Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A Novel Technique to Design a Fuzzy Logic Controller Using Q(λ)-learning and Genetic Algorithms in The Pursuit-Evasion Game

Sameh F. Desouky
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
sameh@sce.carleton.ca

Howard M. Schwartz
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
schwartz@sce.carleton.ca

***Abstract*—This paper presents a novel technique to tune the parameters of a fuzzy logic controller using a combination of reinforcement learning and genetic algorithms. The proposed technique is called a Q(λ)-learning based genetic fuzzy logic controller (QLBGFLC). The proposed technique is applied to a pursuit-evasion game in which the pursuer does not know its control strategy. We assume that we do not even have a simplistic PD controller strategy. The learning goal for the pursuer is to self-learn its control strategy. The pursuer should do that on-line by interaction with the environment; in this case the evader. Our proposed technique is compared with the optimal strategy, Q(λ)-learning only, and unsupervised genetic algorithm learning. Computer simulations show the usefulness of the proposed technique.**

***Index Terms*—Fuzzy control, genetic algorithms, pursuit-evasion game, Q(λ)-learning, reinforcement learning.**

I. Introduction

Reinforcement learning (RL) is a computational approach to learning through interaction with the environment [1], [2]. The main advantage of RL is that it does not need either a teacher, training data or known model. RL is suitable for intelligent robot control especially in the field of autonomous mobile robots [3]–[7].

Limited studies have applied RL alone to solve environmental problems but its use with other learning algorithms has increased. In [8], a RL approach is used to tune the parameters of a fuzzy logic controller (FLC). This approach is applied to a single case of one robot following another along a straight line. In [7], [9], the authors proposed a hybrid learning approach that combines a neuro-fuzzy system with RL in a two-phase structure applied to an obstacle avoidance mobile robot. In the first phase, supervised learning is used to tune the parameters of a FLC. In the second phase, RL is employed so that the system can re-adapt to a new environment. The problem that could appear in their proposed approach is that if the training data is hard or expensive to obtain then supervised learning can not be applied. In the work by [10], the authors solve this problem by using Q-learning in the first phase as an expert to obtain training data. This training data is used to tune the weights of an artificial neural network (ANN) controller which is applied to a mobile robot path planning problem. In [11], the use of RL in the multi-agent pursuit-evasion problem is discussed. The individual agents learn a particular pursuit strategy. However, the authors do not use a realistic robot model or robot control structure. In [12], a multi-robot pursuit-evasion game is investigated. The model consists of a combination of aerial and ground vehicles. However, the unmanned vehicles are not learning. They just do the actions that they received from a central computer system.

In this paper we design a self-learning FLC using a novel technique called a Q(λ)-learning based genetic fuzzy logic controller (QLBGFLC). The proposed technique is applied to a pursuit-evasion game in which the pursuer does not know its control strategy. We assume that we do not even have a simplistic PD controller strategy. The learning process of the proposed technique consists of two phases. In phase 1, the pursuer self-learns its control strategy on-line by interaction with the evader. In this phase, Q(λ)-learning is used to obtain an estimation for the optimal strategy then the states and their corresponding actions are stored in a lookup table. In phase 2, the state-action pairs stored in the lookup table are used as training data to tune the parameters of a FLC using GAs. Then, we run the pursuit-evasion game with the tuned FLC and use GAs again to fine tune the FLC parameters during the interaction with the evader.

This paper is organized as follows: in Section II, the pursuit-evasion game is described. In Section III, the problem statement is discussed. Some basic terminologies for RL, FLC, and GAs are reviewed in Section IV. The proposed technique is described in Section V. Section VI represents computer simulation and the results are discussed in Section VII.

II. Pursuit-Evasion Game

The pursuit-evasion game is one application of differential games [13] in which a pursuer tries to catch an evader in minimum time where the evader tries to escape from the pursuer. The pursuit-evasion game is shown in Fig. 1. Equations of motion for the pursuer and the evader robots are [14], [15]

$$\begin{aligned} \dot{x}_i &= V_i \cos \theta_i \\ \dot{y}_i &= V_i \sin \theta_i \\ \dot{\theta}_i &= \frac{V_i}{R_i} \tan u_i \end{aligned} \tag{1}$$

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

Our strategies are to make the pursuer faster than the evader ($V_p > V_e$) but at the same time to make it less maneuverable than the evader ($u_{p_{max}} < u_{e_{max}}$). The control strategy for the pursuer is discussed in Section III. The control strategy for the evader is to maximize the distance between them so

$$u_e = \tan^{-1}\left(\frac{y_e - y_p}{x_e - x_p}\right) - \theta_e \tag{2}$$

The capture occurs when the distance between the pursuer and the evader is less than a certain amount, ℓ. This amount is called the capture radius which is defined as

$$\ell = \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} \tag{3}$$

III. Problem Statement

The problem assigned in this paper is that we assume that the pursuer does not know its control strategy. It is not told which actions to take so as to be able to catch the evader. We assume that we do not even have a simplistic PD controller strategy. The learning goal is to make the pursuer able to self-learn its control strategy. It should do that on-line by interaction with the evader.

