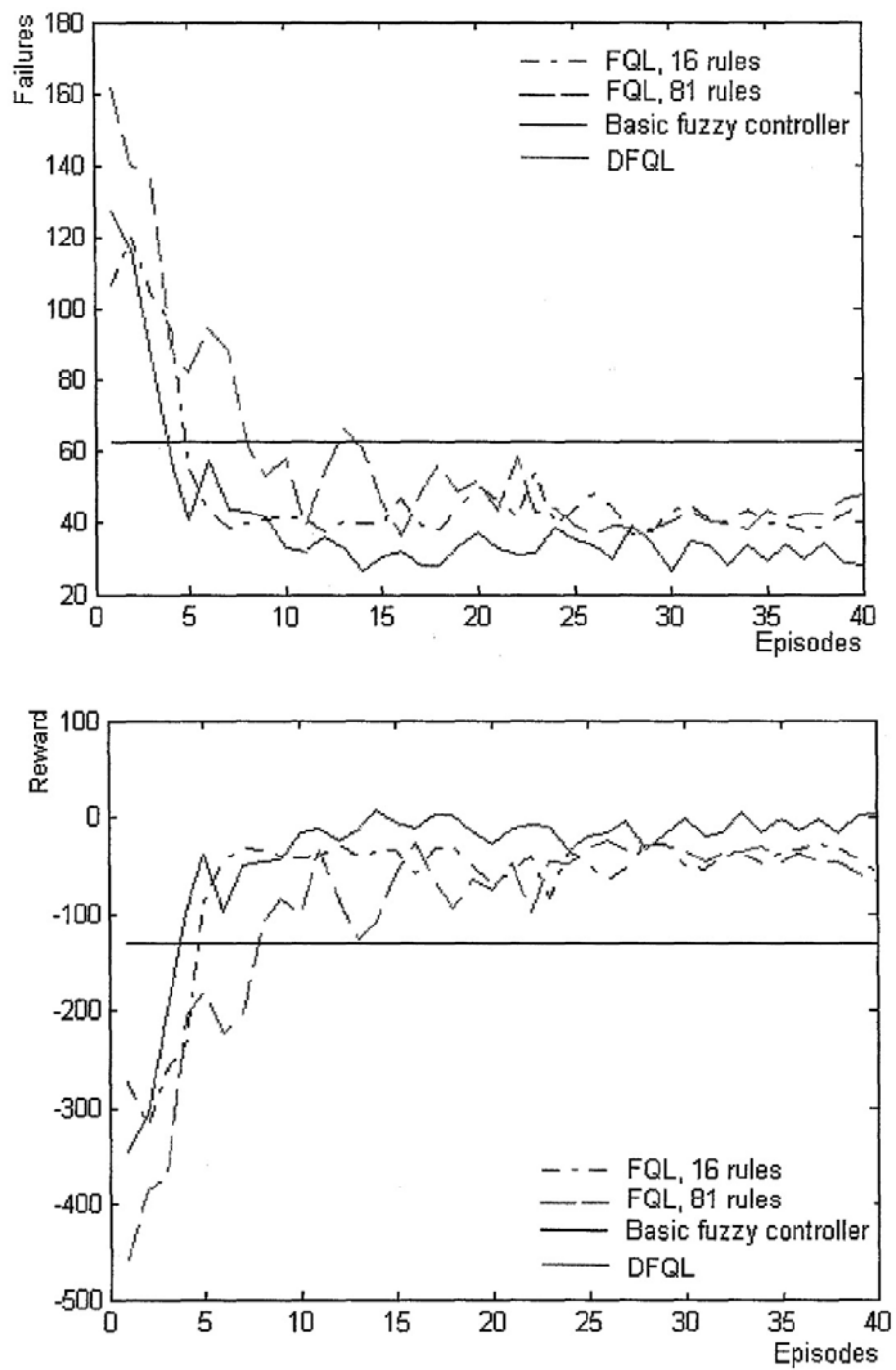


Figure 5.4 Comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL, (b) 81 fuzzy rules based on FQL, (c) Basic fuzzy controller with 16 fuzzy rules, (d) Fuzzy controller based on DFQL

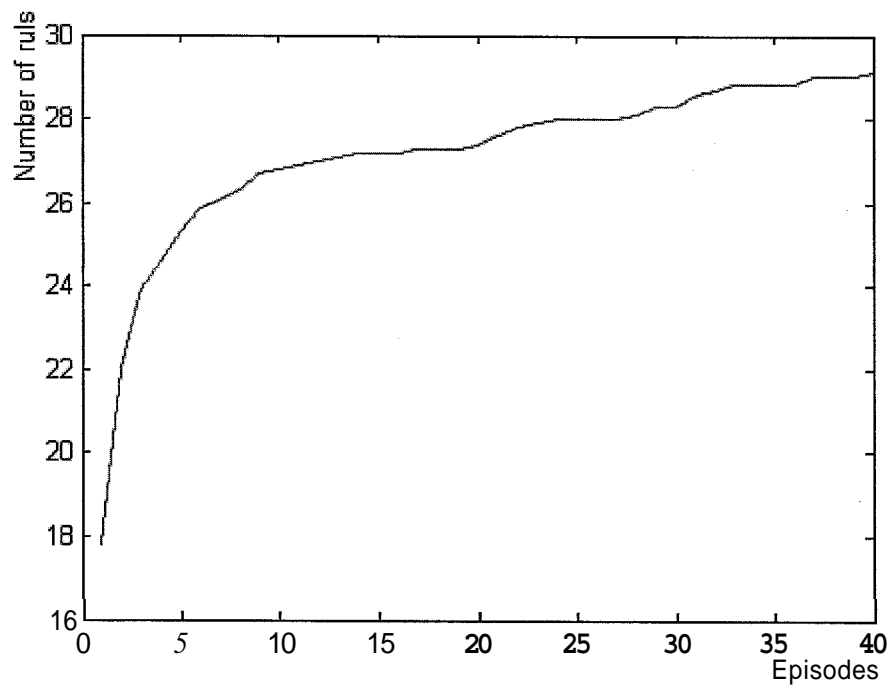


Figure 5.5 Number of fuzzy rules generated by DFQL during learning

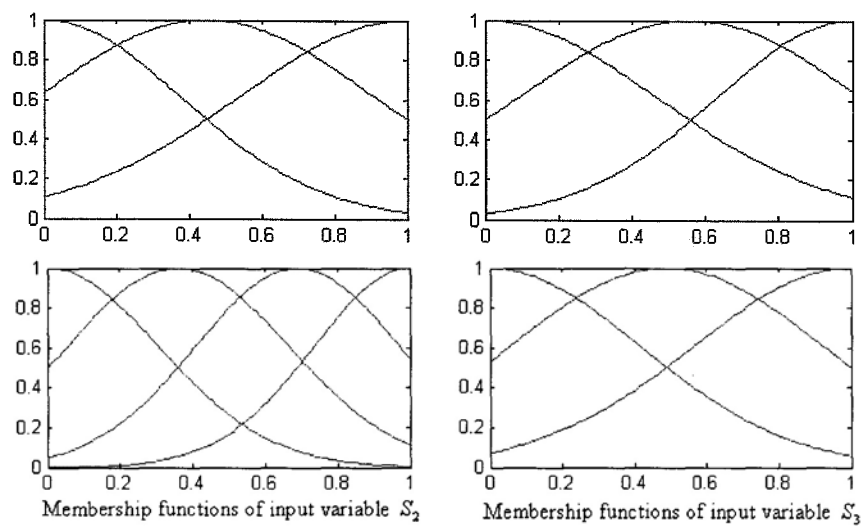


Figure 5.6 Membership functions after learning at one run

5.2.4 DFQL

Now, we assess the performance of DFQL approach. The parameter values are the same as those used in the FQL approach. The other learning parameters for rule generation are: ε -completeness, $\varepsilon = 0.5$; similarity of membership function, $k_{mf} = 0.3$; TD error factor, $K = 50$ and TD error criterion, $k_e = 1$. These values give good performances of the algorithms in an initial phase. However, it should be pointed out that we have not searched the parameter space exhaustively. The performances of the DFQL approach shown in Figure 5.4 are also the mean values during 40 episodes over 10 runs. As expected, the DFQL performs better than the FQL with respect to both failures and reward. In fact, the DFQL outperforms the FQL during the major portion of episodes and asymptotic performance of DFQL is about 30% better than that of FQL according to the performance of the basic fuzzy controller. The number of fuzzy rules generated at every episode is shown in Figure 5.5. The membership functions produced by the DFQL after learning input variables at one run are shown in Figure 5.6. The number of rules can be generated automatically online and does not increase exponentially with increase in the number of input variables. Thus, a compact and excellent fuzzy controller can be obtained online. The reason why the DFQL method outperforms the FQL method is that the DFQL approach is capable of online self-organizing learning. Input-output space partitioning is one of the key issues in fuzzy systems because it determines the structure of a fuzzy system. The common approach of conventional input-space partitioning is the so-called grid-type partitioning. The FQL with 16 rules partition the state space coarsely, on the other hand, the speed of learning 81 rules is slow because a lot more parameters need to be explored. The proposed DFQL need not partition the input space *a priori* and is suitable for RL. It partitions the input space online dynamically according to both the accommodation boundary and the performance of RL. The compact fuzzy system considers

sufficient rules in the critical state space which requires high resolution and does not include the redundant rules in the unimportant or unvisited state space so that the learning is rapid and optimal.

5.2.5 Adaptation in a New Environment

We test the performance of the learned DFQL navigation strategies in an environment different from that used for training. This new environment is similar to the one before, except that it contains some new obstacles depicted in Figure 5.7

