ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

FUZZIFIED Q-LEARNING ALGORITHM IN THE DESIGN OF FUZZY PID CONTROLLER

M.Sc. THESIS

Vahid TAVAKOL AGHAEI

Department of Control Engineering

Control and Automation Engineering Programme

Thesis Advisor: Prof. Dr. İbrahim EKSİN

AUGUST 2013

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

FUZZIFIED Q-LEARNING ALGORITHM IN THE DESIGN OF FUZZY PID CONTROLLER

M.Sc. THESIS

Vahid TAVAKOL AGHAEI
(504091135)

Department of Control Engineering

Control and Automation Engineering Programme

Thesis Advisor: Prof. Dr. İbrahim EKSİN

AUGUST 2013

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BULANIK MANTIK KONTROLÖRÜN TASARIMINDA KULLANILAN BULANIK Q-ÖĞRENME ALGORİTMASI

YÜKSEK LİSANS TEZİ

Vahid TAVAKOL AGHAEİ
(504091135)

Kontrol Mühendisliği Anabilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr. İbrahim EKSİN

Anabilim Dalı : Herhangi Mühendislik, Bilim
Programı : Herhangi Program

AĞUSTOS 2013

Vahid TAVAKOL AGHAEI, a **M.Sc.** student of ITU **Institute of Science and Technology** student ID 504091135, successfully defended the **thesis/dissertation** entitled “**FUZZIFIED QL ALGORITHM IN THE DESIGN OF FUZZY PID CONTROLLER**”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. İbrahim EKSİN** İstanbul Technical University	....
Jury Members :	**Prof. Dr. Müjde GÜZELKAYA** İstanbul Technical University	....
Jury Members :	**Doç. Dr. İlker ÜSTOĞLU** Yıldız Technical University	....

Date of Submission : 3 August 2013
Date of Defense : 28 August 2013

To my parents,

FOREWORD

First of all I would like to thank my advisor Prof. Dr. Ibrahim EKSİN and Prof. Dr. Müjde GÜZELKAYA for their sincere efforts and giving inspiration for doing this thesis.
I am also grateful to my friend, Nasser ARGHAVANİ who was supportive of me through either good or not-so-good times.
At last, I would like to express my endless love towards my family specially my parents for being patient and giving support and encouragement throughout my life.

August 2013 Vahid TAVAKOL AGHAEI

TABLE OF CONTENTS

ABBREVIATIONS

FLC	**:** Fuzzy Logic Controller
ANN	**:** Artificial Neural Network
AI	**:** Artificial Intelligence
FRBS	**:** Fuzzy Rule Based System
FIS	**:** Fuzzy Inference System
GA	**:** Genetic Algorithm
PD	**:** Proportional Derivative
PI	**:** Proportional Integral
PID	**:** Proportional Integral Derivative
PIDC	**:** Proportional Integral Derivative Controller
MIMO	**:** Multi Input Multi Output
MF	**:** Membership Function
FPDC	**:** Fuzzy Proportional Derivative Controller
FPIC	**:** Fuzzy Proportional Integral Controller
I/O	**:** Input/Output
CoG	**:** Center of Gravity
TD	**:** Temporal Difference
ACL	**:** Actor Critic Learning
RL	**:** Reinforcement Learning
DP	**:** Dynamic Programming
AHC	**:** Adaptive Heuristic Critic
MDP	**:** Markov Decision Process
ANFIS	**:** Adaptive Neuro Fuzzy Inference System
FQL	**:** Fuzzified Q-Learning
ISE	**:** Integral Square Error

LIST OF TABLES

LIST OF FIGURES

Page

FUZZIFIED Q-LEARNING ALGORITHM IN THE DESIGN OF FUZZY PID CONTROLLERS

SUMMARY

In the traditional control theory, an appropriate controller is designed based on a mathematical model of the plant under the assumption that the model provides a complete and accurate characterization of the plant. However, in some practical problems, the mathematical models of plants are difficult or time-consuming to be obtained because the plants are inherently nonlinear and/or exhibit uncertainty. Thus, new methods are proposed to process these characteristics. In recent years, increased efforts have been centered on developing intelligent control systems that can perform effectively in real-time. These include the development of non-analytical methods of Artificial Intelligence (AI) such as neural networks, fuzzy logic and genetic algorithms.

Fuzzy logic is a mathematical approach which has the ability to express the ambiguity of human thinking and translate expert knowledge into computable numerical data. It has been shown that fuzzy logic based modeling and control could serve as a powerful methodology for dealing with imprecision and non-linearity efficiently. Also, for real-time applications, its relatively low computational complexity makes it a good candidate. Therefore, fuzzy logic control has emerged as one of the most successful nonlinear control techniques.

Fuzzy Logic Controllers (FLC) are based on *if – then* rules integrating the valuable experiences of human. These rules use linguistic terms to describe systems. Fuzzy logic controllers have shown good performances on the controlling of the complex, ill-defined and uncertain systems.

Three types of fuzzy controllers are known as PI-, PD-, and PID-type controllers which behave somehow similarly to their classical counterparts. Fuzzy PI-type control is known to be more practical than fuzzy PD-type because it is difficult for the fuzzy PD to remove steady-state error. The fuzzy PI-type control is, however, known to give poor performance in transient response for higher order processes due to the internal integration operation. Theoretically, fuzzy PID type control should enhance the performance a lot.

During the building of the FLC, the important tasks are the structure identification and parameters tuning. The structure identification of the FLC includes the input-output variables of a controller, the rule base, the determination of the number of rules, the antecedent and consequent membership functions, the inference mechanism and the defuzzification method. The parameters tuning includes determining the optimal parameters of membership functions antecedent and consequent but also the scaling factors.

Several approaches have been presented to learn and tune the fuzzy rules to achieve the desired performance. These automatic methods may be devised into two categories of supervised and unsupervised learning by whether the teaching signal is needed or not.

In the supervised learning approach, at each time step, if the input-output training data can be acquired, the FLC can be tuned based on the supervised learning methods.

The other category contains reinforcement learning (RL) systems which are unsupervised learning algorithms with the self-learning ability. The RL-based FLCs are learning schemes which need a scalar response from the environment to provide the action performance, that value is easier to collect than the desired-output data pairs in the real application.

The basic idea of the reinforcement learning is to learn, through trial-and-error interaction with a dynamic environment which returns a critic, called reinforcement, In reinforcement learning or learning with a critic the received signal is a behavior sanction (positive, negative or neutral); this signal indicates what you have to do without saying how to do it. The agent uses this signal to determine a policy permitting to reach a long-term objective.

It exist several reinforcement learning algorithms. Some are based on policy iteration such as Actor Critic Learning (ACL) and others on value iterations such as Q-learning. In this work we are interested by the Q-learning, which can be thought of as a reward or a punishment, the control actions to determine desired changes in the control output that will increase the index of performance.

The Q-learning is the most used reinforcement learning algorithm in practice because of its simplicity and the proof of its convergence. In Q-learning algorithm, a Q-value is assigned to each state-action pair. The Q-value reflects the expected long-term reward by taking the corresponding action from the action set in the corresponding state in the state space.

Reinforcement learning techniques assume that, during the learning process, no supervisor is present to directly judge the quality of the selected control actions, and therefore, the final evaluation of process is only known after a long sequence of action. Also, the problem involves optimizing not only the direct reinforcement, but also the total amount of reinforcements the agent can receive in the future.

In this work, we investigate a method for tuning the parameters of the input membership functions and output singletons of fuzzy logic controllers by the Q-learning algorithm in order to minimize or maximize a given cost function of the closed-loop system.

In order to tune the membership functions of the FLC, we choose the vector of parameters to be optimized as universe of discourses of error, change of error and output singletons. To each parameter are associated several competing candidates and to each candidate, is associated a Q-value that is incrementally updated by the QLA. The learning process consists then in determining the best set of parameters, the one that will optimize the future reinforcements.

Thus, as the quantities are initially unknown, the fuzzy controller has to explore and test several actions. The exploration phase is often long. Though, as fuzzy rules are

interpretable and tuning parameters have physical meaning, this phase can be drastically reduced by introducing knowledge in the initial fuzzy controller.

The cost function, which quantifies the effectiveness of the fuzzy controller, is evaluated at the end of a step-response experiment. Without loss of generality, we take the sum of square error cost function or in other words, Integral Square Error (ISE). This cost function then is used to define the reward function of the QL algorithm which we consider it as a scalar reward function to assess the quality of the actions taken by the system.

In this study we propose a manipulated reward function, instead of using a scalar reward function for the Reinforcement QL algorithm to improve both the performance and convergence criteria of the mentioned algorithm which incorporates a fuzzy structure (fuzzy inference system) including more elaborate information about the rewards/punishments assigned to each action in each state which is being taken in each step time by the agent.

Firstly, we apply the proposed algorithm to two distinc second order linear systems, one with time delay and the other one without time delay, and obtain the corresponding unity step responses for the given systems. The obtained results obviously demonstrate a considerable improvement in the performance of the systems closed loop unity step responses in contrast with fuzzy controllers without any tunning schemes.

In the next step, in order to show the effectiveness of the proposed method in a real time case, we apply the algorithm to a nonlinear system. The system to be examined is considered to be an Inverted Pendulum and our goal is to balance it on a vertical position. The resulting simulations clarify that the balancing time considerably reduces in comparison with controlling the system with a non-tuned fuzzy controller scheme.

BULANIK MANTIK KONTROLÖRÜN TASARIMINDA KULLANILAN BULANIK Q-ÖĞRENME ALGORİTMASI

ÖZET

Geleneksel kontrol kuramında, modelin sistemin tüm özelliklerini her an için sağladığı varsayılarak, sisteme ilişkin matematiksel model üzerinden uygun bir kontrolör tasarlanır. Ancak, sistemin matematiksel modelini elde etmek, sistemin lineer olmaması ve/veya sistem parametrelerindeki belirsizlikler yüzünden zor veya çok zaman alıcıdır. Bu nedenle, sistem kontrol tasarımında yeni çeşitli yöntemler önerilmektedir. Son yıllarda, etkili bir gerçek-zaman tasarım yöntemi bulmak için, akıllı kontrol sistemlerini kullanan birçok çalışma yapılmıştır. Sinir ağları, bulanık mantık ve genetik algoritma gibi yapay zeka kapsamında olan ve analitik olmayan yöntemler bu alanda yer almaktadır.

Bulanık mantık bir matematiksel yaklaşımdır ki işlevi insan düşüncesinin anlam karmaşasını algılamak ve uzman bilimini hesaplanabilir biçimde sayısal veriye çevirmektir. Bulanık mantık ile modelleme ve kontrolün, belirsiz ve lineer olmayan sistemler ile ilgilenmek için, güçlü bir yöntem olduğu daha önceden görülmüştür. Ayrıca, gerçek-zaman uygulamalarında, bulanık mantık ile hesaplama işlemlerinin sinir ağlarına nispetle daha az karmaşık olmasından ötürü, daha güçlü bir seçim olarak önerilmektedir. Böylelikle, bulanık mantık ile kontrol lineer olmayan sistemler açısından en başarılı yöntem olarak düşünülebilir.

Bulanık Mantık Kontrolörleri "eğer-ise-o zaman" kurallarına uyarak insanın değerli tecrübelerini sistemlere uyarlamaktadır. Bu kurallar, öncelikle, dilsel değişkenleri kullanarak yapılacak işlemi ifade etmekte kullanılmaktadır. Bu tür kontrolörler karmaşık, eksik-tanımlanmış ve/veya belirsiz sistemlerde çok daha iyi bir başarım göstermektedir.

Bulanık kontrolörün üç özel tipi, PI-, PD- ve PID- tipi bulanık kontrolörler olup bunlar klasik PI, PD ve PID kontrolörlerine çok yakın davranışlar sergiledikleri gösterilmiştir. PD-tipi kotrolör ile sürekli hal hatasını ortadan kaldırmak zor olduğundan, pratikte, bulanık PI-tipi kontrolörler, bulanık PD-tipi kontrolörlere oranla daha fazla kullanılır. Ancak, bulanık PI-tipi kontrolörlerin yüksek mertebeli sistemlerin geçici hal yanıtları üzerindeki başarımı zayıftır. Teorik olarak, bulanık PID-tipi kontrolör yukarıda anlatılan sorunları gidermektedir.

Bulanık mantık kontrolörlerin kurallarını arzu edilen başarıma yaklaştırmak için birçok yaklaşım önerilmiştir. Bu yöntemler gözetimli veya gözetlenmemiş öğrenme olmak üzere iki sınıfta incelenebilir. Eğer giriş-çıkış eğitim verileri her adımda elde edilebiliyorsa, Bulanık Mantık Kontrolü gözetimli öğrenim yöntemine bağlı olarak ayarlanabilir. Denetlenmiş öğrenim yaklaşımının her adımında, eğer giriş-çıkış veriler elde edilebiliyorsa, Bulanık Mantık Kontrolü denetlenmiş öğrenim yöntemine bağlı olarak ayarlanabilir. Gözetimsiz öğrenme algoritmaları ise pekiştirilmiş

öğrenme sistemlerini kapsamaktadır ki bu sistemler kendi kendisini eğitmek gücüne sahiptirler.

Pekiştirilmiş öğrenme yöntemlerinin kullanıldığı Bulanık Mantık Kontrolörleri öğrenim yöntemi olarak davranış başarımını yükseltecek biçimde çevre ile etkileşimlerinden skalar yanıt almağa ihtiyaç duyarlar. Gerçek zaman uygulamalarında bu değerleri elde etmek istenen-çıktı verilerinin elde edilmesinden daha kolaydır.

Pekiştirmeli öğrenmenin temel fikri çevre ile etkileşimde "deneme ve yanılma" yaklaşımının kullanılmasıdır. Pekiştirmeli öğrenmede alınan her işaret bir pozitif, negatif veya nötr davranış onayıdır; yani, bu işaret hangi işlemin yapılması gerektiğini nasıl yapılacağını söylemeden gösterir. Uzun vadeli amaçlara ulaşabilmek için varsayılan kontrolör bu işareti prensip oluşturmak için kullanır.

Bir kaç pekiştirilmiş öğrenme algoritmaları mevcuttur. Bunlardan bazıları politika belirleme iterasyon prensipine uyarak (Actor Critic Learning gibi) diğer bazıları da değer atama iterasyon prensipine uyarak (QL gibi) çalışmaktalar.

Bu tez kapsamında Q-öğrenme algoritması ele alınarak bulanık kontrolörün başarımı artırılmaya çalışılmıştır. Bu amaçla, bir ceza veya ödül ataması ile öğrenim sağlanmaya çalışılır. Bu öğrenim ile bir başarım ölçütünü iyileştirmek üzere sistem çıkışında arzu edilen değişiklikler kontrol çıkış işareti üzerinden yapılır.

Q-öğrenme algoritması kolaylık ve iyi yakınmasından dolayı pratikte pekiştirilmiş öğrenme algoritmaları içinde en yaygın olandır. Q-öğrenme algoritmasında bir "Q-değeri" her durum-eylem çiftine atanmaktadır. Q-değeri beklenilen uzun-vadeli ödül değerini yansıtmaktadır.

Pekiştirilmiş öğrenim tekniğine göre, öğrenim aşamasında hiçbir yönetici veya uzman, seçilen kontrol faaliyetinin kalitesini değerlendirmek için bulunmamaktadır. Bu nedenle, prosedürün değerlendirmesi, sadece uzun bir faaliyet aşamasından sonra tanımlanıyor. Ayrıca problem sadece doğrudan takviye optimizasiyonu değil, belki operatörun(ajan) gelecekte kabul edebileceği tüm takviye'leri içeriyor.

Bu çalışmada, kapalı-çevrimli bir sistemin belli bir davranış ölçütünü maksimize veya minimize etmesi amacıyla, bulanık mantık kontrolörlerinin giriş ve çıkış üyelik fonksiyon parametrelerini, Q-öğrenme algoritmasına dayalı olarak ayarlayan bir yöntem önerilmektedir.

Bulanık Mantık Kontrolörün üyelik fonksiyonlarının parametrelerinin ayarlaması için optimize edilecek vektör parametreleri kontrolör girişi olan hata ve hatanın değişimi ve de çıkış olarak seçilmiştir.

Her üyelik fonksiyon parametresi için birbiri ile rekabette olan çeşitli adaylar ve her bir aday için bir Q-değeri tanımlanmıştır. Bu Q-değerleri adım adım Q-öğrenme algoritması tarafından güncelleştirilmektedir. Böylece öğrenim prosedürü, en iyi üyelik fonksiyon parametre takımını belirlemektedir. Böylece ilk başta, üyelik fonksiyon parametre değerleri belirsizken, bulanık kontrolörün çeşitli şartlar altında çalıştırılmak ve denenmek zorundadır.

Araştırma aşaması çoğu zaman uzundur. Ancak ayar parametreleri fiziksel bir anlam taşıyorsa bu faz kısaltılabilir. Davranış ölçütü kullanılarak farklı parametre değerleri ile elde edilen her basamak yanıtı sonunda bulanık kontrolörün etkinliği bir değer ile ölçülmüş olur. bu çalışmada davranış ölçütü olarak karesel hata integrali kullanılmıştır.

Bu çalışmada, literatürde ilk kez olarak Q-öğrenme algoritmasının ödül fonksiyonunda, skalar değerler ataması yerine bulanık çok değerli atama kullanılmıştır. Böylece öğrenme algoritması daha duyarlı hale gelmiş ve bunun sonucu olarak yakınsama hızlandırılmıştır.

Bulanık kontrolörün üyelik fonksiyon ayarlamasında oluşturulan bulanıklaştırılmış Q-öğrenme algoritması kullanıldığında sistem yanıtlarındaki hataların azaldığı ve davranış ölçütünün çok daha küçük değerlere ulaştığı görülmüştür.

İlk adımda, önerilen yöntem benzetim çalışmaları, iki farklı ikinci dereceden oluşan lineer sisteme uygulanmıştır. Birinci sistem zaman gecikmesi olan, diğeri ise zaman gecikmesi olmayan bir sistem.

Bir sonraki adımda, önerilen yöntemin etkisini göstermek için, bu algoritma lineer olmayan bir sisteme uygulanmıştır. Adı geçen algoritma lineer olmayan ters sarkaç modeli üzerine uygulanmıştır. Amacımız Sarkaçın dikey bir durum'da, dengesini sağlamaktır.

Sabit değerli Q-öğrenme algorirması kullanılan kontrolör, bulanıklaştırılmış Q-öğrenme algoritması kullanılan kontrolör ve ayarlanmamış bulanık kontrolör başarımları tüm sistemler üzerinde karşılaştırılmış ve bulanıklaştırılmış Q-öğrenme algoritması kullanılan kontrolörün başarımının en yüksek olduğu gözlenmiştir.

