Reinforcement Learning in Generating Fuzzy Systems

Yi Zhou[1] and Meng Joo Er[2]
[1]School of Electrical and Electronic Engineering, Singapore Polytechnic
[2]School of Electrical and Electronic Engineering, Nanyang Technological University
Singapore

1. Introduction

Fuzzy-logic-based modelling and control is very efficient in dealing with imprecision and nonlinearity [1]. However, the conventional approaches for designing Fuzzy Inference Systems (FISs) are subjective, which require significant human's efforts. Other than time consuming, the subjective approaches may not be successful if the system is too complex or uncertain. Therefore, many researchers have been seeking automatic methods for generating the FIS [2].

The main issues for designing an FIS are structure identification and parameter estimation. Structure identification is concerned with how to partition the input space and determine the number of fuzzy rules according to the task requirements while parameter estimation involves the determination of parameters for both premises and consequents of fuzzy rules [3]. Structure identification and input classification can be accomplished by Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL). SL is a learning approach that adopts a supervisor, through which, the training system can adjust the structure and parameters according to a given training data set. In [4], the author provided a paradigm of acquiring the parameters of fuzzy rules. Besides adjusting parameters, self-identified structures have been achieved by SL approaches termed Dynamic Fuzzy Neural Networks in [3] and Generalized Dynamic Fuzzy Neural Networks in [5]. However, the training data are not always available especially when a human being has little knowledge about the system or the system is uncertain. In those situations, UL and RL are preferred over SL as UL and RL are learning processes that do not need any supervisor to tell the learner what action to take. Through RL, those state-action pairs which achieve positive reward will be encouraged in future selections while those which produce negative reward will be discouraged. A number of researchers have applied RL to train the consequent parts of an FIS [6.8]. The preconditioning parts of the FIS are either predefined as in [6] or through the ε-completeness and the squared TD error criteria in [7] or through the "aligned clustering" in [8]. Both DFQL and CQGAF methods achieve online structure identification by creating fuzzy rules when the input space is not well partitioned. However, both methods cannot adjust the premise parameters except when creating new rules. The center position and width of fuzzy neurons are allocated by only considering the input clustering. Moreover, both methods cannot delete fuzzy rules once they are generated even when the rules become redundant.

Source: Theory and Novel Applications of Machine Learning, Book edited by: Meng Joo Er and Yi Zhou,
ISBN 978-3-902613-55-4, pp. 376, February 2009, I-Tech, Vienna, Austria

Since RL has been utilized as a good approach to generate well-matched state-action pairs for the consequent parts of an FIS, it is possible to train the premises and generate the structure of an FIS by RL as well. Thus, a novel algorithm termed Dynamic Self-Generated Fuzzy Q-learning (DSGFQL) which is capable of automatically generating and pruning fuzzy rules as well as tuning the premise parameters of an FIS by RL is presented.

2. Architecture of the fuzzy controlling system

The architecture of the fuzzy controlling system is based on extended Ellipsoidal Basis Function (EBF) neural networks, which are functionally equivalent to a simple Takagi-Sugeno-Kan (TSK) fuzzy systems [4]. The neural networks structure of the DSGFQL system is depicted in Figure 1.

Layer one is an input layer and layer two is a fuzzification layer which evaluates the Membership Functions (MFs) of the input variables. Layer one takes in the senor information and normalizes the value to be in range [0,1]. The MF is chosen as a Gaussian function and each input variable x_i (i = 1, 2, ...,N) has L MFs given by

$$\mu_{ij}(x_i) = exp[-\frac{(x_i - c_{ij})^2}{\sigma_{ij}^2}] \quad i = 1, 2...N, j = 1, 2..., L \tag{1}$$

where μ_{ij} is the *jth* MF of x_i, while c_{ij} and σ_{ij} are the center and width of the *jth* Gaussian MF of x_i respectively. Layer three is a rule layer which decides controlling rules. The output of the *jth* rule R_j(j = 1, 2, ...L) in layer 3 is given by

$$f_j(x_1, x_2, ..., x_N) = exp[-\sum_{i=1}^{N} \frac{(x_i - c_{ij})^2}{\sigma_{ij}^2}] \quad j = 1, 2, ..., L \tag{2}$$

if multiplication is adopted for the T-norm operator.

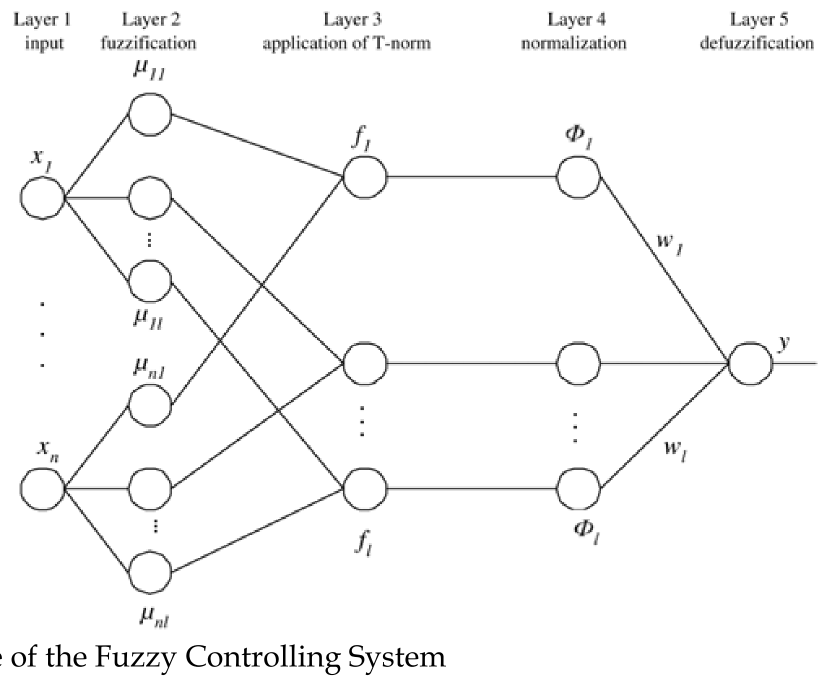


Fig. 1. Structure of the Fuzzy Controlling System

Normalization takes place in layer 4 and we have

$$\phi_j = \frac{f_j}{\sum_{i=1}^{L} f_i} \quad j = 1, 2, ..., L \tag{3}$$

Lastly, nodes of layer five define output variables. If the Center-Of-Gravity (COG) method is performed for defuzzification, the output variable, as a weighted summation of the incoming signals, is given by

$$y = \sum_{j=1}^{L} \phi_j \omega_j \tag{4}$$

where $\omega_j = a_j$ for the Q-learning-based FIS and a_j is the action selected through Q-learning in rule R_j .
Here, the antecedent parts of the fuzzy inference systems are regarded as states and the consequent parts are local actions of each controlling rules. Gaussian MFs are utilized to convert discrete states and actions to continuous ones.
Remark: It should be highlighted that the number of rules does not increase exponentially with the number of inputs as the EBF structure is adopted.