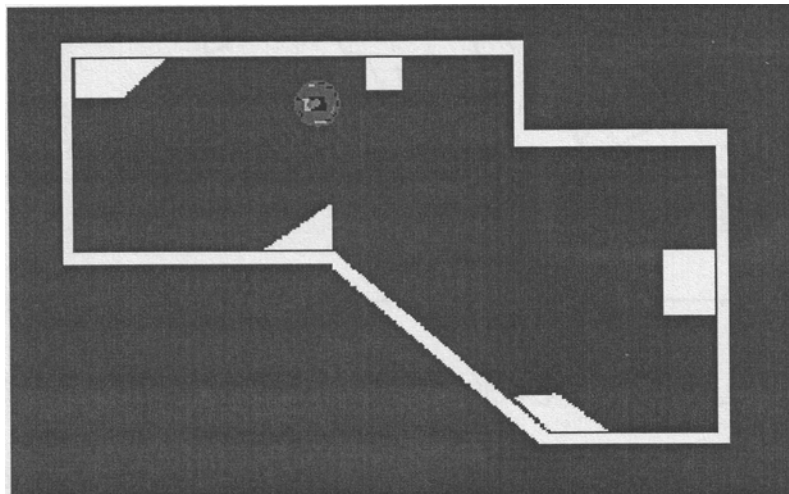
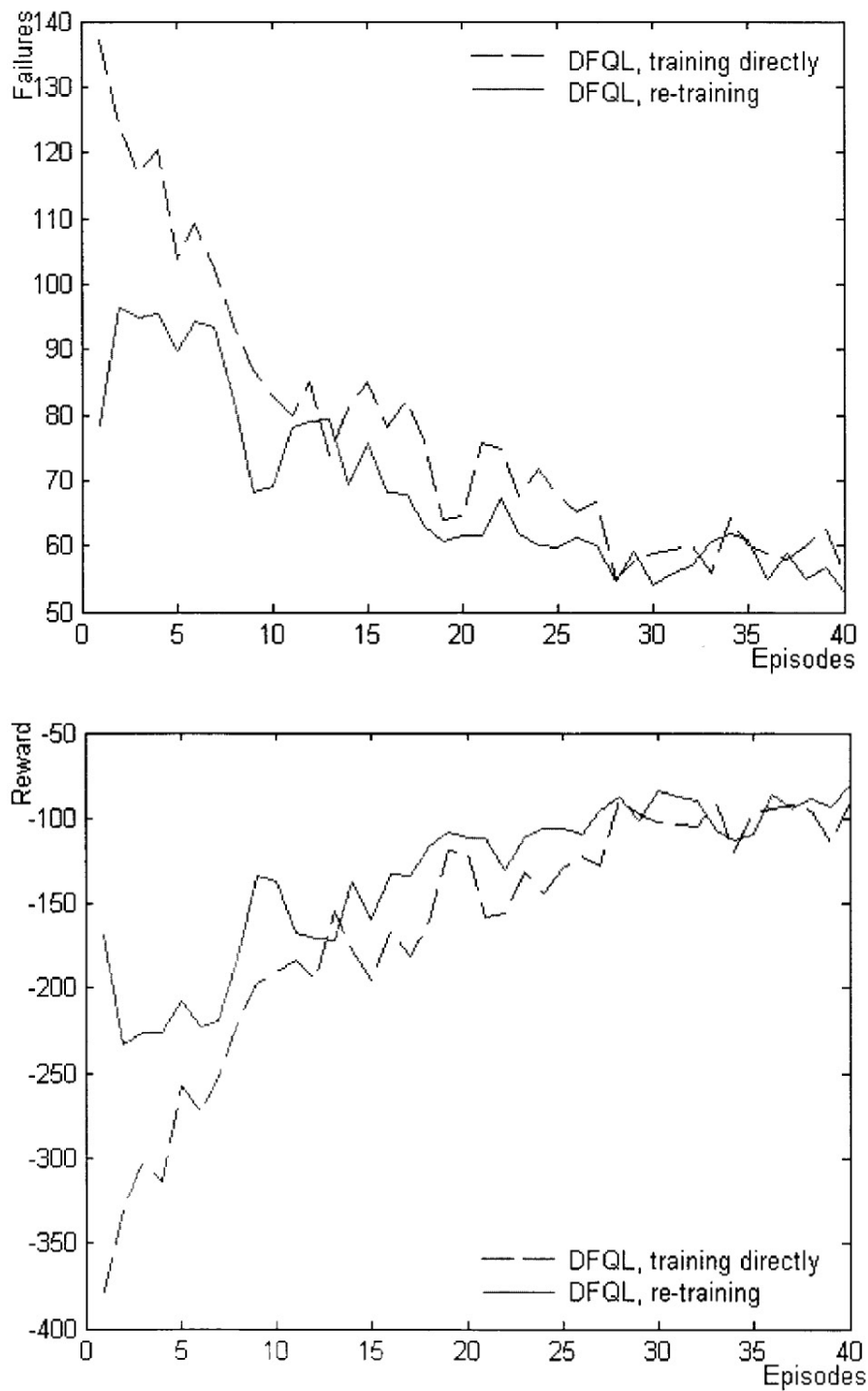


Figure 5.7 The new training environment with obstacles

Figure 5.8 compares the performances (mean values over 20 runs) of the robot during training directly and re-training in the new environment. During re-training, the robot is first trained in the original simple environment and then re-trained in the new environment so as to improve the obtained navigation strategies. In this period, the robot only has a few more fuzzy rules to deal with in the new regions of sensory space, and is able to adapt its previous knowledge to the new environment quickly. It should be highlighted that no more membership functions need to be

generated. As a result, the learning speed of re-training is faster than that of training directly in the new environment.



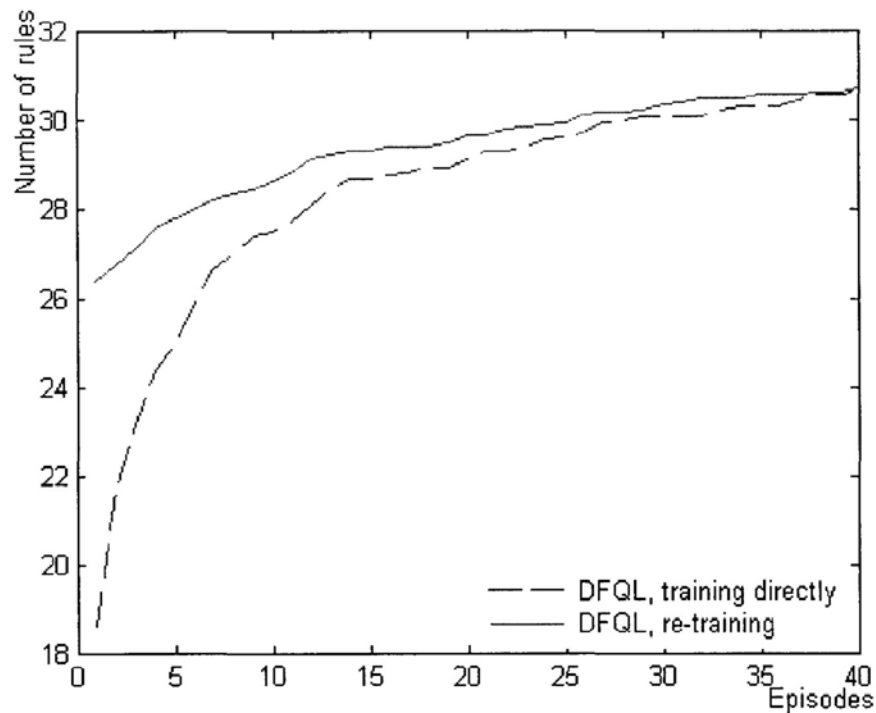


Figure 5.8 Performance comparison of DFQL with training directly and retraining in a new environment

5.2.6 Continuous-Action Q-Learning

The algorithm described in this thesis shows some resemblances with other related works. In particular, the Continuous-Action Q-Learning approach, which is the only approach restricted to the generation of continuous actions by means of Q-learning is proposed in [87]. Bias represents domain knowledge in the form of built-in reflexes, which make learning process rapid and safe. Our idea of incorporating basic fuzzy rules is adopted from this idea. However, our approach differs from it in several aspects. First, fuzzy rules are considered in the DFQL. Second, the DFQL develops fuzzy systems with ellipsoidal regions of rules instead of radial regions. Third, the criteria for rule generation are different. Our criteria

are not only based on the accommodation boundary but also the system performance based on TD errors. On top of this, the most important difference is the updating algorithm of Q-values. In Continuous-Action Q-Learning, only the nearest unit is used to select the action. The resulting continuous action is an average of the discrete actions of the nearest units weighted by their Q-values. On the other hand, in our approach, one discrete action is selected from every fuzzy rule. The resulting continuous action is an average of the actions weighted by the firing strengths of fuzzy rules. Both methods update the Q-values of the actions according to their contributions. For Continuous-Action Q-Learning, if the number of discrete actions is large, more neighboring discrete actions need to be considered. Otherwise, the continuous action is not explored sufficiently. The discrete actions whose Q-values are not good will degrade the continuous action. In our approach, one discrete action is selected according to the exploration-exploitation strategy for every fuzzy rule. It is more efficient to use the firing strengths of fuzzy rules as weights. In order to compare these two algorithms, we only consider the difference of updating Q-values and assume that the others are the same. For Continuous-Action Q-Learning, the exploration rate is $S_p = 0$, and we consider one action to each side of the optimal action, a , according to the following rules:

$$\begin{aligned}
 left &= \frac{1}{3 + (q(S_i, a_m) - q(S_i, a_{m-1}))^2} \\
 right &= \frac{1}{3 + (q(S_i, a_m) - q(S_i, a_{m+1}))^2} \\
 U &= a_m + right * (a_{m+1} - a_m) + left * (a_{m-1} - a_m) \\
 Q(U) &= \frac{q(S_i, a_m) + right * q(S_i, a_{m+1}) + left * q(S_i, a_{m-1})}{1 + right + left}
 \end{aligned} \tag{5.4}$$

For the DFQL, the exploration rate, S_p is set to 0.001 and the global action and its Q-value are given by Eqs. (4.9) and (4.10) respectively. The results (mean values over 10 runs) are shown in Figure 5.9. At the very beginning, the performance of Continuous-Action Q-Learning is better than that of DFQL because it always explores near optimal initial values. However, the performance is worse than that of DFQL later since the learner may get trapped to locally optimal actions. Discrete actions whose Q-values are not good will degrade the continuous action. Of course, we can combine the two methods together. But, it requires more computational time and the learning results are not better than the DFQL significantly.

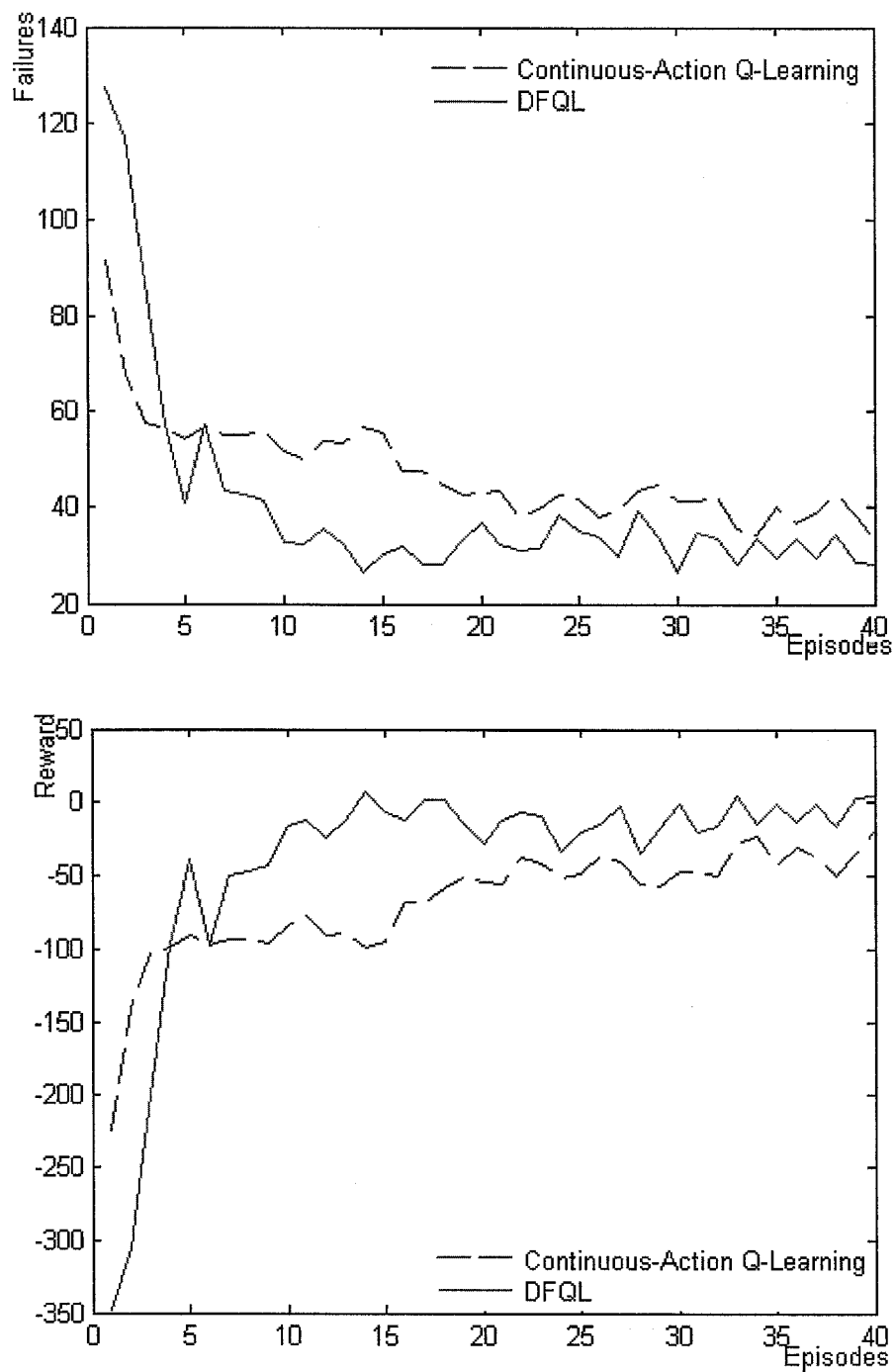


Figure 5.9 Performance comparison of updating Q-values for Continuous-Action Q-Learning and DFQL

Chapter 6

General DFQL with Eligibility Traces

In Chapter 4 and 5, we introduce the DFQL algorithm and a method of embedding initial knowledge in DFQL to speed up learning. In this chapter, we study the effects that combine DFQL with eligibility traces. We extend the learning algorithm to the general version with an eligibility mechanism, leading to faster learning and alleviating the experimentation-sensitive problem. Subsequently, simulation studies of the general DFQL on the optimum path experiments demonstrate the efficiency of the method for learning an appropriate policy.

