Int. J. Fuzzy Syst. (2017) 19(4):1058–1076
DOI 10.1007/s40815-016-0284-8

# A Residual Gradient Fuzzy Reinforcement Learning Algorithm for Differential Games

Mostafa D. Awheda[1] · Howard M. Schwartz[1]

Received: 17 November 2015 / Revised: 2 November 2016 / Accepted: 27 November 2016 / Published online: 16 February 2017

**Abstract** In this work, we propose a new fuzzy reinforcement learning algorithm for differential games that have continuous state and action spaces. The proposed algorithm uses function approximation systems whose parameters are updated differently from the updating mechanisms used in the algorithms proposed in the literature. Unlike the algorithms presented in the literature which use the direct algorithms to update the parameters of their function approximation systems, the proposed algorithm uses the residual gradient value iteration algorithm to tune the input and output parameters of its function approximation systems. It has been shown in the literature that the direct algorithms may not converge to an answer in some cases, while the residual gradient algorithms are always guaranteed to converge to a local minimum. The proposed algorithm is called the residual gradient fuzzy actor–critic learning (RGFACL) algorithm. The proposed algorithm is used to learn three different pursuit–evasion differential games. Simulation results show that the performance of the proposed RGFACL algorithm outperforms the performance of the fuzzy actor–critic learning and the Q-learning fuzzy inference system algorithms in terms of convergence and speed of learning.

**Keywords** Fuzzy control · Reinforcement learning · Pursuit–evasion differential games · Residual gradient algorithms

## 1 Introduction

Fuzzy systems have been widely used in a variety of applications in many different fields in engineering, business, medicine and psychology [1]. Fuzzy systems have also influenced research in other different fields such as in data mining [2]. Fuzzy systems are also known by a number of names such as fuzzy logic controllers (FLCs), fuzzy inference systems (FISs), fuzzy expert systems, and fuzzy models. FLCs have recently attracted considerable attention as intelligent controllers [3, 4]. FLCs have been widely used to deal with plants that are nonlinear and ill-defined [5–7]. They can also deal with plants with high uncertainty in the knowledge about their environments [8, 9]. However, one of the problems in adaptive fuzzy control is which mechanism should be used to tune the fuzzy controller. Several learning approaches have been developed to tune the FLCs so that the desired performance is achieved. Some of these approaches design the fuzzy systems from input–output data by using different mechanisms such as a table lookup approach, a genetic algorithm approach, a gradient-descent training approach, a recursive least squares approach, and clustering [10, 11]. This type of learning is called supervised learning where a training data set is used to learn from. However, in this type of learning, the performance of the learned FLC will depend on the performance of the expert. In addition, the training data set used in supervised learning may be hard or expensive to obtain. In such cases, we think of alternative techniques where neither a priori knowledge nor a training data set is

✉ Mostafa D. Awheda
mawheda@sce.carleton.ca

Howard M. Schwartz
schwartz@sce.carleton.ca

1 Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada

required. In this case, reward-based learning techniques, such as reinforcement learning algorithms, can be used. The main advantage of such reinforcement learning algorithms is that they do not need either a known model for the process nor an expert to learn from [12].

Reinforcement learning (RL) is a learning technique that maps situations to actions so that an agent learns from the experience of interacting with its environment [13, 14]. Reinforcement learning has attracted attention and has been used in intelligent robot control systems [15–26]. Reinforcement learning has also been used for solving nonlinear optimal control problems [27–38]. Without knowing which actions to take, the reinforcement learning agent exploits and explores actions to discover which action gives the maximum reward in the long run. Different from supervised learning, which is learning from input-output data provided by an expert, reinforcement learning is adequate for learning from interaction by using very simple evaluative or critic information instead of instructive information [13]. Most of the traditional reinforcement learning algorithms represent the state/state-action value function as a lookup table for each state/state-action-pair [13]. Despite the theoretical foundations of these algorithms and their effectiveness in many applications, these reinforcement learning approaches cannot be applied to real applications with large state and action spaces [13, 18, 39–42]. This is because of the phenomenon known as the curse of dimensionality caused by the exponentially grown number of states when the number of state variables increases [13]. Moreover, traditional reinforcement learning approaches cannot be applied to differential games, where the states and actions are continuous. One of the possible solutions to the problem of continuous domains is to discretize the state and action spaces. However, this discretization may also lead to the problem of the curse of dimensionality that appears when discretizing large continuous states and/or actions [13, 18, 26]. To overcome these issues that lead to the problem of the curse of dimensionality, one may use a function approximation system to represent the large discrete and/or continuous spaces [10, 13, 43, 44]. Different types of function approximation systems are used in the literature, and the gradient-descent-based function approximation systems are among the most widely used ones [13]. In addition, the gradient-descent-based function approximation systems are well suited to online reinforcement learning [13].