1. INTRODUCTION

The classical linear Proportional Integral Derivative Controllers (PIDCs) are widely used in industrial process control where over 90% of operating controllers are PIDCs [1]. This popularity is due to the simplicity of their structure and the familiarity of engineers and operators with PID algorithms. They are inexpensive and reasonably sufficient for many industrial control systems.

However, the classical PIDCs cannot provide good enough performances in controlling highly complex processes and/or when their parameters are badly tuned, practically when handling MIMO (multiple inputs, multiple outputs) systems. Following the first fuzzy control application carried out by Mamdani [2], fuzzy control [3, 4] has become, in the recent past, an alternative to conventional control algorithms to deal with complex processes and combine the advantages of classical controllers and human operator experience.

In the literature, various types of fuzzy PID (including PI and PD) controllers have been proposed. In general, the application of fuzzy logic to PID controller design can be classified into two major categories according to the way they are constructed [5]:

I. The gains of the conventional PID controller are tuned on-line in terms of the knowledge base and fuzzy inference, and then the conventional PID controller generates the control signal [6, 7].

II. A typical FLC is constructed as a set of heuristic control rules, and the control signal is directly deduced from the knowledge base and the fuzzy inference as it is done in McVicar-Whelan or diagonal rule-base generation approaches [8, 9, 10, 11].

The controllers in the second category are referred to as PID-type FLCs because, from the input–output relationship point of view, their structures are analogous to that of the conventional PID controller.

The formulation of PID-type FLC can be achieved either by combining PI- and PD-type FLCs with two distinct rule-bases or one PD-type FLC with an integrator and a summation unit at the output.

We can summarize the design parameters within two groups:

i. Structural parameters

ii. Tuning parameters

Basically, structural parameters include input/output (I/O) variables to fuzzy inference, fuzzy linguistic sets, membership functions, fuzzy rules, inference mechanism and defuzzification mechanism. Tuning parameters include I/O scaling factors (SF) and parameters of membership functions (MF). Usually the structural parameters are determined during off-line design while the tuning parameters can be calculated during on-line adjustments of the controller to enhance the process performance, as well as to accommodate the adaptive capability to system uncertainty and process disturbance.

The usage of fixed value scaling factors in the case of the real systems which featured nonlinearities, modeling errors, disturbances, parameter changes, etc., may not be sufficient to achieve the desired system performance. Therefore to overcome these kinds of disadvantages, a lot of heuristic and non-heuristic tuning algorithms for the adjustment of scaling factors of fuzzy controllers have been presented in literature [12, 13, 14, 15, 16, 17].

Besides the classical tuning methods, many other tuning methods for classical and fuzzy PID controllers are proposed in the literature; fuzzy supervisors [18, 19], genetic algorithms [20, 21, 22, 23], and the ant colony algorithms [24, 25]. All these methods are capable of generating the optimum or quasi-optimum parameters to the control system in a high-dimensional space, but at the cost of time.

Some popular approaches design the fuzzy systems from input–output data based on the following methods [26]:

i. a table look-up scheme

ii. gradient-descent training

iii. recursive least square

iv. clustering

Unfortunately, the input-output data may not be readily available, and the above methods may belong to supervised learning methods. Another widely used method is to implement adaptive laws based on the Lyapunov synthesis approach [27]. However, a disadvantage of adaptive fuzzy control is that it requires some model information of the system and it may only aim at a specific class of systems.

The automatic methods to learn and tune the fuzzy rules may be divided into two categories of supervised and unsupervised learning by whether the teaching signal is needed or not. In the supervised learning approach, at each time step, if the input-output training data can be acquired, the FLC can be tuned based on the supervised learning methods. The artificial neural network (ANN)-based FLC can automatically determines or modifies the structure of the fuzzy rules and parameters of fuzzy membership functions with unsupervised or supervised learning by representing a FLC in a connectionist way such as ANFIS or other methods [28, 29].

The other category contains genetic algorithm (GA) [30, 31] and reinforcement learning (RL) systems [32, 33], which are unsupervised learning algorithms with the self-learning ability [34].

The difference between the GA-based and RL-based FLCs lies in the manner of state-action space searching. The GA-based FLC is a population based approach that encodes the structure and/or parameter of each FLC into chromosomes to form an individual, and evolves individuals across generations with genetic operators to find the best one. The RL based FLC uses statistical techniques and dynamic programming methods to evaluate the value of FLC actions in the states of the world. However, the pure GA-based FLC cannot proceed to the next generation until the arrival of the external reinforcement signal that is not practical in real time applications. In contrast, the RL-based FLC can be employed to deal with the delayed reinforcement signal that appears in many situations [35].

Reinforcement learning techniques assume that, during the learning process, no supervisor is present to directly judge the quality of the selected control actions, and therefore, the final evaluation of process is only known after a long sequence of action. Also, the problem involves optimizing not only the direct reinforcement, but also the total amount of reinforcements the agent can receive in the future. This leads to the temporal credit assignment problem, i.e., how to distribute reward or punishment to each individual state-action pair to adjust the chosen action and improve its performance [36].

Supervised learning is more efficient than the reinforcement learning when the input-output training data are available [37, 38]. However, in most real-world application, precise training data is usually difficult and expensive to obtain or may not be available at all [39].

For the above reasons, reinforcement learning can be used to tune the fuzzy rules of fuzzy systems. Kaelbling, littman and Moore [40], and more recently Sutton and Barto [33], characterize two classes of methods for reinforcement learning: methods that search the space of value functions and methods that search the space of policies. The former class is exemplified by the temporal difference (TD) method and the latter by the genetic algorithm (GA) approach [41].

To solve the RL problem, the most approach is TD method [42, 43]. Two TD based RL approaches have been proposed the Adaptive Heuristic Critic (AHC) [44, 45, 46, 47].

The definition of controlable factors for the FPID in the reinforcemnt learning algorithms which has been led on discrete levels have been proposed in [48, 49] which are kept unchanged during learning. This is the basic idea of the most existing algorithms, which adopt Q-learning. In general, however, this is not case and then the algorithm may fail. Indeed, for complex systems like static converter, *priori* knowledge are not available, then it becomes difficult to determine a set of parameters in which figure the optimal controlable factors for each fuzzy rule and thus the FPID controller can’t accomplish the given task through FQL. To ensure a fine optimization of the FLC a continuous RL algorithm has been proposed [50].

This thesis is concentrated on the subject of tuning the fuzzy logic controllers by utilizing an unsupervised learning algorithm named reinforcement Q-learning. The proposed optimization method is applied to the input and output membership functions of the main control block, which here is a fuzzy controller, and tries to tune the universe of discourses of the mentioned membership functions.

In order to improve the performance of the Q-learning algorithm an advanced reward function using fuzzy logic is proposed.

For demonstrating the effectiveness of the Q-learning algorithm, it is applied to both linear (with and without time delay) and nonlinear systems. The obtained results are compared simultaneously with that of non-tuned fuzzy controller and also GA method.

The rest of the study is divided into 5 sections. In section 2, we firstly define some preliminaries of fuzzy logic and then extend the subject to the fuzzy controllers and introduce the structure of the standard PID-type fuzzy logic controller. In section 3, the concept of reinforcement learning and Q-learning algorithm is presented. Their application to fuzzy logic control is revealed in the final part of this section. Discussions and related computer simulations are incorporated in section 4. Finally in section 5, appropriate conclusions are drawn and future work is given.

2. FUZZY LOGIC AND CONTROL

One of the more popular new technologies is intelligent control which is defined as a combination of control theory, operations research, and artificial intelligence (AI).

To understand fuzzy logic, it is important to discuss fuzzy sets. In 1965, Zadeh [51] wrote a seminal paper in which he introduced fuzzy sets, nearly 10 years later the fuzzy control technique was first initiated by the pioneering research of Mamdani [52, 53, 54] for systems structurally difficult to model, which was motivated by Zadeh's paper on system analysis with fuzzy sets theory. Since then, fuzzy control theory has become one of the most extensively studied areas in both academia and industry. Many theoretical developments and practical applications have been reported. It is an appealing alternative to conventional control methods since it provides a systematic and efficient framework to deal with uncertainties and nonlinearities in complex systems, when an accurate system analytical model is not available, not possible to obtain, or too complicated to use for control purposes.

Today, in Japan, United States, Europe, Asia, and many other parts of the world, fuzzy control is widely accepted and applied. In many consumer products such as washing machines and cameras, fuzzy controllers are used to obtain intelligent machines (Machine Intelligence Quotient) and user-friendly products.

A few interesting applications can be cited: control of subway systems, image stabilization of video cameras, image enhancement, and autonomous control of helicopters.

2.1 Fuzzy Logic

2.1.1 Fuzzy sets

Zadeh introduced fuzzy set theory as a mathematical discipline, although the underlying ideas had already been recognized earlier by philosophers and logicians.

A broader interest in fuzzy sets started in the seventies with their application to control and other technical disciplines.

In ordinary (non-fuzzy) set theory, elements either fully belong to a set or are fully excluded from it. Recall, that the membership $\mu_A(x)$ of x of a classical set *A*, as a subset of the universe *X*, is defined by:

$$\mu_A(x) = \begin{cases} 1, iff \;\; x \in A \\ 0, iff \;\; x \notin A \end{cases} \qquad \textbf{(2.1)}$$

This means that an element x is either a member of set *A* ($\mu_A(x)=1$) or not ($\mu_A(x)=0$). This strict classification is useful in the mathematics and other sciences that rely on precise definitions. Ordinary set theory complements bi-valent logic in which a statement is either true or false. While in mathematical logic the emphasis is on preserving formal validity and truth under any and every interpretation, in many real-life situations and engineering problems, the aim is to preserve information in the given context. In these situations, it may not be quite clear whether an element belongs to a set or not.

A fuzzy set is a set with graded membership in the real interval:

$$\mu_A(x) \in [1,0] \qquad \textbf{(2.2)}$$

That is, elements can belong to a fuzzy set to a certain degree. As such, fuzzy sets sets can be used for mathematical representations of vague concepts, such as *low temperature*, *fairly tall person*, *expensive car*, etc.

By definition a fuzzy set *A* on universe (domain) *X* is a set defined by the membership function $\mu_A(x)$ which is a mapping from the universe *X* into the unit interval:

$$\mu_A(x)\text{: X} \rightarrow [1,0] \qquad \textbf{(2.3)}$$

A is completely characterized by the set of pairs:

$$\text{A} = \{(x, \mu_A(x)), x \in X\} \qquad \textbf{(2.4)}$$

When *X* is a finite set $\{x_1, \dots, x_n\}$, a fuzzy set on *X* is expressed as:

$$\mu_A(x_1)/x_1 + \dots + \mu_A(x_n)/n \;=\; \textstyle\sum_{i=1}^{i=n} \mu_A(x_i)/x_i \qquad \textbf{(2.5)}$$

When *X* is not finite, we write:

$$A=\int \mu_A(x)/x \qquad \textbf{(2.6)}$$

2.1.2 Operations on fuzzy sets

Definitions of set-theoretic operations such as the complement, union and intersection can be extended from ordinary set theory to fuzzy sets. As membership degrees are no longer restricted to {0, 1} but can have any value in the interval [0, 1], these operations cannot be uniquely defined. It is clear, however, that the operations for fuzzy sets must give correct results when applied to ordinary sets (an ordinary set can be seen as a special case of a fuzzy set).

2.1.2.1 Complement of a fuzzy set

Let A be a fuzzy set in X. The complement of A is a fuzzy set, denoted $\bar{A}$, such that for each $x \in X$:

$$\mu_{\bar{A}}(x) = 1- \mu_A(x) \qquad \textbf{(2.7)}$$

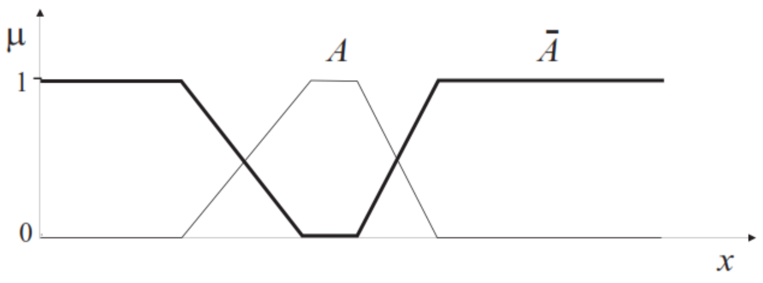


Figure 2.1 : Fuzzy set and complement of it.

2.1.2.2 Intersection of fuzzy sets

Let *A* and *B* be two fuzzy sets in *X*. The intersection of *A* and *B* is a fuzzy set *C*, denoted $C = A \cap B$, such that for each $x \in X$:

$$\mu_c(x) = \min[\mu_A(x), \mu_B(x)] \qquad \textbf{(2.8)}$$

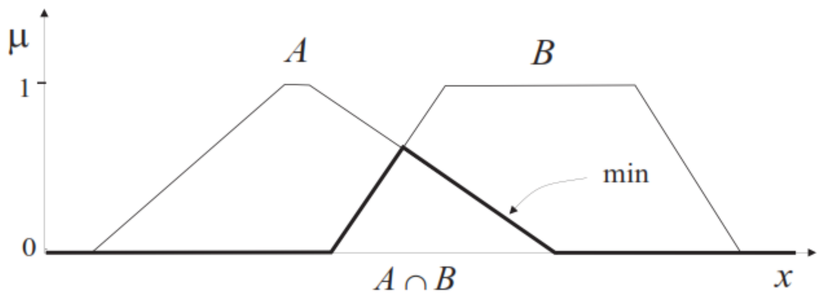


Figure 2.2 : Fuzzy intersection.

2.1.2.3 Union of fuzzy sets

Let A and B be two fuzzy sets in X. The union of A and B is a fuzzy set C, denoted C = A ∪ B, such that for each x ∈ X:

$$\mu_c(x) = \max[\mu_A(x), \mu_B(x)] \tag{2.9}$$

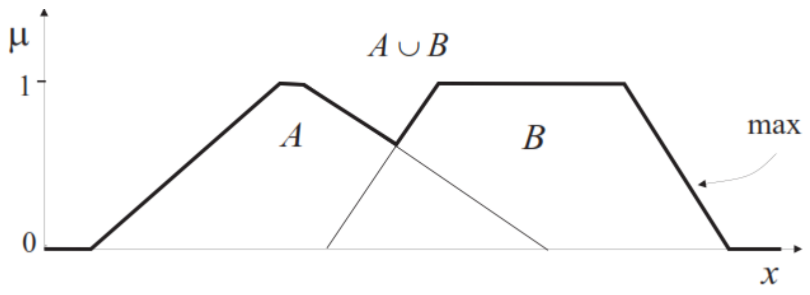


Figure 2.3 : Fuzzy union.

2.1.3 Membership functions

MFs are fuzzy sets that can be represented through mathematical expressions. In general, fuzzy sets can be of triangular, trapezoidal, Gaussian, or other form. In the transition regions, its MF parts can be linear, quadratic, or exponential, depending on the object of interest.

The simplest MF has a triangular shape, which is a function of a support vector x and depends on three scalar parameters (a, b, and c).

The MF is given by the following mathematical expression:

$$\mu(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ \frac{c-x}{c-b} & b \leq x \leq c \\ 0 & c \leq x \end{cases} \tag{2.10}$$

or, more compactly by:

$$\mu(x) = \text{Max}\left[\text{Min}\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right] \tag{2.11}$$

The parameters *a* and *c* locate the left and right limits of the support of the set and the parameter *b* locates the core.

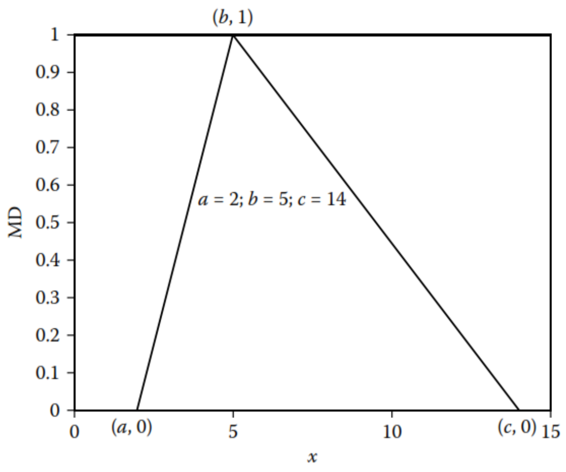


Figure 2.4 : Triangular membership function.

The trapezoidal MF is a function of a support vector x and depends on four scalar parameters (*a*, *b*, *c*, and *d*).

The mathematical expression of this MF is given by:

$$\mu(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ 1 & b \leq x \leq c \\ \frac{d-x}{d-c} & c \leq x \leq d \\ 0 & d \leq x \end{cases} \tag{2.12}$$

or, more compactly by:

$$\mu(x) = \text{Max}\left[\text{Min}\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right] \tag{2.13}$$

The parameters *a* (*b*) and *d* (*c*) locate the left and right limits of the support. It is obvious that the triangular MF is a special case of the trapezoidal MF when *b* = *c*;

and if $a = b = c = d$, then a fuzzy singleton is obtained. Furthermore, a, b, c, and d represent lower modal, left spread, upper modal, and right spread, respectively.

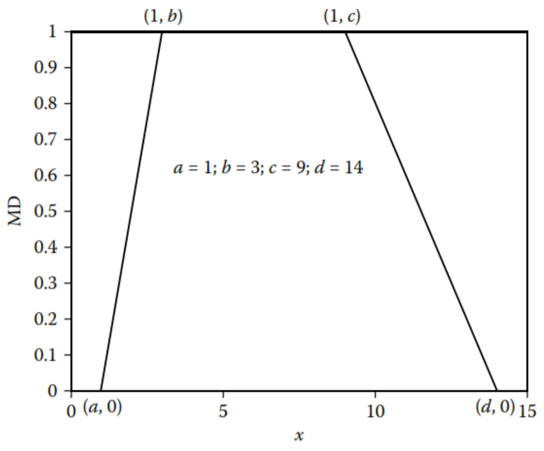


Figure 2.5 : Trapezoidal membership function.

