Robotics and Autonomous Systems 59 (2011) 22–33

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Self-learning fuzzy logic controllers for pursuit–evasion differential games

Sameh F. Desouky *, Howard M. Schwartz

Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, ON, Canada

ARTICLE INFO

Article history:
Received 17 February 2010
Accepted 13 September 2010
Available online 1 October 2010

Keywords:
Differential game
Function approximation
Fuzzy control
Genetic algorithms
Q(λ)-learning
Reinforcement learning

ABSTRACT

This paper addresses the problem of tuning the input and the output parameters of a fuzzy logic controller. The system learns autonomously without supervision or a priori training data. Two novel techniques are proposed. The first technique combines Q(λ)-learning with function approximation (fuzzy inference system) to tune the parameters of a fuzzy logic controller operating in continuous state and action spaces. The second technique combines Q(λ)-learning with genetic algorithms to tune the parameters of a fuzzy logic controller in the discrete state and action spaces. The proposed techniques are applied to different pursuit–evasion differential games. The proposed techniques are compared with the classical control strategy, Q(λ)-learning only, reward-based genetic algorithms learning, and with the technique proposed by Dai et al. (2005) [19] in which a neural network is used as a function approximation for Q-learning. Computer simulations show the usefulness of the proposed techniques.

1. Introduction

Fuzzy logic controllers (FLCs) are currently being used in engineering applications [1,2] especially for plants that are complex and ill-defined [3,4] and plants with high uncertainty in the knowledge about its environment such as autonomous mobile robotic systems [5,6]. However, FLC has a drawback of finding its knowledge base which is based on a tedious and unreliable trial and error process. To overcome this drawback one can use supervised learning [7–11] that needs a teacher or input/output training data. However, in many practical cases the model is totally or partially unknown and it is difficult or expensive and in some cases impossible to get training data. In such cases it is preferable to use reinforcement learning (RL).

RL is a computational approach to learning through interaction with the environment [12,13]. The main advantage of RL is that it does not need either a teacher or a known model. RL is suitable for intelligent robot control especially in the field of autonomous mobile robots [14–18].

1.1. Related work

Limited studies have applied RL alone to solve environmental problems but its use with other learning algorithms has increased. In [19], a RL approach is used to tune the parameters of a FLC. This approach is applied to a single case of one robot following another along a straight line. In [15,20], 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 phase 1, supervised learning is used to tune the parameters of a FLC then in phase 2, RL is employed so that the system can re-adapt to a new environment. The limitation in their approach is that if the training data are hard or expensive to obtain then supervised learning cannot be applied. In [21], the authors overcame this limitation by using Q-learning as an expert to obtain training data. Then the training data are used to tune the weights of an artificial neural network controller applied to a mobile robot path planning problem.

In [22], 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 they received from a central computer system. In [23], 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 [24], RL is used to tune the output parameters of a FLC in a pursuit–evasion game.

A number of articles used the fuzzy inference system (FIS) as a function approximation with Q-learning [25–28] however these works have the following disadvantages: (i) the action space is considered to be discrete and (ii) only the output parameters of the FIS are tuned.

1.2. Paper motivation

The problem assigned in this paper is that we assume that the pursuer/evader does not know its control strategy. It is not told which actions to take so as to be able to optimize its control

* Corresponding author. Tel.: +1 613 520 2600x5725.
E-mail addresses: sameh@sce.carleton.ca, samehfarahat@gmail.com (S.F. Desouky), schwartz@sce.carleton.ca (H.M. Schwartz).

0921-8890/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2010.09.006

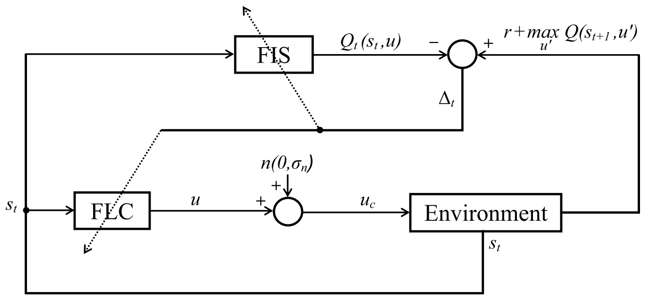


Fig. 1. The proposed QLFIS technique.

strategy. We assume that we do not even have a simplistic PD controller strategy. The learning goal is to make the pursuer/evader able to self-learn its control strategy. It should do that on-line by interaction with the evader/pursuer.

From several learning techniques we choose 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. It is the most appropriate learning technique for our problem.

However, using RL alone, in most cases, has the limitation in that it is too hard to visit all the state–action pairs. We try to cover most of the state–action space but we cannot cover all the space. In addition, there are hidden states that are not taken into consideration due to the discretization process. Hence RL alone cannot find the optimal strategy.

The proposed Q(λ)-learning based genetic fuzzy controller (QLBGFC) and the proposed Q(λ)-learning fuzzy inference system (QLFIS) are two novel techniques used to solve the limitation in RL. The limitation is that the RL method is designed only for discrete state–action spaces. Since we want to use RL in the robotics domain which is a continuous domain, then we need to use some type of function approximation such as FIS to generalize the discrete state–action space into a continuous state–action space. Therefore, from the RL point of view, a FIS is used as a function approximation to compensate for the limitation in RL. And from the FIS point of view, RL is used to tune the input and/or the output parameters of the fuzzy system especially if the model is partially or completely unknown. Also, in some cases it is hard or expensive to get a priori training data or a teacher to learn from. In this case, the FIS is used as an adaptive controller whose parameters are tuned on-line by RL. Therefore, combining RL and FIS has two objectives; to compensate the limitation in RL and to tune the parameters of the FLC.

In this paper, we design a self-learning FLC using the proposed QLFIS and the proposed QLBGFC. The proposed QLBGFC is used when the state and the action spaces can be discretized in such a way that make the resulting state and action spaces have acceptable dimensions. This can be done, as we will see in our case, if the state and the action values are bounded. If not then the proposed QLFIS will be suitable. In this work and for the comparatively purpose, we will use both of the proposed techniques.

The learning process in the proposed QLFIS is performed simultaneously as shown in Fig. 1. The proposed QLFIS is used *directly* with the continuous state and action spaces. The FIS is used as a function approximation to estimate the optimal action-value function, $Q^*(s,a)$, in the continuous state and action spaces while the Q(λ)-learning is used to tune the input and the output parameters of both the FIS and the FLC.

In the proposed QLBGFC, the learning process is performed sequentially as shown in Fig. 2. The proposed QLBGFC can be considered as *indirect* method of using function approximation. First, in phase 1, the state and the action spaces are discretized and Q(λ)-learning is used to obtain an estimate of the desired training data set, (s,a^*). Then this training data set is used by genetic algorithms (GAs) in phase 2 stage 1 to tune the input and the output parameters of the FLC which are used at the same time to generalize the discrete state and action values over the continuous state and action spaces. Finally in phase 2 stage 2, the FLC is further tuned during the interaction between the pursuer and the evader.

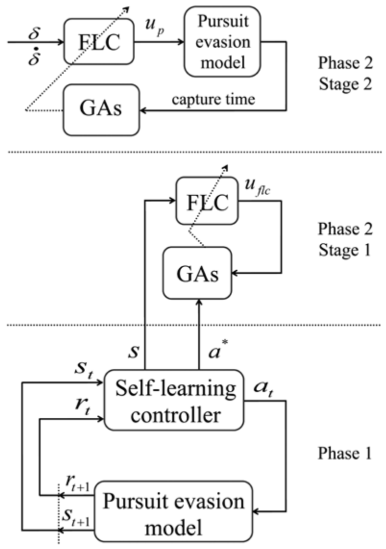


Fig. 2. The proposed QLBGFC technique.

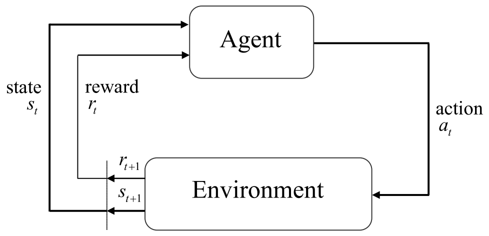


Fig. 3. Agent–environment interaction in RL.

The proposed techniques are applied to two pursuit–evasion games. In the first game, we assume that the pursuer does not know its control strategy whereas in the second game, we increase the complexity of the system by assuming that both the pursuer and the evader do not know their control strategies or the other's control strategy.