From several learning techniques we choose reinforcement learning (RL). RL methods learn without a teacher, without anybody telling them how to solve the problem. RL is related to problems where the learning agent does not know what it must do [16]. It is the most appropriate learning technique for our problem.

One reason for choosing the pursuit-evasion game is that the time-optimal control strategy is known so, it can be a reference for our results. By this way, we can check the validity of our proposed technique.

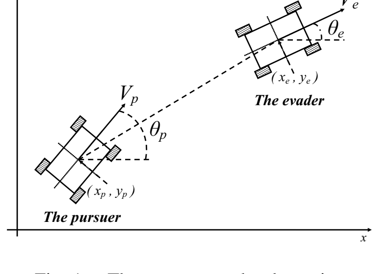


Fig. 1. The pursuer evader dynamics

IV. Preliminaries

A. Reinforcement Learning

Reinforcement learning (RL) is an interaction between agent and environment as shown in Fig. 2 [1]. It consists mainly of two blocks, an agent which tries to take actions so as to maximize the discounted return, R, and an environment which provides the agent with rewards. The discounted return, R_t, at time t is defined as

$$R_t = \sum_{k=0}^{\tau} \gamma^k r_{t+k+1} \tag{4}$$

where γ is the discount factor, $(0 < \gamma \leq 1)$, τ is the terminal point. Any task can be divided into independent episodes, τ is the end of an episode. To be able to maximize the total rewards, the performance of an action, a, taken in a state, s, under policy, π, is evaluated by the action value function, $Q^\pi(s,a)$,

$$\begin{aligned} Q^\pi(s,a) &= E_\pi(R_t | s_t = s, a_t = a) \\ &= E_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} | s_t = s, a_t = a\right) \end{aligned} \tag{5}$$

where $E_\pi(\cdot)$ is the expected value under policy, π.

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

$$D(t) = \sqrt{(x_e(t) - x_p(t))^2 + (y_e(t) - y_p(t))^2} \tag{6}$$

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

$$\Delta D(t) = D(t) - D(t+1) \tag{7}$$

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

$$\Delta D_{max} = V_{rmax} T \tag{8}$$

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

$$r_{t+1} = \Delta D(t) / \Delta D_{max} \tag{9}$$

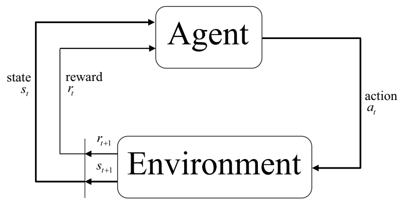


Fig. 2. Agent-environment interaction in RL

The way to choose an action is a trade-off between exploitation and exploration. The $\varepsilon-$greedy action selection method is a common way of choosing the actions. This method can be stated as

$$a_t = \begin{cases} a^*, & \text{with probability } \; 1-\varepsilon; \\ \text{random action}, & \text{with probability } \; \varepsilon. \end{cases} \tag{10}$$

where $\varepsilon \in (0,1)$ and a^* is the greedy action defined as

$$a^* = \arg\max_{a'} Q(s,a') \tag{11}$$

The RL method is searching for the optimal policy, π^*, by searching for the optimal value function, $Q^*(s,a)$ where

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{12}$$

Many methods have been proposed for estimating the optimal value functions. Here, we focus on the temporal difference method (TD). The TD method has several control algorithms. The most widely used and well-known control algorithm is Q-learning which is known as the best RL algorithm [17]. In addition, it is widely used in learning for most mobile robot applications [18], [19].