## 1.1 Related Work

Several fuzzy reinforcement learning approaches have been proposed in the literature to deal with differential games (that have continuous state and action spaces) by using gradient-descent-based function approximation systems [16, 26, 43, 45–47]. Some of these approaches only tune the output parameters of their function approximation systems [16, 43, 46], where the input parameters of their function approximations are kept fixed. On the other hand, the other approaches tune both the input and output parameters of their function approximation systems [26, 45, 47].

In [43], the author proposed a fuzzy actor–critic learning (FACL) algorithm that uses gradient-descent-based FISs as function approximation systems. The FACL algorithm only tunes the output parameters (the consequent parameters) of its FISs (the actor and the critic) by using the temporal difference error (TD) calculated based on the state value function. However, the input parameters (the premise parameters) of its FISs are kept fixed during the learning process. In [46], the authors proposed a fuzzy reinforcement learning approach that uses gradient-descent-based FISs as function approximation systems. Their approach only tunes the output parameters of the function approximation systems based on the TD error calculated based on the state value functions of the two successive states in the state transition. In [16], the authors proposed the generalized probabilistic fuzzy reinforcement learning (GPFRL) algorithm, which is a modified version of the actor–critic learning architecture. The GPFRL algorithm uses gradient-descent-based FISs as function approximation systems. The GPFRL only tunes the output parameters of its function approximation systems based on the TD error of the critic and the performance function of the actor. In [26], the authors proposed a fuzzy learning approach that uses a time delay neural network (TDNN) and a FIS as gradient-descent-based function approximation systems. Their approach tunes the input and output parameters of the function approximation systems based on the TD error calculated based on the state-action value function. In [47], the authors proposed a fuzzy actor–critic reinforcement learning network (FACRLN) algorithm based on a fuzzy radial basis function (FRBF) neural network. The FACRLN uses the FRBF neural networks as function approximation systems. The FACRLN algorithm tunes the input and output parameters of the function approximation systems based on the TD error calculated by the temporal difference of the value function between the two successive states in the state transition. In [45], the authors proposed the QLFIS algorithm that uses gradient-descent-based FISs as function approximation systems. The QLFIS algorithm tunes the input and output parameters of the function approximation systems based on the TD error of the state-action value functions of the two successive states in the state transition. However, all these fuzzy reinforcement learning algorithms [16, 26, 43, 45–47] use what so-called direct algorithms described in [48] to tune the parameters of their function approximation systems. Although direct algorithms have been widely used in the tuning mechanism of the parameters for the function approximation systems, the direct algorithms may lead the function approximation systems to unpredictable results and, in some cases, to divergence [48–52].

### 1.2 Main Contribution

In this work, we propose a new fuzzy reinforcement learning algorithm for differential games, where the states and actions are continuous. The proposed algorithm uses function approximation systems whose parameters are updated differently from the updating mechanisms used in the algorithms proposed in [16, 26, 43, 45–47]. Unlike the algorithms presented in the literature which use the direct algorithms to tune their function approximation systems, the proposed algorithm uses the residual gradient value iteration algorithm to tune the input and output parameters of its function approximation systems. The direct algorithms tune the parameters of the function approximation system based on the partial derivatives of the value function at the current state $s_t$, whereas the residual gradient algorithms tune the parameters of the function approximation system based on the partial derivatives of the value function at both the current state $\mathbf{s}_t$ and the next state $\mathbf{s}_{t+1}$. The direct and residual gradient algorithms are presented in Sect. 2.1. The direct algorithms may not converge to an answer in some cases, while the residual gradient algorithms are always guaranteed to converge to a local minimum [48–52]. Furthermore, we take this opportunity to also present the complete derivation of the partial derivatives that are needed to compute both the direct and residual gradient algorithms. To the best of our knowledge, the derivation of the partial derivatives has never been explicitly shown. Table 1 shows a brief comparison among some fuzzy reinforcement learning algorithms and the proposed algorithm.