3. Fixed fuzzy Q-learning

We first consider the Fuzzy Q-Learning (FQL) approach of [6] that has a fixed number of fuzzy rule sets and the consequents of the rules can be adjusted based on the FQL. The local actions are selected through competitions based on local q-values which indicate the quality of actions at different states (in different fuzzy rule).

3.1 Local action exploration and selection

Here, we simply use the undirected exploration method employed in [6] to select a local action a from possible discrete action vector A, as follows:

$$a_\pi = \Pi_{a \in A}(q(S, a)) = argmax_{a \in A}(q(S, a) + \eta(S, a)) \tag{5}$$

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

$$s_f = \begin{cases} 1 & if\ max(q) = min(q) \\ \frac{s_p(max(q) - min(q))}{max(\psi)} & otherwise \end{cases} \tag{6}$$

$$\eta = s_f \psi \tag{7}$$

where s_p is the noise size and s_f is the corresponding scaling factor. Decreasing the s_p factor implies reducing the undirected exploration. Details of the exploration-exploitation strategy can be found in [6].

3.2 Generation of global actions

As a learner may visit more than one fuzzy states, the system output/global action is determined by weighting all the local actions. Assume that *L* fuzzy rules have been generated. At time step *t*, the input state is X_t and the normalized firing strength vector of each rule is ϕ. Each rule R_i has *M* possible discrete actions and the local action of each rule is selected from the action set *A* by competing with each other based on their *q*-values. The winning local action a_i of every rule cooperates to produce the global action based on the rule's normalized firing strength ϕ. Therefore, the global action is given by

$$U_t(X_t) = \sum_{i=1}^{L} \pi_A(q_t)\phi_t^i = \sum_{i=1}^{L} \phi_i^t a_i^t \tag{8}$$

where a_i^t is the selected action of rule R_i at time step *t* and U_t is the global action which is the same as *y* in Eq (2).

3.3 Update of Q-value

As defined before, local q-value presents the action quality with respect to the state. In the FQL, global Q-values are also obtained similarly as the global actions and they are weighted value of the local q-values of those local winning actions. The global Q function is given by

$$Q_t(X_t, U_t) = \sum_{i=1}^{L} q_t(S_i, a_i^t)\phi_i^t \tag{9}$$

where U_t is the global action, a_i^t is the selected action of rule R_i and q_t is the *q*-value associated with the fuzzy state S_i and action a_i^t at time step *t*.
Q-learning is a type of Temporal Difference (TD) learning [9]. Based on the TD learning, the value function corresponding to the rule optimal actions is defined as follows:

$$V_t(X_t) = \sum_{i=1}^{L} (max_{a\in A} q_t(S_i, a))\phi_i^t \tag{10}$$

and the global TD error is given by

$$TD_{t+1} = r_{t+1} + \gamma V_t(X_{t+1}) - Q_t(X_t, U_t) \tag{11}$$

where r_{t+1} is the reinforcement signal received at time $t + 1$ and γ is the discounted factor utilized to determine the present value of future TD errors. Note that this TD error term only needs to be estimated with quantities available at time step $t + 1$.
This TD error can be used to evaluate the action just selected. If the TD error is positive, it suggests that the quality of this action should be strengthened for future use; whereas if the TD error is negative, it suggests that the quality should be weakened. The learning rule is given by

$$q_{t+1}(S_i, a_i^t) = q_t(S_i, a_i^t) + \alpha TD_{t+1}\phi_i^t, \;\; i = 1, 2, ..., L \tag{12}$$

where α is the learning rate.

3.4 Eligibility traces

In order to speed up learning, eligibility traces are used to memorize previously visited stateaction pairs, weighted by their proximity at time step *t* [6, 7]. The trace value indicates how state-action pairs are eligible for learning. Thus, it permits not only tuning of parameters used at time step *t*, but also those involved in past steps.

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

$$Tr_t(S_i, a_j) = \begin{cases} \lambda Tr_{t-1}(S_i, a_j) + \phi_i^t & if\ a_j = a_i^t \\ \lambda Tr_{t-1}(S_i, a_j) & otherwise \end{cases} \tag{13}$$

where the eligibility rate λ is used to weight time steps. For all rules and actions, the parameter updating law given by Eq (12) becomes

$$q_{t+1}(S_i, a_j) = q_t(S_i, a_j) + \alpha TD_{t+1} Tr_t(S_i, a_j), \quad i = 1, 2, ..., L,\ j = 1, 2, ..., M \tag{14}$$

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

4. Dynamic self-generated fuzzy Q-learning

In this chapter, a Dynamic Self-Generated Fuzzy Q-Learning (DSGFQL) is presented to automatically determine fuzzy controllers based on reinforcement learning. In the DSGFQL method, fuzzy rules are to be created, adjusted and deleted automatically and dynamically according to the system performance and the contributions of each fuzzy rules.

4.1 ε-Completeness criterion for input space partitioning

In the DSGFQL approach, the input variable fuzzy sets are used to represent appropriate high-dimensional continuous sensory spaces.

First of all, the *ε*-completeness criterion for judging clustering of the input space is adopted as in [3, 7, 8, 10]. As pointed out by the author of [8], a rule in a fuzzy controlling system corresponds to a cluster in the input space geometrically. An input data with higher firing strength of a fuzzy rule means that its spatial location is closer to the cluster center compared to those with smaller strengths. The definition of the *ε*-completeness of fuzzy rules is given in [10] as:

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

The *ε*-completeness criterion is to check whether the whole input space has been completely covered with a certain degree (*ε*). If the criterion is not satisfied, it means more fuzzy rules should be created to accomplish the input space.