Q-learning, which was first introduced by Watkins in his Ph.D [20], is an off-policy algorithm. This means that it has the ability to learn without following the current policy. The state and action spaces are discrete and their corresponding value function is stored in a lookup table known as a Q-table. To use Q-learning with continuous systems (continuous state and action spaces), one can discretize the state and action spaces [10], [18], [21]–[23] or use some types of function approximations such as fuzzy systems [4], [24]–[26] and GAs [27], [28]. A one-step update rule for Q-learning is defined as

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \triangle_t \tag{13}$$

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

$$\triangle_t = r_{t+1} + \gamma \max_{\acute{a}} Q(s_{t+1},\acute{a}) - Q(s_t,a_t) \tag{14}$$

Equation (13) is a one-step update rule. It updates the value function according to the immediate reward obtained from the environment. To update the value function based on a multi-step update rule one can use eligibility traces [1].

Eligibility traces are used to modify a one-step TD algorithm, TD(0), to be a multi-step TD algorithm, TD(λ). One type of eligibility traces is the replacing eligibility trace defined for all s, a, as

$$e_t(s,a) = \begin{cases} 1, & \text{if } s = s_t \text{ and } a = a_t; \\ 0, & \text{if } s = s_t \text{ and } a \neq a_t; \\ \lambda\gamma e_{t-1}(s,a), & \text{if } s \neq s_t. \end{cases} \tag{15}$$

where $e_o = 0$, and λ is the trace-decay parameter, $(0 \leq \lambda \leq 1)$. When $\lambda = 0$ that means a one-step update, TD(0), and when $\lambda = 1$ that means an infinite-step update. Eligibility traces are used to speed up the learning process and hence to make it suitable for on-line applications. Now we will modify (13) to be

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\, e_t \triangle_t \tag{16}$$

B. Fuzzy Logic Controller

A block diagram of a fuzzy logic controller (FLC) system is shown in Fig. 3. The FLC has two inputs, the error in the pursuer angle, δ, and its derivative, $\dot{\delta}$, and the output is the steering angle, u_p, where

$$\delta = \tan^{-1}\left(\frac{y_e - y_p}{x_e - x_p}\right) - \theta_p \tag{17}$$

For the inputs of the FLC, we use the gaussian MF described by

$$\mu(x) = \exp\left(-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2\right) \tag{18}$$

where σ is the standard deviation and c is the mean. We use a zero order Takagi-Sugeno-Kang fuzzy inference system (TSK-FIS) in which the consequent part is a constant function and the rule is described as follows

$$R_l : IF \; x_1 \; is \; A_1^l \; AND \; x_2 \; is \; A_2^l \; AND \; ... \; AND \; x_N \; is \; A_N^l \; THEN \; f_l = K_l \tag{19}$$

where R_l is the l^{th} rule, A_i^l is the fuzzy set for the input variable x_i, and K_l is the consequent parameter for the rule output f_l. The crisp output which is the steering angle, u_p, is calculated using the weighted average defuzzification method as follows

$$u_p = \frac{\sum_{l=1}^{L}((\prod_{n=1}^{N}\mu^{A_n^l}(x_n))K_l}{\sum_{l=1}^{L}(\prod_{n=1}^{N}\mu^{A_n^l}(x_n))} \tag{20}$$

where L is the number of rules and N is the number of input variables.

C. Genetic Algorithms

Genetic algorithms (GAs) are search and optimization techniques that are based on a formalization of natural genetics [29], [30]. GAs have been used to overcome the difficulty and complexity in the tuning of the FLC parameters such as MFs, scaling factors and control rules [31]–[36].

GAs search a multidimensional parameter space to find an optimal solution. A given set of parameters is referred to as a chromosome. The parameters can be either decimal or binary numbers. The GA is initialized with a number of randomly selected parameter vectors or chromosomes. This set of chromosomes is the initial population. Each chromosome is tested and evaluated based on a fitness function, in control

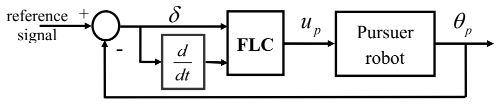


Fig. 3. Block diagram of a FLC system

engineering we would refer to this as a cost function. The chromosomes are sorted based on the lowest cost function or the ranking of the fitness functions. One then selects a number of the best, according to the fitness function, chromosomes to be parents of the next generation of chromosomes. A new set of chromosomes is selected based on reproduction.

In the reproduction process, we generate new chromosomes, which are called children. We use two GA operations. The first operation is a crossover in which we choose a pair of parents and select a random point in all of their chromosomes and make a cross replacement from one parent to another. The second operation is a mutation in which a parent is selected and we change one or more of its parameters to get a new child. Now, we have a new population to test again with the fitness function. The genetic process is repeated until the last iteration is reached. The coding process in the proposed technique using the GAs will be described in detail in Section VI.