6.1 General DFQL

6.1.1 Eligibility Traces

Eligibility traces are one of the basic mechanisms of RL. For example, in the popular $TD(\lambda)$ algorithm, the term λ refers to the use of an eligibility trace. Almost any TD method can be combined with eligibility traces to obtain a more general method that may learn more efficiently. An eligibility trace is a temporary record of the occurrence of an event, such as visiting a state or taking an action.

The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. When a TD error occurs, only the eligible states or actions are assigned credit or blamed for an error. Thus, eligibility traces help bridge the gap between events and training information. Eligibility traces are a basic mechanism for temporal credit assignment.

There are many RL algorithms which employ the eligibility trace mechanism. $TD(\lambda)$ is introduced by Sutton [123], and an alternative version known as a replacing trace is proposed by Singh and Sutton [117]. The TD algorithm has been shown to be convergent by several researchers [32,102,139]. The SARSA algorithm is due to Rummery [110,127], and can also be formulated with an eligibility mechanism $Sarsa(\lambda)$. Q-learning and the eligibility trace method $Q(\lambda)$ are proposed by Watkins [145]. Peng's $Q(\lambda)$ [103] can be thought of as a hybrid arrangement of $Sarsa(\lambda)$ and Watkins's $Q(\lambda)$ and it performs significantly better than Watkins's $Q(\lambda)$ and almost as well as $Sarsa(\lambda)$ although it has not been proven to be convergent in the general case.

All the algorithms presented so far have all been shown to be effective in solving a variety of RL tasks. The $TD(\lambda)$ and $Sarsa(\lambda)$ algorithms are known as on-policy algorithms. The value function that they learn is dependent on the policy that is being followed during learning. Using an on-policy algorithm with an arbitrarily bad training policy might result in non-optimal policy. Eligibility traces used in Watkins's $Q(\lambda)$ are set to zero whenever an exploratory action is taken. Thus, learning may be a little faster than one-step Q-learning in the early stage. On the other hand, the features of Peng's $Q(\lambda)$ make it much more appealing for our purposes, though it cannot be implemented as simply as others.

The main advantage of Peng's $Q(\lambda)$ over other methods with eligibility traces is that it is less experimentation-sensitive and it is able to learn without necessarily following the current policy. This capability makes it much more appealing for the purpose of efficient implementation of RL in real-time applications. We might not know a good policy for the task that we are attempting to learn. Using an experimentation-sensitive algorithm with an arbitrarily bad training policy might result in non-optimal policy. Using an experimentation-insensitive algorithm allows us to alleviate this problem. Another advantage of Peng's $Q(\lambda)$ is that it performs well empirically. Most studies have shown that it performs significantly better than Watkins's $Q(\lambda)$ and almost as well as *Sarsa*(λ).

6.1.2 The General DFQL Learning Algorithm

We extend the DFQL learning algorithm to the general version with an eligibility mechanism based on Peng's $Q(\lambda)$. Under the same assumptions stated in Chapter 4, the one-time-step global working procedure of general learning algorithm is proposed as follows: Let $t+1$ be the current time step and assume that the learner has performed the action U_t and has received a reinforcement signal r_{t+1} .

- a. Check the fuzzy rule generation criteria according to the current state X_{t+1} .
If a new fuzzy rule needs to be generated, tune the structure of the FIS and initialize the parameter vector q of the new rule based on prior knowledge. The initial values of eligibility traces of fuzzy state action pairs are set to zero;
- b. Approximate the optimal evaluation function corresponding to the current state and FIS by using the optimal local action quality i.e. $V_t(X_{t+1})$;

- c. Compute $\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma V_t(X_{t+1}) - Q_t(X_t, U_t)$;
- d. Compute $\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma V_t(X_{t+1}) - V_t(X_t)$;
- e. For all fuzzy state action pairs, update the eligibility traces $Tr(S, a) = \gamma \lambda Tr(S, a)$, where the eligibility rate λ is used to weight time steps and γ is the discount factor for rewards. Next, update all the q values $q_{t+1}(S, a) = q_t(S, a) + \alpha \tilde{\varepsilon}_{t+1} Tr(S, a)$, where α is the learning rate;
- f. Update the q values of "active" fuzzy state action pairs at time step t according to $q_{t+1}(S_i, a_t^i) = q_t(S_i, a_t^i) + \alpha \tilde{\varepsilon}_{t+1} \phi_t^i$, where a_t^i is the selected local action of rule R_i at time step t and ϕ_t^i is the normalized firing strength of rule R_i at time step t . Next, update the eligibility trace of "active" fuzzy state action pairs at time step t according to $Tr(S_i, a_t^i) = Tr(S_i, a_t^i) + \phi_t^i$. Note that eligibility trace values need to be reset to zeros at the end of each episode;
- g. Elect local actions based on the new vector q_{t+1} and compute the global action $U_{t+1}(X_{t+1})$;
- h. Estimate the new evaluation function $Q_{t+1}(X_{t+1}, U_{t+1})$ for the current state and the actions effectively elected and $V_{t+1}(X_{t+1})$ for the current state and the optimal action. Note that $Q_{t+1}(X_{t+1}, U_{t+1})$ and $V_{t+1}(X_{t+1})$ will be used for error computation at the next time step.

NOTE: In the basic DFQL, the TD error is calculated based on the action taken on the step. The traces and q values are all updated corresponding to this TD error, which leads to learning necessarily following the current action. An arbitrarily bad policy might not be exploited for the learning progress.

The main improvement of the general DFQL is that the mixture of updating mechanism is used, which is derived from the unique feature of Peng's $Q(\lambda)$. Unlike the basic DFQL, two TD errors are considered in the general DFQL, respectively

$$\tilde{\varepsilon}_{t+1}' = r_{t+1} + \gamma V_t(X_{t+1}) - Q_t(X_t, U_t)$$

which is based on the Q-value of the actual action on the step and

$$\tilde{\varepsilon}_{t+1} = r_{t+1} + \gamma V_t(X_{t+1}) - V_t(X_t)$$

which is based on the Q-value of the optimal action on the step. All q values associated with all the fuzzy state-action pairs are updated according to the eligibility traces from the TD error $\tilde{\varepsilon}_{t+1}$, i.e.

$$q_{t+1}(S, a) = q_t(S, a) + \alpha \tilde{\varepsilon}_{t+1} Tr(S, a)$$

Next, the q values of "active" fuzzy state-action pairs on the current step are updated from the TD error $\tilde{\varepsilon}_{t+1}'$, i.e.

$$q_{t+1}(S_i, a_t^i) = q_{t+1}(S_i, a_t^i) + \alpha \tilde{\varepsilon}_{t+1}' \phi_t^i$$

The advantage of considering eligibility traces for all state-action pairs leads to faster learning without necessarily following the current action because not only the q values corresponding to the current state-action pair but also the q values associated with all the fuzzy state-action pairs are updated according to the respective TD errors. Any action can be carried out at any time and knowledge is gained from this experience. On the other hand, the general DFQL cannot be implemented as simply as the basic DFQL.

6.2 Experiments

6.2.1 Optimum-Path Task

In this section, we describe the optimum-path experiment performed on the Khepera mobile robot. The task is to take the robot from a starting location to the goal location and attempt to optimize the path. We assume that the goal location is specified in relative Cartesian coordinates with respect to the starting location. The task faced by the robot is to build a self-adaptive controller that is capable of searching an optimal trajectory, which would lead to a minimum cost. We carry out the experiment in simulation environments which can provide the position information of the robot with respect to the starting location. We are not able to implement it in the real robot because the position information cannot be detected due to the hardware limitation. However, positions and orientations of a real robot can be detected in real time if additional devices are equipped, e.g. the laser device on the top and the additional turret of the robot used in [33] or grids lines on the floor and the additional detector of the robot used in [77].

The learning environment consists of an indoor space and a corridor. The task is to generate the shortest possible but safe trajectory from the interior of an office to a point at the end of the corridor, similar to the works of [39,40,86]. It is not easy to implement this seemingly simple task. Firstly, the task is performed using local sensory information. The robot has neither a global view of the environment nor a comprehensive world model. Secondly, the task is a high-dimensional continuous learning task and successful goal reaching requires a non-linear mapping from this space to the space of continuous real-valued actions.

The inputs to the controller are the normalized relative position and orientation of the robot from the goal, i.e., the robot's current position (x,y) and the heading angle θ . Furthermore, each signal is represented as a vector of three components using Millan's codification scheme [86] in order to offer greater robustness and generalization ability. The scheme involves three localized processing units, whose activation values depend on how far the normalized input value is from the respective center positions of the processing units. These units with overlapping localized receptive fields are evenly distributed over the interval of $[0,1]$, and the activation level of the unit located in the point p_i is

$$S_w(p_i, k) = \begin{cases} \frac{1}{w^2} [k - (p_i - w)][(p_i + w) - k] & \text{if } k \in [p_i - w, p_i + w] \\ 0 & \text{otherwise} \end{cases} \quad (6.1)$$

where w is the width of the receptive fields and k is the normalized input. The value of w is 0.45 and the units are located on the points 0.2, 0.5 and 0.8 in the experiments. As an example, the value of $k = 0.4$ is coarse coded into the pattern (0.8025, 0.9506, 0.2099). Thus, the inputs to the controller consist of a vector of nine continuous value components, all of real numbers which are in the interval of $[0,1]$.

The robot's angular rotation, which determines its next direction, is the only output of the controller. For every step, the robot first completely rotates based on the specified angle. After rotation has ceased, it will move to a new location by translating forward a fixed distance if the robot does not collide with obstacles. Whenever the robot detects a collision, an emergency behavior will stop the robot. Here, the emergency activation occurs with the reflectance value of any IR sensors greater than 1000.

The robot receives reinforcement signals when it approaches the goal location and avoids obstacles. It is important to note that the robot does not seek to optimize each immediate reinforcement signal, but to optimize the total amount of reinforcement obtained along the path to the goal. The reinforcement function is directly derived from the task definition, which is to reach the goal along trajectories that are sufficiently short and, at the same time, have a wide clearance to the obstacles. Thus, the reinforcement signal r has two components. The first component penalizes the robot whenever it collides with or approaches an obstacle. If the robot collides, it is penalized by a fixed value; otherwise, if the distance between the robot and obstacles is less than a certain threshold, $d_k=300$, the generated penalty increases as the distance between the robot and the obstacle decreases. The component of the reinforcement that teaches the robot to keep away from obstacles is:

$$r_1 = \begin{cases} -3 & \text{if collision} \\ -1 + (1023 - d_s)/1023 & \text{if } d_s > d_k \\ 0 & \text{otherwise} \end{cases} \quad (6.2)$$

where d_s is the shortest distance, i.e. the maximum reading value, provided by any IR sensors while performing the action. It should be pointed out that only virtual collision occurs, which makes the learning process safe. The other component teaches the robot how to approach the goal. The second component of the reinforcement function is proportional to the angle between the robot heading θ_h and the line connecting the goal and the robot location θ_g , which is given by

$$r_2 = \frac{-\text{abs}(\theta_h - \theta_g)}{180} \quad (6.3)$$

The total immediate reinforcement r is the sum of the two components, $r = r_1 + r_2$. This reinforcement function does not teach the robot directly how to reach the goal; it only trains the robot how to approach the goal without collisions.

6.2.2 Learning Details

The learner's aim is to learn to perform those actions that optimize the total reinforcement in the long term. That is, the learner has to learn a policy that maximizes the total reinforcement which is the sum of immediate reinforcements the learner receives till the robot reaches the goal, i.e.

$$V = \sum_{t=0}^T r(t) \quad (6.4)$$

where T is the total number of moves required to reach the goal. Here, the discount factor is $\gamma = 1$.

