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

A Novel Hybrid Learning Technique Applied to a Self-Learning Multi-Robot System

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 mainly discusses learning in pursuit-evasion game. In the pursuit-evasion model, one robot pursues another one in a partially known environment. Partially known environment means that each robot knows the instant position of the other robot but at the same time none of them knows its control strategy. Therefore, both robots have to self-learn their control strategies on-line by interaction with each other. A new hybrid learning technique is proposed. The proposed technique combines reinforcement learning with both a fuzzy controller and genetic algorithms in a two-phase structure. The proposed technique is called a Q(λ)-learning based genetic fuzzy controller (QLBGFC). To test the performance of our proposed technique, it is compared with the optimal strategy, the Q(λ)-learning, and the reward-based genetic algorithms. Computer simulations show the usefulness of the proposed technique. In addition, the convergence and the boundedness of the Q-learning algorithm used in the proposed technique are shown.**

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

I. Introduction

Reinforcement learning (RL) is a computational approach to learning through interaction with the environment. The main advantage of RL is that it does not need either a teacher, training data or known model [1], [2]. RL is suitable for intelligent robot control [3], [4] especially in the field of autonomous mobile robots [5]–[10].

Limited studies have applied RL alone to solve environmental problems but its use with other learning algorithms has increased [11]. In [2], 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 [12], 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 [13], 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 the work by [14], Q-learning is used 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 this paper we investigate a new scenario where both the pursuer and the evader simultaneously and independently learn their optimal control strategies. To accomplish this, we propose a novel hybrid learning technique called a Q(λ)-learning based genetic fuzzy controller (QLBGFC). The proposed technique is applied to a pursuit-evasion game in a partially known environment. Partially known environment means that each robot can know the instant position of the other robot but at the same time none of them knows 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 and the evader self-learn their control strategies on-line by interaction with each other. 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 process between the pursuer and 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 [15] in which a pursuer tries to catch an evader in minimum time where the evader tries to escape from the pursuer [16]. The pursuit-evasion game is shown in Fig. 1. Equations of motion for the pursuer and the evader robots are [17], [18]

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

where "i" is "p" for the pursuer and is "e" for the evader, (x_i, y_i) is the position of the robot, V_i is the velocity, θ_i is the orientation, R_i is the turning radius, and u_i is the steering angle

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

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 strategies of the pursuer and the evader are discussed in Section III. 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

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

III. Problem Statement

The problem assigned in this paper is that we assume that the pursuer and the evader do not know their control strategies. The pursuer and the evader are not told which is the correct action to take. We assume that we do not even have a simplistic PD controller strategy. The learning goal is to make both the pursuer and the evader able to self-learn their control strategies. They should do that on-line by interaction with each other.

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 [19]. 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 [20] so, it can be a reference for our results. By this way, we can check the validity of our proposed technique.

IV. Preliminaries

A. Reinforcement Learning

Reinforcement learning (RL) is an interaction between agent and environment as shown in Fig. 2 [21]. 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

$$R_t = \sum_{k=0}^{\tau} \gamma^k r_{t+k+1} \tag{3}$$

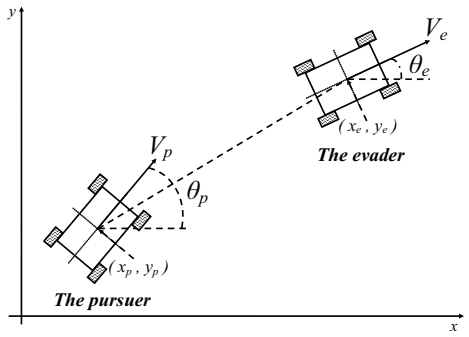


Fig. 1. The pursuer evader dynamics

where r_{t+1} is the immediate reward, 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. The performance of an action, a, taken in a state, s, under policy, π, is evaluated by the action value function, $Q^\pi(s,a)$,

$$\begin{aligned} Q^\pi(s,a) &= E_\pi(R_t | s_t = s, a_t = a) \\ &= E_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} | s_t = s, a_t = a\right) \end{aligned} \tag{4}$$