The generalized bell MF depends on three parameters (a, b, and c) as given by:

$$f(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}} \quad \textbf{(2.14)}$$

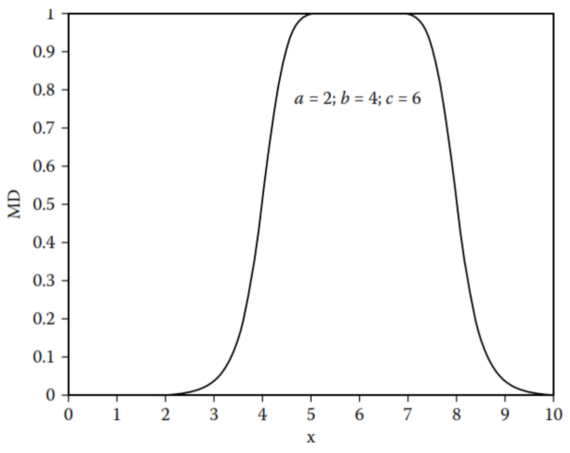


Figure 2.6 : Bell-shaped membership function.

The S-shape MF is defined by three parameters (a, b, and c) and its general shape is shown in Figure 2.6. The mathematical form of an S-shape MF is given as:

$$f(x; a, b, c) = \begin{cases} 0 & x < a \\ 2(\frac{x-a}{c-b})^2 & a < x < b \\ 1 - 2\left(\frac{x-a}{c-b}\right)^2 & b < x < c \\ 1 & x < c \end{cases} \quad \textbf{(2.15)}$$

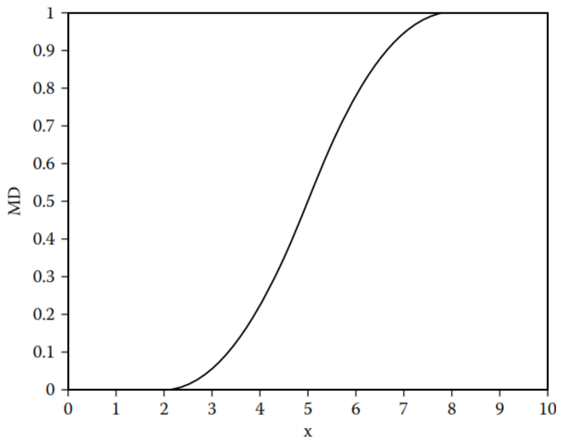


Figure 2.7 : S-shaped membership function.

The Z-shape MF is the asymmetrical polynomial curve open to the left as shown in Figure 2.7 and its mathematical function has the following form:

$$f(x; a, b, c) = \begin{cases} 0 & x < a \\ 1 - 2\left(\frac{x-a}{c-b}\right)^2 & a < x < b \\ 2\left(\frac{x-a}{c-b}\right)^2 & b < x < c \\ 1 & x < c \end{cases} \tag{2.16}$$

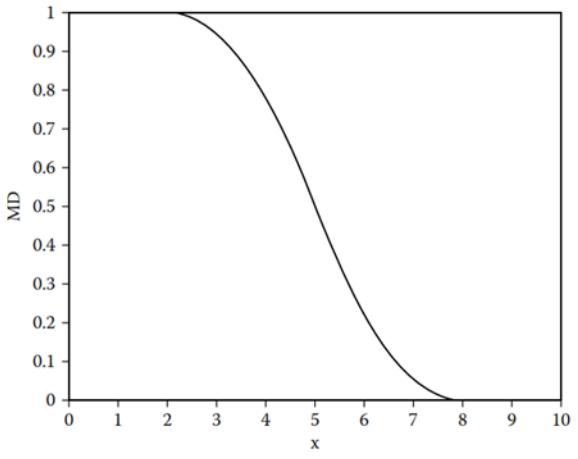


Figure 2.8 : Z-shaped membership function.

2.2 Fuzzy Systems and Control

A static or dynamic system which makes use of fuzzy sets and of the corresponding mathematical framework is called a *fuzzy system*. Fuzzy systems can serve different purposes, such as modeling, data analysis, prediction or control.

Fuzzy sets can be involved in a system in a number of ways, such as:

- In the description of the system: A system can be defined, for instance, as a collection of if-then rules with fuzzy predicates, or as a fuzzy relation.
- In the specification of the system's parameters: The system can be defined by an algebraic or differential equation, in which the parameters are fuzzy numbers instead of real numbers.
- The input, output and state variables of a system may be fuzzy sets. Fuzzy inputs can be readings from unreliable sensors ("noisy" data), or quantities related to human perception, such as comfort, beauty, etc. Fuzzy systems can process such as information, which is not the case with conventional (crisp) systems.

2.2.1 Structural design techniques for fuzzy systems

Two common sources of information for building fuzzy systems are *prior knowledge* and *data* (measurements). Prior knowledge tends to be of a rather approximate nature (qualitative knowledge, heuristics), which usually originates from "experts", i.e., process designers, operators, etc. In this sense, fuzzy models can be regarded as simple *fuzzy expert systems*

Two main approaches to the integration of knowledge and data in a fuzzy model can be distinguished:

1) The expert knowledge expressed in a verbal form is translated into a collection of if–then rules. In this way, a certain model structure is created. Parameters in this structure can be fine-tuned using input-output data.
2) No prior knowledge about the system under study is initially used to formulate the rules, and a fuzzy model is constructed from data. It is expected that the extracted rules and membership functions can provide an a posteriori interpretation of the system's behavior. An expert can confront this

information with his own knowledge, can modify the rules, or supply new ones, and can design additional experiments in order to obtain more informative data. This approach can be termed rule extraction.

2.2.2 Fuzzy controller and its architecture

A fuzzy controller is a controller that contains a (nonlinear) mapping that has been defined by using fuzzy if-then rules.

The general form of the linguistic rules is:

If premise **Then** consequent

The premises (which are sometimes called “antecedents”) are associated with the fuzzy controller inputs and are on the left-hand-side of the rules. The consequents (sometimes called “actions”) are associated with the fuzzy controller outputs and are on the right-hand-side of the rules. Notice that each premise (or consequent) can be composed of the conjunction of several “terms” (e.g.“error is poslarge and change-in-error is negsmall” is a premise that is the conjunction of two terms).

A block diagram of a fuzzy control system is shown in Figure 2.9. The fuzzy controller is composed of the following four elements:

1) A rule-base (a set of If-Then rules), which contains a fuzzy logic quantification of the expert’s linguistic description of how to achieve good control.
2) An inference mechanism which emulates the expert’s decision making in interpreting and applying knowledge about how best to control the plant.
3) A fuzzification interface, which converts controller inputs into information that the inference mechanism can easily use to activate and apply rules.
4) A defuzzification interface, which converts the conclusions of the inference mechanism into actual inputs for the process.

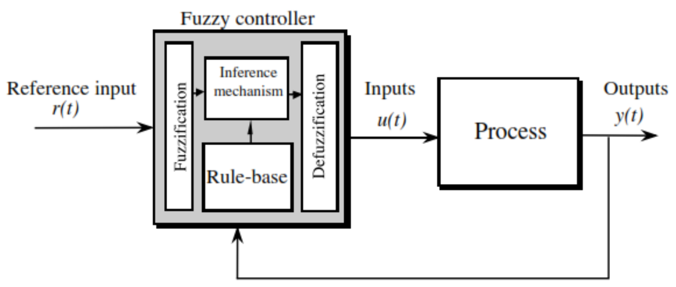


Figure 2.9 : Fuzzy controller architecture.

2.2.2.1 Fuzzification

Fuzzification is the process of decomposing a system input and/or output into one or more fuzzy sets. Many types of curves can be used, but triangular or trapezoidal shaped membership functions are the most common because they are easier to represent in embedded controllers. Each fuzzy set spans a region of input (or output) value graphed with the membership. Any particular input is interpreted from this fuzzy set and a degree of membership is interpreted. The membership functions should overlap to allow smooth mapping of the system. The process of fuzzification allows the system inputs and outputs to be expressed in linguistic terms so that rules can be applied in a simple manner to express a complex system.

2.2.2.2 Rule base

The human operator can be replaced by a combination of a fuzzy rule-based system (FRBS). The input sensory (crisp or numerical) data are fed into the FRBS, where physical quantities are represented or compressed into linguistic variables with appropriate membership functions. These linguistic variables are then used in the antecedents (IF part) of a set of fuzzy rules within an inference engine to result in a new set of fuzzy linguistic variables, or the consequent (THEN part).

2.2.2.3 Inference mechanism

The cornerstone of any expert controller is its inference engine, which consists of a set of expert rules, which reflect the knowledge base and reasoning structure of the solution of any problem.

Depending on the consequent of the fuzzy rule, the inference mechanism can be categorized into two commonly used methods:

Mamdani Inference Mechanism: Under Mamdani rules, the antecedents and the consequent parts of the rule are expressed using linguistic labels.

Inputs $x_1$and x_2 are crisp values and the *max-min* inference method is used. Based on the Mamdani implication method of inference, the aggregated output for the rules will be given by:

$$\mu_{B^i}(y) = \mathrm{Max}\left[\mathrm{Min}\left[\mu_{A_1{}^i}(x_1), \mu_{2^i}(x_2)\right]\right] \text{ for i = 1, 2,…, l} \qquad \textbf{(2.17)}$$

If the max-product inference method is used the aggregated output for the rules will be given by:

$$\mu_{B^i}(y) = \mathrm{Max}\left[\mu_{A_1{}^i}(x_1).\, \mu_{2^i}(x_2)\right] \text{ for i = 1, 2,…, l} \qquad \textbf{(2.18)}$$

Figure 2.10 and 2.11 show graphical illustration of Mamdani-type rules using max-min and max-product inference, respectively, for l = 2, where $A_1$and A_2 refer to the first and second fuzzy antecedents of the first rule and B_1 refers to the fuzzy consequent of the first rule. Similarly, A_1 and A_2 refer to the first and second fuzzy antecedents of the second rule, respectively, and B_2 refers to the fuzzy consequent of the second rule.

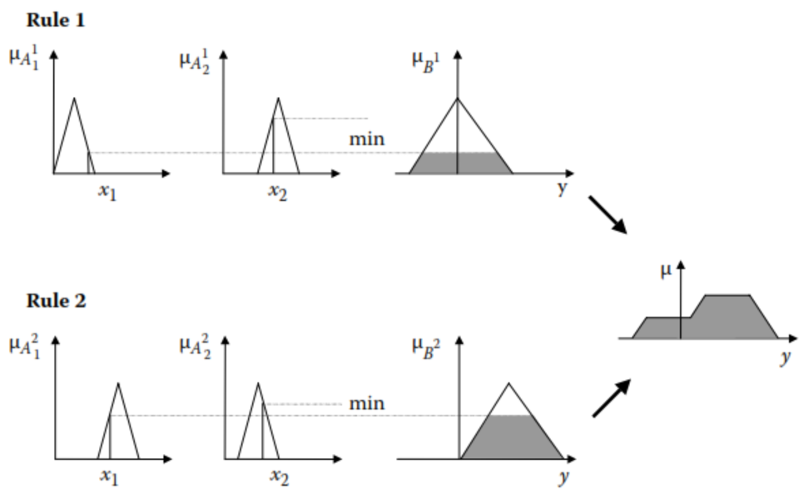


Figure 2.10 : Graphical execution of a min-max inference in a Mamdani rule.

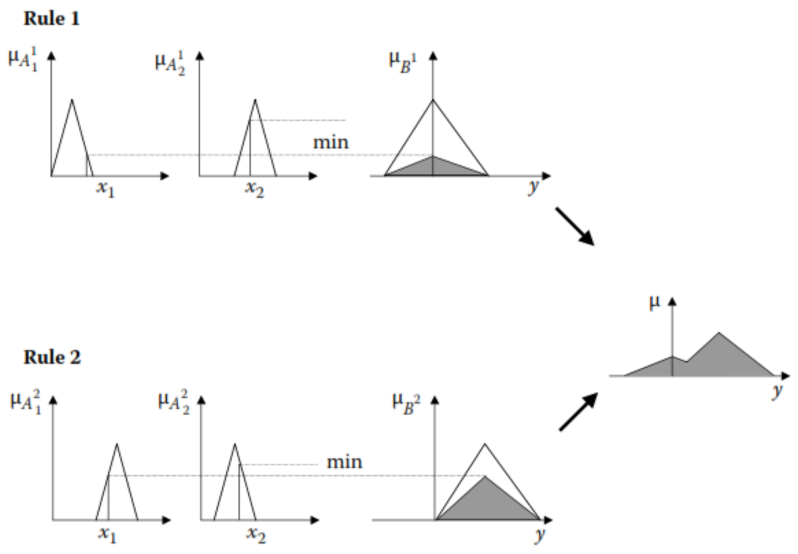
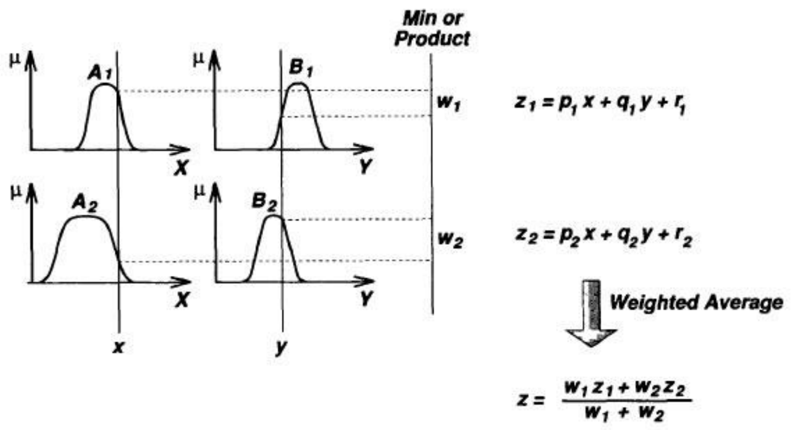


Figure 2.11 : Graphical execution of a min-product inference in a Mamdani rule.

Takagi-Sugeno Inference Mechanism: Another form is Takagi-Sugeno rules, under which the consequent part is expressed as an analytical expression or equation. Figure 2.11 represents a graphical illustration of Takagi-Sugeno inference method.



Min or Product
μ A_1 X
μ B_1 Y
w_1
$z_1 = p_1 x + q_1 y + r_1$
μ A_2 X
μ B_2 Y
w_2
$z_2 = p_2 x + q_2 y + r_2$
Weighted Average
x
y
$z = \frac{w_1 z_1 + w_2 z_2}{w_1 + w_2}$

Figure 2.12 : Takagi-Sugeno inference mechanism.

2.2.2.4 Defuzzification

The result of fuzzy inference is the fuzzy set B′. If a crisp (numerical) output value is required, the output fuzzy set must be defuzzified. Defuzzification is a transformation that replaces a fuzzy set by a single numerical value representative of that set. Figure 2.12 shows one of the most commonly used defuzzification methods: the center of gravity (CoG).

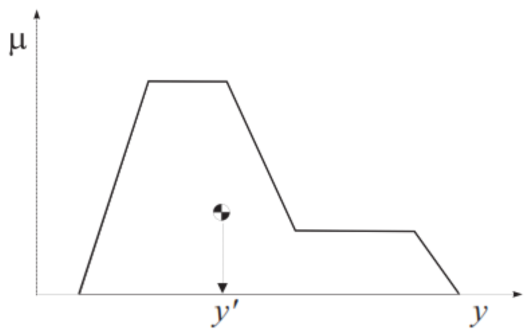


Figure 2.13 : The center-of-gravity defuzzification method.

The COG method calculates numerically the y coordinate of the center of gravity of the fuzzy set B'.

$$y' = \frac{\sum_{j=1}^{F} \mu_{B'}(y_j) y_j}{\sum_{j=1}^{F} \mu_{B'}(y_j)} \tag{2.19}$$

Where F is the number of elements y_j in Y. Continuous domain Y thus must be discretized to be able to compute the center of gravity.

2.2.3 Fuzzy control design

One of the first steps in the design of any fuzzy controller is to develop a knowledge base for the system to eventually lead to an initial set of rules. There are at least five different methods to generate a fuzzy rule base [55].

1) Simulate the closed-loop system through its mathematical model.
2) Interview an operator who has had many years of experience controlling the system.
3) Generate rules through an algorithm using numerical input/output data of the system.
4) Use learning or optimization methods such as neural networks or genetic algorithms to create the rules.
5) In the absence of all of the above, if a system does exist, experiment with it in the laboratory or factory setting and gradually gain enough experience to create the initial set of rules.

2.2.3.1 Standard PID-type fuzzy logic controller

A suitable choice of control variables is important in fuzzy control design. Typically, the inputs to the fuzzy controllers are the error and the change of error. This choice is physically related to classical PID controllers. Usually, a fuzzy controller is either a PD-or a PI-type depending on the output of fuzzy control rules; if the output is the control signal it is said to be PD-type fuzzy controller (FPDC) and if the output is the change of control signal it is said to be PI-type fuzzy controller (FPIC) [56, 57].

Fuzzy PI-type control is known to be more practical than fuzzy PD-type because it is difficult for the fuzzy PD to remove steady-state error. The fuzzy PI-type control is, however, known to give poor performance in transient response for higher order processes due to the internal integration operation. Theoretically, fuzzy PID-type control should enhance the performance a lot. It should be pointed out that, for fuzzy PID controllers, normally a 3-D rule base is required. This is difficult to obtain since 3-D information is usually beyond the sensing capability of a human expert. To obtain proportional, integral and derivative control action all together, it is intuitive and convenient to combine PI and PD actions together to form a fuzzy PID controller. The formulation of PID-type FLC can be achieved either by combining PI- and PD-type FLCs with two distinct rule-bases or one PD-type FLC with an integrator and a summation unit at the output.

A fuzzy PID controller is inherently a piecewise linear PID controller by the fuzzy rule base establishment and fuzzy inference mechanism. It has a better control capability than a conventional linear PID controller within the overall operating range, especially for nonlinear systems, since the nonlinear terms are initially included in the fuzzy rule table construction. In the region close to the origin, a fuzzy PID controller functionally behaves approximately as a linear PID controller. When the system cannot be represented by an explicit analytical model, a fuzzy PID control will provide its superior, effective performance and the ease of implementation for real-time application. Compared to other complicated fuzzy control strategies, a fuzzy PID controller has a simpler structural configuration and hence is more practically useful.

Here we consider a standard fuzzy PID-type controller structure as it is shown in Figure 2.14.