4.2 Allocation of newly generated rules

4.2.1 Assignment of membership functions

If the existing system does not satisfy the *ε*-completeness criterion, a new rule should be considered. If the existing fuzzy system passes a similarity test [5, 7], a new Gaussian MF is allocated whose center is with

$$c_{i(L+1)} = x_i \tag{15}$$

and the widths of MFs in the *i*th input variable are adjusted as follows:

$$\sigma_{ik} = \frac{max(|c_{ik} - c_{i(k-1)}|, |c_{i,(k+1)} - c_{ik}|)}{\sqrt{ln(1/\epsilon)}}, \quad k = 1, ..., L+1 \tag{16}$$

where $c_{i(k-1)}$ and $c_{i(k+1)}$ are the two centers of adjacent MFs of the middle MF whose center is c_{ik}. Note that only the new MF and its neighboring MFs need to be adjusted. The main result concerning adjusting MFs to satisfy the ε-completeness of fuzzy rules has been proved in [2, 5]

4.2.2 Novel sharing mechanism for initialization of new q-values

Instead of randomly assigning q-values for the newly created rules, a novel initialization approach is presented here. In order to reduce the training time, the initial q-values for the newly generated fuzzy rule are set by the nearest neighbors. By this means, the newly generated rules learn/share the training experience from the neighboring rules. For instance, if rules R_m and R_n are the two nearest neighbors to the newly generated rule R_i, then

$$q(S_i, a_j) = \frac{q(S_m, a_j) f_m(C_i) + q(S_n, a_j) f_n(C_i)}{f_m(C_i) + f_n(C_i)}, \quad j = 1, 2, \cdots M. \tag{17}$$

where $f_m(C_i)$ and $f_n(C_i)$ stand for the firing strengths of the newly generated center C_i to rules R_m and R_n, which can be obtained from Eq (2).

4.3 Global TD error evaluation

The objective of RL is to maximize the expected value function. Q-learning is a TD-based approach which utilizes the TD error for updating the estimated reward value function. In TD-based methods, the estimated value function is updated, as in [9], as follows:

$$V(s) \longleftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{18}$$

where $TD = r + \gamma V(s') - V(s)$ and $V(s)$ is the value function of state s.
Thus, we have

$$V(s) \longleftarrow V(s) + \alpha TD \tag{19}$$

If the initial value of $V(s, 0)$ is 0, the value function can be regarded as

$$\begin{aligned} V(s,k) &= \alpha \textstyle\sum_{t=0}^{k} [r(t) + \gamma V(s', t+1) - V(s,t)] \\ &= \alpha \textstyle\sum_{t=0}^{k} TD(t) \end{aligned} \tag{20}$$

It can be seen that the TD error estimates the value function and it becomes a criterion for measuring the learning system as well the performance of fuzzy controller. The bigger the TD error is, the greater the value function will be. Therefore, TD error can be considered as a criterion for measuring the system performance in RL.
As a criterion of system performance, the TD error is to be checked periodically, e.g. a number of training steps or an episode. At the end of each training episode or after several steps, the performance of the fuzzy system is examined by the TD error obtained during that period. Average TD error is adopted if the training environment is static. However, discounted TD error is to be considered for dynamic environment, i.e.

$$TD_{avg} = \begin{cases} \frac{\sum_{t=0}^{t=T} TD(t)}{T} & Static\ environment \\ \frac{\sum_{t=0}^{t=T} \gamma^t TD(t)}{T} & Dynamic\ enviroment \end{cases} \tag{21}$$

where *T* is the training time of each episode, *TD*(*t*) is the TD error at time *t* and is the discounted factor which is less than 1.
By this means, the averaged system TD error is adopted as an index of the system performance.

4.4 Local TD error evaluation

Besides evaluation of system performance via reinforcement signals, contributions of each fuzzy rules are also measured through reinforcement signals in the DSGFQL system. In order to evaluate the individual contributions of each fuzzy rules, a local TD error criterion is adopted. By this means, contributions or significance of each fuzzy rules are evaluated through the local TD errors. Inspired from the TD error sharing mechanism in [11], a local TD error scheme is adopted here to share the reinforcement with each local rule according to its contributions. The local TD error for each fuzzy rule is given, similarly as in [11], as follows:

$$TD_j^{local}(t) = \phi_j^t r(t+1) + \gamma \phi_j^{t+1} \max_{a_j \in A_j} q_j^t(s^t, a_j) - \phi_j^t q_j^t(s^t, a_j^t) \tag{22}$$

where $j = 1, 2, ..., L$, and a_j^t is the action selected at time *t*.

Remarks: The local TD error adopted here is different from that in [11] in that the original form is multiplied by the firing strength of the corresponding fuzzy rule. In [11], the local TD error is used for selecting local actions while the local TD error is adopted as an evaluation criterion of individual rules in this thesis.

4.5 Reinforcement learning on input space partitioning

In the DSGFQL approach, the system/global TD error is selected as a measurement of system performance, the local TD error is adopted as an evaluation for individual contributions, while an averaged firing strength is adopted as a measurement for participation. If the averaged firing strength is lower than a threshold value, it means the significance of that rule is too low. Therefore, the rule can be eliminated to reduce the computational cost.
If the global TD error is less than a threshold value, it means that the overall performance of the entire system is unsatisfactory. Therefore, the FNNs need to be adjusted.

4.6 Restructure of FNNs via reinforcement learning

In the DSGFQL system, if the global average TD error is too negative e.g. it is less than a threshold value k_g, the overall performance is considered unsatisfactory. If the FNN passes the ε-completeness criterion but fails the average TD error test, it means that the input space is well partitioned but the overall performance needs to be improved.
To resolve this problem, the weights of some good rules should be increased which means that the system will be modified by promoting the rule with the best performance, e.g. the best local TD error, to a more important position. As a result, the overall performance is

improved as the rules with good contributions participate more in the system while those rules with poor contributions participate less, correspondingly. In this case, the width of the *j*th fuzzy rule's MF (the one with the best local TD error) will be increased as follows

$$\begin{aligned} \sigma_{ij} &\longleftarrow \kappa\sigma_{ij}, \qquad i = 1,2...N \quad if\ \kappa\sigma_{ij} \le \sigma_{max} \\ &\longleftarrow \sigma_{max}, \qquad\qquad\qquad\quad if\ \kappa\sigma_{ij} > \sigma_{max} \end{aligned} \tag{23}$$

where κ is slightly larger than 1 and σ_{max} is the maximum width allowed for Gaussian MFs, which can be set to 1.2 if ε is chosen as 0.5 $\frac{1}{\sqrt{ln(1/\varepsilon)}}$).