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, the pursuer wants to catch the evader in minimum time. In other words, the pursuer wants 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

$$D(t) = \sqrt{(x_e(t) - x_p(t))^2 + (y_e(t) - y_p(t))^2} \tag{5}$$

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

$$\Delta D(t) = D(t) - D(t+1) \tag{6}$$

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

$$\Delta D_{max} = V_{rmax} T \tag{7}$$

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

$$r_{t+1} = \begin{cases} \dfrac{\Delta D(t)}{\Delta D_{max}}, & \text{for the pursuer;} \\ -\dfrac{\Delta D(t)}{\Delta D_{max}}, & \text{for the evader.} \end{cases} \tag{8}$$

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

$$a_t = \begin{cases} a^*, & \text{with probability} \quad 1-\varepsilon; \\ \text{random action}, & \text{with probability} \quad \varepsilon. \end{cases} \tag{9}$$

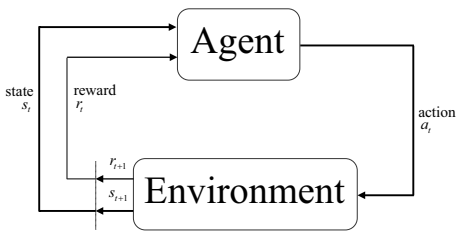


Fig. 2. Agent-environment interaction in RL

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

$$a^* = \arg\max_{a'} Q(s,a') \tag{10}$$

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

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

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 [22]. In addition, it is widely used in learning for most mobile robot applications [23], [24].

Q-learning, which was first introduced by Watkins in his Ph.D [25], 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 [4], [10], [14], [23], [26] or use some types of function approximations such as fuzzy systems [3], [27]–[29] and GAs [30], [31].

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

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \triangle_t \tag{12}$$

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

$$\triangle_t = r_{t+1} + \gamma \max_{\acute{a}} Q(s_{t+1},\acute{a}) - Q(s_t,a_t) \tag{13}$$

Here, (12) is a one-step update rule that 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 [21].

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

$$e_t(s,a) = \begin{cases} 1, & \text{if } s = s_t \text{ and } a = a_t; \\ 0, & \text{if } s = s_t \text{ and } a \neq a_t; \\ \lambda\gamma e_{t-1}(s,a), & \text{if } s \neq s_t. \end{cases} \tag{14}$$

where $e_o = 0$, and λ is the trace-decay parameter, $(0 \leq \lambda \leq 1)$.

Eligibility traces are used to speed up the learning process and hence to make it suitable for on-line applications. Now we will modify (12) to be

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha e_t \triangle_t \tag{15}$$

B. Fuzzy Logic Controller

A block diagram of a fuzzy logic controller (FLC) system is shown in Fig. 3. We will construct two FLCs, one for the pursuer and the other for the evader. Each FLC has two inputs, the error in angle, δ_i, and its derivative, $\dot{\delta}_i$, and the output is the steering angle, u_i, where

$$\delta_i = \tan^{-1}\left(\frac{y_e - y_p}{x_e - x_p}\right) - \theta_i \tag{16}$$

where "i" is "p" for the pursuer and is "e" for the evader. For the inputs of the FLC, we use the gaussian MF described by

$$\mu(x) = \exp\left(-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2\right) \tag{17}$$

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

$$R_l : IF\ x_1\ is\ A_1^l\ AND\ x_2\ is\ A_2^l\ AND\ ...\ AND\ x_N\ is\ A_N^l \quad THEN\ f_l = K_l \tag{18}$$

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

$$u_i = \frac{\sum_{l=1}^{L}((\prod_{n=1}^{N}\mu^{A_n^l}(x_n))K_l}{\sum_{l=1}^{L}(\prod_{n=1}^{N}\mu^{A_n^l}(x_n))} \tag{19}$$

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 [32], [33]. 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 [34]–[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 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.

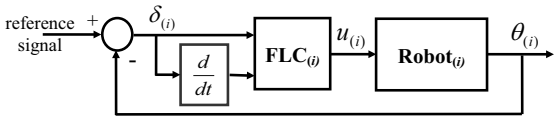


Fig. 3. Block diagram of a FLC system

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.

V. The proposed Q(λ)-learning Based Genetic Fuzzy Controller (QLBGFC)

We propose a novel hybrid learning technique that combines Q(λ)-learning with FLC and GAs. The proposed technique is called a Q(λ)-learning based genetic fuzzy controller (QLBGFC). The proposed technique is applied to a pursuit-evasion game in a partially known environment. Partially known environment means that each robot can know the instant position of the other robot but at the same time none of them knows 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/evader. The state, s, consists of the error, δ_i, defined by (16) and its derivative, $\dot{\delta}_i$, and the action, a, is the steering angle, u_i. The states and their corresponding greedy actions are then stored in a lookup table. The reward function used is defined by (8).