V. Q(λ)-LEARNING BASED GENETIC FUZZY LOGIC CONTROLLER

We design a self-learning FLC using a novel technique that combine Q(λ)-learning with FLC and GAs. The proposed technique is called a Q(λ)-learning based genetic fuzzy logic controller (QLBGFLC). The proposed technique is applied to a pursuit-evasion game in which the pursuer does not know its control strategy. The learning process passes through two phases as shown in Fig. 4.

In phase 1, Q(λ)-learning is used to obtain a suitable estimation for the optimal strategy of the pursuer. The state, s, consists of the error, δ, defined by (17) and its derivative, $\dot{\delta}$, and the action, a, is the steering angle of the pursuer, u_p. The states and their corresponding greedy actions are then stored in a lookup table. The reward function used is defined by (9).

Phase 2 consists of two stages. In stage 1, the state-action pairs stored in the lookup table are used as the training data to tune the parameters of a FLC using GAs. In this stage, the mean square error (MSE) defined by (21) is used as the fitness function in which the error is the difference between the greedy action, a^*, obtained from phase 1 and the output of the FLC, u_{flc}. GAs in this stage are used as a supervised learning technique. In stage 2, we run the pursuit-evasion game with the tuned FLC. GAs are then used to fine tune the parameters of the FLC during the interaction with the evader. In this stage, the capture time which we want to minimize is used as the fitness function. In this stage, GAs are used as an unsupervised learning technique with a priori knowledge obtained from stage 1. The coding process in the proposed technique using the GAs will be described in details in Section VI.

Q-learning alone has the limitation in that it is too hard to visit all the state-action pairs [10]. We try to cover most of the state-action space but we can not cover all the space. In addition, there are hidden states that are not taken into consideration due to the discretization process. Hence Q-learning alone can not find the optimal strategy. Therefore, we extend the learning algorithm by using a FLC to generalize Q-learning over the continuous state space [37]. In addition, GAs are used as an optimization technique to tune the parameters of the FLC. Both FLC and GAs are used to compensate for the limitation of the Q-learning. The learning process in phase 1 and phase 2 is described in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 (Phase 1: Q(λ)-learning)

```
1) Discretize the state space, $S$, and the action space, $A$.
2) Initialize $Q(s,a) = 0$.
3) For each episode
    a) Initialize $e(s,a) = 0$.
    b) Initialize $(x_p, y_p) = (0,0)$.
    c) Initialize $(x_e, y_e)$ randomly.
    d) Initialize $s_t = (\delta, \dot{\delta})$ randomly.
    e) Select $a_t$ using the $\varepsilon-$greedy method.
    f) For each play
        i) Receive $r_{t+1}$.
        ii) Observe $s_{t+1}$.
        iii) Calculate $e_{t+1}$.
        iv) Update $Q(s_t, a_t)$.
    g) End
4) End
5) $Q \leftarrow Q^*$.
6) Assign a greedy action, $a^*$, to each state, $s$.
7) Store the state-action pairs in a lookup table.
```

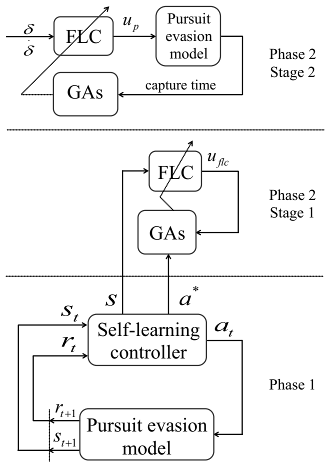


Fig. 4. The proposed QLBGFLC technique

In the case of large continuous state space, which is not our case, the discrete representation of RL is intractable. In our case, the state values represent the error in angle, δ, and its derivative, $\dot{\delta}$, where $\delta \in [-1,1]$ and $\dot{\delta} \in [-1,1]$. The action is the steering angle of the pursuer, u_p, where $u_p \in [-0.5, 0.5]$. These values are relatively coarsely discretized such that the state and the action spaces are not prohibitively large.

A. Comparison of Unsupervised GA Learning to the Proposed Technique

For comparative purpose, we will also implement a general unsupervised GA learning method. The unsupervised GA learning method will be initialized with randomly chosen FLC parameters. The GAs adjust the parameters to maximize the closing distance given by (9). Therefore, (9) acts as the fitness function for the unsupervised GA learning method.

In the proposed technique the GAs are used in phase 2 stage 1 to tune the FLC parameters as determined from $Q(\lambda)$-learning in phase 1. The GAs use an MSE criterion given by (21) that measures the difference between control or action defined by $Q(\lambda)$-learning and the output of the FLC. The FLC parameters are then tuned by the GAs to achieve the greedy actions defined by $Q(\lambda)$-learning in phase 1. Then, in phase 2 stage 2, the GAs fine tune the FLC parameters to achieve a minimizing capture time.