We seek to make RL effective for real robots and require that learning takes place online from a relative small amount of experiences. As we consider sophisticated tasks, it will be almost certain that the learner will not be effective in a reasonable amount of time. It either collides with obstacles that terminate the learning process or explores aimlessly without ever reaching the goal that can take unacceptable long time to discover interesting parts of the space. As mentioned in Chapter 5, a way of alleviating the problem of slow convergence of RL is to use bias from prior knowledge to figure out which part of the action space deserves attention in each situation. The architecture of the bias component is similar to [39] and shown in Figure 6.1. It consists of two fuzzy behavior, namely obstacle avoidance and goal following. The output of the total behavior is obtained by combining corresponding priority functions for each behavior. Since the bias is used to provide only an initial value, it suffices to consider a fixed blending scheme with constant desirability parameters $p_a = 0.9$ and $p_g = 0.1$, one for each behavior. The blender fuses the outputs of each behavior according to

$$u = p_a u_a + p_g u_g \quad (6.5)$$

where u_a and u_g are the outputs of obstacle avoidance and goal following respectively.

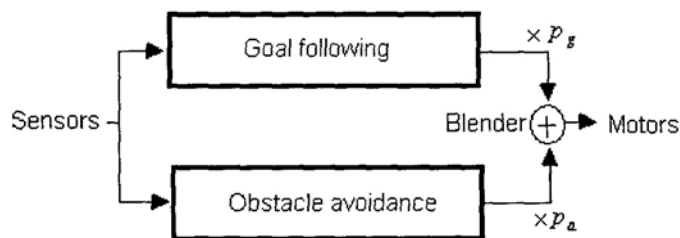


Figure 6.1 Architecture of bias component

Here we use the simple basic fuzzy controller for this specific task because it is sufficient to provide the starting points for learning. As we consider sophisticated environments, any efficient navigation strategy, e.g. those of [25,76], can be regarded as the basic fuzzy controller, as long as it provides at least one free way or path through which the robot can reach the goal without collisions. The basic fuzzy controller, though eventually overwritten and improved by more accurate learned actions through exploration, keep the robot safe and direct it in the right direction while it is trying to learn.

This control task takes place in multi-dimensional continuous state space and prefers continue actions. Millan's method [86] uses the unique feature of the Nodmad 200 robot: the turret motor. Since the turret motor orients the sensors independent of the robot heading, the robot can take similar actions for similar situations independently of its current direction of travel. However, for most robots, the state-space data generated would be different every time a robot visits the same location at different heading angles. Therefore, the entire state space is extremely huge but many states will never be visited. It is necessary to choose the way of using an online adaptive state construction algorithm instead of partitioning the state space evenly prior to learning. Hence, the fuzzy control rules are not

predefined, but are created dynamically when the robot explores its environment. As foreshadowed, a fuzzy rule is a local representation over a region defined in the input space. When a new fuzzy rule is generated, membership functions of the input variables are chosen in the form of Gaussian functions and the receptive fields of this model in the input space can be regarded as radial basis function (RBF) units. The strength of the activation value of the RBF basis function, i.e., the overall truth value of the premise of fuzzy rules is given by

$$f_i(x) = \exp(-\|x - c_i\|^2 / \sigma^2) \quad (6.6)$$

where c_i is the center vector of the i th RBF unit and σ is the receptive width of the unit. In order to avoid complex computation, the receptive widths are kept fixed to $\sigma = 0.3$ in this case. When a new input situation arrives, check the two criteria of rule generation. If the highest firing strength value of fuzzy rules is less than 0.5 or collisions are detected based on the reinforcement signal received, a new fuzzy rule, i.e., a new RBF unit is generated.

The local action space for every rule is a set of rotation angles $A = [-20, -10, 0, 10, 20]$. The selected local action of every rule cooperates to produce the continuous global action based on the rules' normalized firing strengths. The local actions are selected using an exploration-exploitation strategy based on the state-action quality, i.e., q values. Here, the simple ε -greedy method is used for action selection: a greedy action is chosen with the probability of $1 - \varepsilon$, and a random action is used with the probability of ε . The exploration probability is set by $\varepsilon = \frac{2}{10+T}$, where T is the number of trials. The exploration probability is intended to control the necessary trade-off between exploration and control, which is gradually eliminated after each trial.

The q values of the fuzzy state-action pairs can be set to optimistic initial values when the fuzzy state-actions are visited at first in order to accelerate the learning speed. When a new fuzzy rule is generated, the action selected based on the basic fuzzy controller from prior knowledge and the q value is estimated on the basis of the distance from the location to the goal. This enables the basic fuzzy controller to control the robot when new an input state is encountered. When the fuzzy state has been visited before but the local action is selected at first, the q value is initialized to the minimum value of q values in this state. After initialization, all q values of the fuzzy state-action pairs are updated according to the algorithm described in Section 6.1.2 and the learning step size is set to $\alpha = 0.3$ in this task.

In the works of [39,86], the simplest TD method, i.e. $TD(0)$ is used. In order to speed up learning, whenever the goal is reached, the learning algorithm updates the utility values of all RBF units that are along the path to the goal in reverse chronological order. Towards this objective, the robot has to store all information along the current path. Here, however, the eligibility trace method is incorporated into our algorithm and it is not necessary to store data of the current path and update values after reaching the goal. Methods using eligibility traces offer significantly faster learning, particularly when rewards are delayed by many steps and they are suitable for online applications. Furthermore, they should perform better in non-Markovian environments than the $TD(0)$ method. The term λ refers to the use of an eligibility trace and we choose $\lambda = 0.9$ first.

6.2.3 Learning Results

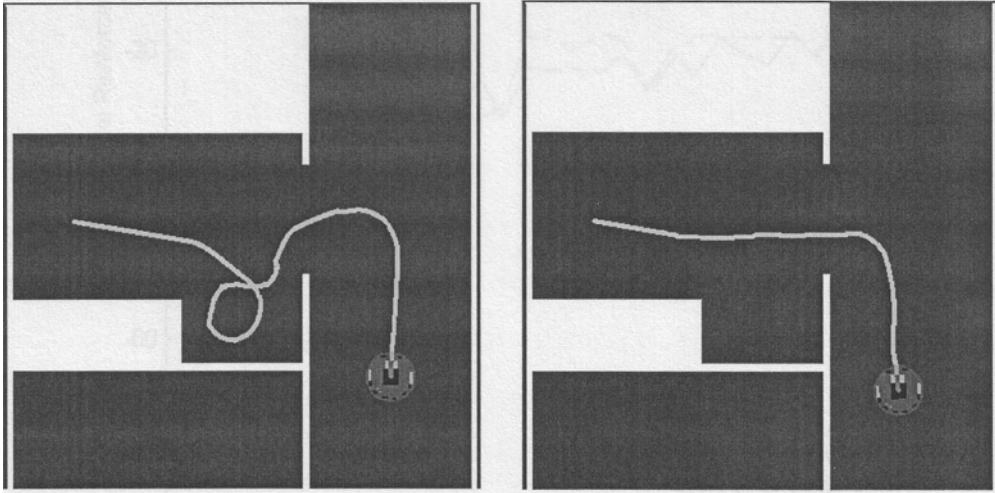


Figure 6.2 Sampling trajectory generated during first and final episodes

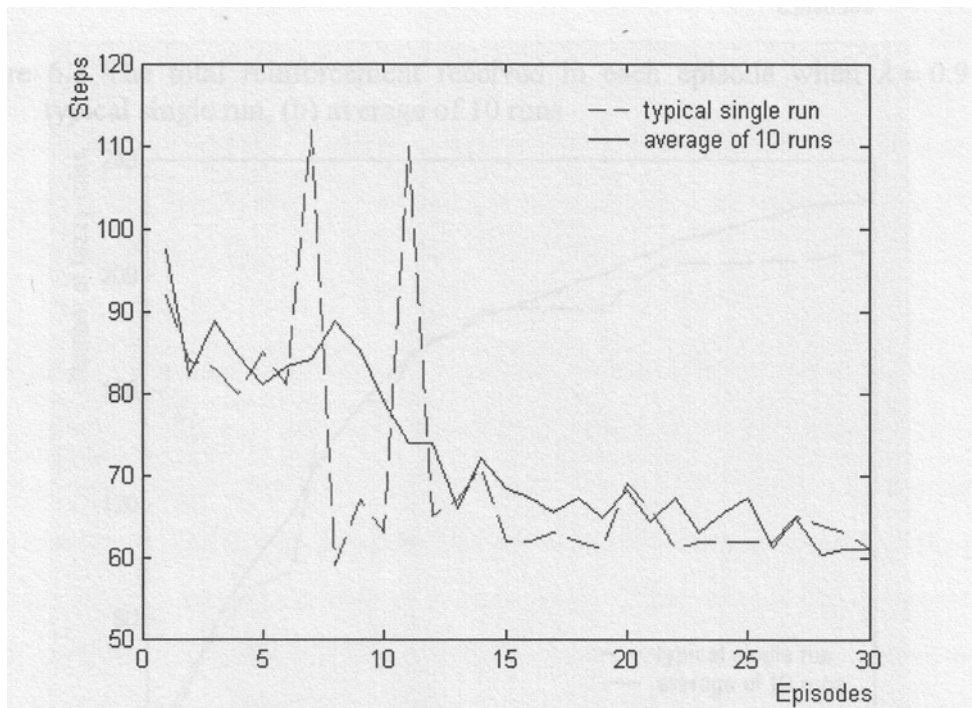


Figure 6.3 Number of steps taken in each episode when $\lambda = 0.9$ (a) typical single run, (b) average of 10 runs

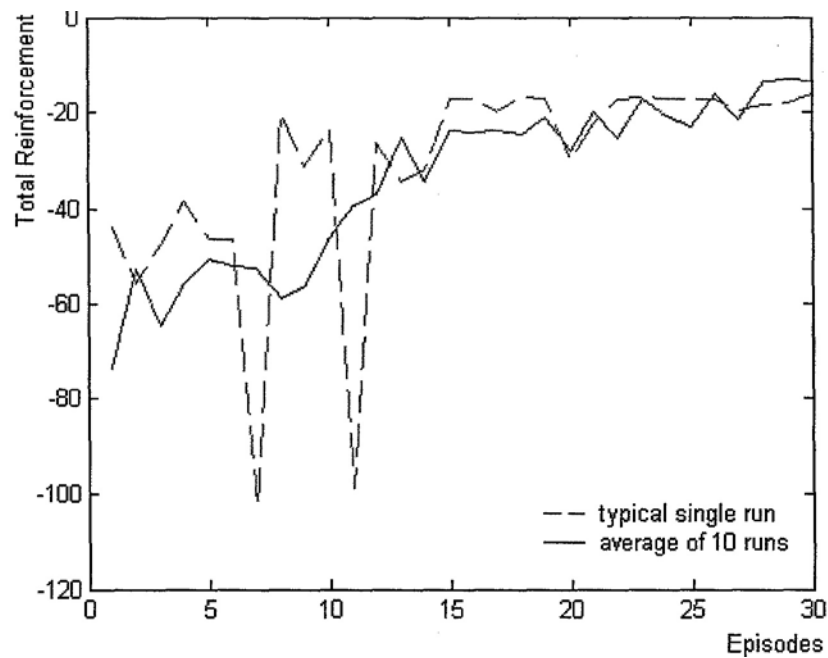


Figure 6.4 The total reinforcement received in each episode when $\lambda = 0.9$ (a) typical single run, (b) average of 10 runs