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 (20) is used as the fitness function. GAs in this stage are used as supervised learning. 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 process between the pursuer and the evader. In this stage, the capture time, which the pursuer wants to minimize and the evader wants to maximize, is used as the fitness function. In other words, at this stage we do the final search for the Nash equilibrium. Here, GAs are used as reward-based learning with a priori knowledge obtained from stage 1.

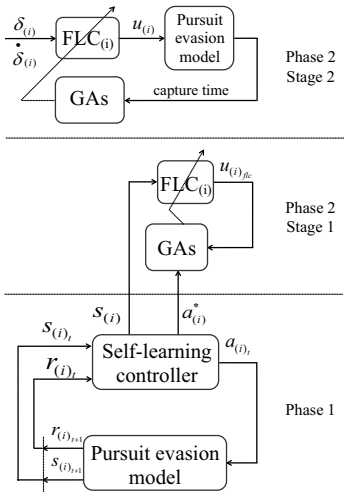


Fig. 4. The proposed QLBGFC technique

Q-learning alone has the limitation in that it is too hard to visit all the state-action pairs [14]. 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 are described in Algorithm 1 and Algorithm 2, respectively. Note that the procedures in the two algorithms are used for both the pursuer and the evader. In these algorithms "i" is "p" for the pursuer and is "e" for the evader.

Algorithm 1 (Phase 1: Q(λ)-learning)

1) Discretize the state space, $S^{(i)}$, and the action space, $A^{(i)}$.
2) Initialize $Q_{(i)}(s^{(i)},a^{(i)}) = 0$.
3) Initialize $e^{(i)}(s^{(i)},a^{(i)}) = 0$.
4) **For** each episode
   a) Initialize $(x_p,y_p) = (0,0)$.
   b) Initialize (x_e,y_e) randomly.
   c) Calculate $s_t^{(i)} = (\delta_i,\dot{\delta}_i)$ according to (16).
   d) Select $a^{(i)}$ randomly.
   e) **For** each play
      i) Receive r_{t+1}.
      ii) Observe $s_{t+1}^{(i)}$.
      iii) Select $a_{t+1}^{(i)}$ using the $\varepsilon-$greedy method.
      iv) Calculate $e_{t+1}^{(i)}$.
      v) Update $Q_{(i)}(s_t^{(i)},a_t^{(i)})$.
   f) **End**
5) **End**
6) $Q_{(i)} \leftarrow Q^*_{(i)}$.
7) Assign a greedy action, $a^{(i)*}$, to each state, $s^{(i)}$.
8) Store the state-action pairs in a lookup table.

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_i, and its derivative, $\dot{\delta}_i$, where $\delta_i \in [-1,1]$ for both the pursuer and the evader. The action is the steering angle, u_i, where $u_p \in [-0.5,0.5]$ and $u_e \in [-1,1]$. These values are relatively coarsely discretized such that the state and the action spaces are not prohibitively large.

Algorithm 2 (Phase 2: GAs learning)

- **Stage 1**
  1) Get the state-action pairs from the lookup table.
  2) Initialize a population, $P^{(i)}$, randomly.
  3) **For** each iteration
     a) **For** each set of chromosomes in the population
        i) Construct two FLCs from the population chromosomes (one for the pursuer and the other for the evader).
        ii) **For** each state, $s^{(i)}$,
        – Calculate the FLC output, $u_{flc}^{(i)}$.
        iii) **End**
        iv) Calculate fitness function using MSE as

$$MSE^{(i)} = \frac{1}{2L}\sum_{l=1}^{L}(u_d^{(i)l} - u_{flc}^{(i)l})^2 \qquad (20)$$

        where $u_d^{(i)l}$ is the l^{th} greedy action, $a^{(i)*}$.
     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, $P^{(i)}$, from the tuned FLCs obtained from stage 1.
  2) **For** each iteration
     a) Initialize (x_e, y_e) randomly.
     b) **For** each set of chromosomes in the population
        i) Initialize $(x_p, y_p) = (0,0)$.
        ii) Compute the state $(\delta_i, \dot{\delta}_i)$ according to (16).
        iii) Construct two FLCs from the population chromosomes (one for the pursuer and the other for the evader).
        iv) **For** each play
           A) Calculate the FLC output, u_i.
           B) Observe the next state.
        v) **End**
        vi) Observe the fitness function which is the capture time.
     c) **End**
     d) Sort the entire chromosomes of the population according to their fitness values (ascending for the pursuer and descending for the evader).
     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**

A. Comparison of reward-based GA Learning to the Proposed Technique

For comparative purposes, we will also implement a general reward-based GA learning method. The reward-based GA learning method will be initialized with randomly chosen FLC parameters. The GAs adjust the parameters using the closing distance reward given by (8) as the fitness function which both the pursuer and the evader want to maximize.