The output of the fuzzy PID-type controller is given by:

$$u = \alpha U + \beta \int U\,dt \tag{2.20}$$

Where U is the output of the FLC. It has been shown in [16, 58] that the fuzzy controllers with product–sum inference method, center of gravity defuzzification method and triangular uniformly distributed membership functions for the inputs and a crisp output, the relation between the input and the output variables of the FLC is given by:

$$U = A + PE + D\dot{E} \tag{2.21}$$

Where $E = k_e e$ and $\dot{E} = k_d \dot{e}$. Therefore, from (2.20) and (2.21) the controller output is obtained as:

$$u = \alpha A + \beta At + \alpha k_e Pe + \beta k_d De + \beta k_e P \int e dt + \alpha k_d \dot{e} \tag{2.22}$$

Thus, the equivalent control components of the PID-type FLC are obtained as follows:

Proportional gain: $\alpha k_e P + \beta k_d D$

Integral gain: $\beta k_e P$

Derivative gain: $\alpha k_d D$

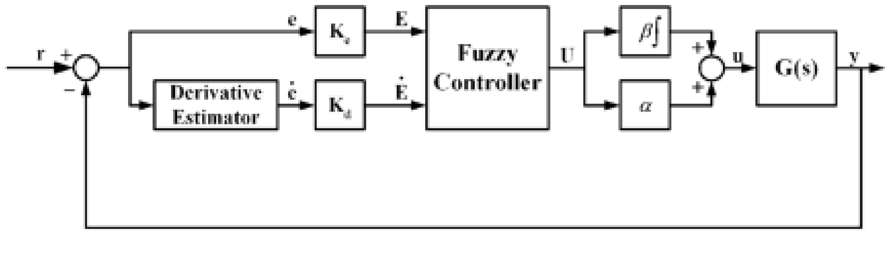


Figure 2.14 : Closed loop control structure of standard PID-type FLC.

3. REINFORCEMENT LEARNING AND DYNAMIC PROGRAMMING

In dynamic programming (DP) and reinforcement learning (RL), a controller (agent, decision maker) interacts with a process (environment), by means of three signals: *a state signal*, which describes the state of the process, *an action signal*, which allows the controller to influence the process, and *a scalar reward signal*, which provides the controller with feedback on its immediate performance. At each discrete time step, the controller receives the state measurement and applies an action, which causes the process to transition into a new state. A reward is generated that evaluates the quality of this transition. The controller receives the new state measurement, and the whole cycle repeats. This flow of interaction is represented in Figure 3.1

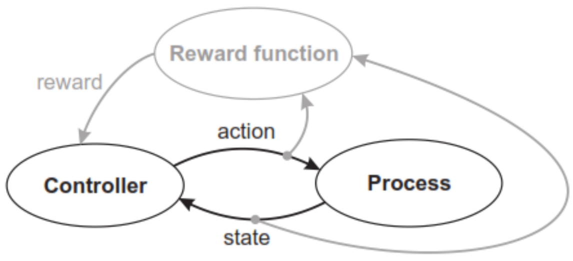


Figure 3.1 : The flow of interaction in DP and RL.

The DP/RL framework can be used to address problems from a variety of fields, including, e.g., automatic control, artificial intelligence, operations research, and economics. Automatic control and artificial intelligence are arguably the most important fields of origin for DP and RL. In automatic control, DP can be used to solve nonlinear and stochastic optimal control problems [59], while RL can alternatively be seen as adaptive optimal control. In artificial intelligence, RL helps to build an artificial agent that learns how to survive and optimize its behavior in an unknown environment, without requiring prior knowledge. Because of this mixed inheritance, two sets of equivalent names and notations are used in DP and RL, e.g., “controller” has the same meaning as “agent,” and “process” has the same meaning as “environment.”

DP algorithms require a model of the environment, including the transition dynamics and the reward function, to find an optimal policy. The model DP algorithms work offline, producing a policy which is then used to control the process. Usually, they do not require an analytical expression of the dynamics. Instead, given a state and an action, the model is only required to generate a next state and the corresponding reward. Constructing such a generative model is often easier than deriving an analytical expression of the dynamics, especially when the dynamics are stochastic.

RL algorithms are model-free [60, 33] which makes them useful when a model is difficult or costly to construct. RL algorithms use data obtained from the process, in the form of a set of samples, a set of process trajectories, or a single trajectory. So, RL can be seen as model-free, sample based or trajectory based DP, and DP can be seen as model-based RL. While DP algorithms can use the model to obtain any number of sample transitions from any state-action pair, RL algorithms must work with the limited data that can be obtained from the process – a greater challenge. Note that some RL algorithms build a model from the data; we call these algorithms "model-learning.

3.1 Deterministic Setting

A deterministic setting for RL is defined by the *state space* X of the process, the *action space* U of the controller, the *transition function* f of the process (which describes how the state changes as a result of control actions), and the *reward function* r (which evaluates the immediate control performance). As a result of the action $\mathrm{u_k}$ applied in the state $\mathrm{x_k}$ at the discrete time step *k*, the state changes to $\mathrm{x_{k+1}}$, according to the transition function f: X × U →X

$$\mathrm{x_{k+1}} = \mathrm{f}(\mathrm{x_k}, \mathrm{u_k}) \qquad \textbf{(3.1)}$$

At the same time, the controller receives the scalar reward signal r_{k+1}, according to the reward function r: X ×U →R

$$\mathrm{r_{k+1}} = \rho(x_k, u_k) \qquad \textbf{(3.2)}$$

The reward evaluates the immediate effect of action u_k namely the transition from x_k to x_{k+1}, but in general does not say anything about its long-term effects. The controller chooses actions according to its policy h: X →U, using:

$$u_k = \mathrm{h}(x_k) \qquad \textbf{(3.3)}$$

3.1.1 Optimality in the deterministic setting

In DP and RL, the goal is to find an optimal policy that maximizes the return from any initial state x_0. The return is a cumulative aggregation of rewards along a trajectory starting at x_0. It concisely represents the reward obtained by the controller in the long run. Several types of return exist, depending on the way in which the rewards are accumulated. The infinite-horizon discounted return is given by:

$$R^h(x_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} \qquad \textbf{(3.4)}$$

Where $\gamma \in [0, 1)$ is the *discount factor*. The discount factor can be interpreted intuitively as a measure of how "far-sighted" the controller is in considering its rewards, or as a way of taking into account increasing uncertainty about future rewards. From a mathematical point of view, discounting ensures that the return will always be bounded if the rewards are bounded. The goal is therefore to maximize the long-term performance (return), while only using feedback about the immediate, one-step performance (reward). This leads to the so-called challenge of delayed reward, actions taken in the present affect the potential to achieve good rewards far in the future, but the immediate reward provides no information about these long-term effects.

3.1.2 Value functions and the Bellman equations in the deterministic setting

A convenient way to characterize policies is by using their value functions. Two types of value functions exist: state-action value functions (Q-functions) and state value functions (V-functions). The Q-function Q^{h}: X ×U → R of a policy *h* gives the return obtained when starting from a given state, applying a given action, and following *h* thereafter:

$$Q^h(x,u) = \rho(x,u) + \gamma R^h\big(f(x,u)\big) \qquad \textbf{(3.5)}$$

Here, $R^h\big(f(x,u)\big)$ is the return from the next state $f(x,u)$.

The optimal Q-function is defined as the best Q-function that can be obtained by any policy:

$$Q^*(x,u) = \max Q^h(x,u) \qquad \textbf{(3.6)}$$

Any policy h^* that selects at each state an action with the largest optimal Q-value, i.e., that satisfies:

$$h^*(x) \in \arg\max Q^*(x,u) \qquad \textbf{(3.7)}$$

is optimal (it maximizes the return). The Q-functions Q^h and Q^* are recursively characterized by the Bellman equations, which are of central importance for value iteration algorithms The Bellman equation for Q^h states that the value of taking action *u* in state *x* under the policy *h* equals the sum of the immediate reward and the discounted value achieved by *h* in the next state:

$$Q^h(x,u) = \rho(x,u) + \gamma Q^h\Big(f(x,u), h\big(f(x,u)\big)\Big) \qquad \textbf{(3.8)}$$

The Bellman optimality equation characterizes Q^*, and states that the optimal value of action *u* taken in state *x* equals the sum of the immediate reward and the discounted optimal value obtained by the best action in the next state:

$$Q^*(x,u) = \rho(x,u) + \gamma max_{u'} Q^*(f(x,u),u') \qquad \textbf{(3.9)}$$

3.2 Model-Free Value Iteration and the Need for Exploration

Here we consider RL, model-free value iteration algorithms, and discuss Q-learning, the most widely used algorithm from this class. Q-learning starts from an arbitrary initial Q-function Q_0 and updates it without requiring a model, using instead observed state transitions and rewards, i.e., data tuples of the form (x_k, u_k,

x_{k+1},r_{k+1}) [46, 47]. After each transition, the Q-function is updated using such a data tuple, as follows:

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k[r_{k+1} + \gamma max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k)] \quad \textbf{(3.10)}$$

Where $\alpha_k \in (0, 1]$ is the learning rate. The term between square brackets is the temporal difference, i.e., the difference between the updated estimate $r_{k+1} + \gamma max_{u'} Q_k(x_{k+1}, u')$ of the optimal Q-value of (x_k, u_k) and the current estimate $Q_k(x_k, u_k)$.

3.3 Convergence

As the number of transitions *k* approaches infinity, Q-learning asymptotically converges to Q^*if the state and action spaces are discrete and finite, and under the following conditions [47, 62, 63]:

- The sum $\sum_{k=0}^{\infty} {\alpha_k}^2$ produces a finite value, whereas the sum $\sum_{k=0}^{\infty} \alpha_k$ produces an infinite value.
- All the state-action pairs are (asymptotically) visited infinitely often.

In practice, the learning rate schedule may require tuning, because it influences the number of transitions required by Q-learning to obtain a good solution. A good choice for the learning rate schedule depends on the problem at hand.

The second condition can be satisfied if, among other things, the controller has a nonzero probability of selecting any action in every encountered state; this is called *exploration*. The controller also has to *exploit* its current knowledge in order to obtain good performance, e.g., by selecting *greedy actions* in the current Q-function. This is a typical illustration of the exploration-exploitation trade-off in online RL.

A classical way to balance exploration with exploitation in Q-learning is *ε-greedy exploration* [33], which selects actions according to:

$$u_k = \begin{cases} u \in argmax_{u'} Q_k(x_k, u_k) & \text{probability } 1 - \varepsilon_k \\ \text{A uniformly random action in U} & \text{probability } \varepsilon_k \end{cases} \quad \textbf{(3.11)}$$

Where $\varepsilon_k \in (0, 1)$, is the exploration probability at step *k*.

3.4 Q-learning Algorithm

```
1) initialize Q-function $Q_0 \leftarrow 0$
2) measure initial state $x_0$
3) **for** every time step $k$ = 0,1,2, . . . **do**
4) $u_k = \begin{cases} \mathrm{u} \in \mathrm{argmax}_{\mathrm{u}'} \mathrm{Q}_\mathrm{k}(\mathrm{x}_\mathrm{k}, \mathrm{u}_\mathrm{k}) & \text{probability } 1 - \varepsilon_\mathrm{k} \text{ (exploit)} \\ \text{random action selection} & \text{probability } \varepsilon_\mathrm{k} \text{ (explore)} \end{cases}$
5) apply $u_k$, measure next state $x_{k+1}$and reward $r_{k+1}$
6) $Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k[r_{k+1} + \gamma max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k)]$
7) **end for**
```

Greedy action selection leads to exploring novel actions. Confidence intervals for the returns can be estimated, and the action with largest upper confidence bound, i.e., with the best potential for good returns, can be chosen.

The idea of RL can be generalized into a model, in which there are two components: an agent that makes decisions and an environment in which the agent acts. For every time step t, the agent is in a state $x_k \in$ X where X is the set of all possible states, and in that state the agent can take an action $u_k \in$ U where U is the set of all possible actions in state x_k. As the agent transits to a new state $\mathrm{x}_{\mathrm{k}+1}$ at time $t + 1$ it receives a numerical reward r_{k+1}. It up to date then its estimate of the evaluation function of the action $Q_k(x_k, u_k)$ using the immediate reinforcement, r_{k+1}, and the estimated value of the following state, $Q^*{}_k(x_{k+1}, u')$,

$$Q^*{}_k(x_{k+1}, u') = max_{u'} Q_k(x_{k+1}, u') \quad \textbf{(3.12)}$$

Where $u' \in U_{k+1}$. The Q-value of each state/action pair is updated by

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k\left[r_{k+1} + \gamma Q^*{}_k(x_{k+1}, u') - Q_k(x_k, u_k)\right] \quad \textbf{(3.13)}$$

This algorithm is called Q-Learning. It shows several interesting characteristics. The estimates of the function Q, also called the Q-values, are independent of the policy pursued by the agent. To calculate the evaluation function of a state, it is not necessary to test all the possible actions in this state but only to take the maximum

Q-value in the new state (3.14). However, the too fast choice of the action having the greatest Q-value can lead to local minima.

$$u' = \arg max_u Q_k(x_{k+1}, u_k) \quad \textbf{(3.14)}$$

To obtain a useful estimate of Q, it is necessary to sweep and evaluate the whole of the possible actions for all the states: it is what one calls the phase of exploration [33].

3.4.1 Exemplification of QL by a simple robot problem

In this part we introduce the concept of Q-learning through a simple but comprehensive numerical example. The example describes an agent which uses unsupervised training to learn about an unknown environment. Suppose we have 5 rooms in a building connected by doors as shown in the figure below. We will number each room 0 through 4. The outside of the building can be thought of as one big room (5). Notice that doors 1 and 4 lead into the building from room 5 (outside).

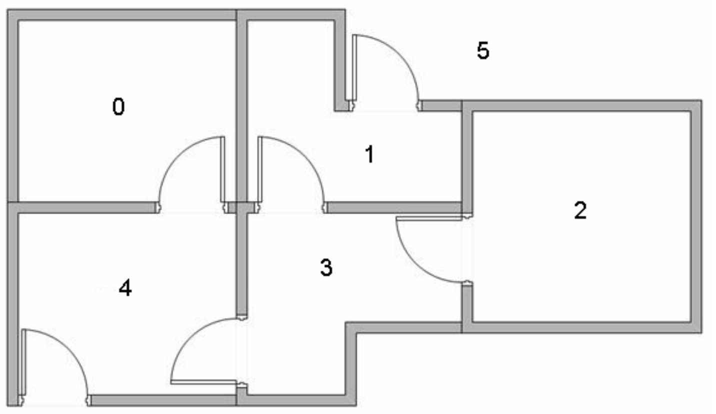


Figure 3.2 : Structure of the environment.

We can represent the rooms on a graph, each room as a node, and each door as a link.

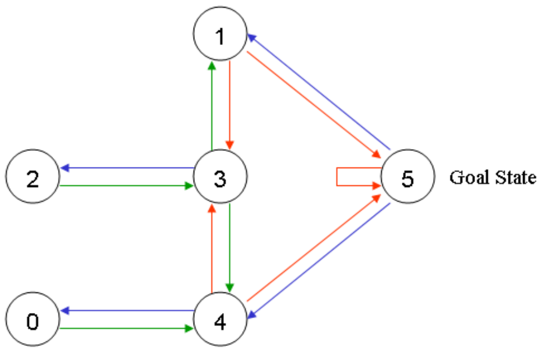


Figure 3.3 : Graphical representation of environment.

For this example, we would like to put an agent in any room, and from that room, go outside the building (this will be our target room). In other words, the goal room is number 5. To set this room as a goal, we will associate a reward value to each door (i.e. link between nodes). The doors that lead immediately to the goal have an instant reward of 100. Other doors not directly connected to the target room have zero reward. Because doors are two-way (0 leads to 4, and 4 leads back to 0), two arrows are assigned to each room. Each arrow contains an instant reward value, as shown below:

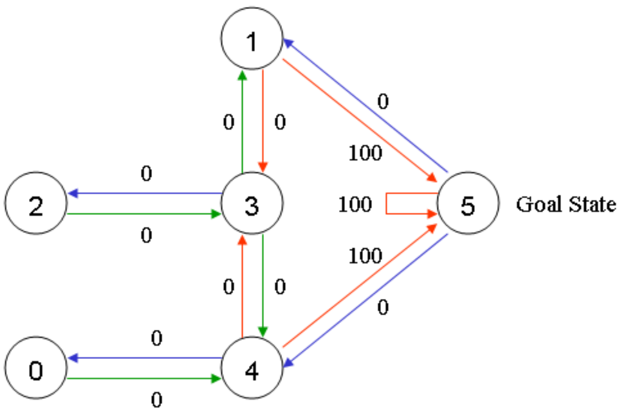


Figure 3.4 : Reward assigned to each action.

Of course, Room 5 loops back to itself with a reward of 100, and all other direct connections to the goal room carry a reward of 100. In Q-learning, the goal is to reach the state with the highest reward, so that if the agent arrives at the goal, it will remain there forever.

Imagine our agent as a dumb virtual robot that can learn through experience. The agent can pass from one room to another but has no knowledge of the environment, and does not know which sequence of doors lead to the outside.

Suppose we want to model some kind of simple evacuation of an agent from any room in the building. Now suppose we have an agent in Room 2 and we want the agent to learn to reach outside the house (5).

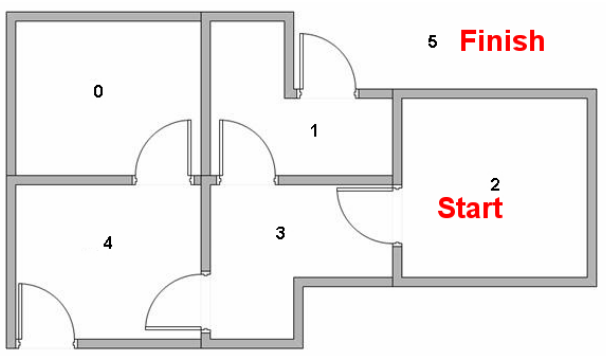


Figure 3.5 : Structure of the problem.

The terminology in Q-Learning includes the terms "state" and "action", so we will call each room, including outside, a "state", and the agent's movement from one room to another will be an "action". In our diagram, a "state" is depicted as a node, while "action" is represented by the arrows.

Suppose the agent is in state 2. From state 2, it can go to state 3 because state 2 is connected to 3. From state 2, however, the agent cannot directly go to state 1 because there is no direct door connecting room 1 and 2 (thus, no arrows). From state 3, it can go either to state 1 or 4 or back to 2 (look at all the arrows about state 3). If the agent is in state 4, then the three possible actions are to go to state 0, 5 or 3. If the agent is in state 1, it can go either to state 5 or 3. From state 0, it can only go back to state 4.

We can put the state diagram and the instant reward values into the following reward matrix R.

$$R = \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \overset{\text{Action}}{\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ \end{array}} \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \quad \textbf{(3.15)}$$

The -1's in the table represent null values where there isn't a link between nodes. Now we will add a similar matrix, "Q", to the brain of our agent, representing the memory of what the agent has learned through experience. The rows of matrix Q represent the current state of the agent, and the columns represent the possible actions leading to the next state.