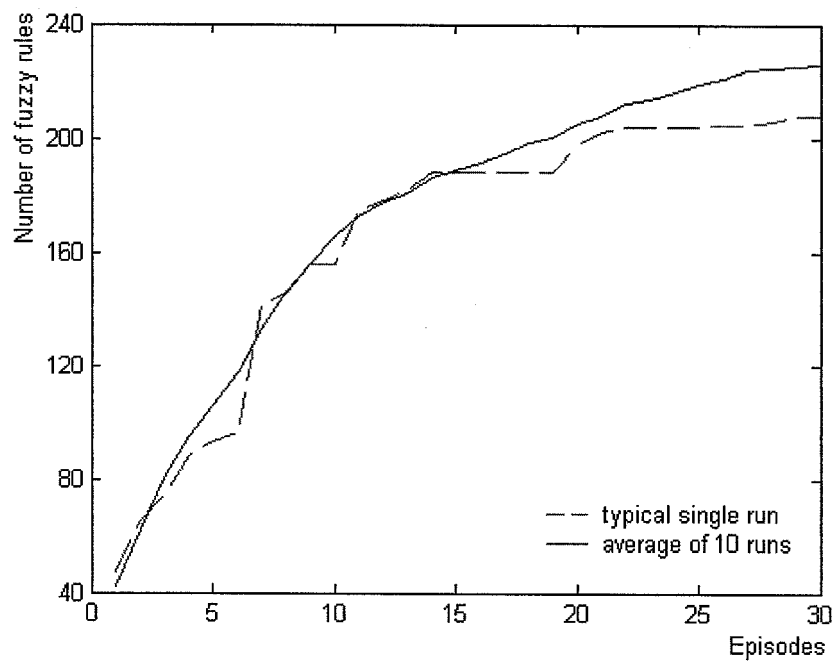


Figure 6.5 Number of fuzzy rules generated at the end of each episodes $\lambda = 0.9$ (a) typical single run, (b) average of 10 runs

Figure 6.2 shows sample robot trajectories during first and final episodes. The first time the robot tries to reach the goal, it depends almost all the time on the basic fuzzy controller, which forces it to go into the concave region since the information of the environment is unknown. In the final robot trajectory, the robot has learned to smooth out the trajectory by circumventing the concave region and to avoid colliding with the door edges by passing through the middle of the door. The learning curves which correspond to the mean values during 30 episodes over 10 runs and one typical single run are shown in Figure 6.3-6.5. During the first few episodes, the total reinforcement obtained is worse and more steps are taken along each trajectory. When the episodes proceeded, the performance of the robot is gradually improved. The number of fuzzy rules grows when the robot is exploring the environment. The average performance after 30 learning episodes is much better than that in the initial phase. Note from the single run curve that the system performances on single episodes are sometimes extremely bad. On these episodes, the robot practically takes a different action and departs from the already learned path. In the following episodes, however, it returns to its previous performance and follows the learned path.

In order to examine the effects of A values on the learning speed and quality, various values of A are used while the other parameters are left the same. Smaller learning step size, α might be used for bigger values of λ , but it is kept constant for consistency. The learning curves that correspond to the mean values during 30 episodes over 10 runs are shown in Figures 6.6-6.7. The value of $A = 0$ gives the worst performance. Increasing A improves the leaning speed. The values of λ equal to 0.9 or equal to 1.0 are similarly effective, greatly outperforming the performance for $\lambda = 0$ and better than that for $A = 0.5$. The main result is that using large A always significantly improves the performance because the parameter λ is used to distribute credit throughout the sequence of actions, leading

to faster learning and also help to alleviate the non-Markovian effect [103]. But, it is not quite consistent with the empirical results of [128], in which the performance is best for intermediate λ near 1 but the worst for $\lambda = 1$. It seems more likely that the optimal value of λ simply depends strongly on the particular problem. Another point is that bias values are used and this task actually is in non-Markovian environments.

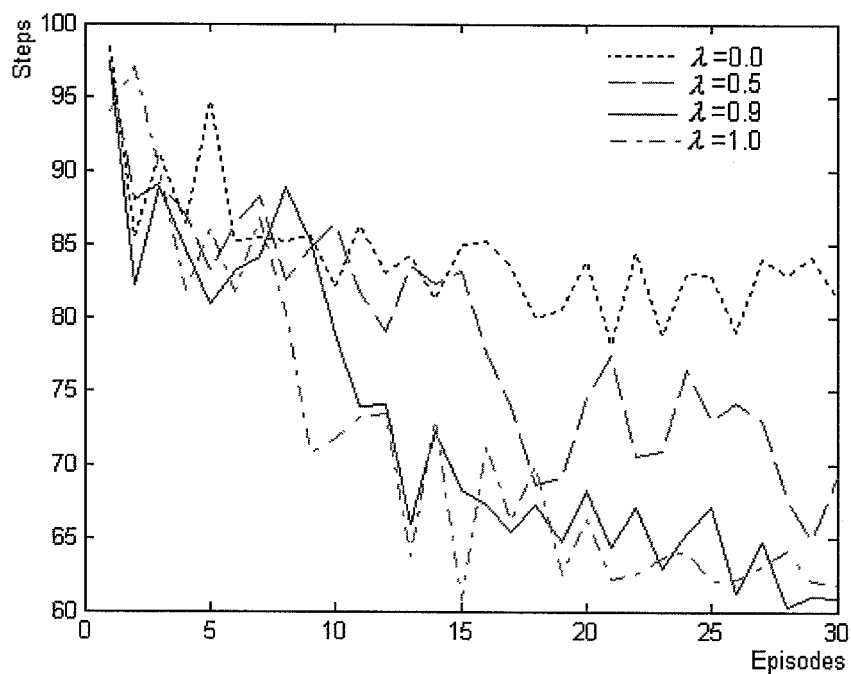


Figure 6.6 Comparison of number of steps taken during learning when different λ are used (a) $\lambda = 0.0$ (b) $\lambda = 0.5$ (c) $\lambda = 0.9$ (d) $\lambda = 1.0$

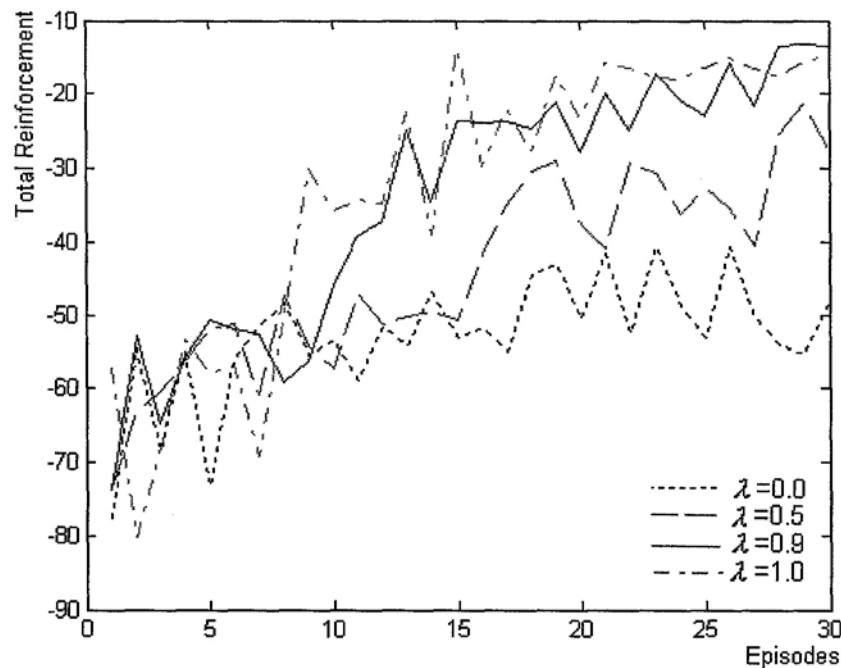


Figure 6.7 Comparison of the total reinforcement received during learning when different λ are used (a) $\lambda = 0.0$ (b) $\lambda = 0.5$ (c) $\lambda = 0.9$ (d) $\lambda = 1.0$

6.3 Discussions

As mentioned in Section 3.1.3, there are mainly two prevalent approaches to RL, namely Q-learning and actor-critic learning. The advantage of DFQL is the generation of continuous actions by means of Q-learning. There are other RL algorithms for handling continuous space and action spaces, but almost all of them are based on actor-critic architecture, e.g. the two representative fuzzy RL algorithms [10,68] discussed in Section 2.2.2. These works are based on Williams's REINFORCE algorithms [148]. Actions are generated with a normal distribution whose mean and variance vary according to the success or failure of actions. The drawback of these actor-critic architectures is that they usually suffer from local minima problems in network learning due to the use of gradient descent

learning method. Actor-critic architectures seem to be more difficult to work with than Q-learning in practice. It can be very difficult to get the relative learning rates right in actor-critic architectures so that the two components converge together.

Q-learning learns the values of all actions, rather than just finding the optimal policy. The main advantage of Q-learning over actor-critic learning is exploration insensitivity, i.e. any action can be carried out at any time and knowledge is gained from this experience. For these reasons, Q-learning is the most popular and seems to be the most effective model-free algorithm for learning from delayed reinforcement. On the other hand, because actor-critic learning updates the state value at any state based on the actual action selected, it is exploration-sensitive.

As we noted in the previous sections, we might not know a good policy for the task that we are attempting to learn. Using experimentation-sensitive algorithm with an arbitrarily bad training policy might result in a non-optimal policy. However, using experimentation-insensitive method, frees us from worrying about the quality of the policy that we adopt during training. In the works of [39,86], the learning architecture is also an actor-critic system. In order to avoid bad effects of an exploration policy, only the simplest TD method, i.e., TD(0), is used. In our algorithm, the general learning algorithm is extended to the version with an eligibility mechanism based on Peng's $Q(\lambda)$. The $Q(\lambda)$ learning algorithm exhibits beneficial performance characteristics attributable to the use of $TD(\lambda)$ returns for $\lambda > 0$. At the same time, similar Q-learning, $Q(\lambda)$ learning constructs the function of state-action pairs rather than the state, making it capable of discriminating between the effects of choosing different actions in each state. Thus, while $Q(\lambda)$ learning is experimentation-sensitive, unlike Q-learning, it seems reasonable to expect it to be less so than actor-critic learning [103].

Chapter 7

Conclusions and Future Works

This chapter summarizes the contributions made by this thesis. We then go on to discuss several possible ways in which this work might be extended in the future.

7.1 Conclusions

In this thesis, a novel algorithm termed Dynamic Fuzzy Q-Learning (DFQL) has been designed and developed.

There are two main research tracks that influence our work. The first is related to learning paradigms of fuzzy systems. Chapter 2 introduces the basic concept of FISs and discusses several issues concerning the learning ability of fuzzy systems based on different families of learning methods characterized by the information source used for learning. The second track is related to the use of generalization in reinforcement learning. Chapter 3 presents the basic framework of reinforcement learning and discusses the problem of generalization in large continuous spaces. Furthermore, problems in applying reinforcement learning to robotics are described.

In Chapter 2, we describe the learning methods of fuzzy systems based on Q-learning. However, all the algorithms described only adjust the parameters of fuzzy systems and do not involve structure identification. In Chapter 3, we introduce algorithms for generalization of experiences in RL. However, most of these works assume discrete actions. In order to cope with these problems, Chapter 4 introduces the development of the proposed DFQL. Detailed descriptions of the DFQL architecture, on-line structure and parameter learning algorithm and modeling method are presented. From the point of view of fuzzy systems, the DFQL method is a learning method capable of generating and tuning fuzzy rules automatically based on simple reinforcement signals. From the point of view of machine learning, the DFQL method is a mechanism of introducing generalization in the state-space and generating continuous actions in RL problems. The DFQL generalizes the continuous input space with fuzzy rules and generates continuous-valued actions using fuzzy reasoning. Based on the criteria of ε -completeness and the TD errors, new fuzzy rules can be generated automatically, which allows us to circumvent the problem of setting up fuzzy rules by hand.

One of the main hurdles to implementing RL systems is overcoming the lack of initial knowledge. If we know nothing of the task beforehand, it is often difficult to make any progress with learning or to keep the robot safe during the early stages of learning. Chapter 5 describes the natural framework of incorporating the initial knowledge as bias to the learning system based on fuzzy rules. It focuses on the search process on promising parts of the action space immediately and reduces the training time significantly.

In order to learn more efficiently, especially from delayed reinforcement signals, an RL system can be combined with eligibility traces, which are a basic mechanism for temporal credit assignment. Chapter 6 extends the DFQL to the

general version with an eligibility mechanism, leading to faster learning and alleviating the experimentation-sensitive problem where an arbitrarily bad training policy might result in a non-optimal policy.