The rest of this paper is organized as follows: some basic terminologies for RL, FIS and GAs are reviewed in Section 2, Section 3 and Section 4, respectively. In Section 5, the pursuit–evasion game is described. The proposed QLFIS and the proposed QLBGFC techniques are described in Section 6 and Section 7, respectively. Section 8 presents the computer simulation and the results are discussed in Section 9. Finally, conclusion and future work are discussed in Section 10.

2. Reinforcement learning

Agent–environment interaction in RL is shown in Fig. 3 [12]. 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

$$R_t = \sum_{k=0}^{\tau} \gamma^k r_{t+k+1} \qquad (1)$$

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 and τ is the end of an episode. If τ is finite then the model is called a finite-horizon model [13]. If $\tau \to \infty$ then the model is called an infinite-horizon discounted model and in this case $\gamma < 1$ to avoid infinite total rewards.

The performance of an action, a, taken in a state, s, under policy, π, is evaluated by the action value function, $Q^{\pi}(s,a)$,

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}(R_t | s_t = s, a_t = a) \\ &= E_{\pi}\left(\sum_{k=0}^{\tau} \gamma^k r_{k+t+1} | s_t = s, a_t = a\right) \end{aligned} \tag{2}$$

where $E_{\pi}(\cdot)$ is the expected value under policy, π. The way to choose an action is a trade-off between exploitation and exploration. The ε-greedy action selection method is a common way of choosing the actions. This method can be stated as

$$a_t = \begin{cases} a^*, & \text{with probability } 1-\varepsilon; \\ \text{random action}, & \text{with probability } \varepsilon \end{cases} \tag{3}$$

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

$$a^* = \arg\max_{a'} Q(s, a'). \tag{4}$$

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

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{5}$$

Many algorithms have been proposed for estimating the optimal value functions. The most widely used and well-known control algorithm is Q-learning [29].

Q-learning, which was first introduced by Watkins in his Ph.D. [30], is an off-policy algorithm. Therefore, 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 what is 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 [31–34,21,35] or use some type of function approximation such as FISs [36,26,37], neural networks (NNs) [12,38,19], or use some type of optimization technique such as GAs [39,40]. A one-step update rule for Q-learning is defined as

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \Delta_t \tag{6}$$

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

$$\Delta_t = r_{t+1} + \gamma \max_{\acute{a}} Q_t(s_{t+1}, \acute{a}) - Q_t(s_t, a_t). \tag{7}$$

Eq. (6) 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 [12].

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 [41] defined as: $\forall\ s, a$,

$$e_t(s,a) = \begin{cases} 1, & \text{if } s = s_t \text{ and } a = a_t; \\ 0, & \text{if } s = s_t \text{ and } a \neq a_t; \\ \lambda \gamma e_{t-1}(s,a), & \text{if } s \neq s_t \end{cases} \tag{8}$$

where $e_0 = 0$, 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 (6) to be

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha e_t \Delta_t. \tag{9}$$

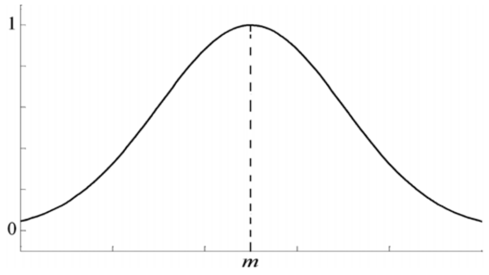


Fig. 4. Gaussian MF.

For the continuous state and action spaces, the eligibility trace is defined as

$$e_t = \gamma \lambda e_{t-1} + \frac{\partial Q_t(s_t, a_t)}{\partial \phi} \tag{10}$$

where ϕ is the parameter to be tuned.

3. Fuzzy inference system

The most widely used FIS models are Mamdani [42] and Takagi–Sugeno–Kang (TSK) [43]. In this work we are interesting in using the TSK model. A first-order TSK means that the output is a linear function of its inputs while a zero-order TSK means that the output is a constant function. For simplicity purpose we use a zero-order TSK model. A Rule used in zero-order TSK model for N inputs has the form

$$R_l:\ \text{IF } x_1 \text{ is } A_1^l \text{ AND } \ldots \text{ AND } x_N \text{ is } A_N^l \qquad \text{THEN } f_l = K_l \tag{11}$$

where A_i^l is fuzzy set of the ith input variable, x_i, in rule R_l, $l = 1, 2, \ldots, L$, K^l is the consequent parameter of the output, f_l, in rule R_l.

The fuzzy output can be defuzzified into a crisp output using one of the defuzzification techniques. Here, the weighted average method is used and is defined as follows

$$f(\bar{x}) = \frac{\sum_{l=1}^{L}\left(\prod_{i=1}^{N} \mu^{A_i^l}(x_i)\right) K_l}{\sum_{l=1}^{L}\left(\prod_{i=1}^{N} \mu^{A_i^l}(x_i)\right)} \tag{12}$$

where $\mu^{A_i^l}(x_i)$ is the membership value for the fuzzy set A_i^l of the input x_i in rule R_l. Due to its simple formulas and computational efficiency, the Gaussian membership function (MF) has been used extensively especially in real-time implementation, and control. The Gaussian MF depicted in Fig. 4 is defined as

$$\mu^{A_i^l}(x_i) = \exp\left(-\left(\frac{x_i - m_i^l}{\sigma_i^l}\right)^2\right) \tag{13}$$

where σ and m are the standard deviation and the mean, respectively.

The structure of the FIS used in this work is shown in Fig. 5. Without loss of generality, we assume that the FIS model has 2 inputs, x_1 and x_2, and one output, f, and each input has 3 Gaussian MFs. The structure has two types of nodes. The first type is an adaptive node (a squared shape) whose output need to be adapted (tuned) and the second type is a fixed node (a circled shape) whose output is a known function of its inputs.

The structure has 5 layers. In layer 1, all nodes are adaptive. This layer has 6 outputs denoted by O^1, The output of each node in layer 1 is the membership value of its input defined by (13). In layer 2, all nodes are fixed. The AND operation (multiplication) between the inputs of each rule is calculated in this layer. This layer has 9 outputs denoted by O_l^2, $l = 1, 2, \ldots, 9$. The output of each node in

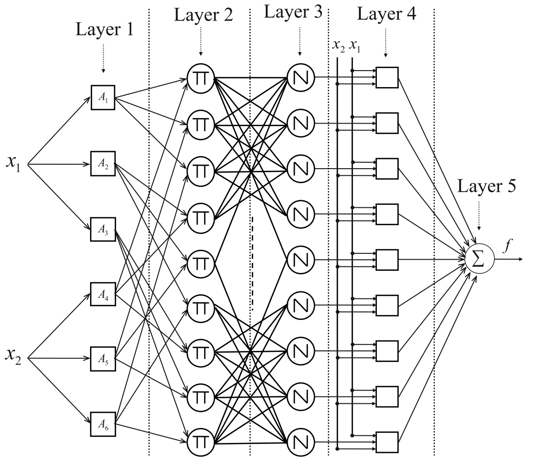


Fig. 5. Structure of a FIS model.

layer 2, known as the firing strength of the rule, ω_l, is calculated as follows

$$O_l^2 = \omega_l = \prod_{i=1}^{2} \mu^{A_i^l}(x_i). \tag{14}$$

In layer 3, all nodes are fixed. This layer has 9 outputs denoted by O_l^3. The output of each node in layer 3 is the normalized firing strength, $\overline{\omega}_l$, which is calculated as follows

$$O_l^3 = \overline{\omega}_l = \frac{O_l^2}{\sum_{l=1}^{9} O_l^2} = \frac{\omega_l}{\sum_{l=1}^{9} \omega_l}. \tag{15}$$

In layer 4, all nodes are adaptive. The defuzzification process that uses the weighted average method, defined by (12), is performed in this layer and the next layer. Layer 4 has 9 outputs denoted by O_l^4. The output of each node is

$$O_l^4 = O_l^3 K_l = \overline{\omega}_l K_l. \tag{16}$$

Layer 5 is the output layer and has only one fixed node whose output, f, is the sum of all its inputs as follows

$$O^5 = f = \sum_{l=1}^{9} O_l^4 = \sum_{l=1}^{9} \overline{\omega}_l K_l \tag{17}$$

which is the same as (12).

4. Genetic algorithms

Genetic algorithms (GAs) are search and optimization techniques that are based on a formalization of natural genetics [44,45]. 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 [46,8,47,5,48,6].

A GA searches 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 real 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 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.