Remarks: The restructuring is only adopted when the system under-performs. Enlarging the MF of the rule will increase the firing strength of the rule. As a result, the global action is improved by becoming closer to the best local action.

On the other hand, if the local TD error is larger than a heavy threshold value k_{lh}, but smaller than a light threshold value k_{ll}, a punishment will be given to the rule by decreasing its width of the MF as follows:

$$\sigma_{ik} \longleftarrow \tau\sigma_{ik}, \qquad i = 1,2...N \tag{24}$$

where τ is a positive value less than 1.

Remarks: This is a performance-oriented approach, in which fuzzy rules with the best local TD error values are to be promoted and those with unsatisfactory local TD error are to be demoted or even removed from the system.

4.7 Pruning of redundant fuzzy rules

The local TD error criterion offers a direct evaluation of contributions of fuzzy rules. The more negative the local TD error is, the worse result is offered by the fuzzy rule and vice versa. Therefore, a fuzzy rule should be punished if the local TD error is extremely poor. If the local TD error of a fuzzy rule is less than a heavy threshold value k_{lh} ($k_{lh} < k_{ll}$), the individual contributions are unsatisfactory and the fuzzy rule will be deleted as a serious punishment.

Removing problematic fuzzy rules can help to improve the overall performance of the FNN and new rules may be generated if the ε-completeness criterion fails due to the elimination of rules at the next step. By this means, performance can be improved as "black sheep" has been eliminated and it is better to restart the learning rather than staying in the problematic region.

Besides the TD error, firing strength should also be considered for system evaluation as it is a measurement for participation. If a fuzzy rule has very low firing strength during the entire episode or a long period of time recently, it means that this rule is unnecessary for the system. As more fuzzy rules mean more computation and longer training time, the rule whose mean firing strength over a long period of time is less than a threshold value, k_f, should be deleted.

Remarks: If a fuzzy rule keeps failing the light local TD error check, its firing strength will be reduced by decreasing width of its MF. When the width is reduced to a certain level, it will fail the firing strength criterion and will be deleted. The light local test gives a certain tolerance, which means the fuzzy rule is not deleted due to one slight fault. However, the fuzzy rule which does not provide satisfactory result is still punished by being demoted and it will be deleted if it keeps failing the light local test.

4.8 Gradualism of learning

The values of the thresholds for the TD error criteria are set according to the expection of the task. At the early stage of training, each fuzzy rule needs time to adjust its own performance. For a training system, it is natural to set the demanding requirement small at the beginning and increase it later when the training becomes stable. Thus, the idea of gradualism learning in [12], which uses coarse learning in the early training stage and fine learning in the later stage, is adopted here. Moreover, the elimination process is frozen in the early stage and rules are only pruned after a certain period of time.
In this chapter, a linear gradualism learning is introduced and the values of the thresholds for global and local TD error are set as follows:

$$k_g = k_g^{min} + (k_g^{max} - k_{gr}^{min})\frac{episodes}{episodes + K_r} \tag{25}$$

$$k_{lh} = k_{lh}^{min} + (k_{lh}^{max} - k_{lh}^{min})\frac{episodes}{episodes + K_r} \tag{26}$$

$$k_{ll} = k_{ll}^{min} + (k_{ll}^{max} - k_{ll}^{min})\frac{episodes}{episodes + K_r} \tag{27}$$

where k_g^{min} and k_g^{max} are the minimal and maximal values for the global TD error threshold values respectively, k_{lh}^{min} and k_{lh}^{max} are the minimal and maximal values for the heavy local TD error threshold values respectively, and k_{ll}^{min} and k_{ll}^{max} are the minimal and maximal values for the light TD error threshold values respectively. The term *episodes* is the number of training episodes or periods and *K*r is a controlling constant which can be set according to the number of training episodes. Those threshold values are set according to the target of the training system. If the total TD error (value function) is expected to be *V* in *T* training steps, the values of thresholds of the TD error criterion should be set around the value *V/T* . The maximal threshold values should be slightly bigger than the minimal ones as the system trained by the DSGFQL is supposed to obtain better results via the training. Another suggested guidance is to check the performances of some classical controllers (such as basic fuzzy Controller and FQL controller) and set the results as the minimal threshold values.
Remarks: The gradualism learning is optional for applying the DSGFQL method. The DSGFQL can also be applied with unique threshold values throughout the training, if users do not like to adjust the thresholds. Gradualism learning offers a framework or guideline in case that users are keen on adjusting the thresholds during the learning process.

5. The Khepera robot

The robot employed in this chapter is a miniature mobile robot called Khepera [13] shown in Figure 2. The Khepera mobile robot is cylindrical in shape, with 55 mm in diameter and 30 mm in height weighting only 70g. Its small size allows experiments to be performed in a small work area. The robot is supported by two lateral wheels that can rotate in both directions and two rigid pivots in the front and back.

Fig. 2. The miniature mobile robot: Khepera

The basic configuration of Khepera is composed of the CPU and the sensory/motor boards. The micro-controller includes all the features needed for easy interfacing with memories, with I/O ports and with external interrupts.
The sensory/motor board includes two DC motors coupled with incremental sensors, eight analogue infra-red (IR) proximity sensors denoted by (S_0, ..., S_7) in Figure 3, and an on-board power supply. Each IR sensor is composed of an emitter and an independent receiver. The dedicated electronic interface uses multipliers, sample/holds and operational amplifiers. This allows absolute ambient light and estimation, by reflection, of the relative position of an object to the robot to be measured. By this estimation, the distance between the robot and the obstacle can be derived. This estimation gives, in fact, information about the distance between the robot and the obstacle. The sensor readings are integer values in the range of [0, 1023]. A sensor value of 1023 indicates that the robot is very close to the object, and a sensor value of 0 indicates that the robot does not receive any reflection of the IR signal.