The main characteristics of DFQL are summarized as follows:

- The DFQL is able to construct a FIS based on evaluative scalar reinforcement signals.
- New fuzzy rules can be generated based on the distance driven and error driven criteria so as to adjust the structure and parameter of FIS automatically.
- Continuous states are handled and continuous actions are generated through fuzzy reasoning in the DFQL.
- The if-then fuzzy rules allow the addition of initial knowledge as biases to the DFQL for rapid and safe learning during the early stages of learning.
- The general method of DFQL with an eligibility mechanism leads to more efficient learning and the ability to learn without necessarily following the current policy.

In order to test the performance of DFQL, three typical behaviors of mobile robots have been investigated. In Chapter 4, experiments performed on the Khepera robot for the obstacle avoidance behavior demonstrate the efficiency of DFQL. Compared with the random policy, the Q-learning method and the Q-KOHON method, the DFQL method is superior because of its capability of handling continuous-valued states and actions. In Chapter 5, the wall-following behavior of the Khepera robot is investigated in experiments. Thanks to the flexibility of DFQL, experimental results and comparative studies show the superiority of the proposed DFQL over the conventional Fuzzy Q-Learning in terms of both asymptotic performance and speed of learning. A comparative study of the

Continuous-Action Q-Learning and our approach also demonstrates the superiority of the DFQL method. Furthermore, the adaptive capability of DFQL has been tested in a new environment. In Chapter 6, simulation studies on optimum path experiments demonstrate that the robot is able to learn the appropriate navigation policy with a few trials. We examine the issues of efficient and general implementation of the DFQL for different eligibility rates for optimizing the sum of rewards.

7.2 Recommendations for Further Research

There are several promising directions for further work based on the results presented in this thesis. We look briefly at some of these directions and discuss their potential usefulness

7.2.1 The Convergence Property

The DFQL method is a heuristic learning method for real-life applications, where state spaces and action spaces are continuous, especially for robotics systems. Experiments have been carried out to demonstrate its usefulness. Although the DFQL method has been shown to work in a number of real and simulated domains, there is no formal guarantee of convergence. As described in Section 3.2, the analysis of the performance of general function approximation based on nonlinear architecture in RL is still an open question, although there are a large number of successful applications in practice. On the other hand, a fuzzy system can be represented as a linear architecture with fuzzy basis functions. The DFQL can be regarded as a useful method for selecting features, i.e. fuzzy basis functions and

finding initial values of parameters. By virtue of the value iteration algorithm with linear architectures [140], the convergence property can be established.

Tsitsiklis and Roy [140] discuss compact representations which approximate a value function using a linear combination of features. We consider compact representations consisting of linear combinations of fuzzy basis functions. Let us view the state space as $S = \{1, \dots, n\}$. With the fuzzy basis functions architecture [143], the state-action value takes on the following form:

$$\tilde{V}_x(W) = \sum_{k=1}^K W_k \phi_k(s) = \sum_{k=1}^K \frac{W_k f_k(s)}{\sum_{k=1}^K f_k(s)}$$

where K is the number of pre-selected fuzzy rules, W is the parameter vector, f_k is the firing strength of the k th rule. For convenience, we will assume that $f_k(s_k) = 1$ for $k \in \{1, \dots, K\}$, where s_1, \dots, s_K are pre-selected states in S . We can define a fuzzy basis function $\phi_k(s)$ as a feature and a feature mapping $\Phi(s) = (\phi_1(s), \dots, \phi_K(s))$. If with Ψ defined by $\Psi \equiv \min_{k \in \{1, \dots, K\}} \phi_k(s_k)$, there exists a $\gamma' \in [\gamma, 1)$ such that $\Psi \geq 0.5 \left(1 + \frac{\gamma}{\gamma'}\right)$, the assumption in [140] which restricts the type of features is satisfied. Based on the value iteration algorithm of [140], the convergence property can be established.

7.2.2 Partially Observable Environments

In many real-world environments, it will not be possible for the learner to have perfect and complete perception of the state of the environment. Unfortunately, complete observability is necessary for learning methods based on MDPs. The model in which the learner makes observations of the state of environment but

these observations provide incomplete information is called a partially observable Markov decision process. The way to behave effectively in hidden state tasks is to use memory of previous actions and observations to disambiguate the current state [83,122,147]. However, most of these methods are based on discrete states and actions. It would be also considered to employ fuzzy logic to deal with continuous state and action spaces.

7.2.3 Integrating Planning and Learning

As foreshadowed, it is possible for the learner to learn an optimal policy without knowing the models of environments and without learning those models. However, these methods make inefficient use of the data they gather and therefore often require a great deal of experiences to achieve good performance. The other kind of learning methods uses experience to learn the model of the environment and improve the policy based on RL at the same time [91,124], so as to achieve a better policy with fewer environmental interactions. However, these algorithms rely on the assumption of discrete states. Additional research based on our approach may produce more general results.

7.2.4 Multi-Agent Systems

In the last few years, research on multi-agent systems has become increased important. Problems are better solved by teams of agents, such as parking cleaning, vigilance of large spaces and distributed artificial intelligence. RL agents come forward as an interesting option, due to their implicit capacity to act in environments. This capacity is very attractive in multi-agent systems, because the dynamics of the environment makes the creation of a model extremely difficult.

Many researchers have tackled the problem in [1,44,73]. The DFQL learning algorithm can be further deployed for multi-agent systems.

Author's Publications

Journal Papers

Meng Joo Er and Chang Deng, "Online Tuning of Fuzzy Inference Systems Using Dynamic Fuzzy Q-Learning," IEEE Transactions on Systems, Man and Cybernetics, Part B ,Vol. 34, No. 3, pp. 1478-1489, June, 2004.

Meng Joo Er and Chang Deng, "Obstacle Avoidance of a Mobile Robot Using Hybrid Learning Approach," IEEE Transactions on Industrial Electronics, Vol. 52, No. 3, pp. 898-905, June, 2005.

Conference Papers

Chang Deng and Meng Joo Er, "Supervise Learning Neuro-Fuzzy Control of Mobile Robots," accepted for presentation at the 4th Asian Control Conference (ASCC), 2002, Singapore, 24-27 Sep., 2002.

Meng Joo Er and Chang Deng, "Comparative Study of Tracking Control Schemes for Robotic Manipulators," accepted for presentation at the 4th Asian Control Conference (ASCC), 2002, Singapore, 24-27 Sep., 2002.

Chang Deng and Meng Joo Er, "An Intelligent Robotic System Based on Neuro-Fuzzy Approach," accepted for presentation at the 7th International Conference on Control, Automation, Robotics and Vision (ICARCV), 2002, 2-5 Dec., Singapore, 2002.

Chang Deng and Meng Joo Er, "Automatic Generation of Fuzzy Rules Using Dynamic Fuzzy Neural Networks with Reinforcement Learning," accepted for presentation at IFAC International Conference on Intelligent Control Systems and Signal Processing (ICONS) 2003, pp. 520-525, 08-11 Apr., Faro, Portugal, 2003.

Meng Joo Er and **Chang Deng**, "Mobile Robot Control Using Generalized Dynamic Fuzzy Neural Networks" (invited paper), accepted for presentation at IFAC International Conference on Intelligent Control Systems and Signal Processing (ICONS) 2003, pp. 191-196, 08-11 Apr., Faro, Portugal, 2003.

Chang Deng and Meng Joo Er, "Automatic Generation of Fuzzy Rules Using Dynamic Fuzzy Q-Learning", accepted for presentation at IEEE International Conference on Systems, Man and Cybernetics, SC Track, 5-8 Oct., Washington, D. C., USA, 2003.

Chang Deng and Meng Joo Er, "Mobile Robot Control Using Dynamic Fuzzy Q-Learning," accepted for presentation at the 2nd WSEAS on E-Activities (E-Activities 2003) and the 2nd WSEAS on Electronics, Control and Signal Processing (ICECS 2003), 7-9 Dec, Singapore, 2003.

Chang Deng and Meng Joo Er, "Efficient Implementation of Dynamic Fuzzy Q-Learning," accepted for presentation at 2003 Joint Conference of the Fourth International Conference on Information, Communications and Signal Processing and Fourth Pacific-Rim Conference on Multimedia, 15-18 Dec, Singapore, 2003.

Meng Joo Er and **Chang Deng**, "Obstacle Avoidance of a Mobile Robot Using Hybrid Learning Approach," accepted for presentation at the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS), 15-18 Dec, Singapore, 2003.

Chang Deng and Meng Joo Er, "Real-time Dynamic Fuzzy Q-Learning and Control of Mobile Robots," accepted for presentation at the 5th Asian Control Conference (ASCC), 11-15 July, Australia, 2004.

Chang Deng and Meng Joo Er, "Dynamic Fuzzy Q-Learning and Control of Mobile Robots," accepted for presentation at the 8th International Conference on Control, Automation, Robotics and Vision (ICARCV), 6-9 Dec, China, 2004.

Bibliography

- [1] O. Abul, F. Polat and R. Alhajj, "Multiagent reinforcement learning using function approximation," *IEEE Trans. on Systems, Man and Cybernetics, Part C*, Vol. 30, pp. 485-496, 2000.
- [2] R. C. Arkin, *Behavior-Based Robotics*, Cambridge, Mass.: MIT Press, 1998.
- [3] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Purposive behavior acquisition for a real robot by vision-based reinforcement learning," *Machine Learning*, Vol. 23, pp. 279-303, 1996.
- [4] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, Vol. 11, pp. 11-113, 1997.
- [5] L. Baird and A. Moore, "Gradient descent for general reinforcement learning," in *Advances in Neural Information Processing Systems*, Vol. 11, MIT Press, 1999.
- [6] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Systems, Man and Cybernetics*, Vol. 13, pp. 834-846, 1983.
- [7] A. G. Barto and M. I. Jordan, "Gradient following without back propagation in layered network", in *Proc. IEEE 1st Annu. Conf. Neural Network*, San Diego, CA, pp. 629-636, 1987.
- [8] A. G. Barto, "Connectionist learning for control: an overview," in W. T. Miller R. S. Sutton and P. J. Werbos (Eds.), *Neural Networks for Control*, MIT Press, Cambridge, MA, 1990.
- [9] R. Bellman, *Dynamic programming*, Princeton University Press, Princeton, NJ, 1957.
- [10] H. R. Berenji and P. Khedkar, "learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, Vol. 5, pp. 724-740, 1992.

- [11] H. R. Berenji, "Fuzzy Q-Learning: a new approach for fuzzy dynamic programming," in Proc. IEEE Int. Conf. Fuzzy Systems, pp. 486-491, 1994.
- [12] D. P. Bertsekas, Dynamic programming and optimal control, Athena Scientific, Belmont, MA, 1995.
- [13] D. P. Bertsekas and J. N. Tsitsiklis, Neuro-Dynamic Programming, MIT, Athena Scientific, Belmont, MA, 1996.
- [14] M. J. L. Boada, R. Barber and M. A. Salichs, "Visual approach skill for a mobile robot using learning and fusion of simple skills," Robotics and Autonomous Systems, Vol. 38, pp. 157-170, 2002.
- [15] A. Bonarini, "Evolutionary learning, reinforcement learning, and fuzzy rules for knowledge acquisition in agent-based systems," Proceeding of IEEE, Vol. 89, pp. 1334-1346, 2001,
- [16] A. Bonarini and C. Bonacina and M. Matteucci, "An approach to the design of reinforcement functions in real world, agent-based applications", IEEE Trans. Systems, Man and Cybernetics, Part B, Vol. 31, pp. 288-300, 2001.
- [17] K. M. Bossley, D. J. Mills, M. Brown and C. J. Harris, "Construction and design of parsimonious neuro-fuzzy systems", in Neural Network Engineering in Dynamic Control Systems, K. J. Hunt, G. W. Irwin and K. Warwick (Ed.), Berlin, Spring Verlag, pp. 153-178, 1995.
- [18] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: safely approximating the value function," in Advances in Neural Information Processing Systems, G. Tesauro, D. S. Touretzky, and T. K. Leen, (Eds), Cambridge, MA, MIT Press, 1995.
- [19] S. J. Bradtke, "Reinforcement learning applied to linear quadratic regulation," in S. J. Hanson, J. d. Cowan and C. L. Giles (Eds.), Advances in Neural Information Processing systems, San Mateo, CA, Morgan Kaufmann, Vol. 5, pp. 295-302, 1993.