Algorithm 2 (Phase 2: GAs learning)

```
• Stage 1
    1) Get the state-action pairs from the lookup table.
    2) Initialize a population, $P$, randomly.
    3) For each iteration, $I$
        a) For each set of chromosomes in the population
            i) Construct a FLC from the population chromosomes.
            ii) For each state, $s$,
                – Calculate the FLC output, $u_{flc}$.
            iii) End
            iv) Calculate fitness function using MSE as
                $$MSE = \frac{1}{2L}\sum_{l=1}^{L}(u_d^l - u_{flc}^l)^2 \qquad (21)$$
                where $L$ is the number of input/output data pairs and $u_d^l$ is the $l^{th}$ greedy action, $a^*$.
        b) End
        c) Sort the entire chromosomes of the population according to their fitness values.
        d) Select a portion of the sorted population as the new parents.
        e) Create a new generation for the remaining portion of population using crossover and mutation.
    4) End
• Stage 2
    1) Initialize a population, $P$, from the tuned FLC obtained from stage 1.
    2) For each iteration, $I$
        a) Initialize $(x_e, y_e)$ randomly.
        b) For each set of chromosomes in the population
            i) Initialize the state $(\delta, \dot{\delta}) = (0,0)$.
            ii) Initialize $(x_p, y_p) = (0,0)$.
            iii) Construct a FLC from the population chromosomes.
            iv) For each play
                A) Calculate the FLC output, $u_p$.
                B) Observe the next state.
            v) End
            vi) Observe the fitness function which is the capture time that we want to minimize.
        c) End
        d) Sort the entire chromosomes of the population according to their fitness values.
        e) Select a portion of the sorted population as the new parents.
        f) Create a new generation for the remaining portion of population using crossover and mutation.
    3) End
```

VI. Computer Simulation

At the beginning of each episode, the pursuer starts motion from the position $(0,0)$ with an initial orientation $\theta_p = 0$ rad and turning radius $R_p = 2$ m. The velocity of the pursuer $V_p = 2$ m/s and the steering angle $u_p \in [-0.5, 0.5]$ rad. The evader starts motion from a random position for each episode with an initial orientation $\theta_e = 0$ rad and turning radius $R_e = 0.5$ m. The velocity of the evader $V_e = 1.0$ m/s which is half that of the pursuer (slower) and the steering angle $u_e \in [-1, 1]$ rad which is twice that of the pursuer (more maneuverable). The capture radius, $\ell = 0.10$ m.

To build the state space we discretize the ranges of the inputs, δ and $\dot{\delta}$, by 0.2. The ranges of δ and $\dot{\delta}$ are set to be from -1 to 1 so the discretized values for δ and $\dot{\delta}$ will be $(-1.0, -0.8, -0.6, \ldots, 0.8, 1.0)$. There are 11 discretized values for δ and 11 discretized values for $\dot{\delta}$. These values are combined to form $11 \times 11 = 121$ states. To build the action space we discretize the range of the action, u_p, by 0.1. The range of the action is set to be from -0.5 to 0.5 so the discretized values in the action space are $(-0.5, -0.4, -0.3, \ldots, 0.4, 0.5)$. There are 11 actions and the dimension of the Q-table will be 121×11. We choose the number of games or episodes to be 800, the number of plays or steps in each episode is 10000, the discount factor $\gamma = 0.9$, and the trace-decay parameter $\lambda = 0.8$. We make the learning rate, α, decrease with each episode such that

$$\alpha = 0.2\,e^{-0.005\,i} \qquad (22)$$

and we also make ε decrease with each episode such that

$$\varepsilon = 0.1\,e^{-0.03\,i} \qquad (23)$$

where i is the current episode.

In phase 1, the initial state of the system, $(\delta, \dot{\delta})$, and the initial position of the evader, (x_e, y_e), are chosen randomly at the beginning of each episode to cover most of the situations and states. The states and their corresponding greedy actions, a^*, are stored in a lookup table and then used for tuning the parameters of the FLC using GAs in phase 2. Since we have 121 states so the number of state-action pairs in the lookup table is 121.