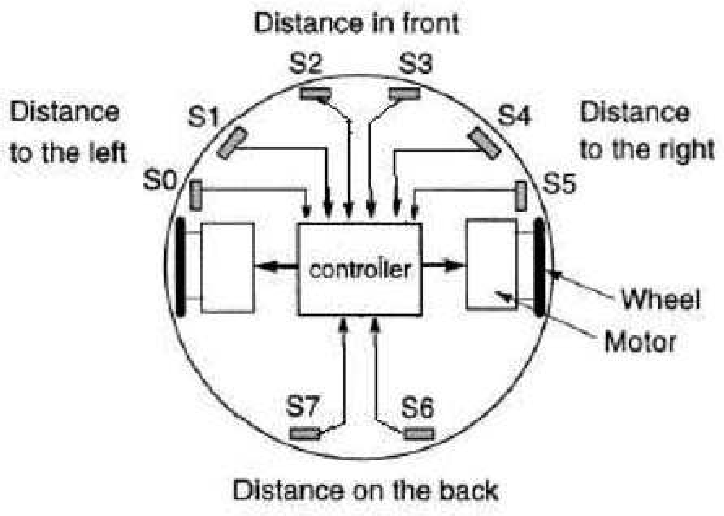


Fig. 3. Position and orientation of sensors on the Khepera

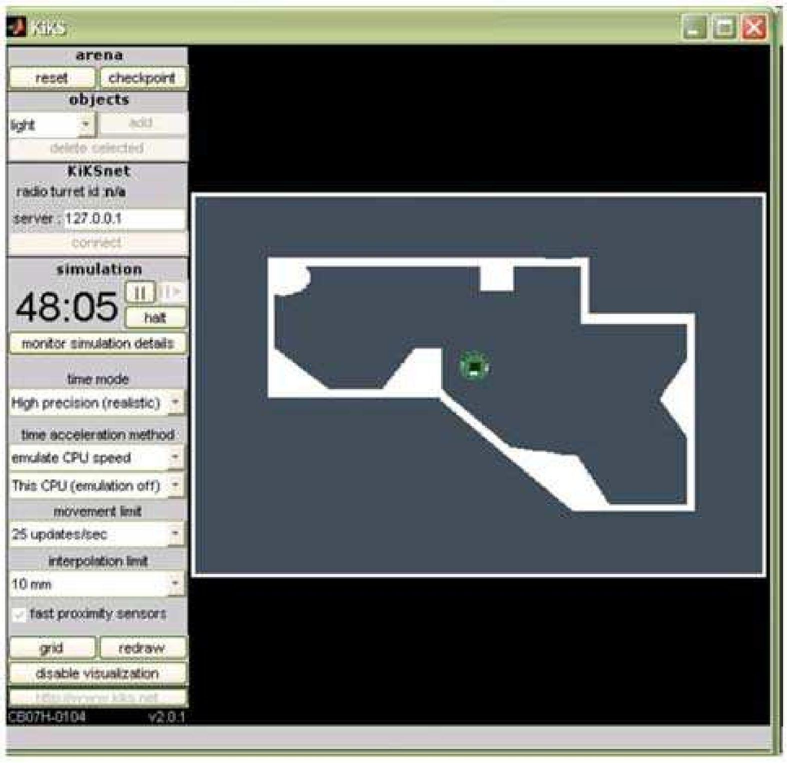
Simulation version of the Khepera [14] is used for carrying out a comprehensive numerical comparison of different approaches. The program simulates Kheperas in the MATLAB environment. Simulated Kheperas are controlled in the same way as real physical Kheperas. Simulation studies on a wall following task are presented in the following sections.

6. Wall following task for Khepera robot

In this chapter, the Khepera robot is to be applied for a wall-following task. The aim of the experiment is to design a controller for wall following. In order to simplify the problem, we only consider robots moving in clockwise direction at a constant speed as that in [7]. Thus, we only need to deal with four input variables, which are the values of sensor S_i (i = 0, 1, 2, 3).
All these sensor values can be normalized within the interval [0, 1]. The output of the controller is the steering angle of the robot. In order for the robot to follow a wall, it must keep the distance from the wall while staying between a maximum distance, d_+, and a minimum distance, $d\_$. The distance to the wall being followed, d, can be calculated via the sensor values. The robot receives a reward evaluation after performing every action U. The reward function depends on this action and the next situation as given in [7]:

$$r = \begin{cases} 0.1, & if(d\_ < d < d_+)\ and\ (U \in [-8^o, +8^o]) \\ -3.0, & if(d \le d\_)\ or\ (d_+ \le d) \\ 0.0, & otherwise. \end{cases} \tag{28}$$

Here, $d\_$ = 0.15 and d_+ = 0.85, which are normalized values, and U is the steering angle of the robot. These values are the same as the settings used in [7].
The training environment with lighted shaped walls used for simulation studies is shown in Figure 4.



light
radio turret id n/a
server : 127.0.0.1
48:05
time mode
High precision (realistic)
time acceleration method
emulate CPU speed
This CPU (emulation off)
movement limit
25 updates/sec
interpolation limit
10 mm
fast proximity sensors
CB07H-0104 v2.0.1

Fig. 4. MATLab simulation environment for wall-following experiments

The performances of different approaches are evaluated at every episode of 1000 control steps according to two criteria, namely the number of failures which correspond to the total number of steps the robot has left the "lane" and rewards which are accumulated. In order to compare the different approaches systematically and find appropriate parameters, we have done a number of comparison studies of these methods on simulations.

7. Simulation results and comparison studies

7.1 Basic fuzzy controller

First, a fuzzy controller based on intuitive experiences is designed. For each input linguistic variable, we define two linguistic values: *Small* and *Big*, whose MFs cover the region of the input space evenly with the match degree set to 0.5. This means that there are 16 (2^4) fuzzy rules. Through trial and error, 16 fuzzy rules are obtained as a special case of the TSK fuzzy controller, whose consequents are constant, as in [7]. The 16 fuzzy rules are listed in Table 1.

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

Table 1. Basic fuzzy control rules for wall following

If the robot only uses the basic fuzzy controller, it can actually follow the wall, but along inefficient trajectories. When only the basic fuzzy controller is used, the robot encounters 79 failures and -164.0 of reward value per episode on average. Certainly, we can provide finer partitioning of the input space or tune the parameters of the MFs and consequents so as to obtain better performances. However, the number of rules will increase exponentially with increase in the number of input variables. Furthermore, tuning consequents of rules is time consuming because of the risk of creating conflicts among the rules. It is almost impossible or impractical to design an optimal fuzzy controller by hand due to a great number of variables involved. Therefore, automatic tuning and determination of fuzzy controlling rules are desired.