- [20] R. A. Brooks, "A robust layered control system for a mobile robot," IEEE Tran. Robot. Automat. Vol. 2, pp. 14-23, 1986.
- [21] R. A. Brooks, "Intelligence without representation," Artificial Intell. J., Vol. 47, pp. 139-159, 1991.
- [22] R. A. Brooks and M. J. Mataric, "Real robots, real learning problems," in Robot Learning, J. H. Connell and S. Mahadevan (Eds.), Kluwer Academic, Boston, 1993.
- [23] J. Bruske, I. Ahrns and G. Sommer, "An integrated architecture for learning of reactive behaviors based on dynamic cell structures", Robotics and Autonomous Systems, Vol. 22, pp. 87-101, 1997.
- [24] C. T. Chao, Y. J. Chen and C. C. Teng, "Simplification of fuzzy-neural systems using similarity analysis", IEEE Trans. Systems, Man and Cybernetics, B, Vol. 26, pp. 344-354, 1996.
- [25] R. Chatterjee and F. Matsuno, "Use of single side reflex for autonomous navigation of mobile robots in unknown environments," Robotics and Autonomous Systems, Vol. 35, pp. 77-96, 2001.
- [26] C. K. Chiang, H. Y. Chung and J. J. Lin, "A self-learning fuzzy logic controller using genetic algorithms with reinforcements," IEEE Trans. Fuzzy Systems, Vol. 5, pp. 460-467, 1997.
- [27] K. B. Cho and B. H. Wang, "Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction", Fuzzy Sets and Systems, Vol. 83, pp. 325-339, 1996.
- [28] J. Connell and S. Mahadevan, Robot Learning, Kluwer Academic, Boston, 1993.
- [29] O. Cordon and F. Herrera, "A hybrid genetic algorithms evolution strategy process for learning fuzzy logic controller knowledge bases", in Genetic Algorithms and Soft Computing, Berlin, Germany, Physica Verlag, pp. 251-278, 1996.

- [30] O. Cordon, F. Herrera, and M. Lozano, "On the bidirectional integration of fuzzy logic and genetic algorithms," in 2nd Online Workshop Evolutionary Computat. (WEC2), Nagoya, Japan, pp. 13–17, 1996.
- [31] R. H. Crites and A. G. Barto, "Improving elevator performance using reinforcement learning," In D. S. Touretzky, M. C. Mozer and M. E. Hasselmo (Eds), *Advances in Neural Information Processing Systems: 1995 Conference*, pp. 1017-1023, MIT Press, 1996.
- [32] P. Dayan, "The convergence of $TD(\lambda)$ for general A," *Machine Learning*, Vol. 8, pp. 341-362, 1992.
- [33] D. Floreano and F. Mondada, "Evolution of homing navigation in a real mobile robot," *IEEE Trans. Systems, Man and Cybernetics, Part B*, Vol. 26, 1996.
- [34] T. Fukuda and N. Kubota, "An intelligent robotic system based on a fuzzy approach", *Proceeding of IEEE*, Vol. 87, pp.1448-1470, 1999.
- [35] Y. Gao, "Adaptive Identification and Control of Nonlinear Systems Using Generalized Fuzzy Neural Network," PhD thesis, Nanyang Technological University, Singapore, 2003.
- [36] P. Gaussier, A. Revel, C. Joulain and S. Zrehen, "Living in a partially structured environment: How to bypass the limitations of classical reinforcement techniques," *Robotics and Autonomous Systems*, Vol. 20, pp. 225-250, 1997.
- [37] P.Y. Glorennec and J. Jouffe, "Fuzzy Q-learning", *Proceedings of the sixth IEEE international conference on fuzzy systems*, 1997.
- [38] V. Gullapalli, J. A. Frankin and H. Benbrahim, "Acquiring robot skills via reinforcement learning," *IEEE Control Systems Magazine*, Vol. 14, pp. 13-24, 1994.
- [39] G. Hailu and G. Sommer, "Learning by biasing", *Proceeding of the 1998 IEEE Inter. Conf. on robotics and automation*, pp. 2168-2173, 1998.

- [40] G. Hailu, "Symbolic structures in numeric reinforcement for learning optimum robot trajectory," *Robotics and Autonomous Systems*, Vol. 37, pp. 53-68, 2001.
- [41] T. Horiuchi, A. Fujino, O. Katai and T. Sawaragi, "Fuzzy interpolation-based Q-learning with profit sharing plan scheme", in *Proc. IEEE Int. Conf. Fuzzy Systems*, pp. 1707-1712, 1997.
- [42] W. R. Hwang and W. E. Thompson, "Design of intelligent fuzzy logic controllers using genetic algorithms," in *Proc. IEEE Int. Conf. Fuzzy Systems*, Orlando, FL, pp. 1383-1388, 1994.
- [43] H. Inoue, K. Kamei, and K. Inoue, "Automatic generation of fuzzy rules using hyper-elliptic cone membership function by genetic algorithms," in *Proc. 7th IFSA World Congress*, Prague, Czech Republic, Vol. II, pp. 383–388, 1997.
- [44] Y. Ishiwaka, T. Sato and Y. Kakazu, "An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning," *Robotics and Autonomous*, Vol. 43, pp. 245-256, 2003.
- [45] J. S. R. Jang and C. T. Sun, "Functional equivalence between Radial Basis Function networks and fuzzy inference systems", *IEEE Trans. Neural Networks*, Vol. 4, pp. 156-158, 1993.
- [46] J. S. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system", *IEEE Trans. Systems, Man and Cybernetics*, Vol. 23, pp. 665-684, 1993.
- [47] J. S. R. Jang, C. T. Sun and E. Mizutani, "Neuro-fuzzy and soft computing". Englewood Cliffs, New Jersey, Prentice Hall, 1997.
- [48] L. Jouffe, "Fuzzy inference system learning by reinforcement methods", *IEEE Trans. Systems, Man, and Cybernetics, Part C*, Vol. 28, pp. 338-355, 1998.
- [49] C. F. Juang and C. T. Lin, "An on-line self-constructing neural fuzzy inference network and its applications", *IEEE Trans. Fuzzy Systems*, Vol. 6, pp. 12-32, 1998.

- [50] K-Team S. A., "Khepera 2 user manual," Switzerland, 2002.
- [51] L. P. Kaelbling, M. L. Littman and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligent Research*, Vol. 4, pp. 237-285, 1996.
- [52] Z. Kalmar and C. Szepesvari and A. Lorinca, "Module-based reinforcement learning: experiments with a real robot," *Machine Learning*, Vol. 31, pp. 55-85, 1998.
- [53] R. M. Kandadai and J. M. Tien, "A knowledge-base generation hierarchical fuzzy-neural controller," *IEEE Trans. Neural Networks*, Vol. 8, pp. 1531-1541, 1997.
- [54] C. Karr, "Applying genetics to fuzzy logic", *AI Expert*, Vol. 6, pp. 38-43, 1991.
- [55] C. L. Karr, "Adaptive control with fuzzy logic and genetic algorithms", in *Fuzzy Sets, Neural Networks, and Soft Computing*, R. R. Yager and L. A. Zadeh, Eds. New York: Van Nostrand Reinhold, 1993.
- [56] C. L. Karr and E. J. Gentry, "Fuzzy control of pH using genetic algorithms," *IEEE Trans. Fuzzy Systems*, vol. 1, pp. 4653, 1993.
- [57] E. Kim, S. Ji and M. Park, "A new approach to fuzzy modeling", *IEEE Trans. Fuzzy Systems*, Vol.5, pp.328-337, 1997.
- [58] T. Kohonen, *Self-organization and associative memory*, Springer-Verlag, 1988.
- [59] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller", *IEEE Trans. Systems, Man and Cybernetics*, Vol. 20, pp.404-436, 1990.
- [60] C. C. Lee, "A self learning rule-based controller employing approximate reasoning and neural net concepts," *Int. J. Intell. Syst.*, Vol. 6, pp. 71-93, 1991.
- [61] M. A. Lee and H. Takagi, "Integrating design stages of fuzzy systems using genetic algorithms", in *Proceeding of 2nd IEEE Int. Conf. Fuzzy Systems*, pp. 612-617, 1993.

- [62] M. A. Lee, H. Takagi, "Dynamic control of genetic algorithms using fuzzy logic techniques," in Proc. Int. Conf. Genetic Algorithms, Urbana-Champaign, IL, pp. 76–83, 1993.
- [63] R. P. Li and M. Mukaidono, "A new approach to rule learning based on fusion of fuzzy logic and neural networks", IEICE Trans. Inf. Syst., Vol. E78-d, pp. 1509-1514, 1995.
- [64] Y. Li, N. Sundararajan and P. Saratchandran, "Analysis of minimal radial basis function network algorithm for real-time identification of nonlinear dynamic systems," IEE Proc. Control Theory Appl., Vol. 147, pp. 476-484, 2000.
- [65] L. J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," Machine Learning, Vol. 8, pp.293-321, 1992.
- [66] L. J. Lin, "Hierarchical learning of robot skills by reinforcement," Proceeding of IEEE Conf. Neural Network, pp. 181-187, 1993.
- [67] C. T. Lin, "Neural fuzzy control systems with structure and parameter learning", Singapore, World scientific, 1994.
- [68] C. T. Lin and C. S. G. Lee, "Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems," IEEE Trans. Fuzzy Systems, Vol. 2, pp. 46-63, 1994.
- [69] Y. H. Lin and G. A. Cunningham, "A new approach to fuzzy-neural system modeling", IEEE Trans. Fuzzy Systems, Vol. 3, pp. 190-197, 1995.
- [70] C. T. Lin and C. P. Jou, "GA-based fuzzy reinforcement learning for control of a magnetic bearing system," IEEE Trans. Systems, Man and Cybernetics, Vol. 30, pp. 276-289, 2000.
- [71] D. A. Linkens and H. O Nyongesa, "Genetic algorithms for fuzzy control, offline/online system development and application," IEE Proc. Control Theory and Applications, Vol. 142, pp. 161-185, 1995.

- [72] D. A. Linkens and H. O. Nyongesa, "Learning systems in intelligent control: On appraisal of fuzzy, neural and genetic algorithm control applications", in Proc. Inst. Elect. Eng. Control Theory Applications, Vol. 143, pp. 367-386, 1996.
- [73] M. L. Littman and C. Szepesvari, "A generalized reinforcement learning model: convergence and applications," Proc. The 13th Intl. Conf. Machine Learning, pp. 310-318, 1996.
- [74] Y. Lu, N. Sundararajan and P. Saratchandran, "Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm," IEEE Trans. Neural Networks, Vol. 9, pp. 308-318, 1998.
- [75] J. Lygeros, "A formal approach to fuzzy modeling", IEEE Trans. Fuzzy Systems, Vol. 5, pp.37-327, 1997.
- [76] H. Maaref and C. Barret, "Sensor-based fuzzy navigation of an autonomous mobile robot in an indoor environment," Control Engineering Practice, Vol. 8, pp. 757-768,2000.
- [77] P. Machler, "Robot odometry correction using grid lines on the floor," MCPA, Italy, 1997.
- [78] Y. Maeda, "Modified Q-Learning method with fuzzy state division and adaptive rewards," Proceeding of the 2002 IEEE world congress on computational intelligence, pp. 1556-1561,2002.
- [79] P. Maes and R. A. Brooks, "Learning to coordinate behaviors," in Proceedings of the Eighth National Conference on Artificial Intelligence, pp.796-802, Menlo Park, C A , A A A I Press, 1990.
- [80] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," Artificial Intelligence, Vol. 55, pp. 311-365,1992.
- [81] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller", Int. J. Man-Machine Studies, Vol. 7, pp.1 - 13,1975.