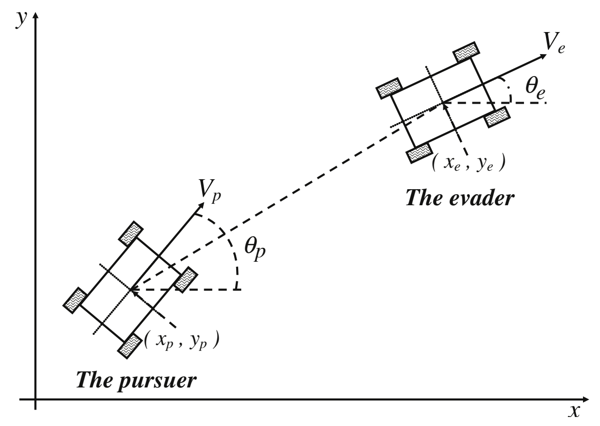


Fig. 6. The pursuit–evasion model.

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 a termination condition is met. There are different conditions to terminate the genetic process such as: (i) the maximum iteration is reached, (ii) a fitness threshold is achieved, (iii) a maximum time is reached and (iv) a combination of the previous conditions. In our work we cannot use the maximum time condition since we use the learning time as a parameter in our comparison. We use the maximum iteration condition which is actually based on a threshold condition. The number of GA iterations is determined based on simulations. We tried different numbers of iterations and we determined the number of iterations for which further training would not have any significant improvement in performance. The coding process in the proposed QLBGFC technique using GAs will be described in detail in Section 7.

5. Pursuit-evasion differential game

The pursuit–evasion differential game is one application of differential games [49] in which a pursuer tries to catch an evader in the minimum time where the evader tries to escape from the pursuer. The pursuit–evasion game is shown in Fig. 6. Equations of motion for the pursuer and the evader robots are [50,51]

$$\begin{aligned} \dot{x}_i &= V_i \cos(\theta_i) \\ \dot{y}_i &= V_i \sin(\theta_i) \\ \dot{\theta}_i &= \frac{V_i}{L_i} \tan(u_i) \end{aligned} \tag{18}$$

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, L_i is the distance between the front and rear axle, and u_i is the steering angle where $u_i \in [-u_{i_{\max}}, u_{i_{\max}}]$. The minimum turning radius is calculated as

$$Rd_{i_{\min}} = \frac{L_i}{\tan(u_{i_{\max}})}. \tag{19}$$

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 that we compare our results with are defined by

$$u_i = \begin{cases} -u_{i_{\max}} & : \quad \delta_i < -u_{i_{\max}} \\ \delta_i & : \quad -u_{i_{\max}} \le \delta_i \le u_{i_{\max}} \\ u_{i_{\max}} & : \quad \delta_i > u_{i_{\max}} \end{cases} \tag{20}$$

where

$$\delta_i = \tan^{-1}\left(\frac{y_e - y_p}{x_e - x_p}\right) - \theta_i \tag{21}$$

where "i" is "p" for the pursuer and is "e" for the evader. 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

$$\ell = \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2}. \tag{22}$$

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 techniques.

6. The proposed Q(λ)-learning fuzzy inference system

A FIS is used as a function approximation for Q(λ)-learning to generalize the discrete state and action spaces into continuous state and action spaces and at the same time Q(λ)-learning is used to tune the parameters of the FIS and the FLC. The structure of the proposed QLFIS is shown in Fig. 1 which is a modified version of the proposed techniques used in [19,24].

The difference between the proposed QLFIS and that proposed in [24] is that in [24], the authors used FIS to approximate the value function, $V(s)$, but the proposed QLFIS is used to approximate the action-value function, $Q(s, a)$. In addition in [24], the authors tune only the output parameters of the FIS and the FLC while in this work the input and the output parameters of the FIS and the FLC are tuned. The reason for choosing Q-learning in our work is that it outperforms the actor-critic learning [52]. The main advantage of Q-learning over actor-critic learning is exploration insensitivity since Q-learning is an off-policy algorithm (see Section 2) whereas actor-critic learning is an on-policy algorithm.

The difference between the proposed QLFIS and that proposed in [19] is that in [19], the authors used neural networks (NNs) as a function approximation but here we use the FIS as a function approximation. There are some advantages of using FIS rather than NNs such that: (i) linguistic fuzzy rules can be obtained from human experts [53] and (ii) the ability to represent fuzzy and uncertain knowledge [54]. In addition, our results show that the proposed QLFIS outperforms the technique proposed in [19] in both the learning time and the performance.

Now we will derive the adaptation laws for the input and the output parameters of the FIS and the FLC. The adaptation laws will be derived only once and are applied for both the FIS and the FLC. Our objective is to minimize the TD-error, Δ_t, and by using the mean square error (MSE) we can formulate the error as

$$E = \frac{1}{2}\Delta_t^2. \tag{23}$$

We use the gradient descent approach and according to the steepest descent algorithm, we make a change along the $-ve$ gradient to minimize the error so,

$$\phi(t+1) = \phi(t) - \eta\frac{\partial E}{\partial \phi} \tag{24}$$

where η is the learning rate and ϕ is the parameter vector of the FIS and the FLC where $\phi = [\sigma, m, K]$. The parameter vector, ϕ, is to be tuned. From (23) we get

$$\begin{aligned}\frac{\partial E}{\partial \phi} &= \Delta_t\frac{\partial \Delta_t}{\partial \phi}\\ &= \Delta_t\frac{\partial \Delta_t}{\partial Q_t(s_t,u_t)}\frac{\partial Q_t(s_t,u_t)}{\partial \phi}\end{aligned} \tag{25}$$

Then from (7),

$$\frac{\partial E}{\partial \phi} = -\Delta_t\frac{\partial Q_t(s_t,u_t)}{\partial \phi}. \tag{26}$$

Substituting in (24), we get

$$\phi(t+1) = \phi(t) + \eta\Delta_t\frac{\partial Q_t(s_t,u_t)}{\partial \phi}. \tag{27}$$

We can obtain $\partial Q_t(s_t,u_t)/\partial\phi$ for the output parameter, K_l, from (17), where f is $Q_t(s_t,u_t)$ for the FIS and f is u for the FLC, as follows

$$\frac{\partial Q_t(s_t,u_t)}{\partial K_l} = \sum_l \overline{\omega}_l. \tag{28}$$

Then we can obtain $\partial Q_t(s_t,u_t)/\partial\phi$ for the input parameters, σ_i^l and m_i^l, based on the chain rule,

$$\frac{\partial Q_t(s_t,u_t)}{\partial \sigma_i^l} = \frac{\partial Q_t(s_t,u_t)}{\partial \omega_l}\frac{\partial \omega_l}{\partial \sigma_i^l} \tag{29}$$

$$\frac{\partial Q_t(s_t,u_t)}{\partial m_i^l} = \frac{\partial Q_t(s_t,u_t)}{\partial \omega_l}\frac{\partial \omega_l}{\partial m_i^l}. \tag{30}$$

The term $\partial Q_t(s_t,u_t)/\partial\omega_l$ is calculated from (17) and (15). The terms $\partial\omega_l/\partial\sigma_i^l$ and $\partial\omega_l/\partial m_i^l$ are calculated from both (14) and (13) so

$$\frac{\partial Q_t(s_t,u_t)}{\partial \sigma_i^l} = \frac{(K_l - Q_t(s_t,u_t))}{\sum_l \omega_l}\omega_l\frac{2(x_i - m_i^l)^2}{(\sigma_i^l)^3} \tag{31}$$

$$\frac{\partial Q_t(s_t,u_t)}{\partial m_i^l} = \frac{(K_l - Q_t(s_t,u_t))}{\sum_l \omega_l}\omega_l\frac{2(x_i - m_i^l)}{(\sigma_i^l)^2}. \tag{32}$$

Substituting from (28), (31) and (32) in (10) and modifying (27) to use an eligibility trace, the update law for the FIS parameters becomes

$$\phi_Q(t+1) = \phi_Q(t) + \eta\Delta_t e_t. \tag{33}$$

The update law in (27) is applied also to the FLC by replacing $Q_t(s_t,u_t)$ with the output of the FLC, u. In addition and as shown from Fig. 1, a random Gaussian noise, $n(0,\sigma_n)$, with zero mean and standard deviation σ_n is added to the output of the FLC in order to solve the exploration/exploitation dilemma as for example the ϵ-greedy exploration method used in the discrete state and action spaces. Then the update law for the FLC parameters is defined by

$$\phi_u(t+1) = \phi_u(t) + \xi\Delta_t\frac{\partial u}{\partial \phi}\left(\frac{u_c - u}{\sigma_n}\right) \tag{34}$$

where u_c is the output of the random Gaussian noise generator and ξ is the learning rate for the FLC parameters. The term $\partial u/\partial\phi$ can be calculated by replacing $Q_t(s_t,u_t)$ with the output of the FLC, u, in (28), (31) and (32).