7.2 Simulation results for the FQL controller

The set of discrete actions is given by A = [-30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30]. The initial q-values for incorporated fuzzy rules are set as, k_q = 3.0. Other parameters in the learning algorithm are: Discounted factor, γ = 0.95; Trace-decay factor, λ = 0.7; TD learning rate, α = 0.05. The controller with 81 (3^4) fuzzy rules whose MFs is with 0.5 match degree of the input space is also considered. Average performances of the two controllers during 40 episodes over 30 runs are shown in Figures 5 and 6.

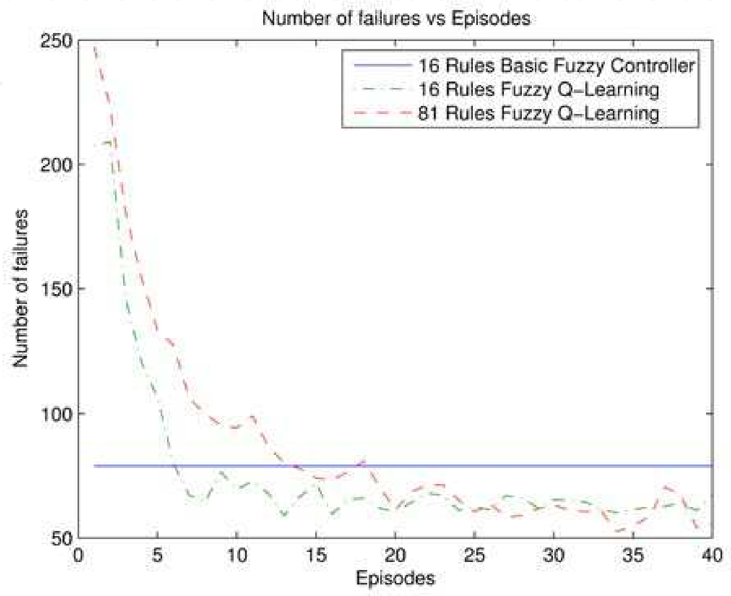


Fig. 5. Number of failures for comparison of performances of fuzzy controllers for (a)Basic fuzzy controller with 16 fuzzy rules, (b)16 fuzzy rules based on FQL, (c)81 fuzzy rules based on FQL

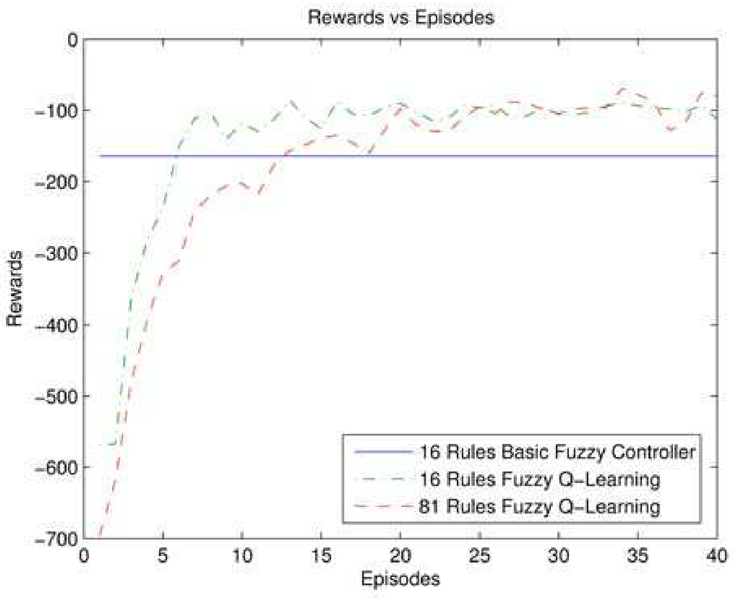


Fig. 6. Reward values for comparison of performances of fuzzy controllers for (a)Basic fuzzy controller with 16 fuzzy rules, (b)16 fuzzy rules based on FQL, (c)81 fuzzy rules based on FQL

At the very beginning, performances of the two controllers based on the FQL are worse than that of the basic fuzzy controller due to the exploration feature of RL. The robot has to explore different actions in order to ensure that better actions are selected in the future. However, the performance of the robot is improved gradually and is better than that of the basic fuzzy controller. Therefore, it can be concluded that performance can be improved by explorations in the action set and tuning consequent parameters.

To assess the effectiveness of finer partitioning of the input space, we compare the performances of the FQL using 16 rules and 81 rules. The speed of learning 81 rules is slower than that of 16 rules because a lot more parameters need to be explored. However, asymptotic performances of these two methods are almost the same. It is impractical to partition the input space further due to the curse of dimensionality. Therefore, automatic generation of premises of fuzzy controllers and automatic partitioning of input space are desired.

7.3 Simulation results of the DSGFQL controller

The DSGFQL is presented as a novel approach of determining premise parts of fuzzy controllers. In this section, the performance of the DSGFQL approach is to be assessed. The parameter values for consequents training are the same as those used in the FQL approach. Other learning parameters for rule generation are ε-completeness, $\varepsilon = 0.5$ and similarity of MF, $k_{mf} = 0.3$. The training aim is to limit the number of failures to 60; therefore, the threshold values of the TD error criterion should be set around $(-3 \times 60/1000) = -0.18$. If it requires each rule to be active at least with an average firing strength for 10 times in the 1000 training steps among about 50 rules, the firing strength threshold value is set as $10.(1000 \times 50) = 0.0002$. Therefore, the global TD error threshold values are $k_g^{max} = -0.05$ and $k_g^{min} = -0.45$; heavy local threshold values are $k_{lh}^{max} = -0.10$ and $k_{lh}^{min} = -0.30$; light local TD error threshold values are $k_{ll}^{max} = 0$ and $k_{ll}^{min} = -0.20$; the firing strength threshold values are $k_f = 0.0002$ and $K_r = 20$, $\kappa = 1.05$ and $\tau = 0.95$. These values give good performances of the algorithms in an initial phase. However, it should be pointed out that we have not searched the parameter space exhaustively.

The performances of the DSGFQL approach shown in Figures 7 and 8 which are also the mean values during 40 episodes over 30 runs.

As expected, the DSGFQL performs better than the FQL with respect to both number of failures and reward values. The number of fuzzy rules generated at every episode is shown in Figure 9. The MFs produced by the DSGFQL after learning input variables at one run are shown in Figure 10. The number of rules can be generated automatically online and does not increase exponentially with the increase in the number of input variables. At the same time, MFs can be adjusted dynamically according to the performance of controlling rules. Thus, a compact and excellent fuzzy controller can be obtained online.