The agent starts out knowing nothing, the matrix Q is initialized to zero. In this example, for the simplicity of explanation, we assume the number of states is known (to be six). If we didn't know how many states were involved, the matrix Q could start out with only one element. It is a simple task to add more columns and rows in matrix Q if a new state is found.

The transition rule of Q learning is updated according to (3.10) where α_k=1 as following:

$$Q(state, action) = R(state, action) + \gamma Max[Q(next\ state, all\ actions)] \quad \textbf{(3.16)}$$

Our virtual agent will learn through experience, without a teacher (this is called unsupervised learning). The agent will explore from state to state until it reaches the goal. We'll call each exploration an episode. Each episode consists of the agent moving from the initial state to the goal state. Each time the agent arrives at the goal state, the program goes to the next episode.

Algorithm to utilize the Q matrix:

1. Set current state = initial state.
2. From current state, find the action with the highest Q value.
3. Set current state = next state.
4. Repeat Steps 2 and 3 until current state = goal state

To understand how the Q-learning algorithm works, we'll go through a few episodes step by step. We will start by setting the value of the learning parameter γ = 0.8, and the initial state as Room 1.

First we Initialize matrix Q as a zero matrix:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \qquad \textbf{(3.17)}$$

Look at the second row (state 1) of matrix R. There are two possible actions for the current state 1: go to state 3, or go to state 5. By random selection, we select to go to 5 as our action.

Now let's imagine what would happen if our agent were in state 5. Look at the sixth row of the reward matrix R (i.e. state 5). It has 3 possible actions: go to state 1, 4 or 5. By using (3.16) we will have:

$$Q(1,5) = R(1,5) + 0.8Max[Q(5,1), Q(5,4), Q(5,5)] = 100 \qquad \textbf{(3.18)}$$

The next state, 5, now becomes the current state. Because 5 is the goal state, we've finished one episode. Our agent's brain now contains an updated matrix Q as:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \qquad \textbf{(3.19)}$$

For the next episode, we start with a randomly chosen initial state. This time, we have state 3 as our initial state. Look at the fourth row of matrix R (state 3) ; it has 3 possible actions: go to state 1, 2 or 4. By random selection, we select to go to state 1 as our action.

Now we imagine that we are in state 1. Look at the second row of reward matrix R (i.e. state 1). It has 2 possible actions: go to state 3 or state 5. Then, we compute the Q value by utilizing (3.16):

$$Q(3,1) = R(3,1) + 0.8Max[Q(1,3), Q(1,5)] = 80 \qquad \textbf{(3.20)}$$

Then the matrix Q is updated as:

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 100 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 80 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \qquad (3.21)$$

The next state, 1, now becomes the current state. We repeat the inner loop of the Q learning algorithm because state 1 is not the goal state.

If our agent learns more through further episodes, it will finally reach convergence values in matrix Q like:

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 80 & 0 \\ 1 & 0 & 0 & 0 & 64 & 0 & 100 \\ 2 & 0 & 0 & 0 & 64 & 0 & 0 \\ 3 & 0 & 80 & 51 & 0 & 80 & 0 \\ 4 & 64 & 0 & 0 & 64 & 0 & 100 \\ 5 & 0 & 80 & 0 & 0 & 80 & 100 \end{array} \qquad (3.22)$$

Once the matrix Q gets close enough to a state of convergence, we know our agent has learned the most optimal paths to the goal state. Tracing the best sequences of states is as simple as following the links with the highest values at each state.

Finally the learned path can be depicted in the following graph:

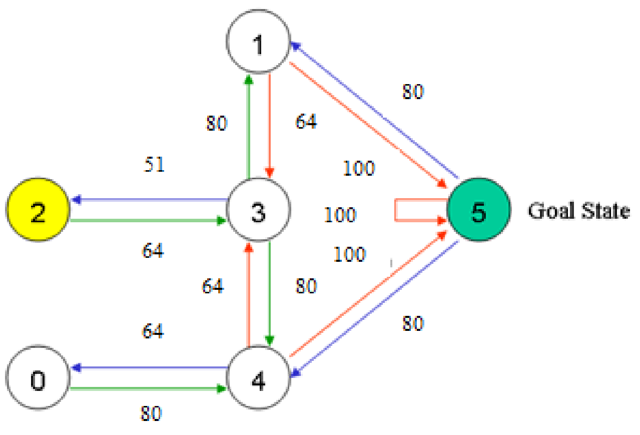


Figure 3.6 : Learned path.

4. QL-BASED FLC DESIGN

In order to minimize or maximize a given cost function of the closed-loop system, QLA makes it practical to tune FPD/FPI/FPID controllers.

The user can decide about the vector of the tuning parameters depending on the control objectives and the available knowledge.

One can distinguish different types of tuning parameters:

i. Parameter vector composed by the universe of discourses of error, change of error and control signal.

ii. Parameter vector composed by the conclusions of fuzzy rules when the premise parameters are fixed.

iii. Parameter vector composed by the positions of the triangular membership functions when the conclusions are fixed.

iv. Parameter composed by all the parameters of both the premises and conclusions.

To each candidate, is associated a Q-value that is incrementally updated by the QLA. The learning process consists then in determining the best set of parameters, the one that will optimize the future reinforcements. Thus, as the quantities are initially unknown, the fuzzy controller has to explore and test several actions. The exploration phase is often long. Though, as fuzzy rules are interpretable and tuning parameters have physical meaning, this phase can be drastically reduced by introducing knowledge in the initial fuzzy controller.

4.1 Parameter discretization

Consider the problem of representing a Q-function, not necessarily the optimal one. Since no prior knowledge about the Q-function is available, the only way to guarantee an exact representation is to store distinct values of the Q-function (Q-values) for every state-action pair, so it is required to have discretized parameters of states and actions.

Let P_{min} and P_{max} mark the boundary of the variation domain of a parameter. The J parameter candidates are distributed in $[P_{min}, P_{max}]$. Note that a very approximate knowledge permits to shrink this interval and/or to eliminate undesirable or physically nonsensical solutions. To simplify, we consider that the proposed values are equally distributed between these bounds. Let $P_{min}{}^{i}$, $P_{max}{}^{i}$ be the bounds of ith element of the parameters' vector P. Therefore, we can take:

$$P_{i1} = P_{min}{}^{i} \text{ , } P_{i2} = P_{i1} + \frac{P_{max}{}^{i} - P_{min}{}^{i}}{J-1} \text{ ,…, } P_{ij} = P_{max}{}^{i} \qquad \textbf{(4.1)}$$

Different Q-values which have been stored for corresponding parameters (here namely P) are depicted graphically in figure (4.1) where we have our states and actions as x= $\{x_1, x_2, x_3, \dots, x_k\}$ and u = $\{u_1, u_2, u_3, \dots, u_k\}$, respectively.

In each state, among the available actions from the action set u, one action is selected according to a desired action selection policy and the resulting action is applied to the system and after that the corresponding Q-value is stored in the predetermined Q matrix. This process continues until all of the states from the state space are met or the convergence criteria of the algorithm has been satisfied. After that in each state, the maximum Q-value which exists among the all available Q-values is recognized by making the use of the updated Q matrix. This maximum Q-value is being considered to be the optimal Q function (optimum Q-value) and therefore the corresponding state-action pair which has been resulted to obtaining of the optimal values of Q matrix is then choosen as the optimized values for the given parameter, namely, P. This procedure is called parameter discretization in a state and action space (set)and the outcome of this process is a kind of algorithmic parameter optimization through the Reinforcement Q-learning algorithm which will be used in our investigated method for optimizing the universe of discourses of the input-output membership functions of the fuzzy controller.

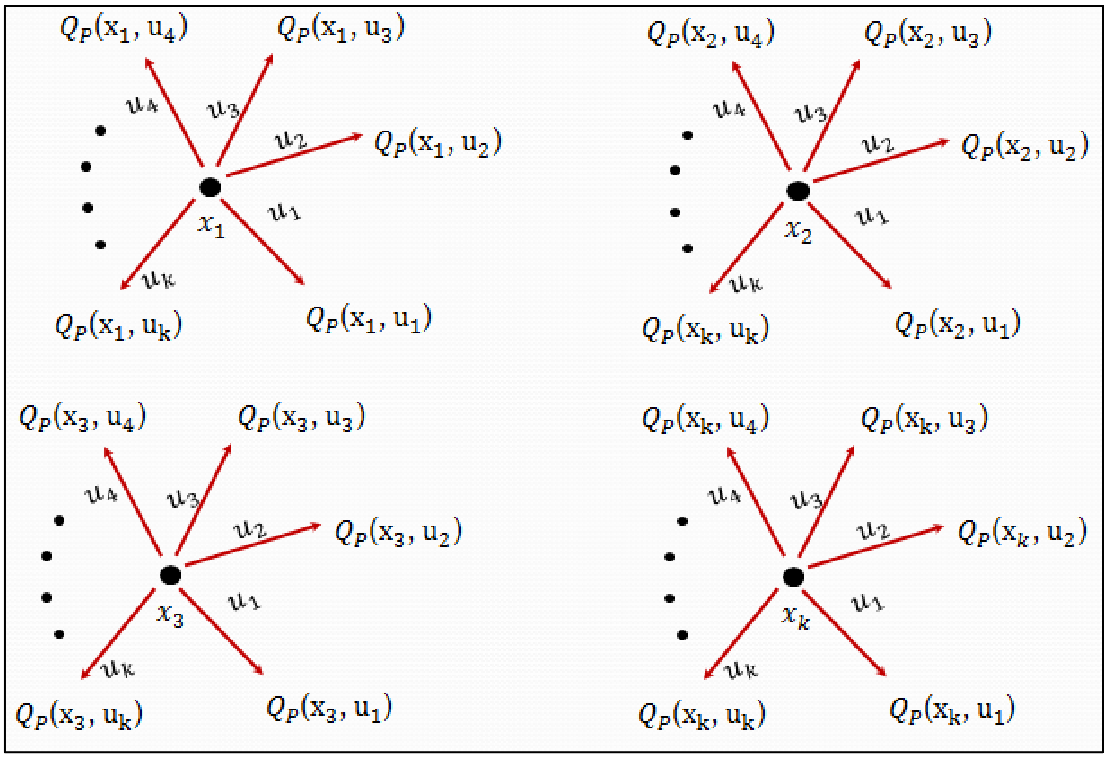


Figure 4.1 : Q-values associated with each parameter P in each state.

4.2 Reward Function

In reinforcement learning, the purpose or goal of the agent is formalized in terms of a special reward signal passing from the environment to the agent. At each time step, the reward is a simple number. Informally, the agent's goal is to maximize the total amount of reward it receives. This means maximizing not immediate reward, but cumulative reward in the long run.

The use of a reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning. Although this way of formulating goals might at first appear limiting, in practice it has proved to be flexible and widely applicable. The best way to see this is to consider examples of how it has been, or could be, used. For example, to make a robot learn to walk, researchers have provided reward on each time step proportional to the robot's forward motion. In making a robot learn how to escape from a maze, the reward is often zero until it escapes, when it becomes +1. Another common approach in maze learning is to give a reward of -1 for every time step that passes prior to escape; this encourages the agent to escape as quickly as possible. To make a robot learn to find and collect empty soda cans for recycling, one might give it a reward of zero most of the time, and then a reward of +1 for each can collected (and confirmed as empty). One might also want to give the robot negative rewards when it bumps into things or when somebody yells at it. For an agent to learn to play checkers or chess, the natural rewards are +1 for winning, -1 for losing, and 0 for drawing and for all nonterminal positions.

You can see what is happening in all of these examples. The agent always learns to maximize its reward. If we want it to do something for us, we must provide rewards to it in such a way that in maximizing them the agent will also achieve our goals. It is thus critical that the rewards we set up truly indicate what we want accomplished. In particular, the reward signal is not the place to impart to the agent prior knowledge about *how* to achieve what we want it to do. For example, a chessplaying agent should be rewarded only for actually winning, not for achieving subgoals such taking its opponent's pieces or gaining control of the center of the board. If achieving these sorts of subgoals were rewarded, then the agent might find a way to achieve them without achieving the real goal. For example, it might find a way to take the opponent's pieces even at the cost of losing the game. The reward signal is your way of communicating to the robot *what* you want it to achieve, not *how* you want it achieved.

According to the distribution of rewards in the space of states, reward function can be classified as two basic forms, dense function and sparse function. Sparse function takes advantage of the easy implementation, but its learning efficiency is relatively low. As for dense function, it is very difficult to design when the size of state space is very large. In response to these defects, we will present a methodology for designing a fuzzy reward functions by the use of fuzzy inference system.

4.2.1 QL-based FLC with scalar reward function

Consider a control system composed by a SISO unknown plant and a FPI or FPD controller. The cost function, which quantifies the effectiveness of the fuzzy controller, is evaluated at the end of a step-response experiment. Without loss of generality, we take the sum of square error cost function given by:

$$\mathrm{I} = \frac{1}{NT - N_0 T} \sum_{k=N_0}^{N} e^2\,(k) \quad \textbf{(4.2)}$$

Where N is the total number of samples, N_0 is the number of transient samples and T the sampling time.

In order to minimize the cost function *I* we use the reinforcement signal *r* given by:

$$r = \begin{cases} -R_1\, if\ \ I > I^* \\ +R_2\, if\ I \leq I^* \end{cases} \tag{4.3}$$

where R_1 and R_2 are positive coefficient and I^* is a satisfaction threshold. The signal *r* is perceived as a reward or a punishment depending on whether it is superior or inferior to a satisfaction threshold.

It is a simple form of a reward function and may ignore some useful information about the interactions between agent (controller) and environment (plant), which can result in an acceptable performance. In order to overcome this disadvantage we will suggest a methodology to invent an innovative kind of reward function which incorporates more elaborate information to how best achieve our goal.

4.2.2 QL-based FLC with fuzzy reward function

Instead of comparing the obtained cost function with a given satisfaction threshold (I^*) and assigning a reward or a punishment to the reinforcement signal, we propose a fuzzy system which its inputs can be considered to be some features of the system's closed loop response such as settling time, overshoot, rise time, etc. and reward signal *r* is seen as the output of the fuzzy inference system.

4.3 Tuning FLCs by the FQL algorithm

The discrete Q-Learning such as we described it uses a discrete space of states and actions which must have reasonable size to enable the algorithms to converge in an acceptable time in practice. In this case, a look-up table can be built up by listing the state/action pairs with their Q values. However in many applications, the number of state/action pairs is very large. Thus a method that is able to make Q-Learning applicable to the continuous problem domain is necessary. The Fuzzy Inference System (FIS) learner is one existing generalization methods which can be introduce generalization in the state space and generate continuous actions in the reinforcement-learning problem.

The principle of Fuzzy Q-Learning (FQL) proposed by Jouffe [64] is RL method that tunes with only *the consequent part* in an incremental way based only on reinforcement signals. Each fuzzy rule R_i has possible k discrete candidate consequents (actions) U_i= $({u^i}_1, {u^i}_2, \dots , {u^i}_k)$ and it memorizes the parameter vector Q associated with each of these actions. Local actions $(u_1, u_2, \dots, u_k)$ selected from U compete with each other based on their Q-values so as to maximize the discounted sum of rewards obtained while achieving the task.

Each Rule R_i of the FPID can be described as follow:

If e is ${L^i}_1$ and de is ${L^i}_2$ then u^i is ${u^i}_1$with Q(s, ${u^i}_1$)

Or u^i is ${u^i}_2$ with Q(s, ${u^i}_2$)

…. ….

Or u^i is ${u^i}_k$ with Q(s, ${u^i}_k$)

Where ${L^i}_m$ is linguistic label related to each input state, ${u^i}_j$ is a rule consequent (action) which corresponds to the consequent part of i-th fuzzy rule R_i and has its own Q-value ${q^i}_j$.

In the previous method, the antecedent parameters are set using the *a priori* task knowledge of the user. To restrict a FPID optimization to the only tuning of the parameters of the consequent part is often insufficient to reach high performances. The principle of the FQL algorithm applied to the antecedent parameters will consist in selecting for each membership function a modal point from a possible discrete candidate modal points set basing on the evaluation function of the action Q(s_t, u_t).

Let us consider the FPID **c**ontroller with the two input variables *e* or *de*. Each universe of discourse of *e* or *de* is partitioned in Nmf_i membership functions ${F^j}_i$:$\{{F^1}_i, {F^2}_i, \dots, {F^j}_i, \dots, {F^{Nmf_i}}_i\}$. Each ${F^j}_i$ has a possible discrete modal point set A(i,j):$\{{a^{j,1}}_i, {a^{j,2}}_i, \dots, {a^{j,k}}_i, \dots, {a^{j,Nniv_{i,j}}}_i\}$ where ${a^{j,k}}_i$ is the kth point modal possible of the jth membership function of the input i. ${F^j}_i$ and $Nni_{v_{i,j}}$ are the cardinals of the set.

Each *Rule* $R_i$of the FPID can be described as follow:

If *e* is ${F_1}^i({a_1}^{i,1})$ and *de* is ${F_2}^i({a_2}^{i,1})$ then *u* is ${u^i}_t$

or is ${F_1}^i({a_1}^{i,2})$ or is ${F_2}^i({a_2}^{i,2})$

…. ….

or is ${F_1}^i({a_1}^{i,k})$ or is ${F_2}^i({a_2}^{i,k})$

According to the FQL algorithm, local modal points $({a^{j,1}}_i, {a^{j,2}}_i, \dots, {a^{j,k}}_i, \dots, {a^{j,Nniv_{i,j}}}_i)$ selected from $\mathrm{A}(i,j)$ compete with each other based on their q-values so as to maximize the discounted sum of rewards obtained while achieving the task. Global quality Q for the state s_t is then defined by the inference of these qualities locally elected:

$$Q_t(s_t) = \sum_{i=1}^{2} \sum_{j=1}^{Nmf_i} q_t\left({a_i}^j\right) . \, {\mu_t}^{{F_i}^j}(x_i) \qquad \textbf{(4.4)}$$

Where ${a_i}^j$ is the elected modal point at the time step *t* for the membership function ${F_i}^j$ by an exploration-exploitation policy and ${\mu_t}^{{F_i}^j}(x_i)$ is the membership degree of the input variable $x_i(x_1{=}e,\ x_2{=}de)$ to ${F_i}^j$ and it measures the contribution of the modal points to the generation of the global action.