7. The proposed Q(λ)-learning based genetic fuzzy logic controller

The proposed QLBGFC combines Q(λ)-learning with GAs to tune the parameters of the FLC. The learning process passes through two phases as shown in Fig. 2. Now, we describe the FLC used in the proposed technique then we will discuss the learning in the two phases.

7.1. Fuzzy logic controller

A block diagram of a FLC system is shown in Fig. 7. The FLC has two inputs, the error in the pursuer angle, δ, defined in (21), and its

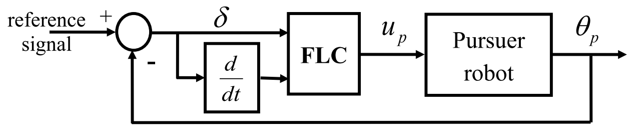


Fig. 7. Block diagram of a FLC system.

derivative, $\dot{\delta}$, and the output is the steering angle, u_p. For the inputs of the FLC, we use the Gaussian MF described by (13). For the rules we modify (11) to be

$$R_l:\ \text{IF } \delta \text{ is } A_1^l \text{ AND } \dot{\delta} \text{ is } A_2^l \text{ THEN } f_l = K_l \tag{35}$$

where $l = 1, 2, \ldots, 9$. The crisp output, u_p, is calculated using (12) as follows

$$u_p = \frac{\sum_{l=1}^{9}\left(\prod_{i=1}^{2}\mu^{A_i^l}(x_i)\right)K_l}{\sum_{l=1}^{9}\left(\prod_{i=1}^{2}\mu^{A_i^l}(x_i)\right)}. \tag{36}$$

7.2. Learning in phase 1

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 in angle of the pursuer, δ, and its derivative, $\dot{\delta}$, and the action, a, is the steering angle of the pursuer, u_p. The states, $(\delta, \dot{\delta})$, and their corresponding greedy actions, a^*, are then stored in a lookup table.

7.2.1. Building the discrete state and action spaces

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 -1.0 to 1.0 so the discretized values for δ and $\dot{\delta}$ will be $(-1.0, -0.8, -0.6, \ldots, 0.0, \ldots, 0.8, 1.0)$. 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 for u_p will be $(-0.5, -0.4, -0.3, \ldots, 0.0, \ldots, 0.4, 0.5)$. There are 11 actions and the dimension of the Q-table will be 121-by-11.

7.2.2. Constructing the reward function

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 the 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

$$D(t) = \sqrt{(x_e(t) - x_p(t))^2 + (y_e(t) - y_p(t))^2}. \tag{37}$$

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

$$\Delta D(t) = D(t) - D(t+1). \tag{38}$$

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

$$\Delta D_{\max} = V_{r\max} T \tag{39}$$

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

$$r_{t+1} = \frac{\Delta D(t)}{\Delta D_{\max}}. \tag{40}$$

The learning process in phase 1 is described in Algorithm 1.

Algorithm 1 (Phase 1: Q(λ)**-learning)**

```
1: Discretize the state space, $S$, and the action space, $A$.
2: Initialize $Q(s,a) = 0 \quad \forall s \in S, a \in A$.
3: Initialize $e(s,a) = 0 \quad \forall s \in S, a \in A$.
4: For each episode
   a: Initialize $(x_p, y_p) = (0, 0)$.
   b: Initialize $(x_e, y_e)$ randomly.
   c: Compute $s_t = (\delta, \dot{\delta})$ according to (21).
   d: Select $a_t$ using (3).
   e: For each play
        i: Receive $r_{t+1}$ according to (40).
       ii: Observe $s_{t+1}$.
      iii: Select $a_{t+1}$ using (3).
       iv: Calculate $e_{t+1}$ using (8).
        v: Update $Q(s_t, a_t)$ according to (9).
   f: End
5: End
6: $Q \leftarrow Q^*$.
7: Assign a greedy action, $a^*$, to each state, $s$ using (4).
8: Store the state–action pairs in a lookup table.
```

7.3. Learning in phase 2

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 the FLC using GAs. Stage 1 is an off-line tuning. In this stage, the fitness function used is the mean square error (MSE) defined as

$$\text{MSE} = \frac{1}{2M}\sum_{m=1}^{M}\left(a^{*^m} - u_{flc}^m\right)^2 \tag{41}$$

where M is the number of input/output data pairs and is equivalent to the number of states, a^{*^m} is the mth greedy action obtained from phase 1, and u_{flc} is the output of the FLC. The GA in this stage is used as supervised learning 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, we run the pursuit–evasion game with the tuned FLC as the controller. The GA is then used to fine tune the parameters of the FLC during the interaction with the evader. In this stage, the capture time which the pursuer wants to minimize is used as the fitness function. In this stage, the GA is used as a reward-based learning technique with a priori knowledge obtained from stage 1.

7.3.1. Coding a FLC into a chromosome

In phase 2, we use a GA to tune the input and the output parameters of the FLC. Now, we will describe the coding of the FLC parameters using the GA. Note that the coding process of the FLC is the same for all the different GAs used in this paper so we will describe it in general. 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 the Gaussian MF defined by (13) which has 2 parameters to be tuned. These parameters are the standard deviation, σ, and the mean, m. The total number of input parameters to be tuned is 12 parameters. The rules being used is defined by (35) that has a parameter, K, to be tuned with a total number of 9 parameters to be tuned for the output part.

A FLC will be coded into a chromosome with length $12 + 9 = 21$ genes as shown in Fig. 8. We use real numbers in the coding process. The population consists of a set of chromosomes, P, (coded FLCs). In the reproduction process, we generate new chromosomes by using two GA operations. The first operation is crossover in which we choose a pair of parents and select a random gene, g, between 1 and 20 and make a cross replacement from one parent to another as shown in Fig. 9. The second operation is mutation

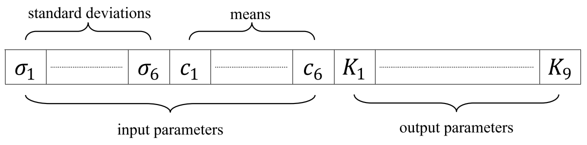
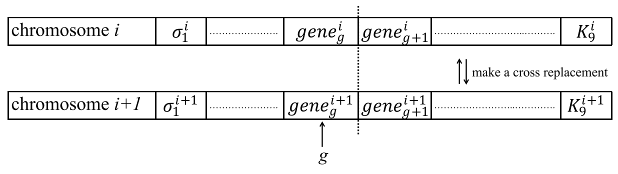
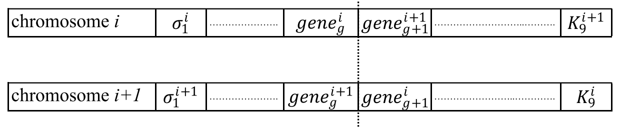


Fig. 8. A FLC coded into a chromosome.



(a) Old chromosomes.



(b) New chromosomes.

Fig. 9. Crossover process in a GA.

in which we generate a chromosome randomly to avoid a local minimum/maximum for the fitness function. Now, we have a new population to test again with the fitness function. The genetic process is repeated until the termination condition is met (see Section 4). The learning process in phase 2 with its two stages is described in Algorithms 2 and 3.

Algorithm 2 (Phase 2 Stage 1: GA learning)

```
1: Get the state–action pairs from the lookup table.
2: Initialize a set of chromosomes in a population, $P$, randomly.
3: **For** each iteration
   a: **For** each chromosome in the population
      i: Construct a FLC.
      ii: **For** each state, $s$,
         • Calculate the FLC output, $u_{flc}$, using (36).
      iii: **End**
      iv: Calculate the fitness value using (41).
   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 the population using crossover and mutation.
4: **End**
```

7.4. Reward-based genetic algorithm learning

For comparative purpose, we will also implement a general reward-based GA learning technique. The reward-based GA learning will be initialized with randomly chosen FLC parameters (chromosomes). The GA adjusts the parameters to maximize the closing distance given by (40). Therefore, (40) acts as the fitness function for the reward-based GA learning.

In the proposed QLBGFC, A GA is used in phase 2 stage 1 to tune the FLC parameters as determined from $Q(\lambda)$-learning in phase 1. The GA uses an MSE criterion given by (41) 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 GA to achieve the greedy actions defined by $Q(\lambda)$-learning in phase 1.