The reason why the DSGFQL method outperforms the FQL method is that the DSGFQL approach is capable of online self-organizing learning. Besides the input space partitioning, both overall and local performances have been evaluated to determine the structure of a fuzzy system. The common approach of conventional input-space partitioning is the socalled grid-type partitioning. The FQL with 16 rules partitions the state space coarsely, on the other hand, the speed of learning 81 rules is slow because a lot more parameters need to be explored. The DSGFQL does not need to partition the input space a *priori* and is suitable for RL. It partitions the input space online dynamically according to both the accommodation boundary and the reinforcement criteria. The compact fuzzy system considers sufficient rules in the critical state space which requires high resolution and does not include redundant rules in unimportant or unvisited state space so that learning is rapid and efficient.

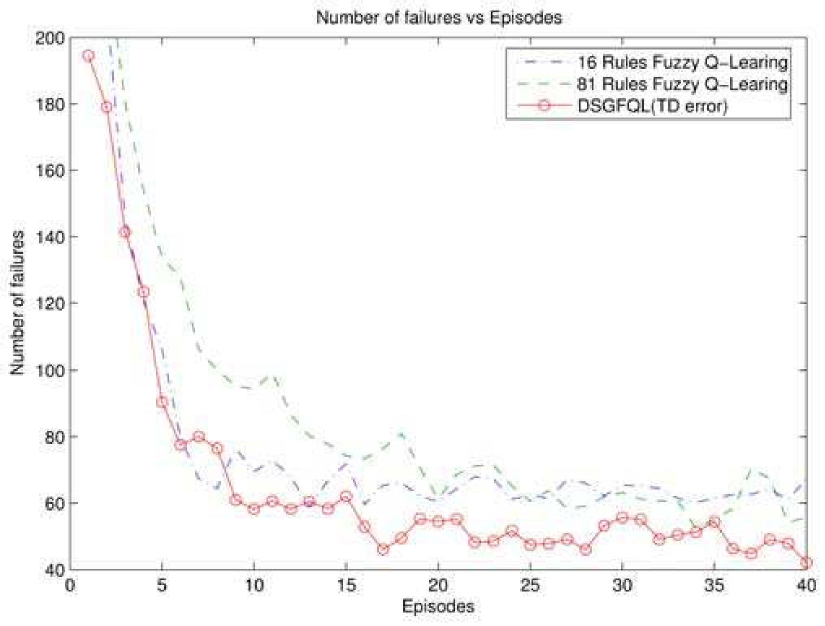


Fig. 7. Number of failures for comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL, (b) 81 fuzzy rules based on FQL, (c) DSGFQL

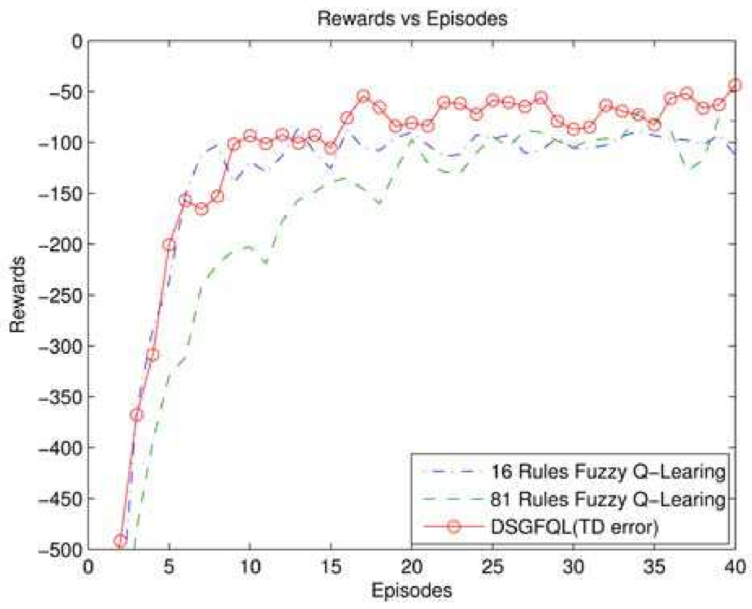


Fig. 8. Reward values for comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL, (b) 81 fuzzy rules based on FQL, (c) DSGFQL