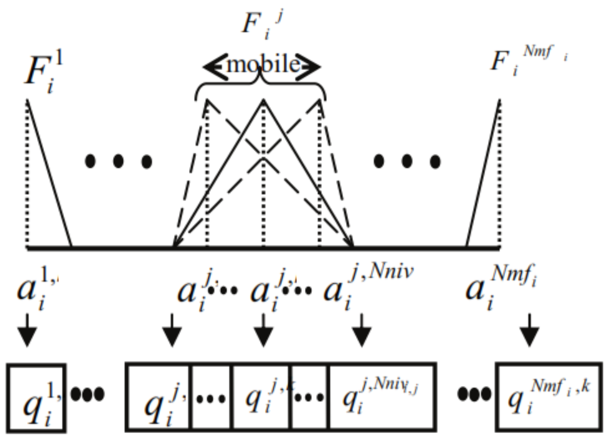


Figure 4.2 : FQL Structure for the antecedent part.

5. SIMULATION STUDIES

5.1 Standard Fuzzy PID Controller

In this study, we will deal with standard fuzzy PID controllers, formed using a fuzzy PD controller with an integrator and a summation unit at the output, as it is shown in Figure 5.1.

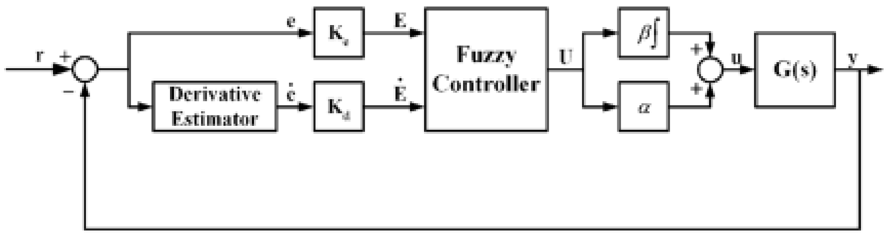


Figure 5.1 : Closed-loop control structure for PID-type FLC.

The output of the fuzzy PID controller is given by:

$$u = \alpha U+\beta \int U\, dt \qquad \textbf{(5.1)}$$

The error e and the derivative of error $\dot{e}$ are used as the inputs and the change of the control signal U is used as the output of the FLC. The input scaling factors k_e for error and k_d for the derivative of error normalize the inputs to the range in which the membership functions of the inputs are defined.

The corresponding symmetrical rule base which has beed used for the FLC is shown in Table 5.1. And also three equidistant triangular membership functions for the inputs of fuzzy controller and five singletons for the output of it have been utilized.

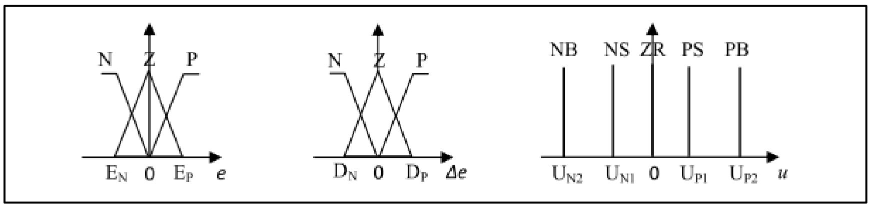


Figure 5.2 : Input-Output MFs to the FLC.

Table 5.1 : Rule base for the FLC.

e	**de**		
	ℕ	ℤ	ℙ
ℕ	NB	NS	ZR
ℤ	NS	ZR	PS
ℙ	ZR	PS	PB

In practice, fuzzy control is applied using local inferences. That means each rule is inferred and the results of the inferences of individual rules are then aggregated. The most common inference methods are: the max-min method, the max-product method and the sum-product method, where the aggregation operator is denoted by either max or sum, and the fuzzy implication operator is denoted by either min or product. Finally, a defuzzification method is needed to obtain a crisp output from the aggregated fuzzy result. Popular defuzzification methods include maximum matching and centriod defuzzification. The centroid defuzzification is widely used for fuzzy control problems where a crisp output is needed, and maximum matching is often used for pattern matching problems where we need to need to know the output class. In this thesis, the fuzzy reasoning results of outputs are gained by aggregation operation of fuzzy sets of inputs and designed fuzzy rules, where *max-product* aggregation method and *average-sum* defuzzification method are used.

5.2 Application of fuzzified QL to membership function tuning

The application of the QL algorithm for tunning the membership functions of the FLC is to optimize the universe of discourse of the input-output membership functions by the suggested method and finally update the table 5.1 for the rule base of the FLC.

The cost function to be minimized, which quantifies the efficiency of the fuzzy controller, is evaluated at the end of a step-response experiment. Without loss of generality, we take the sum of square error cost function given by:

$$\mathrm{I} = \frac{1}{NT-N_0T}\sum_{k=N_0}^{N} e^2\,(k) \qquad \textbf{(5.2)}$$

Where N is the total number of samples, N_0 is the number of transient samples and T is the sampling time.

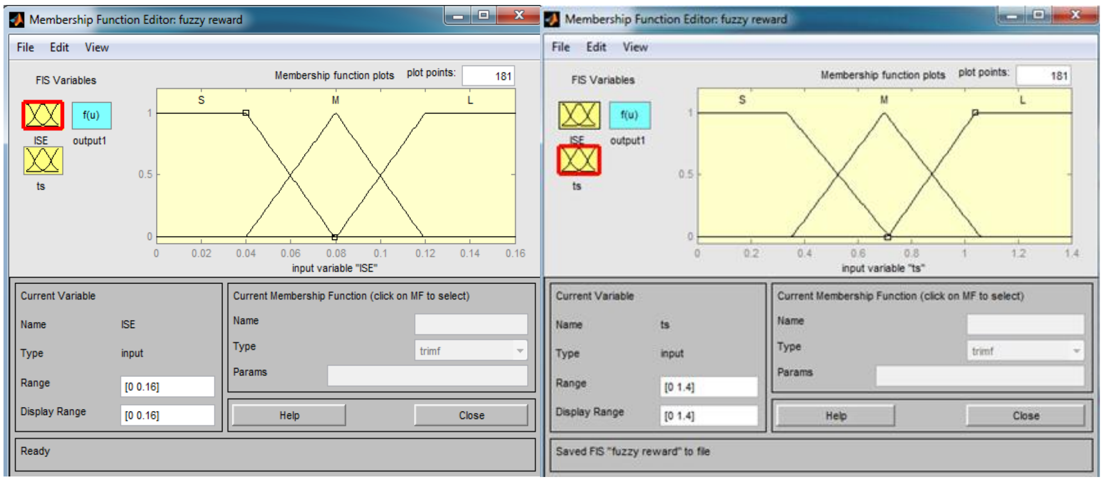
In order to demonstrate that the performance of the proposed QL method can be significantly improved by manipulating the reward function of the algorithm, we propose a fuzzy reward function for the QL algorithm which incorporates a fuzzy structure including more elaborate information about the rewards/punishments assigned to each action which is being taken in each step time and make use of this fuzzy reward function in our simulations.

Besides enhancing the performance of the system's unit step response, Q-learning with a fuzzy reward function reaches to its optimal value and meets its convergence criteria faster than the Q-learning with a scalar reward function.

5.3 Structure of the proposed fuzzy reward function

To build a fuzzy reward function for the QL algorithm, we consider the inputs to our fuzzy system to be Integral Square Error (ISE) and settling time(t_s). Consequently, the output of the fuzzy system is considered to be reward/punishment.

For our fuzzy reward function, three equidistant triangular membership functions for the inputs and seven singletons for the outputs have been used. The corresponding input MFs have been shown in the following:


Membership Function Editor: fuzzy reward

Current Variable: Name ISE; Type input; Range [0 0.16]; Display Range [0 0.16]

Current Membership Function (click on MF to select): Type trimf

plot points: 181

Ready

Membership Function Editor: fuzzy reward

Current Variable: Name ts; Type input; Range [0 1.4]; Display Range [0 1.4]

Current Membership Function (click on MF to select): Type trimf

plot points: 181

Saved FIS "fuzzy reward" to file

Figure 5.3 : Input MFs for fuzzy reward function.

The performance of the fuzzy reward function will be evaluated according to ISE and settling time of the given system. Therefore, we can build a desired fuzzy rule base according to these informations as the following:

If ISE is Small and t_s is Small then reward is Positive Large.

If ISE is Small and t_s is Medium then reward is Positive Medium.

If ISE is Small and t_s is Large then reward is Positive Small.

If ISE is Medium and t_s is Small then reward is Positive Small.

If ISE is Medium and t_s is Medium then reward is Zero.

If ISE is Medium and t_s is Large then reward is Negative Small.

If ISE is Large and t_s is Small then reward is Negative Medium.

If ISE is Large and t_s is Medium then reward is Negative Medium.

If ISE is Large and t_s is Large then reward is Negative Large.

The resulting rule base for our fuzzy reward function is assembled in the following table.

Table 5.2 : Rule base for fuzzy reward function.

ISE	t_s		
	S	**M**	**L**
S	PL	PM	PS
M	PS	Z	NS
L	NM	NM	NL

5.4 Simulations

At first we will apply the proposed method to two second order linear systems with and without time delay and their unity step responses will be demonstrated. After that, we consider a non-linear system and try to utilize the QL in order to tune the membership functions of the corresponding fuzzy controller.

Note that in all of the simulations in order to simulate the given plant and its corresponding time delay, Fourth-Order Runge-Kutta and Pade Approximation methods have been used, respectively.

5.4.1 System I

Firstly, we apply the method to a second order linear system without time delay:

$$G(s) = \frac{16}{s^2 + 3s + 2} \tag{5.3}$$

The scaling factors of the FLC are taken as α = 0.22, β = 1, k_e= 0.7, k_d= 0.2. The unit step response of the system is represented in the following figure:

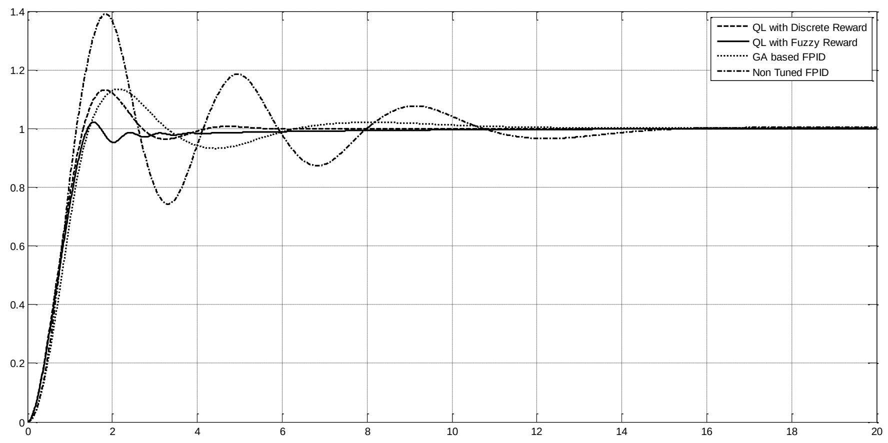


Figure 5.4 : Closed-loop unity step response for system I.

As can be seen from the Figure 5.4, the overshoot of the response has been decreased to 2.2% and settling time has become 3.57 seconds by means of the proposed method with a fuzzy structure for its reward function.

5.4.2 System II

The other system to be considered is a second order plus dead time (SOPDT) linear system which is given by the following transfer function:

$$G(s) = \frac{2e^{-1.5s}}{s^2 + 3s + 2} \tag{5.4}$$

The scaling factors for this system are taken as: α = 0.2, β = 0.4, k_e= 1, k_d= 0.1. The unit step response for the given plant is shown in the following figure:

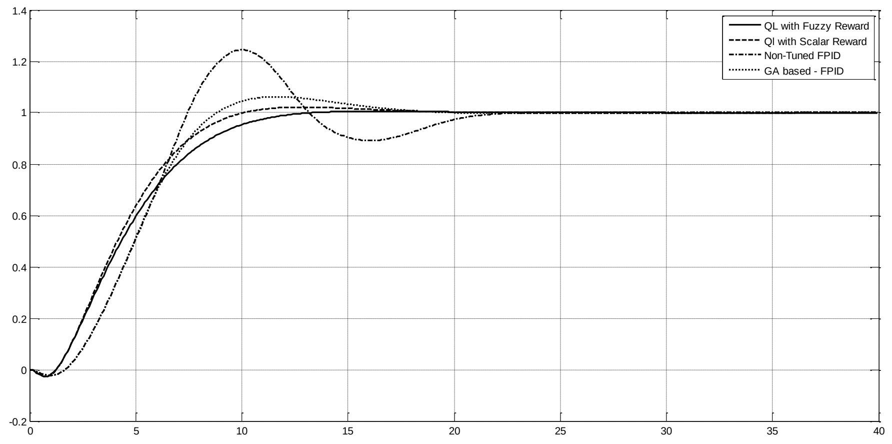


Figure 5.5 : Closed-loop unity step response for system II.

By comparing the results of the figure 5.5 it can be observed that by the proposed method the overshoot of the response has been completely eliminated and also settling time has decreased significantly to 11.22 sec.

The learned parameters and obtained rule bases for the FLC have been demonstrated in the following for both systems I and II.

1) System I

QL with fuzzy reward: $E_P = |E_N| = 0.3$, $D_P = |D_N| = 0.02$, $U_{P2} = |U_{N2}| = 1$

QL with scalar reward: $E_P = |E_N| = 0.4556$, $D_P = |D_N| = 0.1289$,

$U_{P2} = |U_{N2}| = 0.45$

2) System II

QL with Fuzzy reward: $E_P = |E_N| = 1.864$, $D_P = |D_N| = 0.871$,

$U_{P2} = |U_{N2}| = 1.578$

QL with scalar reward: $E_P = |E_N| = 1.457$, $D_P = |D_N| = 0.743$,

$U_{P2} = |U_{N2}| = 1.367$

Table 5.3 : Learned rule bases for SysI (QL with fuzzy and scalar reward).

e	de		
	N	Z	P
N	-1	-0.5	0
Z	-0.5	0	0.5
P	0	0.5	1

e	de		
	N	Z	P
N	-0.45	-0.22	0
Z	-0.22	0	0.22
P	0	0.22	0.45

Table 5.4 : Learned rule bases for SysII (QL with fuzzy and scalar reward).

e	de		
	N	Z	P
N	-1.57	-0.78	0
Z	-0.78	0	0.78
P	0	0.78	1.57

e	de		
	N	Z	P
N	-1.36	-0.68	0
Z	-0.68	0	0.68
P	0	0.68	1.36

5.4.3 System III

In this part we apply the proposed method to a nonlinear system. We consider the problem of stabilization of an inverted pendulum in the vertical position to demonstrate the efficiency of the proposed tuning method. We use a fuzzy PD controller as shown in figure 5.6 with triangular membership functions for inputs and singletons for output and try to tune the universe of discourse of input-output MFs. For our simulations we use an Inverted Pendulum which its dynamics is described by the following equations:

$$\ddot{y} = \frac{9.8\sin y + \cos y\left(\frac{-\bar{u} - 0.25\dot{y}^2 \sin y}{1.5}\right)}{0.5\left(\frac{4}{3} - \frac{1}{3}\cos^2 y\right)} \qquad \textbf{(5.5)}$$

$$\dot{\bar{u}} = -100\bar{u} + 100u \quad \textbf{(5.6)}$$

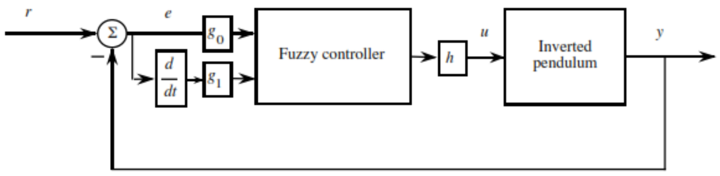


Figure 5.6 : Fuzzy PD controller for balancing an Inverted Pendulum with scaling gains g_0, g_1 and h.

As shown in figure 5.7, y denotes the angle that pendulum makes with the vertical (in radians), l is the half-pendulum length (in meters) and u is the force input that moves the cart (in Newtons).

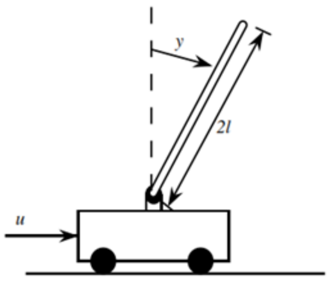


Figure 5.7 : Inverted Pendulum.

Here, the first order filter on u to produce $\bar{u}$ represents an actuator. For our problem $y(0)$ takes different values and $\dot{y}(0)$=0. The initial condition for the actuator state is also zero.

During the simulations scaling gains for the FPD are taken as: g_0= 12.23, g_1= 0.712 and h = 0.601.

We consider four different cases depending on the initial state of the pendulum's vertical position and gradually increase it in order to observe both the output and control signal of the inverted pendulum's behavior as following:

x_0=0.2 (rad), x_0=0.4 (rad), x_0=0.5 (rad) and x_0=0.8 (rad).

i. *Initial state for the Pendulum's vertical position is* 0.2(rad) = 11.46(deg):

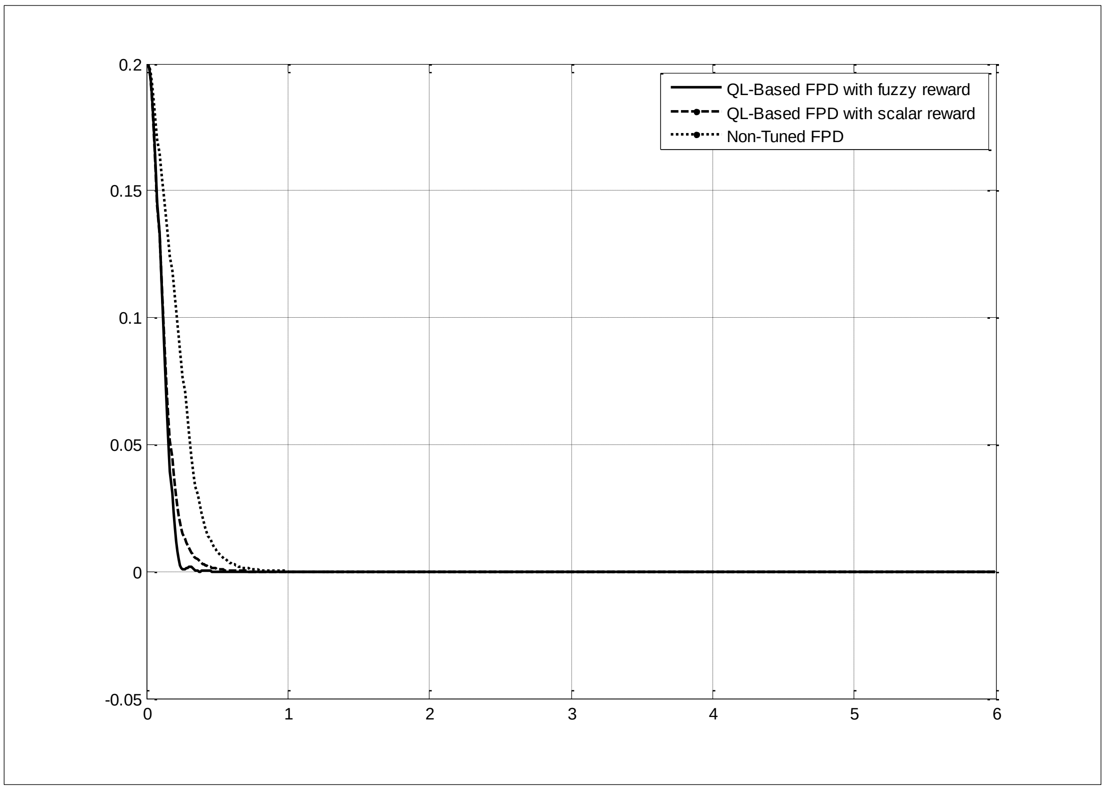


Figure 5.8 : Inverted Pendulum balancing for case (i).