Algorithm 3 (Phase 2 Stage 2: GA learning)

```
1: Initialize a set of chromosomes in a population, $P$, from the tuned FLC obtained from stage 1.
2: **For** each iteration
   a: Initialize $(x_e, y_e)$ randomly.
   b: Initialize $(x_p, y_p) = (0, 0)$.
   c: Calculate $s_t = (\delta, \dot{\delta})$ according to (21).
   d: **For** each chromosome in the population
      i: Construct a FLC.
      ii: **For** each play
         • Calculate the FLC output, $u_p$, using (36).
         • Observe $s_{t+1}$.
      iii: **End**
      iv: Observe the fitness value which is the capture time that the pursuer wants to minimize.
   e: **End**
   f: Sort the entire chromosomes of the population according to their fitness values.
   g: Select a portion of the sorted population as the new parents.
   h: Create a new generation for the remaining portion of the population using crossover and mutation.
3: **End**
```

In phase 2 stage 2, the GA fine tunes the input and the output parameters of the FLC to achieve a minimizing capture time. The learning process in the reward-based GA learning is described in Algorithm 4.

Algorithm 4 (Reward-based GA learning)

```
1: Initialize a set of chromosomes in a population, $P$, randomly.
2: **For** each iteration
   a: Initialize $(x_e, y_e)$ randomly.
   b: Initialize $(x_p, y_p) = (0, 0)$.
   c: Calculate $s_t = (\delta, \dot{\delta})$ according to (21).
   d: **For** each chromosome in the population
      i: Construct a FLC.
      ii: **For** each play
         • Calculate the FLC output, $u_p$, using (36).
         • Observe $s_{t+1}$
      iii: **End**
      iv: Observe the fitness value defined by (40).
   e: **End**
   f: Sort the entire chromosomes of the population according to their fitness values.
   g: Select a portion of the sorted population as the new parents.
   h: Create a new generation for the remaining portion of the population using crossover and mutation.
3: **End**
```

8. Computer simulation

We use a core 2 duo with a 2.0 GHz clock frequency and 4.0 Gigabytes of RAM. We do computer simulation with MATLAB software. Q(λ)-learning and GAs have many parameters to be set a priori therefore we tested computer simulation for different parameter values and different parameter value combinations and chose the values that give the best performance. The initial position of the evader is randomly chosen from a set of 64 different positions in the space.

8.1. The pursuit–evasion game

The pursuer starts motion from the position $(0, 0)$ with an initial orientation $\theta_p = 0$ and with a constant velocity $V_p = 1$ m/s.

Table 1
Values of GAs parameters.

	Phase 2	
	Stage 1	Stage 2
Number of iterations	800	200
Population size	40	10
Number of plays	–	300
Crossover probability	0.2	0.2
Mutation probability	0.1	0.1
Fitness function	MSE defined by (41)	Capture time
Fitness function objective	Minimize	Minimize

The distance between the front and rear axle $L_p = 0.3$ m and the steering angle $u_p \in [-0.5, 0.5]$. From (19), $Rd_{p_{\min}} \simeq 0.55$ m.

The evader starts motion from a random position for each episode with an initial orientation $\theta_e = 0$ and with a constant velocity $V_e = 0.5$ m/s which is half that of the pursuer (slower). The distance between the front and rear axle $L_e = 0.3$ m and the steering angle $u_e \in [-1, 1]$ which is twice that of the pursuer (more maneuverable). From (19), $Rd_{e_{\min}} \simeq 0.19$ m which is about one third that of the pursuer. The duration of a game is 60 s. The game ends when 60 s passed without capture or when the capture occurs before the end of this time. The capture radius $\ell = 0.10$ m. The sampling time is set to 0.1 s.

8.2. The proposed QLFIS

We choose the number of episodes (games) to be 1000, the number of plays (steps) in each episode is 600, $\gamma = 0.95$, and $\lambda = 0.9$. We make the learning rate for the FIS, η, decrease with each episode such that

$$\eta = 0.1 - 0.09\left(\frac{i}{\text{Max. episodes}}\right) \tag{42}$$

and also make the learning rate for the FLC, ξ, decrease with each episode such that

$$\xi = 0.01 - 0.009\left(\frac{i}{\text{Max. episodes}}\right) \tag{43}$$

where i is the current episode. Note that the value of η is 10 times the value of ξ i.e. the FIS converges faster than the FLC to avoid instability in tuning the parameters of the FLC. We choose $\sigma_n = 0.08$.

8.3. The proposed QLBGFC

We choose the number of episodes to be 200, the number of plays in each episode is 6000, $\gamma = 0.5$, and $\lambda = 0.3$. We make the learning rate, α, decrease with each episode such that

$$\alpha = \frac{1}{(i)^{0.7}} \tag{44}$$

and we also make ε decrease with each episode such that

$$\varepsilon = \frac{0.1}{i} \tag{45}$$

where i is the current episode. The position of the evader, (x_e, y_e), is chosen randomly at the beginning of each episode to cover most of the states. Table 1 shows the values of GAs parameters used in phase 2 stage 1 and stage 2.

8.4. Compared techniques

To validate the proposed QLFIS and QLBGFC techniques, we compare their results with the results of the classical control strategy, Q(λ)-learning only, the technique proposed in [19], and the reward-based GA learning. The classical control strategies of the pursuer and the evader are defined by (20) and (21). The parameters of Q(λ)-learning only have the following values: the number of episodes is set to 1000, the number of plays in each episode is 6000, $\gamma = 0.5$ and $\lambda = 0.3$. The learning rate, α, and ε are defined by (44) and (45), respectively.

For the technique proposed in [19], we choose the same values for the parameters of the NN. The NN has a three-layer structure with 7-21-1 nodes. The RL parameters and the initial values of the input and the output parameters of the FLC are all chosen to be the same as those chosen in the proposed QLFIS. We choose $\sigma_n = 0.1$ which is decreasing each episode by $1/i$ where i is the current episode. The parameters of the reward-based GA learning are chosen as follows: the number of iterations = 1000, the population size = 40, the number of plays = 300, the probability of crossover = 0.2 and the probability of mutation = 0.1.

Note that in phase 2 stage 2 of the proposed QLBGFC we already have a tuned FLC (obtained from phase 2 stage 1) 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. This random initialization of the FLC will make the pursuer not be able or take a long time to catch the evader for some iterations and therefore the learning time will increase as we see in Section 9.

Table 2
Fuzzy decision table after tuning using the proposed QLFIS.

	$\dot{\delta}_p$		
δ_p	**N**	**Z**	**P**
N	−0.5452	−0.2595	−0.0693
Z	−0.2459	0.0600	0.2299
P	0.0235	0.3019	0.5594

Table 3
Fuzzy decision table after tuning using the proposed QLBGFC.

	$\dot{\delta}_p$		
δ_p	**N**	**Z**	**P**
N	−1.0927	−0.4378	−0.7388
Z	−0.5315	−0.2145	0.0827
P	0.9100	0.1965	0.0259

9. Results

Fig. 10 and Table 2 show the input and the output parameters of the FLC after tuning using the proposed QLFIS, respectively where "**N**", "**Z**", and "**P**" are referred to the linguistic values "**N**egative", "**Z**ero", and "**P**ositive". Fig. 11 and Table 3 show the input and the output parameters of the FLC after tuning using the proposed QLBGFC.

Table 4 shows the capture times for different initial positions of the evader using the classical control strategy of the pursuer, the Q(λ)-learning only, the technique proposed in [19], the reward-based GA, the proposed QLFIS and the proposed QLBGFC. In addition, the learning times for the different techniques are also shown in this table. From Table 4 we can see that although the Q(λ)-learning only has the minimum learning time, it is not enough to get the desired performance in comparison with the classical control strategy and the other techniques. The reward-based GA gets the best performance in comparison to the other techniques and its performance approaches that of the classical control strategy. However, the learning process using the reward-based GA takes a comparatively long learning time. The proposed QLFIS outperforms the technique proposed in [19] in both performance and learning time and both of them have better performance than using the Q(λ)-learning only. We can also see

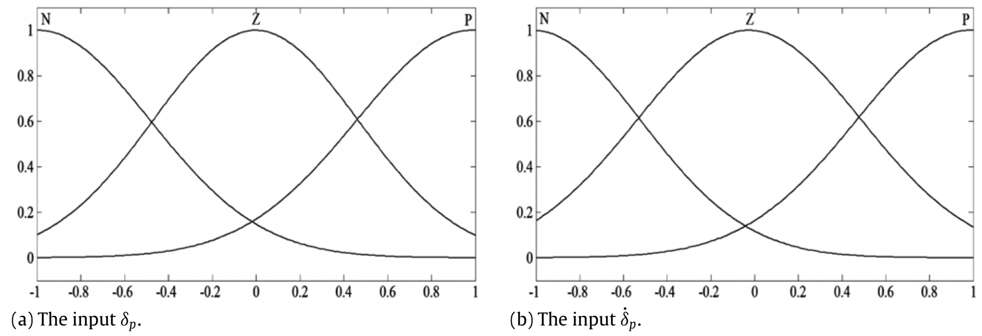

(a) The input δ_p. (b) The input $\dot{\delta}_p$.