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 (20) 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 for the pursuer and a maximizing capture time for the evader.

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 $R_p = 1$ m. The velocity of the pursuer $V_p = 1$ 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 $R_e = 1$ m. The velocity of the evader $V_e = 0.5$ 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 spaces for the pursuer, we discretize the ranges of the inputs, δ_p and $\dot{\delta}_p$, by 0.2. The ranges of δ_p and $\dot{\delta}_p$ are set to be from -1 to 1 so the discretized values for δ_p and $\dot{\delta}_p$ will be $(-1.0, -0.8, -0.6, \ldots, 0.8, 1.0)$. There are 11 discretized values for δ_p and 11 discretized values for $\dot{\delta}_p$. These values are combined to form $11 \times 11 = 121$ states for the pursuer. The state space of the evader is identical to that of the pursuer since the ranges of δ_e and $\dot{\delta}_e$ are set also to be from -1 to 1. To build the action space of the pursuer, 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 of the pursuer will be 121×11. To build the action space of the evader, we discretize the range of the action, u_e, by 0.1. The range of the action is set to be from -1 to 1 so the discretized values in the action space are $(-1, -0.9, -0.8, \ldots, 0.9, 1)$. There are 21 actions and the dimension of the Q-table of the evader will be 121×21. We choose the number of games or episodes to be 1000, the number of plays or steps in each episode is 6000, the discount factor $\gamma = 0.8$, and the trace-decay parameter $\lambda = 0.3$. We make the learning rate, α, decrease with each episode such that

$$\alpha = \frac{1}{i} \qquad (21)$$

and we also make ε decrease with each episode such that

$$\varepsilon = 0.2 - 0.2\left(\frac{i}{\text{maximum \# of episodes}}\right) \tag{22}$$

where i is the current episode.

In phase 1, the position of the evader, (x_e, y_e), and the actions taken by both the pursuer and the evader are chosen randomly at the beginning of each episode to cover most of the states and the actions. At the end of phase 1, the states of the pursuer and the evader and their corresponding greedy actions, a^*, are stored in two lookup tables to be used in the learning process of phase 2.

In phase 2, GAs are used to tune the parameters of two FLCs (one for the pursuer and the other for the evader). Each FLC to be learned has 2 inputs, $(\delta_p, \dot{\delta}_p)$ for the pursuer and $(\delta_e, \dot{\delta}_e)$ for the evader. Each input variable has 3 MFs for a total of 6 MFs. We use gaussian MFs which have 2 coefficients to be tuned. The total number of premise parameters to be tuned per each FLC is $6 \times 2 = 12$ parameters for a total of 24 premise parameters for the two FLCs. We have $3 \times 3 = 9$ rules. We use the zero-order TSK-FIS defined in (18) and modify it to be

$$R_l : IF\ \delta\ is\ A_1^l\ AND\ \dot{\delta}\ is\ A_2^l\ THEN\ f_l = K_l \tag{23}$$

where $l = 1, 2, ..., 9$. The crisp output of the system, which is the steering angle , u_i, is calculated using (19) as follows

$$u_i = \frac{\sum_{l=1}^{9}((\prod_{n=1}^{2}\mu^{A_n^l}(x_n))K_l}{\sum_{l=1}^{9}(\prod_{n=1}^{2}\mu^{A_n^l}(x_n))} \tag{24}$$

The consequent parameters to be tuned are 9 for each FLC with a total number of 18 consequent parameters to be tuned for the two FLC. The total number of FLC parameters to be tuned is $24 + 18 = 42$ parameters for the inputs and the output of the two FLCs.

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 (20). 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 those of phase 1 therefore we need to perform stage 2. In stage 2, GAs are used as reward-based 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 for the pursuer and to maximize for the evader, 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, the Q(λ)-learning only, and the reward-based GA learning where the reward function is given by (8). The values of the GAs parameters used in stage 1 of phase 2, stage 2 of phase 2, and reward-based 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 the reward-based 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.