In phase 2, we use GAs to tune the parameters of a FLC. Now, we will describe the coding of the FLC parameters using the GAs. The FLC to be learned has 2 inputs, the error, δ, and its derivative, $\dot{\delta}$. Each input variable has 3 MFs with a total of 6 MFs. We use gaussian MFs defined in (18) which have 2 parameters to be tuned. These parameters are the the standard deviation, σ, and the mean, c. The total number of input parameters to be tuned is 6 standard deviations $+6$ means $= 12$ parameters. The crisp output of the system, which is the steering angle of the pursuer, u_p, is calculated using (20). We use the zero-order TSK-FIS defined in (19) and modify it to be

$$R_l : IF\ \delta\ is\ A_1^l\ AND\ \dot{\delta}\ is\ A_2^l\ THEN\ f_l = K_l \quad (24)$$

where $l = 1, 2, ..., 9$ so we have 9 rules. Each rule has a parameter, K_l, to be tuned with a total number of 9 parameters to be tuned for the output part.

Now, a FLC will be coded to a chromosome with length $6+6+9 = 21$ genes. We use decimal numbers in the coding process. The population consists of a set of chromosomes (codded FLCs). In the reproduction process, we generate new chromosomes. We use two GA operations. The first operation is a crossover in which we choose a pair of parents and select a random point, r, in all of their chromosomes and make a cross replacement from one parent to another. The second operation is a mutation in which we generate a chromosome randomly to avoid a local minimum for the fitness function. Now, we have a new population to test again with the fitness function. The genetic process is repeated until the last iteration is reached.

Learning in phase 2 consists of two stages. In stage 1, GAs are used as supervised learning. All the FLC parameters are initialized randomly. The fitness function used is the MSE defined in (21). The desired output, u_d, is the greedy action, a^*, obtained from the lookup table of phase 1 and the actual output is the output of the FLC, u_{flc}, so the results of this stage will not be better than that of phase 1 so we need to perform stage 2. In stage 2, GAs are used as unsupervised learning with a priori knowledge obtained from stage 1 i.e. we initialize the FLC parameters from those obtained in stage 1. In this stage, the capture time which we want to minimize is used as the fitness function instead of the MSE used in stage 1.

To validate our proposed technique, we compare its results with the results of the optimal strategy, Q(λ)-learning only, and unsupervised GA learning. The values of the GAs parameters used in stage 1 of phase 2, stage 2 of phase 2, and unsupervised GA learning are shown in Table I. Notice that in stage 2 of phase 2 we already have a tuned FLC and we just fine tune it but in unsupervised GA learning we have no idea about the FLC parameters so we initialize them randomly and of course this random initialization will increase the learning time as we see in Section VII.

VII. Results

To execute our computer simulations, we use a pentium 4 computer with a 3.2 GHz clock frequency anf 2.0 Gigabytes of RAM. In phase 1, we run the simulation of the Q(λ)-learning which takes on average 2.5 minutes. At the end of phase 1, we store the states, $s = (\delta, \dot{\delta})$, and their corresponding greedy actions, a^*, to use them as training data in phase 2.

Phase 2 consists of two stages. In stage 1, the state-action pairs stored at the end of phase 1 are used to tune the parameters of a FLC using GAs. This process takes on average 0.16 minutes. In stage 2, We put the FLC tuned in stage 1 in our pursuit-evasion game then we run the system simulation and fine tune the parameters of the FLC during the interaction with the evader using GAs. The initial population of the GAs is generated from the FLC tuned in stage 1. The learning process in this stage takes on average 3 minutes with a total time of 5.5 minutes for the learning process in our proposal QLBGFLC technique.

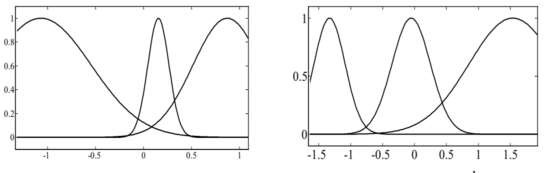
Fig. 5 shows the MFs for the two inputs, δ and $\dot{\delta}$, after the learning process. Table II shows the fuzzy decision table after the learning process. Table III shows the capture times for different initial positions of the evader using the optimal strategy of the pursuer, Q(λ)$-$learning only, unsupervised GA learning, and our proposed QLBGFLC. From Table III we can see that Q(λ)-learning only is not enough to get the desired performance in comparison with the optimal strategy and the other techniques. Unsupervised GA learning gets better performance than the Q(λ)-learning only and its performance approaches the performance of our proposed technique but it takes on average 51.7 minutes in the learning process. Our proposed technique has the best performance which approaches the performance of the optimal strategy. We can see that there is a slight difference in capture time between the optimal strategy and the proposed QLBGFLC technique. This difference in the capture time does not exceed 0.6 seconds in the worst case i.e. there is almost no difference. In addition, it takes only 5.5 minutes in the learning process.