Fig. 10. MFs for the inputs after tuning using the proposed QLFIS.

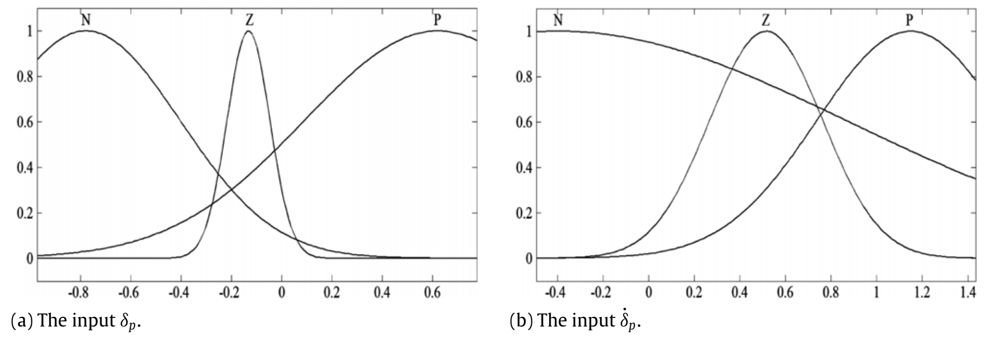

(a) The input δ_p. (b) The input $\dot{\delta}_p$.

Fig. 11. MFs for the inputs after tuning using the proposed QLBGFC.

Table 4
Capture time, in seconds, for different evader initial positions and learning time, in seconds, for the different techniques.

	Evader initial position				Learning time
	(−6,7)	(−7,−7)	(2,4)	(3,−8)	
Classical control strategy	9.6	10.4	4.5	8.5	–
Q(λ)-learning only	12.6	15.6	8.5	11.9	32.0
Technique proposed in [19]	10.9	12.9	4.7	9.1	258.6
Reward-based GA	9.7	10.5	4.5	8.6	460.8
Proposed QLFIS	10.0	10.7	4.6	8.8	65.2
Proposed QLBGFC	9.9	10.5	4.6	8.7	47.8

Table 5
Fuzzy decision table for the pursuer after tuning using the proposed QLFIS.

	$\dot{\delta}_p$		
δ_p	**N**	**Z**	**P**
N	−1.2990	−0.6134	−0.4064
Z	−0.3726	−0.0097	0.3147
P	0.3223	0.5763	0.9906

that the proposed QLBGFC has the best performance as the reward-based GA. In addition, it takes only 47.8 s in the learning process which is about 10% of the learning time taken by the reward-based GA and about 18% of the learning time taken by the technique proposed in [19].

Now, we will increase the complexity of the model by making both the pursuer and the evader self-learn their control strategies simultaneously. The difficulty in the learning process is that each robot will try to find its control strategy based on the control strategy of the other robot which, at the same time, is still learning. Fig. 12 and Table 5 show the input and the output parameters of the FLC for the pursuer using the proposed QLFIS. Fig. 13 and Table 6 show the input and the output parameters of the FLC for the evader using the proposed QLFIS. Fig. 14 and Table 7 show the input and the output parameters of the FLC for the pursuer using the proposed QLBGFC. Fig. 15 and Table 8 show the input and the output parameters of the FLC for the evader using the proposed QLBGFC.

Table 6
Fuzzy decision table for the evader after tuning using the proposed QLFIS.

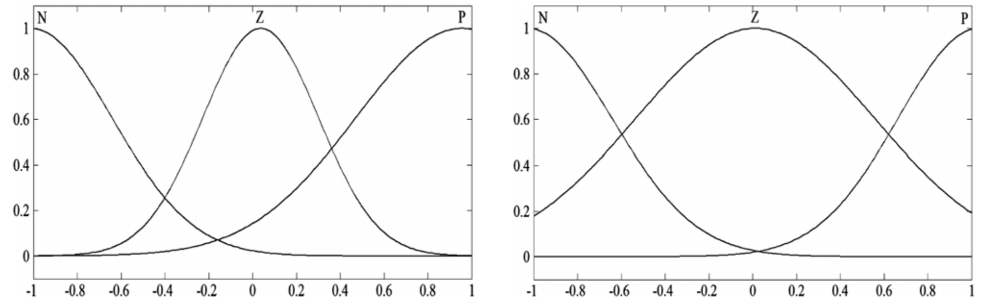
	$\dot{\delta}_e$		
δ_e	**N**	**Z**	**P**
N	−1.4827	−0.4760	−0.0184
Z	−0.5365	−0.0373	0.5500
P	−0.0084	0.4747	1.1182

Table 7
Fuzzy decision table for the pursuer after tuning using the proposed QLBGFC.

	$\dot{\delta}_p$		
δ_p	**N**	**Z**	**P**
N	−0.4677	−0.2400	−0.7332
Z	−0.8044	−1.2307	−0.2618
P	0.8208	0.1499	0.9347

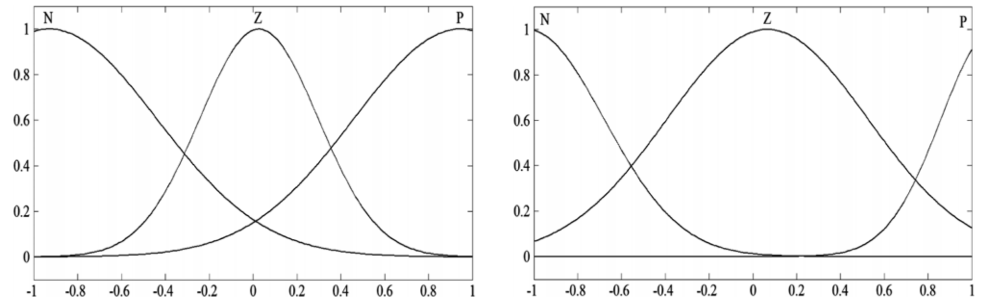
Table 8
Fuzzy decision table for the evader after tuning using the proposed QLBGFC.

	$\dot{\delta}_e$		
δ_e	**N**	**Z**	**P**
N	−1.1479	−0.0022	−0.2797
Z	−0.0529	−0.9777	−0.1257
P	0.4332	0.4061	0.7059



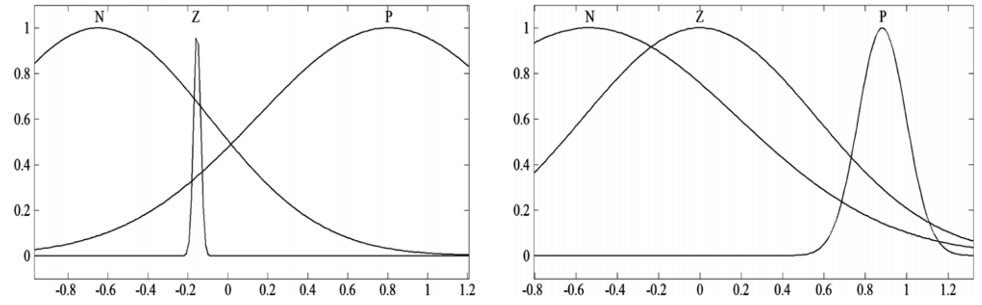
(a) The input δ_p. (b) The input $\dot{\delta}_p$.

Fig. 12. MFs for the inputs of the pursuer after tuning using the proposed QLFIS.