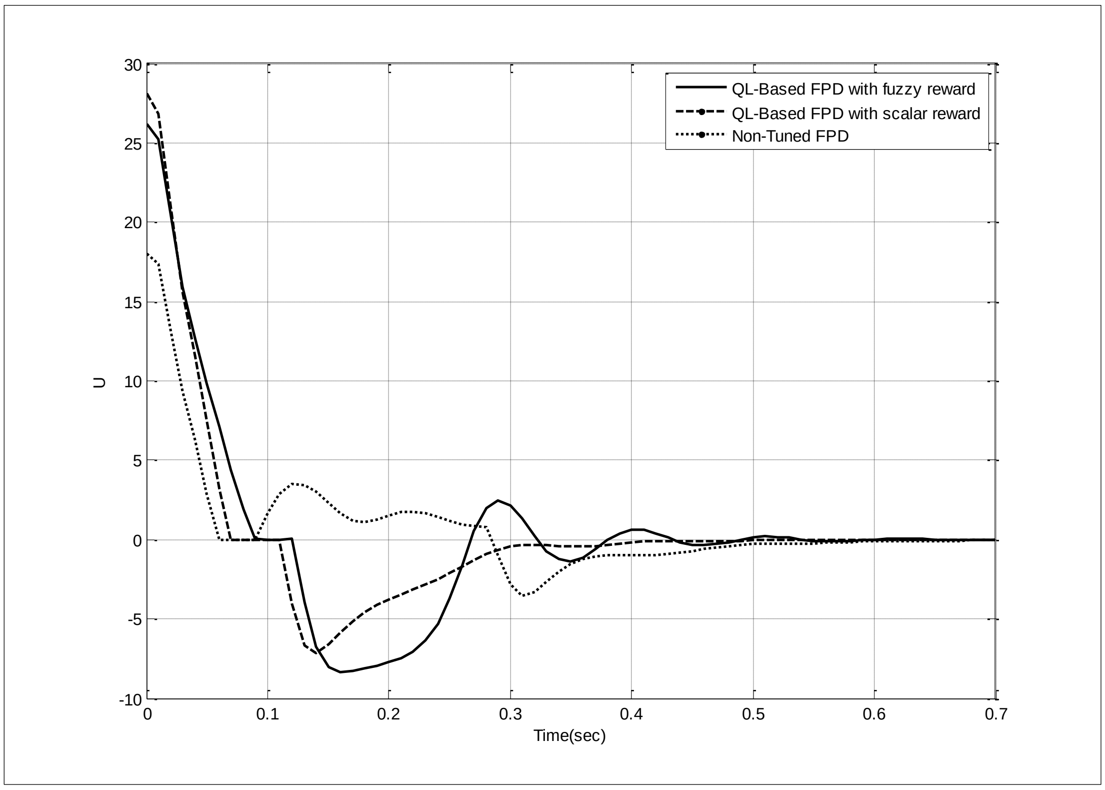


Figure 5.9 : Control signal for case (i).

ii. *Initial state for the Pendulum's vertical position is* 0.4(rad) = 22.92(deg):

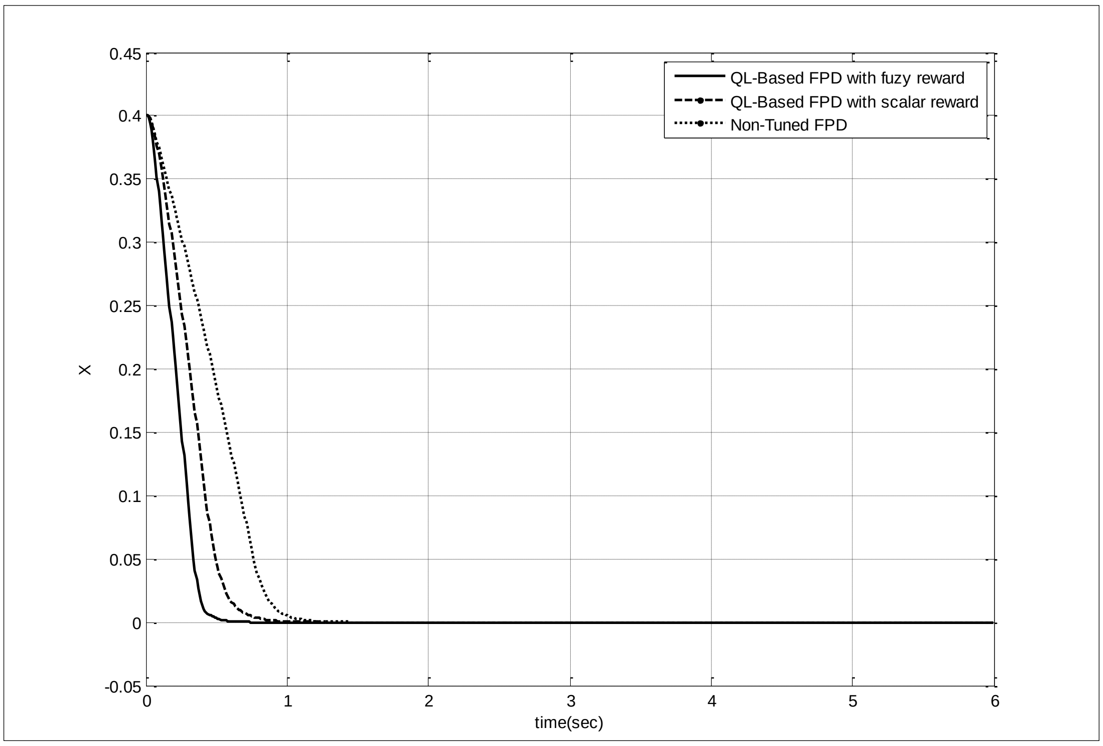


Figure 5.10 : Inverted Pendulum balancing for case (ii).

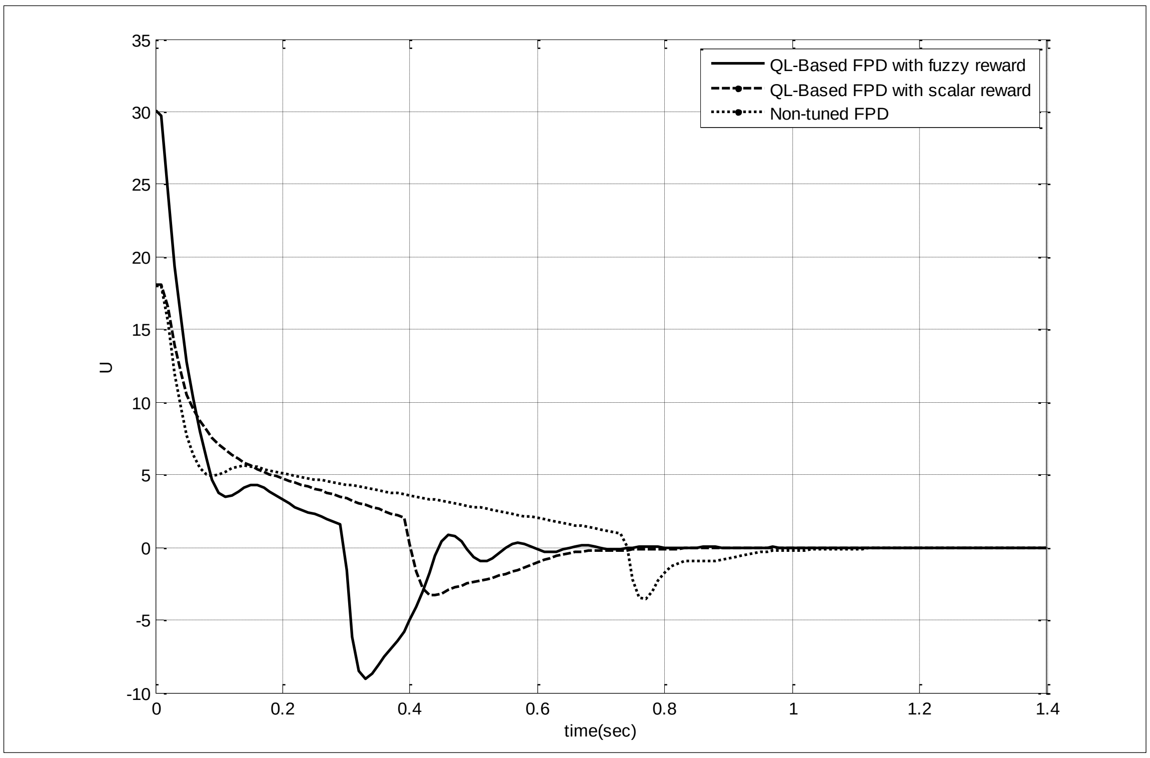


Figure 5.11 : Control signal for case (ii).

iii. *Initial state for the Pendulum's vertical position is* 0.5 (rad) = 28.66 (deg):

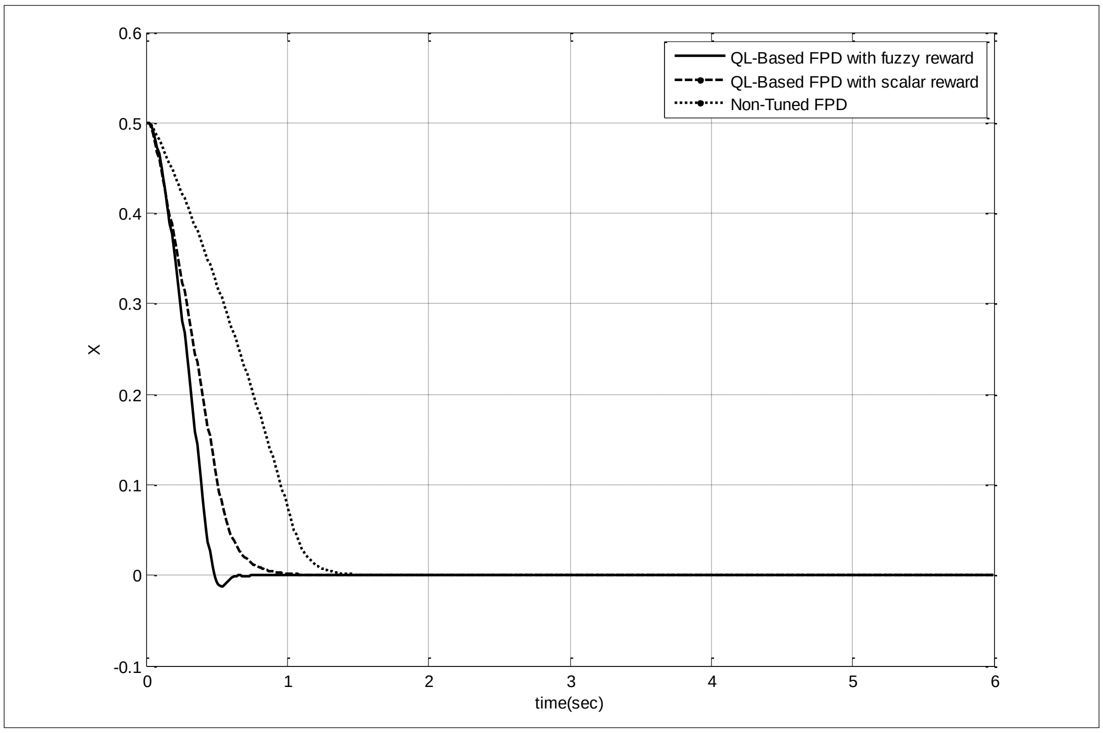


Figure 5.12 : Inverted Pendulum balancing for case (iii).

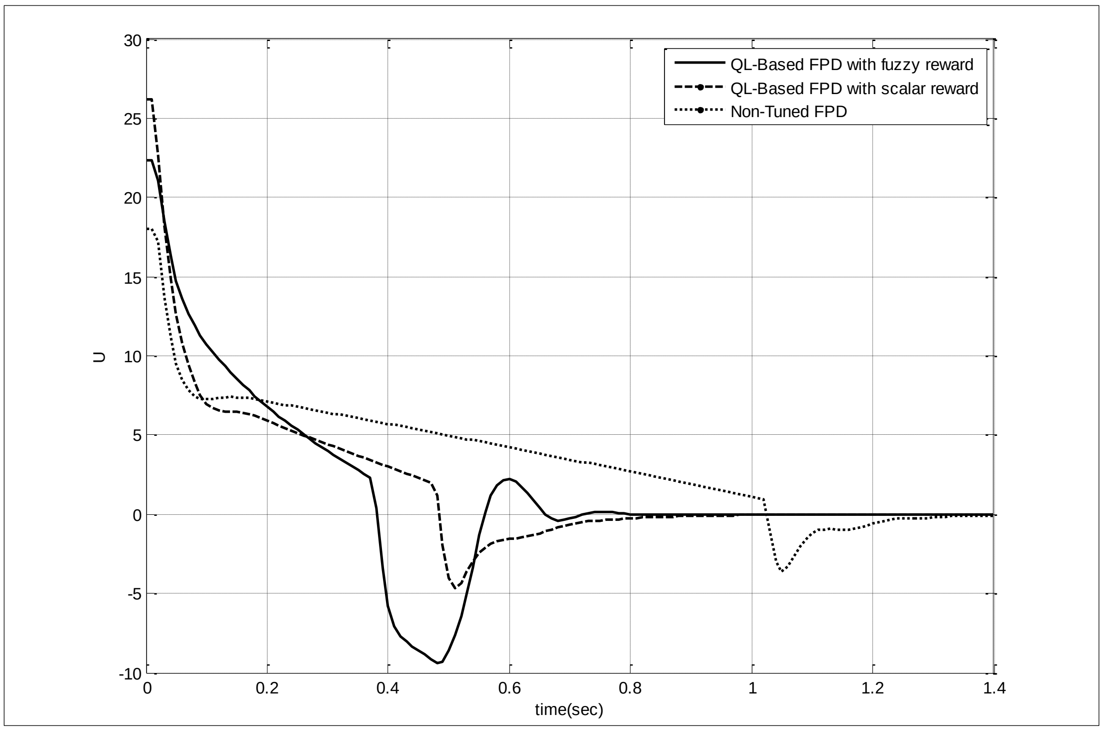


Figure 5.13 : Control signal for case (iii).

iv. *Initial state for the Pendulum's vertical position is* 0.8(rad) = 45.85(deg):

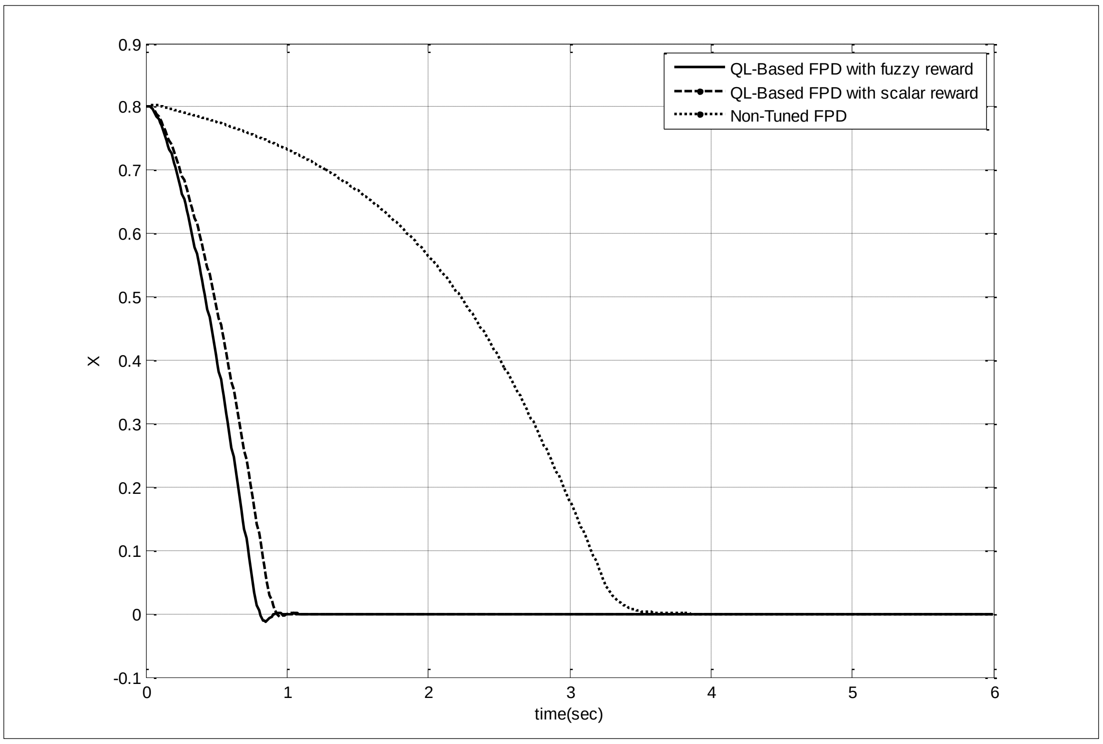


Figure 5.14 : Inverted Pendulum balancing for case (iv).