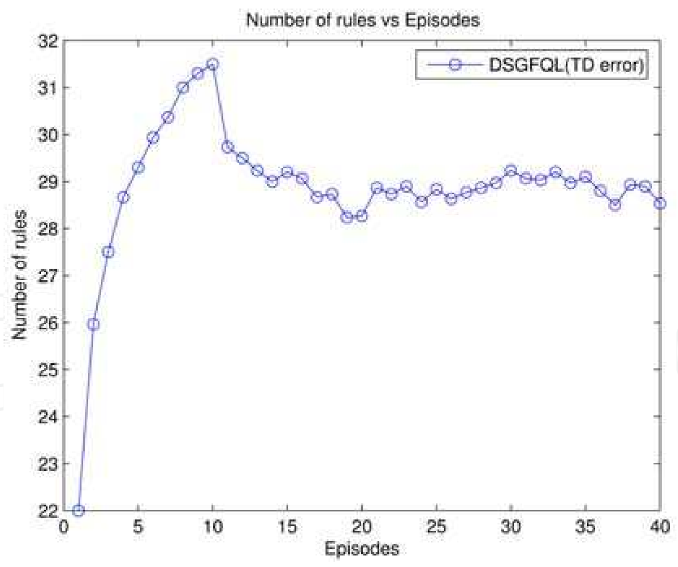


Fig. 9. Number of rules generated by the DSGFQL approach

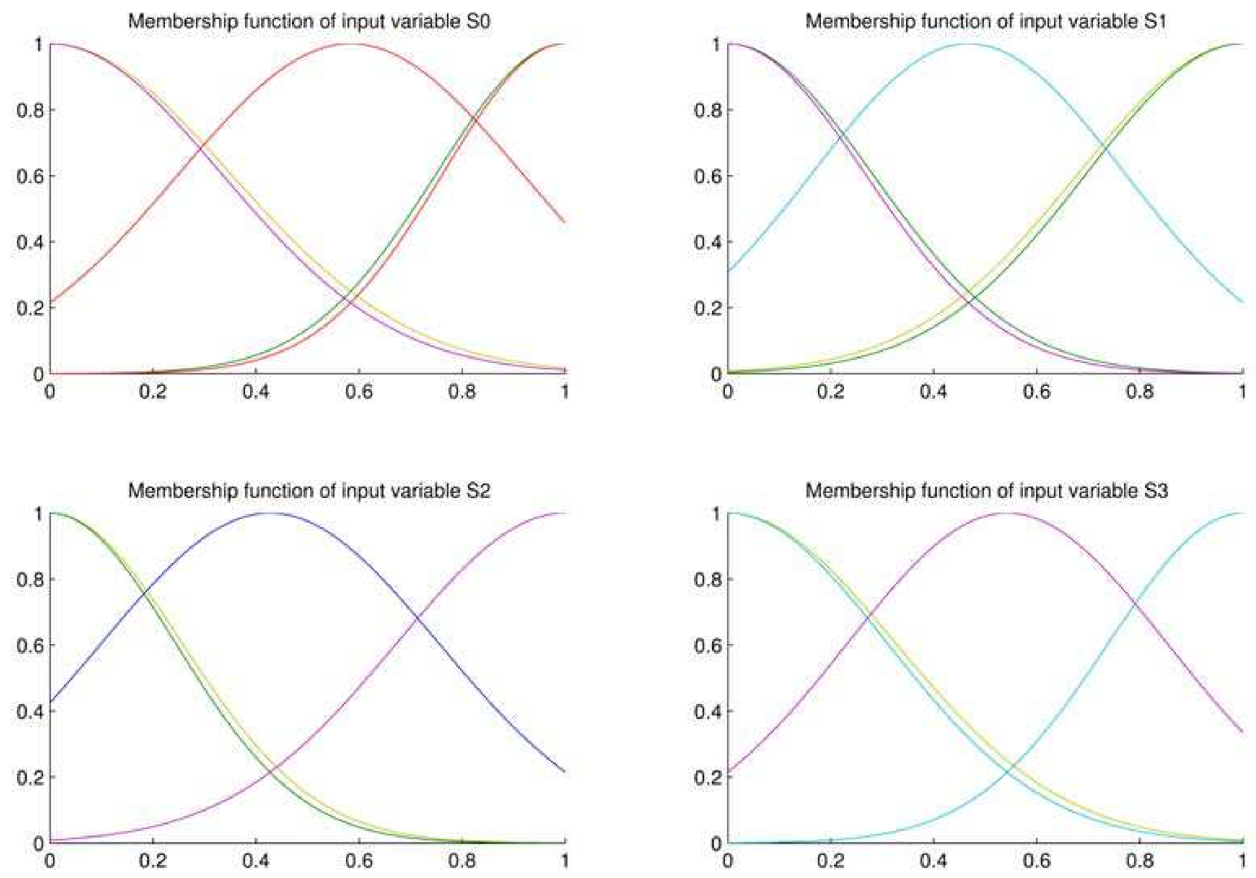


Fig. 10. Membership functions after learning at one run

8. Conclusions

In this chapter, a novel reinforcement learning method termed (DSGFQL) is introduced for navigation tasks of mobile robots. In the DSGFQL approach, continuous sensory states and action space for mobile robots can be self-generated. Controlling rules are expressed in fuzzy logic and the rules can be generated, adjusted and pruned automatically and dynamically without a priori knowledge or supervised learning. Online clustering method is adopted for partitioning the input space (sensory space of robots) and fuzzy rules are generated automatically by reinforcement learning. In the DSGFQL, new rules are created if system fails the ε-completeness criterion. At the same time, contributions of existing rules are to be determined through a reinforcement sharing mechanism. When the performance of the control system is poor, controlling rules are adjusted according to reinforcement signals. Q-learning is adopted in selecting optimal local actions in each fuzzy rule and the system action is a weighted continuous action via fuzzy reasoning. Comparative studies with other fuzzy controllers on mobile robot navigation demonstrate that the DSGFQL method is superior.

9. References

[1] L. X. Wang, Universal approximation by hierarchical fuzzy systems, *Fuzzy sets and systems*, vol. 93, no. 22, pp. 223-230, 1998.

[2] M. J. Er, S. Wu, and Y. Gao, *Dynamic fuzzy neural networks: architectures, algorithms and applications*, McGraw-Hill, New York, 2003.

[3] S. Wu and M.J. Er, Dynamic fuzzy neural networks: A novel approach to function approximation, *IEEE Trans. on Systems, Man and Cybernetics, Part B*, vol. 30, no. 2, pp. 358-364, 2000.

[4] J. S. R. Jang, Anfis: Adaptive-network-based fuzzy inference system, *IEEE Trans. System, Man and Cybernetics*, vol. 23, no. 3, pp. 665-684, 1993.

[5] S. Wu, M.J. Er, and Y. Gao, A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks, *IEEE Trans. on Fuzzy Systems*, vol. 5, no. 4, pp. 578-594, 2001.

[6] L. Jouffe, Fuzzy inference system learning by reinforcement methods, *IEEE Trans. Systems, Man, and Cybernetics, Part C*, vol. 28, no. 3, pp. 338-335, 1998.

[7] M. J. Er and C. Deng, Online tuning of fuzzy inference systems using dynamic fuzzy q-learning, *IEEE Trans on Systems, Man and Cybernetics, Part B*, vol. 34, no. 3, pp. 1478-1489, 2004.

[8] C. F. Juang, Combination of online clustering and q-value based ga for reinforcement fuzzy systems, *IEEE Trans on Fuzzy Systems*, vol. 13, no. 3, pp. 289-302, 2005.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, The MIT Press, Cambridge, Massachusetts., 1998.

[10] C. C. Lee, Fuzzy logic in control systems: fuzzy logic controller, *IEEE Trans. System, Man and Cybernetics*, vol. 20, no. 2, pp. 404-435, Mar 1990.

[11] P. Ritthiprava, T. Maneewarn, D. Laowattana, and J. Wyatt, A modi_ed approach to fuzzy-q learning for mobile robots, in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, 2004, vol. 3, pp. 2350 - 2356.

[12] K. B. Cho and B. H. Wang, Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction, *Fuzzy Sets Syst.*, vol. 83, no. 3, pp. 325-339, 1996.

[13] S. A. K-Team, Switzerland, *Khepera 2 user manual*, 2002.
[14] T. Nilsson, [online]. avaiable: http://www.kiks.f2s.com.