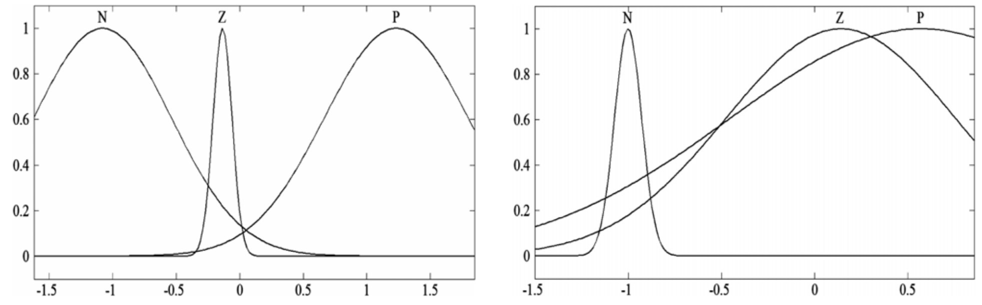
(a) The input δ_e. (b) The input $\dot{\delta}_e$.

Fig. 13. MFs for the inputs of the evader after tuning using the proposed QLFIS.



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

Fig. 14. MFs for the inputs of the pursuer after tuning using the proposed QLBGFC.



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

Fig. 15. MFs for the inputs of the evader after tuning using the proposed QLBGFC.

To check the performance of the different techniques we cannot use the capture time as a measure, as we did in Table 4, because in this game both the pursuer and the evader are learning so we may find capture times that are smaller than those corresponding to the optimal solution. Of course that does not mean that the performance is better than the optimal solution but it means that the evader does not learn well and as a result it is captured in a shorter time. Therefore the measure that we use is the paths of both the pursuer and the evader instead of the capture time. Figs. 16–20 show the paths of the pursuer and the evader of the different techniques against the classical control strategies of the pursuer and the evader. We can see that the best performance is that of the proposed QLFIS and the proposed QLBGFC. We can also see that the performance of the reward-based GA diminishes

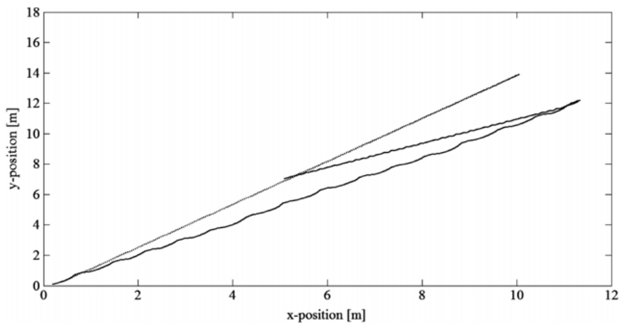


Fig. 16. Paths of the pursuer and the evader (solid line) using the Q(λ)-learning only against the classical control strategies (dotted line).

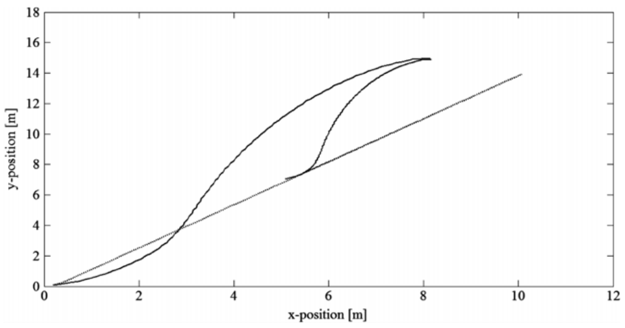


Fig. 17. Paths of the pursuer and the evader (solid line) using the technique proposed in [19] against the classical control strategies (dotted line).

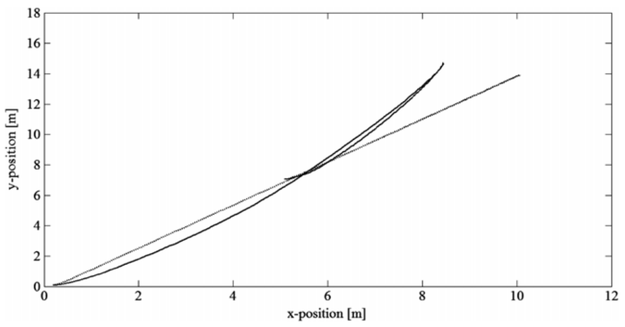


Fig. 18. Paths of the pursuer and the evader (solid line) using the reward-based GA against the classical control strategies (dotted line).

Table 9
Learning time, in seconds, for the different techniques.

	Learning time
Q(λ)-learning only	62.8
Technique proposed in [19]	137.0
Reward-based GA	602.5
Proposed QLFIS	110.3
Proposed QLBGFC	54.7

as a result of increasing the complexity of the system by making both the pursuer and the evader learn their control strategies simultaneously.

Table 9 shows the learning time for the different techniques. Table 9 shows that the proposed QLBGFC has the minimum learning time. Finally, we can conclude that the proposed QLBGFC has the best performance and the best learning time among all the other techniques. We can also see that the proposed QLFIS still outperforms the technique proposed in [19] in both performance and learning time.

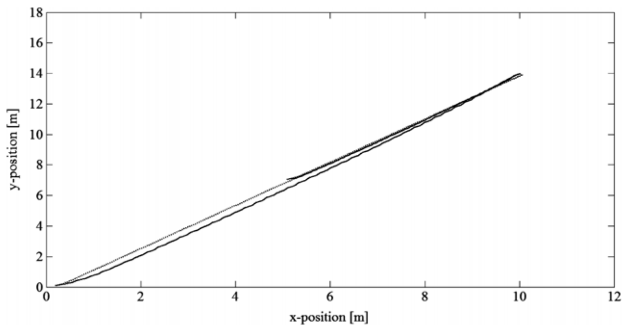


Fig. 19. Paths of the pursuer and the evader (solid line) using the proposed QLFIS against the classical control strategies (dotted line).

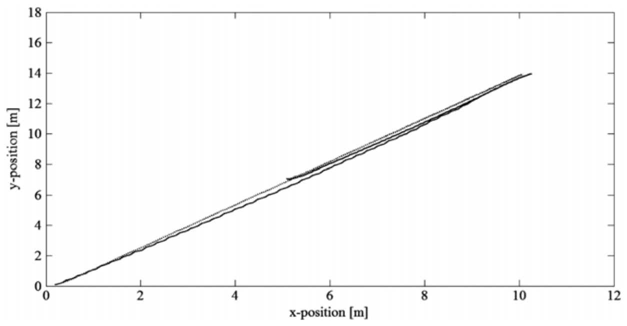


Fig. 20. Paths of the pursuer and the evader (solid line) using the proposed QLBGFC against the classical control strategies (dotted line).

10. Conclusion

In this paper, we proposed two novel techniques to tune the parameters of FLC in which RL is combined with FIS as a function approximation to generalize the state and the action spaces to the continuous case. The second technique combines RL with GAs as a powerful optimization technique. The proposed techniques are applied to a pursuit–evasion game. First, we assume that the pursuer does not know its control strategy. However it can self-learn its control strategy by interaction with the evader. Second, we increase the complexity of the model by assuming that both the pursuer and the evader do know their control strategies. Computer simulation and the results show that the proposed QLFIS and the proposed QLBGFC techniques outperform all the other techniques in performance when compared with the classical control strategy and in the learning time which is also an important factor especially in on-line applications.

In future work, we will test our proposed technique in a more complex pursuit–evasion games in which multiple pursuers and multiple evaders self-learn their control strategies.

References

[1] S. Micera, A.M. Sabatini, P. Dario, Adaptive fuzzy control of electrically stimulated muscles for arm movements, Medical and Biological Engineering and Computing 37 (1999) 680–685.
[2] F. Daldaban, N. Ustkoyuncu, K. Guney, Phase inductance estimation for switched reluctance motor using adaptive neuro-fuzzy inference system, Energy Conversion and Management 47 (2006) 485–493.
[3] S. Labiod, T.M. Guerra, Adaptive fuzzy control of a class of SISO nonaffine nonlinear systems, Fuzzy Sets and Systems 158 (10) (2007) 1126–1137.
[4] H.K. Lam, F.H.F. Leung, Fuzzy controller with stability and performance rules for nonlinear systems, Fuzzy Sets and Systems 158 (2007) 147–163.
[5] H. Hagras, V. Callaghan, 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 141 (2004) 107–160.
[6] M. Mucientes, D.L. Moreno, A. Bugarin, S. Barro, Design of a fuzzy controller in mobile robotics using genetic algorithms, Applied Soft Computing 7 (2) (2007) 540–546.