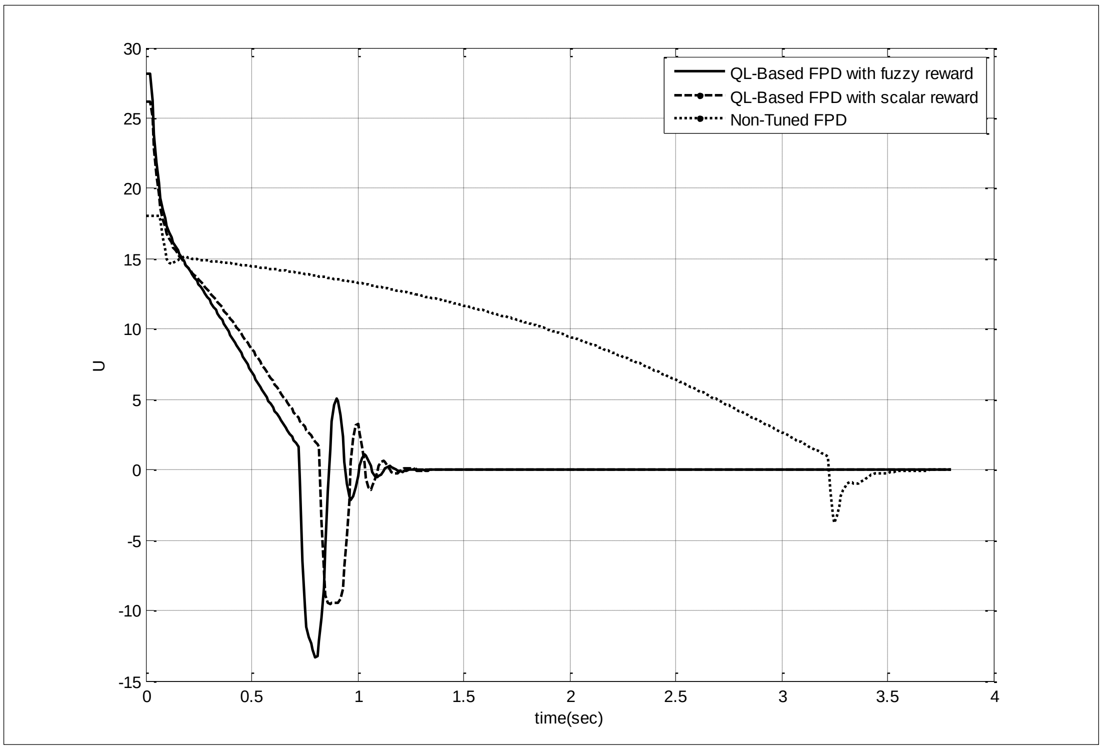


Figure 5.15 : Control signal for case (iv).

From the given figures, it is obviously observable that the balancing time of the Inverted Pendulum is considerably decreased when using the proposed QL method in comparison with the non-tuned FPD controller.

The learned parameters and rule bases obtained by the QLA for the Inverted Pendulum balancing problem are given as following:

1) Case (i)

   QL with fuzzy reward: $E_P = |E_N| = 1.15$, $D_P = |D_N| = 0.884$,

   $U_{P2} = |U_{N2}| = 43.57$

   QL with scalar reward: $E_P = |E_N| = 1.318$, $D_P = |D_N| = 0.785$,

   $U_{P2} = |U_{N2}| = 46.79$

2) Case (ii)

   QL with fuzzy reward: $E_P = |E_N| = 1.234$, $D_P = |D_N| = 0.883$,

   $U_{P2} = |U_{N2}| = 50$

   QL with scalar reward: $E_P = |E_N| = 1.56$, $D_P = |D_N| = 0.687$,

   $U_{P2} = |U_{N2}| = 30.07$

3) Case (iii)

   QL with fuzzy reward: $E_P = |E_N| = 1.49$, $D_P = |D_N| = 1.08$,

   $U_{P2} = |U_{N2}| = 37.14$

   QL with scalar reward: $E_P = |E_N| = 1.58$, $D_P = |D_N| = 0.69$,

   $U_{P2} = |U_{N2}| = 43.57$

4) Case (iv)

   QL with fuzzy reward: $E_P = |E_N| = 1.15$, $D_P = |D_N| = 1.1$,

   $U_{P2} = |U_{N2}| = 46.78$

   QL with scalar reward: $E_P = |E_N| = 1.23$, $D_P = |D_N| = 0.981$,

   $U_{P2} = |U_{N2}| = 43.58$

Table 5.5 : Rule bases for case i (QL with fuzzy and scalar reward).

e	**de**		
	N	**Z**	**P**
N	-43.57	-21.79	0
Z	-21.79	0	21.79
P	0	21.79	43.57

e	**de**		
	N	**Z**	**P**
N	-46.79	-23.40	0
Z	-23.40	0	23.40
P	0	23.40	46.79

Table 5.6 : Rule bases for case ii (QL with fuzzy and scalar reward).

e	**de**		
	N	**Z**	**P**
N	-50	-25	0
Z	-25	0	25
P	0	25	50

e	**de**		
	N	**Z**	**P**
N	-30.07	-15.03	0
Z	-15.03	0	15.03
P	0	15.03	30.07

Table 5.7 : Rule bases for case iii (QL with fuzzy and scalar reward).

e	**de**		
	N	**Z**	**P**
N	-37.14	-18.57	0
Z	-18.57	0	18.57
P	0	18.57	37.14

e	**de**		
	N	**Z**	**P**
N	-43.57	-21.79	0
Z	-21.79	0	21.79
P	0	21.79	43.57

Table 5.8 : Rule bases for case iv (QL with fuzzy and scalar reward).

e	**de**		
	N	**Z**	**P**
N	-46.78	-23.39	0
Z	-23.39	0	23.39
P	0	23.39	46.78

e	**de**		
	N	**Z**	**P**
N	-43.58	-21.8	0
Z	-21.8	0	21.8
P	0	21.8	43.58

6. CONCLUSIONS AND RECOMMENDATIONS

In this study an optimization method for tuning the parameters of membership functions of fuzzy logic controllers based on Reinforcement Learning (Q-Learning) is proposed. This approach is very powerful in stochastic control problems, and in those without an exemplary supervisor or a complete lack of model of the environment. In essence, the agent does not need to know the very nature of the controlled system but only rewards from interactions.

Firstly, we applied the QL based optimization method to different second order linear systems with time delay and without time delay for tuning the universe of discourses of error, change of error and output singletons of triangular membership functions of the fuzzy controllers. The simulations revealed that the overshoot and settling time of the closed-loop unity step response has decreased in a considerable manner.

Afterwards, the proposed algorithm is applied to a non-linear system and it has been observed that the reinforcement Q-learning algorithm with its very simple algorithm, is a systematic method that saves us a great deal of time in comparison with its counterparts when performing complex simulations and also its convergence is guaranteed.

For future work, the proposed method might be applied to a practical system to explore the efficiency of it. The other area to be concentrated on can be proposing more sophisticated reward functions for the algorithm in order to examine its performance. Besides that there exist some other action-state selection methods other than greedy policy, which can be used in the suggested algorithm.

REFERENCES

[1] Åström, K. J. and Hägglund, T. (1995). PID controllers, theory, design and tuning. Research Triangle Park, North Carolina: Instrument Society of America.

[2] Mamdani, E. H. (1974). Application of fuzzy algorithm for simple dynamic plant. Proc. IEEE, 1585-1588.

[3] Jantzen, J. (2007). Foundations of fuzzy control. West Sussex (England): John Wiley & Sons Ltd.

[4] Yager, R. R. and Filev, D. P. (1994). Essential of fuzzy modeling and control. John Wiley & Sons Inc**.**

[5] Xu, J.X., Hang, C.C., Liu, C. (2000). Parallel structure and tuning of a fuzzy PID controller. Automatica 36, 673–684.

[6] He, S.Z., Shaoua, T.D., Xu, F.L. (1993). Fuzzy self-tuning of PID. Fuzzy Sets and Systems 56, 37–46.

[7] Zhao, Z.Y., Tomizuka, M., Isaka, S. (1993). Fuzzy gain scheduling of PID controllers. IEEE Transactions on Systems Man and Cybernetics 23 (5), 1392–1398.

[8] Mizumoto, M. (1992). Realization of PID controls by fuzzy control methods. Fuzzy Sets and Systems 70, 171–182.

[9] Qin, S.J., Borders, G. (1994). A multi region fuzzy logic controller for nonlinear process control. IEEE Transactions on Fuzzy Systems 2, 74–81.

[10] Palm, R., Driankov, D. (1996). Model Based Fuzzy Control. Springer, Berlin.

[11] Xu, J.X., Liu, C., Hang, C.C. (1998). Tunning and analysis of a fuzzy PI controller based on gain and phase margins. IEEE Transactions on Systems Man and Cybernetics—Part A 28 (5), 685 691.

[12] Chung, H. Y., Chen, B. C., & Lin, J. J. (1998). A PI-type fuzzy controller with self-tuning scaling factors. Fuzzy Sets and Systems, 93, 23–28.

[13] Guzelkaya, M., Eksin, I., & Yesil, E. (2003). Self-tuning of PID-type fuzzy logic controller coefficients via relative rate observer. Engineering Applications of Artificial Intelligence, 16, 227–236.

[14] Karasakal, O., Guzelkaya, M., Yesil, E., & Eksin, I. (2005). Implementation of a new self-tuning fuzzy PID controller on PLC. Turkish Journal of Electrical Engineering and Computer Science, 13, 277–286.

[15] Mudi, R. K., and Pal, N. R. (1999). A robust self-tuning scheme for PI- and PD-type fuzzy controllers. IEEE Transactions on Fuzzy Systems, 7(1), 2–16.

[16] Qiao, W. Z., and Mizumoto, M. (1996). PID type fuzzy controller and parameters adaptive method. Fuzzy Sets and Systems, 78, 23–35.

[17] Woo, Z. W., Chung, H. Y., & Lin, J. J. (2000). A PID-type fuzzy controller with self-tuning scaling factors. Fuzzy Sets and Systems, 115, 321–326.

[18] Chung, H.Y., Chen, B.C., Lin, J.J. (1998). A PI-type fuzzy controller with self-tuning scaling factors. Fuzzy Sets and Systems ; 93:23_8.

[19] Zheng, L., Yamatake, Honeywell. (1992). A practical guide to tune of proportional and integral (PI) like fuzzy controllers. In: Proc. IEEE international conference on fuzzy systems. p. 633_40.

[20] Adams, J.M., Rattan, K.S. (2001). A genetic multi-stage fuzzy PID controller with a fuzzy switch. In: Proc. IEEE Int. conference on systems, man, and cybernetics, vol. 4. p. 2239_ 44.

[21] Ko, C.N., Lee, T.L., Fan, H.T., Wu, C.J. (2006). Genetic auto-tuning and rule reduction of fuzzy PID controllers. In: Proc. IEEE international conference on systems, man, and cybernetics. p. 1096_101.

[22] Tang, K.S., Man, K.M., Chen, G. (2001). An optimal fuzzy PID controller. IEEE Transactions on Industrial Electronics; 48:757_65.

[23] Wu, C.J., Lee, T.L., Fu, Y.Y., Lai, L.C. (2007). Auto-tuning fuzzy PID control of a pendubot system. In: Proc. 4th IEEE international conference on mechatronics. p. 1_6.

[24] Duan, H.B., Wang, D.B., Yu, X.F. (2006) Novel approach to nonlinear PID parameter optimization using ant colony optimization algorithm. Journal of Bionic Engineering; 3:73_8.

[25] Varol, H.A., Bingul, Z. (2004). A new PID tuning technique using ant algorithm. In: Proc. American control conference. p. 2154_9.

[26] Wang, L.X. (1997). A Course in Fuzzy System and Control. Upper Saddle River, NJ: Prentice-Hall.

[27] Wang, L.X. (1993). Stable adaptive fuzzy control of nonlinear systems, IEEE Trans. Fuzzy Syst., vol. 1, no. 2, pp. 146–155.

[28] Jang, J. C. (1993). ANFIS: Adaptive-Neural-Based Fuzzy Inference System, IEEE Trans. On SMC, Vol. 23, pages 665- 685.

[29] Anderson, C.W. (1987). Strategy learning with multilayer connectionist representations, in Proc. 4th Int. Workshop on Mach. Learn, Irvine, CA, pp. 103–114.

[30] Bonarini, A. (1997). Evolutionary Learning of Fuzzy rules: competition and cooperation, In Pedrycz, W. (Ed.), FuzzyModelling: Paradigms and Practice, Kluwer Academic Press, Norwell, MA, pp 265 – 284.

[31] Kawabe, T., Tagami, T., Katayama, T. (1996). A genetic algorithm based minimax optimal design of robust I-PD controller, in Proc. IEE Int.Conf. Control, London, U.K. pp. 436–441.

[32] Sutton, R.S., Barto, A.G., Williams, R.J. (1992). Reinforcement learning is direct adaptive optimal control, IEEE Control Syst. Mag., vol. 12, no. 2, pp. 19–22.

[33] Sutton, R.S., Barto, A.G. (1998). Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press.

[34] Chiang, C.K., Chung, H.Y., Lin, J.J. (1997). A self-learning fuzzy logic controller using genetic algorithms with reinforcements, IEEE Trans. Fuzzy Syst., vol. 5, no. 3, pp. 460–467.

[35] Dongbing, Gu., Huosheng, Hu. (2002). Reinforcement Learning of Fuzzy Logic Controllers for Quadruped Walking Robots, IFAC 15th Triennial World Congress, Barcelona, Spain.

[36] Byung Jun, Hyo., Kwee, B.S. (1997). Behavior Learning and Evolution of Collective Autonomous Mobile Robotsbased on Reinforcement Learning and Distributed Genetic Algorithms, IEEE International Workshop on Robot and Communication IEEE, pp 248,253.

[37] Barto, A. G., Jordan, M.I. (1987). Gradient following without backpropagation in layered networks," in Proc. of IEEE, Annual Con$ Neural Networks, vol. 2, San Diego, CA, pp. 629-636. Fuzzy Sets and Systems, 93, 23–28.

[38] Anderson, C.W. (1986). Learning and Problem Solving With Multilayer Connectionist Systems, Ph.D. disseration, Univ. Massachusetts, Amherst, 16, 227–236.

[39] Chia, F.J. (2001). Construction of Dynamic Fuzzy If-Then Rules through Genetic reinforcement Learning for Temporal Problems Solving, IEEE, pp 2341-23466.

[40] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey.Journal of Artificial Intelligence Research, 4, 237-285.

[41] Moriarty, D.E., Schultz, A.C., Grefenstette, J.J. (1999). Evolutionary Algorithms for Reinforcement Learning, Journal of Artificial Intelligence Research. 241-276 .

[42] Sutton, R.S. (1998). Learning to predict by the methods of temporal differences, Mach. Learn., vol. 3, no. 1, pp. 9–44.

[43] Tsitsiklis, J.N., Roy, B.V. (1997) An analysis of temporal-difference learning with function approximation, IEEE Trans. Autom. Control, vol. 42, no. 5, pp. 674–690.

[44] Barto, A.G., Sutton, R.S., Anderson, C.W. (1983). Neuron like adaptive elements that can solve difficult learning control problems, IEEE Trans. Systems, Man, Cybern., vol. SMC-13, no. 5, pp. 834–846.

[45] Anderson, C.W. (1989). Learning to control an inverted pendulum using neural networks, IEEE Control Syst. Mag., vol. 9, pp. 31–37.

[46] Wakins, C.J. (1989). Learning from delayed rewards, Ph.D. dissertation, King's College, Cambridge, U.K.

[47] Wakins, C.J., Dayan, P. (1992). Technical note: Q-learning, Mach. Learn., vol. 8, pp. 279–292.

[48] Koike, Y., Doya, K. (1999). Multiple state estimation reinforcement learning for driving model: Driver model of automobile, in Proc. IEEE Int. Conf. Systems, Man, and Cybernetics (SMC), Tokyo, Japan, vol. 5, pp. 504–509,

[49] Suh, I.H., Kim, J.H., Oh, S.R. (2001). Region-based Q-learning for intelligent robot systems, in IEEE Int. Symp. Computational.

[50] Xiaohui, Dai., Kwong Li, Chi. (2005). An Approach to Tune Fuzzy Controllers Based on Reinforcement Learning for Autonomous Vehicle Control, IEEE Transactions On Intelligent Transportation Systems, Vol. 6, No. 3, pp 285-293

[51] Zadeh, L.A. (1965). Fuzzy sets, Information and Control 8, 338–353.

[52] Mamdani, E.H. (1974). Application of fuzzy algorithms for the control of a dynamic plant, Proc. IEEE 121, No. 12-1585-1588.

[53] Mamdani, E.H. (1977). Application of fuzzy logic to approximate reasoning using linguistic synthesis, IEEE Tr. Comp. C-26, 1182-1191

[54] Mamdani, E.H., Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. International Journal of Man-Machine Studies 7(1), 1–13.

[55] Aminzadeh, F., Jamshidi, M. (1993). Soft computing, Prentice Hall Series on Environmental and Intelligent Manufacturing Systems, Vol. 4 (M. Jamshidi, Ed.). Prentice-Hall, Englewood Cliffs, NJ.

[56] Jantzen, J. (2007). Foundations of fuzzy control. West Sussex (England): John Wiley & Sons Ltd.

[57] Yager, R.R., Filev, D.P. (1994). Essential of fuzzy modeling and control. John Wiley & Sons Inc.

[58] Galichet, S., Foulloy, L. (1995). Fuzzy controllers: Synthesis and Equivalences. IEEE Trans. On Fuzzy systems, 3:140-148.

[59] Bertsekas, D.P. (2007). Dynamic Programming and Optimal Control.

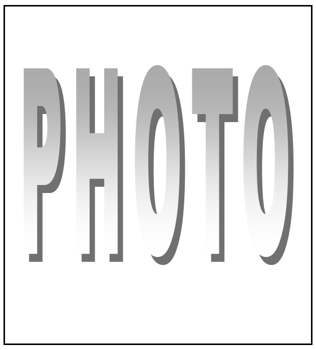
[60] Bertsekas, D.P., Tsitsiklis, J. (1996) Neuro-Dynamic Programming, 512 pages, hardcover.

[61] Tsitsiklis, J. (1994). Asynchronous stochastic approximation and Q-learning. Machime Learning, 3(16):185-202.

[62] Jaakkola, T., Jordan, M.I., Singh, S.P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. Massachusetts Institute of Technology, Vol. 6, No. 6, Pages 1185-1201.

[63] Jouffe, L. (1998). Fuzzy inference system learning by reinforcement methods," IEEE Trans. Syst., Man, Cybern. C, Appl. Rev., vol. 28, no. 3, pp. 338–355. Neural Computation, 6(6):1185-1201.

CURRICULUM VITAE



Name Surname: Vahid Tavakol Aghaei

Place and Date of Birth: Tebriz-1985

E-Mail: vahid.tavakol64@gmail.com

B.Sc. : Electrical and Electronics Engineering

M.Sc. : Control and Automation Engineering