We will test the convergence of the Q-learning algorithms used in phase 1. In addition we will test the boundedness of the Q values. The importance of these tests is that the learning in phase 2 with its two stages depends on the results obtained from phase 1. Therefore we want to be sure that the Q-learning algorithms for both the pursuer and the evader did well in phase 1. To test the convergence of the Q-learning, we calculate L^1 norm, $\|Q\|_1$, which is defined as

$$\|Q\|_1 = \sum_{s,a}|Q(s,a)| \qquad \text{for all } s,a \tag{25}$$

VII. Results

To execute our computer simulations, we use a core 2 duo computer with a 2.0 GHz clock frequency and 4.0 Gigabytes of RAM. We do the computer simulations in MATLAB program. In phase 1, we run the simulation of the Q(λ)-learning which takes on average 2.88 minutes. At the end of phase 1, we store the states, $s_p = (\delta_p, \dot{\delta}_p)$ and $s_e = (\delta_e, \dot{\delta}_e)$, and their corresponding greedy actions, a_p^* and a_e^*, in two look-up tables 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 process between the pursuer and 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 12.91 minutes with a total time of 15.95 minutes for the learning process in our proposed QLBGFC technique.

Fig. 5 and Fig. 6 show the tuned MFs for the two inputs, δ_p and $\dot{\delta}_p$ for the pursuer and δ_e and $\dot{\delta}_e$ for the evader, respectively. Table II and Table III show the fuzzy decision tables after the learning process for the pursuer and the evader, respectively. Fig. 7 shows the paths of the pursuer and the evader using the Q(λ)$-$learning only. Fig. 8 shows the paths of the pursuer and the evader using the optimal strategy, the reward-based GA learning, and our proposed QLBGFC. The pursuer starts from position (0,0) and the

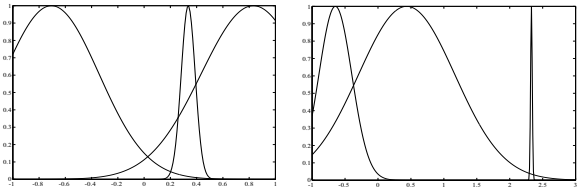
TABLE I
VALUES OF GAS PARAMETERS

	Phase 2		Reward-based GA learning
	Stage 1	*Stage 2*	
Number of iterations	300	200	500
Population size	200	100	200
Number of plays	-	600	600
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 for pursuer & maximize for evader	maximize
Learning time (minutes)	0.16	12.91	29.55

evader starts from position (6,7). From Fig. 7 we can see that the Q(λ)-learning only is not enough to get the desired performance in comparison with the optimal strategy and the other techniques. From Fig. 8 we can see that the reward-based 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 29.55 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 only a slight difference in the paths between the optimal strategy and the proposed QLBGFC technique. In addition, it takes 15.95 minutes in the learning process which is about 54 % of the time taken by the reward-based GA learning. Fig. 9 shows the convergence of the Q-learning for both the pursuer and the evader. We can see that the Q-learning of the pursuer converges to a value of 24 and the Q-learning of the evader converges to a value of 15. Fig. 10 shows the maximum and the minimum Q values for the pursuer and the evader. Notice that the Q values of the pursuer are in the range from -0.6 to 0.45 and the Q values of the evader are in the range from -0.28 to 0.37 and therefore they are bounded.

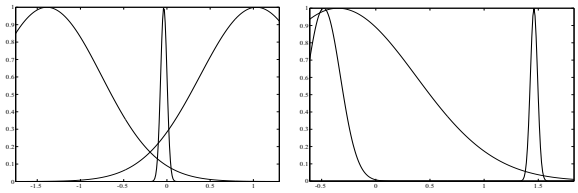
VIII. Conclusion

In this paper we propose a novel hybrid learning technique in which RL is combined with FLC and GAs. This combination exploits the advantage of RL in which no expert or training data is needed, the advantage of FLC as a function approximation, 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 and the evader do not know their control strategies. However they can self-learn their control strategies by interaction with each other.