VIII. Conclusion

In this paper we proposed a novel technique to tune the parameters of FLC in which RL is combined with GAs. This combination exploits the advantage of RL in which no expert

TABLE I
Values of GAs parameters

	Phase 2		Unsupervised GA learning
	Stage 1	Stage 2	
Number of iterations	200	200	600
Population size	100	40	80
Number of plays	-	300	300
Crossover probability	0.2	0.2	0.2
Mutation probability	0.1	0.1	0.1
Fitness function	MSE	capture time	reward function
Fitness function objective	minimize	minimize	maximize



(a) The input δ (b) The input $\dot{\delta}$

Fig. 5. MFs for the inputs after tuning

TABLE II
Fuzzy decision table after tuning

δ \ $\dot{\delta}$	N	Z	P
N	-0.2421	-0.6559	-0.5929
Z	-0.6135	0.1584	0.2843
P	-0.5153	0.6205	0.2992

TABLE III
CAPTURE TIME, IN SECONDS, FOR DIFFERENT EVADER INITIAL POSITIONS AND LEARNING TIME, IN MINUTES, FOR THE DIFFERENT TECHNIQUES

	Evader initial position				Learning
	(-9,-3)	(-7,2)	(2,4)	(0,-9)	Time
Optimal strategy	19.6	18.1	5.8	11.6	-
Q(λ)-learning only	29.9	25.0	9.6	13.0	2.5
Unsupervised GA learning	20.4	18.5	6.1	11.9	51.7
Proposed QLBGFLC	19.7	18.7	6.1	11.8	5.5

or training data is needed and the advantage of GAs which is a powerful optimization technique. The proposed technique is applied to a pursuit-evasion game. We assume that the pursuer does not know its control strategy. However it can self-learn its control strategy by interaction with the evader. Computer simulations and the results show that the proposed QLBGFLC technique has the best performance among all other techniques compared with the optimal strategy. When compared with unsupervised GA learning, our proposed technique takes about 10 % the time that the unsupervised GA learning takes in the learning process.

In future work, we will test our proposed technique in more complex multi-robot pursuit-evasion games that have a team of pursuers and a team of evaders.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[2] L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[3] M. Rodríguez, R. Iglesiasa, C. V. Regueirob, J. Correaa, and S. Barroa, "Autonomous and fast robot learning through motivation," *Robotics and Autonomous Systems*, vol. 55, no. 9, pp. 735–740, Sep. 2007.

[4] Y. Duan and X. Hexu, "Fuzzy reinforcement learning and its application in robot navigation," in *International Conference on Machine Learning and Cybernetics*, vol. 2. Guangzhou: IEEE, Aug. 2005, pp. 899–904.

[5] D. A. Gutnisky and B. S. Zanutto, "Learning obstacle avoidance with an operant behavior model," *Artificial Life*, vol. 10, no. 1, pp. 65–81, 2004.

[6] T. Kondo and K. Ito, "A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robots control," *Robotics and Autonomous Systems*, vol. 46, no. 2, pp. 111–124, Feb. 2004.

[7] C. Ye, N. H. C. Yung, and D. Wang, "A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 33, no. 1, pp. 17–27, Jan. 2003.

[8] X. Dai, C. K. Li, and A. B. Rad, "An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 3, pp. 285–293, Sep. 2005.

[9] M. J. Er and C. Deng, "Obstacle avoidance of a mobile robot using hybrid learning approach," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 3, pp. 898–905, Jun. 2005.

[10] H. Xiao, L. Liao, and F. Zhou, "Mobile robot path planning based on Q-ANN," in *IEEE International Conference on Automation and Logistics*, Jinan, China, Aug. 2007, pp. 2650–2654.

[11] Y. Ishiwaka, T. Satob, and Y. Kakazu, "An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning," *Robotics and Autonomous Systems*, vol. 43, no. 4, pp. 245–256, 2003.

[12] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry, "Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 662–669, Oct. 2002.

[13] R. Isaacs, *Differential Games*. John Wiley and Sons, 1965.

[14] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[15] S. H. Lim, T. Furukawa, G. Dissanayake, and H. F. D. Whyte, "A time-optimal control strategy for pursuit-evasion games problems," in *International Conference on Robotics and Automation*, New Orleans, LA, Apr. 2004.