- [82] M. J. Mataric, "Reward functions for accelerated learning," in W. W. Cohen and H. Hirsh (Eds.), *Proceeding of the 11th Inter. Conf. Machine Learning*, Morgan Kaufmann 1994.
- [83] R. A. Mccallum, "Hidden state and reinforcement learning with instance-based state identification," *IEEE Trans. on Systems, Man and Cybernetics*, Part B, Vol. 26, pp. 464-473, 1996.
- [84] Z. Michalewicz, "Genetic Algorithms + Data Structures + Evolution Programs", 3rd Ed. Berlin, Germany: Springer Verlag, 1996.
- [85] J. D. R. Millan, "Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot," *Robotics and Autonomous Systems*, Vol. 15, pp. 275-299, 1995.
- [86] J. D. R. Millan, "Rapid, safe, and incremental learning of navigation strategies," *IEEE Trans. Systems, Man and Cybernetics*, Vol. 26, pp. 408-420, 1996.
- [87] J. D. R. Millan, D. Posenato and E. Dedieu, "Continuous-action Q-learning", *Machine Learning*, Vol. 49, pp. 247-265, 2002.
- [88] T. M. Mitchell and S. Thrun, "Explanation-based neural network learning for robot control", in *Advances in Neural Information Processing Systems*, San Mateo, CA, Morgan Kaufmann, Vol. 5, 1993.
- [89] J. Moody and C. J. Darken, "Fast learning in network of locally-tuned processing units", *Neural Computation*, Vol. 1, pp. 281-294, 1989.
- [90] A. W. Moore, "Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued spaces," in *Proc. 8th International Machine Learning Workshop*, 1991.
- [91] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less real time," *Machine learning* Vol. 13, pp. 103-130, 1993.

- [92] A. W. Moore and C. G. Atkeson, "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces," *Machine Learning*, Vol. 21, pp. 199-233, 1995.
- [93] G. C. Mouzouris and M. Mendel, "Dynamic non-singleton fuzzylogic system for nonlinear modeling", *IEEE Trans. Fuzzy Systems*, Vol. 5, pp. 199-208, 1997.
- [94] R. Munos, "A study of reinforcement learning in the continuous case by the means of viscosity solutions," *Machine Learning*, Vol. 40, pp. 265-299, 2000.
- [95] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Machine Learning*, Vol. 49, pp. 291-323, 2002.
- [96] T. Nilsson, <http://www.kiks.f2s.com>
- [97] S. Nolfi and D. Floreano, *Evolutionary Robotics: the Biology, Intelligence, and Technology of Self-Organizing Machines*, Cambridge, Mass.: MIT Press, 2000.
- [98] D. Ormoneit and S. Sen, "Kernel-based reinforcement learning," *Machine Learning*, Vol. 49, pp. 161-178, 2002.
- [99] A. Parodi and P. Bonelli, "A new approach to fuzzy classifier system⁷", in *Proc. 5th Int. Conf. Genetic Algorithms*, San Mateo, CA, pp. 223-230, 1993.
- [100] K. M. Passino and S. Ywkovich, "Fuzzy control", Addison Wesley Longman, Inc., 1998.
- [101] W. Pedrycz, "Fuzzy modeling: fundamental construction and evaluation", *Fuzzy Sets and Systems*, Vol. 41, pp.1-15, 1991.
- [102] J. Peng, "Efficient dynamic programming-based learning for control", Ph.D thesis, Northeastern University, Boston, MA, 1993.
- [103] J. Peng and R. J. Williams, "Incremental multi-step Q-learning," *Machine Learning*, Vol. 22, pp. 283-290, 1996.
- [104] J. Platt, "A resource allocating network for function interpolation," *Neural Computa.*, Vol. 3, pp. 213-225, 1991.

- [105] M. L. Puterman, Markov decision processes-Discrete stochastic dynamic programming, John Wiley & Sons, Inc., New York, NY, 1994.
- [106] G. V. S. Raju and J. Zhou, "Adaptive hierarchical fuzzy controller", IEEE Trans. Systems, Man and Cybernetics, Vol, 23, pp. 937-980, 1993.
- [107] H. Rak and H. S. Cho, "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning," IEEE Trans. Systems, Man, and Cybernetics, Vol. 25, pp. 464-477, 1995.
- [109] R. Ranscheit and E. M. Schard, "Experiments with the use of a rule-based self-organizing controller for robotics applications", Fuzzy Sets and Systems, Vol. 26, pp. 195-214, 1988.
- [109] C. Ribeiro, "Reinforcement learning agents," Artificial Intelligence Review, Vol. 17, pp. 223-250, 2002.
- [110] G. A. Rummery, "Problem solving with reinforcement learning," Ph.D thesis, Cambridge University, 1995.
- [111] M. Rylatt, C. Czarnecki and T. Routen, "Connectionist learning in behavior-based mobile robot: a survey," Artificial Intelligence Review, Vol. 12, pp. 445-468, 1998.
- [112] J. C. Santamaria, R. S. Sutton and A. Ram, "Experiments with reinforcement learning in problems with continuous state and action spaces," Adaptive Behavior, Vol. 6, pp. 163-217, 1998.
- [113] J. J. Shann and H. C. Fu, "A fuzzy neural networks for rule acquiring in fuzzy control systems", Fuzzy Sets and systems, Vol. 71, pp. 345-357, 1995.
- [114] S. Shao, "Fuzzy self-organizing controller and its application for dynamic processes", Fuzzy Sets and Systems, Vol. 26, pp. 151-164, 1988.
- [115] Y. Shi, R. Eberhart and Y. Chen, "Implementation of evolutionary fuzzy systems", IEEE Trans. Fuzzy Systems, Vol. 7, pp. 109-119, 1999.
- [116] K. Shimojima, T. Fukuda, and Y. Hasegawa, "RBF-fuzzy system with GA based unsupervised/supervised learning method," in Proc. Int. Joint Conf. 4th

- IEEE Int. Conf. Fuzzy Syst./2nd Int. Fuzzy Eng. Symp. (FUZZ/IEEE-TFES), Yokohama, Japan, Vol. I, pp. 253-258., 1995.
- [117] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, Vol. 22, pp. 123-158, 1996.
- [118] S. P. Singh and D. Bertsekas, "Reinforcement learning for dynamic channel allocation in cellular telephone systems," in *Advances in Neural Information Processing Systems: 1996 Conference*, pp. 947-980, MIT Press, 1997.
- [119] M. Sugeno and G. T. Kang, "Structure identification of fuzzy model", *Fuzzy Sets and Systems*, Vol. 28, pp.15-33, 1988.
- [120] M. Sugeno and K. Tanaka, "Successive identification of a fuzzy model and its applications to prediction of a complex system", *Fuzzy Sets and Systems*, Vol. 42, pp. 315-334, 1991.
- [121] C. T. Sun and J. S. Jang, "A neuro-fuzzy classifier and its applications", in *Proc. IEEE Int. Conf. Fuzzy Systems*, Vol. 1, pp. 94-98, 1993.
- [122] R. Sun and C. Seessions, "Self-segmentation of sequences: Automatic formation of hierarchies of sequential behaviors," *IEEE Trans. on Systems, Man and Cybernetics, Part B* , Vol. 30, pp. 403-418, 2000.
- [123] R. S. Sutton, "Learning to predict by the method of temporal differences", *Machine Learning*, Vol. 3, pp. 9-44, 1988.
- [124] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," *Proc. 7th Intl. Conf. Machine Learning*, 1990.
- [125] R. S. Sutton and A. G. Barto, "Time-derivative models of pavlovian reinforcement," *Learning and Computational Neuroscience: Foundations for Adaptive Networks*, MIT Press, 1990.
- [126] R. S. Sutton, "Reinforcement Learning Architectures for Animats," *Proc. of the 1st Inter. Conf. on Simulation of Adaptive behavior, From Animals to Animats*, J. A. Meyer and S. W. Wilson (Eds.), pp. 288-296, 1992.

- [127] R. S. Sutton, "Generalization in reinforcement learning: successful examples using sparse coarse coding", in Touretzky, D. S., Mozer, M. C., and Hasselma, M. E. (eds), *Advances in Neural Information Processing Systems Vol. 8*, pp. 1038-1044, 1996.
- [128] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, the MIT Press, Cambridge, Massachusetts, 1998.
- [129] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control", *IEEE Trans. Systems, Man and Cybernetics*, Vol. 15, pp. 116-132, 1985.
- [130] T. Takagi and M. Sugeno, "NN-driven fuzzy reasoning", *Int. J. Approx. Reason.*, Vol. 5, pp. 191-211, 1991.
- [131] G. J. Tesauro, "Practical issues in temporal difference learning," *Machine Learning*, Vol. 8, pp. 257-277, 1992.
- [132] G. J. Tesauro, "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, Vol. 6, pp. 215-219, 1994.
- [133] C. L. Tham, "Reinforcement learning of multiple tasks using a hierarchical CMAC architecture," *Robotics and Autonomous Systems*, Vol. 15, pp. 247-274, 1995.
- [134] P. Thrift, "Fuzzy logic synthesis with genetic algorithms," in *Proc. 4th Int. Conf. Genetic Algorithms (ICGA)*, San Diego, CA, pp. 509-513, 1991.
- [135] S. B. Thrun, "The role of exploration in learning control", in White, D. A., and Sofge, D. A. (Eds), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold, New York, NY, 1992.
- [136] S. Thrun, "An approach to learning mobile robot navigation", *Robotics and Autonomous Systems* Vol. 15, pp. 301-319, 1995.
- [137] H. Tong and T. X. Brown, "Reinforcement learning for call admission control and routing under quality of service constraints in multimedia networks," *Machine Learning*, Vol. 49, pp. 111-139, 2002.

- [138] C. F. Touzet, "Neural reinforcement learning for behavior synthesis", Robotics and Autonomous, Vol. 22, pp. 251-281, 1997.
- [139] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," Machine Learning, Vol. 16, pp. 185-202, 1994.
- [140] J. N. Tsitsiklis and B. V. Roy, "Feature-based methods for large scale dynamic programming," Machine Learning, Vol. 22, pp. 59-94, 1996.
- [141] Y. Tsukamoto, "An approach to fuzzy reasoning method", in "Advances in Fuzzy Set Theory and Applications", M. M. Gupta, R.K. Ragade and R.R. Yager (Ed.), Amsterdam, North Holland, pp. 137-149, 1979.
- [142] L. X. Wang and J. M. Mendel, "Fuzzy basis function, universal approximation, and orthogonal least squares learning", IEEE Trans. Neural Networks, Vol. 3, pp. 807-814, 1992.
- [143] L. X. Wang, "Adaptive fuzzy systems and control: design and stability analysis", New Jersey, Prentice Hall, 1994.
- [144] L. X. Wang, "A course in fuzzy systems and control", New Jersey, Prentice Hall, 1997.
- [145] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge Univ. Cambridge, U. K., 1989.
- [146] C. J. C. H. Watkins and P. Dayan, "Q-learning," Machine Learning, Vol. 8, pp. 279-292, 1992.
- [147] S. Whitehead and L. Lin, "Reinforcement learning of non-Markov decision process," Artificial Intelligence, Vol. 73, pp. 271-306, 1995.
- [148] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine Learning, Vol. 8, pp. 229-256, 1992.
- [149] S. Wu, M. J. Er and Y. Gao, "A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks", IEEE Trans. Fuzzy Systems, Vol. 9, pp. 578-693, 2001.

-
- [150] L. A. Zadeh, "Fuzzy sets", *Information and control*, Vol. 8, pp. 338-358, 1965.
- [151] W. Zhang and T. G. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *Proceeding of the International Joint Conference on Artificial Intelligence*, 1995.
- [152] Y. Lu, "Development and application of a sequential minimal radial basis function (RBF) neural network learning algorithm," M. Eng Thesis, EEE, NTU, 1997.
- [153] Y. Lu, N. Sundararajan and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function neural networks," *Neural Computation*, Vol. 9, No. 2, pp. 461-478, MIT Press, USA, 1997.