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

Fig. 5. Tuned MFs for the inputs of the pursuer



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

Fig. 6. Tuned MFs for the inputs of the evader

TABLE II
Fuzzy decision table after tuning for the pursuer

δ \ $\dot{\delta}$	N	Z	P
N	-0.6864	-1.0896	0.0105
Z	-0.5703	0.2660	-0.6728
P	0.3509	1.4113	-0.1126

TABLE III
Fuzzy decision table after tuning for the evader

δ \ $\dot{\delta}$	N	Z	P
N	-0.1045	-1.6240	0.2204
Z	-0.8479	-0.4161	0.2949
P	-0.3151	1.4953	1.1489

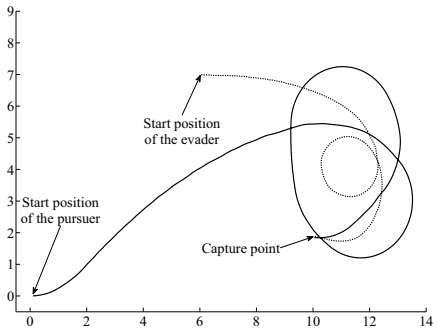


Fig. 7. The paths of the pursuer and the evader using the Q-learning only

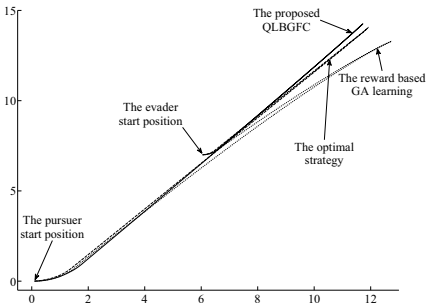


Fig. 8. The paths of the pursuer and the evader using the different techniques

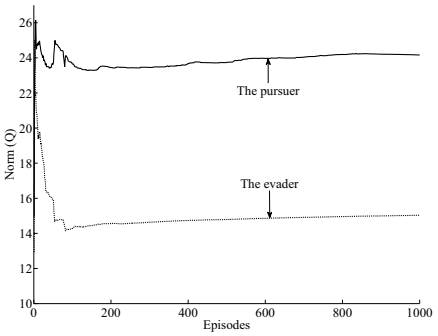


Fig. 9. The convergence of the Q-learning algorithms of both the pursuer and the evader

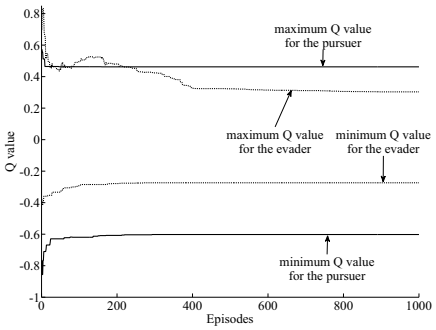


Fig. 10. The upper and lower boundaries of the Q values for both the pursuer and the evader

Computer simulations and the results show that the proposed QLBGFC technique has the best performance among all other techniques compared with the optimal strategy.

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] W. M. V. Buijtenen, G. Schram, R. Babuška, and H. Verbruggen, "Adaptive fuzzy control of satellite attitude by reinforcement learning," *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 2, pp. 185–194, May 1998.

[2] 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.

[3] 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.

[4] 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.

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

[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] 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.

[8] 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.

[9] W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *IEEE International Conference on Robotics and Automation*, vol. 4, Washington, DC, May 2002, pp. 3404–3410.

[10] 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.

[11] S. H. Chen, A. J. Jakeman, and J. P. Norton, "Artificial intelligence techniques: An introduction to their use for modelling environmental systems," *Mathematics and Computers in Simulation*, vol. 78, no. 2–3, pp. 379–400, 2008.

[12] 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.

[13] 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.

[14] 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.

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

[16] J. Ge, L. Tang, J. Reimann, and G. Vachtsevanos, "Hierarchical decomposition approach for pursuit-evasion differential game with multiple players," in *2006 IEEE Aerospace Conference*, Big Sky, MT, Mar. 2006.

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

[18] 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.

[19] 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.

[20] J. Baltes and Y. J. Park, "Comparison of several machine learning techniques in pursuit-evasion games," in *RoboCup-01: Robot Soccer World Cup V*. New York: Springer, 2002.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

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

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

[35] 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.

[36] T. L. Seng, M. Khalid, and R. Yusof, "Tuning of a neuro-fuzzy controller by genetic algorithms with an application to a coupled-tank liquid-level control system," *International Journal of Engineering Applications on Artificial Intelligence*, vol. 11, no. 4, pp. 517–529, Sep. 1998.

[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.