[16] M. Cerrada and J. Aguilar, "Reinforcement learning in system identification," in *Reinforcement Learning Theory and Applications*. Vienna, Austria: I-Tech Education and Publishing, Jan. 2008, ch. 2.

[17] D. B. Aranibar, L. M. G. Gonçalves, and P. J. Alsina, "Learning by experience and by imitation in multi-robot systems," in *Frontiers in Evolutionary Robotics*, H. Iba, Ed. Vienna, Austria: I-Tech Education and Publishing, Apr. 2008, ch. 4.

[18] Y. Yang, Y. Tian, and H. Mei, "Cooperative Q-learning based on blackboard architecture," in *International Conference on Computational Intelligence and Security Workshops*, Harbin, China, Dec. 2007, pp. 224–227.

[19] L. Buşoniu, R. Babuška, and B. de Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 2, pp. 156–172, Mar. 2008.

[20] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge University, 1989.

[21] B. Bhanu, P. Leang, C. Cowden, Y. Lin, and M. Patterson, "Real-time robot learning," in *IEEE International Conference on Robotics and Automation*, vol. 1, Seoul, Korea, May 2001, pp. 491–498.

[22] K. Maček, I. Petrović, and N. Perić, "A reinforcement learning approach to obstacle avoidance of mobile robots," in *7th International Workshop on Advanced Motion Control*, Maribor, Slovenia, 2002, pp. 462–466.

[23] T. M. Marin and T. Duckett, "Fast reinforcement learning for vision-guided mobile robots," in *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, Apr. 2005, pp. 4170–4175.

[24] T. Theodoridis and H. Hu, "The fuzzy sarsa'a'(λ) learning approach applied to a strategic route learning robot behaviour," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2006, pp. 1767–1772.

[25] C. Deng and M. J. Er, "Real-time dynamic fuzzy Q-learning and control of mobile robots," in *5th Asian Control Conference*, vol. 3, Jul. 2004, pp. 1568–1576.

[26] M. J. Er and Y. Zhou, "Dynamic self-generated fuzzy systems for reinforcement learning," in *International Conference on Intelligence For Modelling, Control and Automation. Jointly with International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, 2005, pp. 193–198.

[27] X. W. Yan, Z. D. Deng, and Z. Q. Sun, "Genetic Takagi-Sugeno fuzzy reinforcement learning," in *IEEE International Symposium on Intelligent Control*, Mexico City, Mexico, Sep. 2001, pp. 67–72.

[28] D. Gu and H. Hu, "Accuracy based fuzzy Q-learning for robot behaviours," in *International Conference on Fuzzy Systems*, vol. 3, Budapest, Hungary, Jul. 2004, pp. 1455–1460.

[29] J. H. Holland, "Genetic algorithms and the optimal allocation of trials," *SIAM Journal on Computing*, vol. 2, no. 2, pp. 88–105, 1973.

[30] D. E. Goldberg, *Genetic Algorithms for Search, Optimization, and Machine Learning*. Reading: Addison-Wesley, 1989.

[31] C. Karr, "Genetic algorithms for fuzzy controllers," *AI Expert*, vol. 6, no. 2, pp. 26–33, 1991.

[32] F. Herrera, M. Lozano, and J. L. Verdegay, "Tuning fuzzy logic controllers by genetic algorithms," *International Journal of Approximate Reasoning*, vol. 12, pp. 299–315, 1995.

[33] L. Ming, G. Zailin, and Y. Shuzi, "Mobile robot fuzzy control optimization using genetic algorithm," *Artificial Intelligence in Engineering*, vol. 10, no. 4, pp. 293–298, 1996.

[34] H. Hagras, V. Callaghan, and M. Colley, "Learning and adaptation of an intelligent mobile robot navigator operating in unstructured environment based on a novel online fuzzy-genetic system," *Fuzzy Sets and Systems*, vol. 141, no. 1, pp. 107–160, Jan. 2004.

[35] Y. C. Chiou and L. W. Lan, "Genetic fuzzy logic controller: An iterative evolution algorithm with new encoding method," *Fuzzy Sets and Systems*, vol. 152, no. 3, pp. 617–635, Jun. 2005.

[36] M. Mucientes, D. L. Moreno, A. Bugarin, and S. Barro, "Design of a fuzzy controller in mobile robotics using genetic algorithms," *Applied Soft Computing*, vol. 7, no. 2, pp. 540–546, 2007.

[37] E. Yang and D. Gu, "Multiagent reinforcement learning for multi-robot systems: A survey," Tech. Rep., 2004, [Online]. Available: http://citeseer.ist.psu.edu/yang04multiagent.html.