[7] L.X. Wang, Generating fuzzy rules by learning from examples, IEEE Transactions on Systems, Man, and Cybernetics 22 (1992) 1414–1427.
[8] F. Herrera, M. Lozano, J.L. Verdegay, Tuning fuzzy logic controllers by genetic algorithms, International Journal of Approximate Reasoning 12 (1995) 299–315.
[9] A. Lekova, L. Mikhailov, D. Boyadjiev, A. Nabout, Redundant fuzzy rules exclusion by genetic algorithms, Fuzzy Sets and Systems 100 (1998) 235–243.
[10] S.F. Desouky, H.M. Schwartz, Genetic based fuzzy logic controller for a wall-following mobile robot, in: 2009 American Control Conference, IEEE Press, St. Louis, MO, 2009, pp. 3555–3560.
[11] S.F. Desouky, H.M. Schwartz, Different hybrid intelligent systems applied for the pursuit–evasion game, in: 2009 IEEE International Conference on Systems, Man, and Cybernetics, 2009, pp. 2677–2682.
[12] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
[13] L.P. Kaelbling, M.L. Littman, A.P. Moore, Reinforcement learning: a survey, Journal of Artificial Intelligence Research 4 (1996) 237–285.
[14] W.D. Smart, L.P. Kaelbling, Effective reinforcement learning for mobile robots, in: IEEE International Conference on Robotics and Automation, Washington, DC, vol. 4, 2002, pp. 3404–3410.
[15] C. Ye, N.H.C. Yung, 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 33 (2003) 17–27.
[16] T. Kondo, K. Ito, A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robots control, Robotics and Autonomous Systems 46 (2004) 111–124.
[17] D.A. Gutnisky, B.S. Zanutto, Learning obstacle avoidance with an operant behavior model, Artificial Life 10 (1) (2004) 65–81.
[18] M. Rodríguez, R. Iglesias, C.V. Regueiro, J. Correa, S. Barro, Autonomous and fast robot learning through motivation, Robotics and Autonomous Systems 55 (2007) 735–740.
[19] X. Dai, C.K. Li, A.B. Rad, An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control, IEEE Transactions on Intelligent Transportation Systems 6 (2005) 285–293.
[20] M.J. Er, C. Deng, Obstacle avoidance of a mobile robot using hybrid learning approach, IEEE Transactions on Industrial Electronics 52 (2005) 898–905.
[21] H. Xiao, L. Liao, F. Zhou, Mobile robot path planning based on Q-ANN, in: IEEE International Conference on Automation and Logistics, Jinan, China, 2007, pp. 2650–2654.
[22] R. Vidal, O. Shakernia, H.J. Kim, D.H. Shim, S. Sastry, Probabilistic pursuit–evasion games: Theory, implementation, and experimental evaluation, IEEE Transactions on Robotics and Automation 18 (2002) 662–669.
[23] Y. Ishiwaka, T. Satob, Y. Kakazu, An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning, Robotics and Autonomous Systems 43 (4) (2003) 245–256.
[24] S.N. Givigi, H.M. Schwartz, X. Lu, A reinforcement learning adaptive fuzzy controller for differential games, Journal of Intelligent and Robotic Systems 59 (1) (2010) 3–30.
[25] L. Jouffe, Fuzzy inference system learning by reinforcement methods, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 28 (1998) 338–355.
[26] Y. Duan, X. Hexu, Fuzzy reinforcement learning and its application in robot navigation, in: International Conference on Machine Learning and Cybernetics, vol. 2, IEEE Press, Guangzhou, 2005, pp. 899–904.
[27] L. Buşoniu, R. Babuška, B. de Schutter, A comprehensive survey of multiagent reinforcement learning, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 38 (2008) 156–172.
[28] A. Waldock, B. Carse, Fuzzy Q-learning with an adaptive representation, in: 2008 IEEE 16th International Conference on Fuzzy Systems (FUZZ-IEEE), Piscataway, NJ, 2008, pp. 720–725.
[29] C.J.C.H. Watkins, P. Dayan, Q-learning, Machine Learning 8 (1992) 279–292.
[30] C.J.C.H. Watkins, Learning from delayed rewards. Ph.D. thesis, Cambridge University, 1989.
[31] B. Bhanu, P. Leang, C. Cowden, Y. Lin, M. Patterson, Real-time robot learning, in: IEEE International Conference on Robotics and Automation, vol. 1, Seoul, Korea, 2001, pp. 491–498.
[32] K. Maček, I. Petrović, 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.
[33] T.M. Marin, T. Duckett, Fast reinforcement learning for vision-guided mobile robots, in: IEEE International Conference on Robotics and Automation, Barcelona, Spain, 2005, pp. 4170–4175.
[34] Y. Yang, Y. Tian, H. Mei, Cooperative Q-learning based on blackboard architecture, in: International Conference on Computational Intelligence and Security Workshops, Harbin, China, 2007, pp. 224–227.
[35] S.F. Desouky, H.M. Schwartz, A novel technique to design a fuzzy logic controller using Q(λ)-learning and genetic algorithms in the pursuit–evasion game, in: 2009 IEEE International Conference on Systems, Man, and Cybernetics, San Antonio, TX, 2009, pp. 2683–2689.
[36] C. Deng, M.J. Er, Real-time dynamic fuzzy Q-learning and control of mobile robots, in: 5th Asian Control Conference, vol. 3, 2004, pp. 1568–1576.
[37] M.J. Er, 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.
[38] V. Stephan, K. Debes, H.M. Gross, A new control scheme for combustion processes using reinforcement learning based on neural networks, International Journal of Computational Intelligence and Applications 1 (2001) 121–136.
[39] X.W. Yan, Z.D. Deng, Z.Q. Sun, Genetic Takagi-Sugeno fuzzy reinforcement learning, in: IEEE International Symposium on Intelligent Control, Mexico City, Mexico, 2001, pp. 67–72.
[40] D. Gu, H. Hu, Accuracy based fuzzy Q-learning for robot behaviours, in: International Conference on Fuzzy Systems, Budapest, Hungary, vol. 3, 2004, pp. 1455–1460.
[41] S.P. Singh, R.S. Sutton, Reinforcement learning with replacing eligibility traces, Machine Learning 22 (1996) 123–158.
[42] E.H. Mamdani, S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, International Journal of Man-Machine Studies 7 (1) (1975) 1–13.
[43] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modelling and control, IEEE Transactions on Systems, Man and Cybernetics SMC-15 (1985) 116–132.
[44] J.H. Holland, Genetic algorithms and the optimal allocation of trials, SIAM Journal on Computing 2 (2) (1973) 88–105.
[45] D.E. Goldberg, Genetic Algorithms for Search, Optimization, and Machine Learning, Addison-Wesley, Reading, 1989.
[46] C. Karr, Genetic algorithms for fuzzy controllers, AI Expert 6((2) (1991) 26–33.
[47] L. Ming, G. Zailin, Y. Shuzi, Mobile robot fuzzy control optimization using genetic algorithm, Artificial Intelligence in Engineering 10 (4) (1996) 293–298.
[48] Y.C. Chiou, L.W. Lan, Genetic fuzzy logic controller: An iterative evolution algorithm with new encoding method, Fuzzy Sets and Systems 152 (2005) 617–635.
[49] R. Isaacs, Differential Game, John Wiley and Sons, 1965.
[50] S.M. LaValle, Planning Algorithms, Cambridge University Press, 2006.
[51] S.H. Lim, T. Furukawa, G. Dissanayake, H.F.D. Whyte, A time-optimal control strategy for pursuit–evasion games problems, in: International Conference on Robotics and Automation, New Orleans, LA, 2004.
[52] J. Peng, R.J. Williams, Incremental multi-step Q-learning, Machine Learning 22 (1996) 283–290.
[53] L. Jouffe, Actor-critic learning based on fuzzy inference system, in: IEEE International Conference on Systems, Man, and Cybernetics, Beijing, China, vol. 1, 1996, pp. 339–344.
[54] X.S. Wang, Y.H. Cheng, J.Q. Yi, A fuzzy actor-critic reinforcement learning network, Information Sciences 177 (2007) 3764–3781.

Sameh F. Desouky received his B.Eng. and M.Sc. degree from the military technical college, Cairo, Egypt in June 1996 and January 2002, respectively. He is currently working toward the Ph.D. degree in the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada, and his research interests include adaptive and fuzzy control, genetic algorithms, reinforcement learning, and mobile robots.

Howard M. Schwartz received his B.Eng. degree from McGill University, Montreal, Quebec in June 1981 and his M.Sc. degree and Ph.D. degree from M.I.T., Cambridge, Massachusetts in 1982 and 1987, respectively. He is currently the chief of the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada, and his research interests include adaptive and intelligent control systems, robotics and process control, system modeling, system identification, system simulation and computer vision systems.