We investigate the proposed algorithm on different pursuit–evasion differential games because this kind of games is considered as a general problem for several other problems such as the problems of wall following, obstacle avoidance, and path planning. Moreover, pursuit–evasion games are useful for many real-world applications such as search and rescue, locating and capturing hostile intruders, localizing and neutralizing environmental threads, and surveillance and tracking [18, 45]. The proposed algorithm is used to learn three different pursuit–evasion differential games. In the first game, the evader is following a simple control strategy, whereas the pursuer is learning its control strategy to capture the evader in minimum time. In the second game, it is also only the pursuer that is learning. However, the evader is following an intelligent control strategy that exploits the advantage of the maneuverability of the evader. In the third game, we make both the pursuer and the evader learn their control strategies. Therefore, the complexity of the system will increase as the learning in a multi-robot system is considered as a problem of a "moving target" [53]. In the multi-robot system learning, each robot will try to learn its control strategy by interacting with the other robot which is also learning its control strategy at the same time. Thus, the best-response policy of each learning robot may keep changing during learning in multi-robot system. The proposed algorithm outperforms the FACL and QLFIS algorithms proposed in [43] and [45] in terms of convergence and speed of learning when they all are used to learn the pursuit–evasion differential games considered in this work.

This paper is organized as follows: Preliminary concepts and notations are reviewed in Sect. 2. The QLFIS algorithm proposed in [45] is presented in Sect. 3. Section 4 presents the pursuit–evasion game. The proposed RGFACL algorithm is introduced in Sect. 5. The simulation and results are presented in Sect. 6.

## 2 Preliminary Concepts and Notations

The direct and the residual gradient algorithms and the fuzzy inference systems are presented in this section.

**Table 1** Comparison among some fuzzy reinforcement learning algorithms and the proposed algorithm

| Algorithm | Type of gradient-descent method | Input parameters of function approximation systems | Output parameters of function approximation systems |
|---|---|---|---|
| FACL [43] | Direct | Fixed | Tuned |
| Algorithm proposed in [46] | Direct | Fixed | Tuned |
| GPFRL [16] | Direct | Tuned | Tuned |
| Algorithm proposed in [26] | Direct | Tuned | Tuned |
| FACRLN [47] | Direct | Tuned | Tuned |
| QLFIS [45] | Direct | Tuned | Tuned |
| The proposed algorithm | Residual | Tuned | Tuned |

## 2.1 Direct and Residual Gradient Algorithms

Traditional reinforcement learning algorithms such as the Q-learning algorithm and the value iteration algorithm represent the value functions as lookup tables and are guaranteed to converge to optimal values [13]. These algorithms have to be combined with function approximation systems when they are applied to real applications with large state and action spaces or with continuous state and action spaces. The direct and the residual gradient algorithms described in [48] can be used when the traditional Q-learning and the value iteration algorithms are combined with function approximation systems. In [48], the author illustrated with some examples that the direct algorithms may converge fast but may become unstable in some cases. In addition, the author showed that the residual gradient algorithms converge in those examples and are always guaranteed to converge to a local minimum. Another study presented in [51] shows that the direct algorithms are faster than the residual gradient algorithms only when the value function is represented in a tabular form. However, when function approximation systems are involved, the direct algorithms are not always faster than the residual gradient algorithms. In other words, when function approximation systems are involved, the residual gradient algorithms are considered as the superior algorithms as they are always guaranteed to converge, whereas the direct algorithms may not converge to an answer in some cases. Other different studies [49, 50, 52] confirm the results presented in [48] in terms of the superiority of the residual gradient algorithms as they are always shown to converge. To illustrate the difference between the direct algorithms and the residual gradient algorithms, we will give two different examples of these algorithms: the direct value iteration algorithm (an example of the direct algorithms) and the residual gradient value iteration algorithm (an example of the residual gradient algorithms).

### 2.1.1 The Direct Value Iteration Algorithm

For Markov decision processes (MDPs), the value function $V_t(\mathbf{s}_t)$ at the state $\mathbf{s}_t$ for approximation of the reinforcement rewards can be defined as follows [46, 54],

$$V_t(\mathbf{s}_t) = E\left\{\sum_{i=t}^{\infty} \gamma^{i-t} r_i\right\} \tag{1}$$

where $\gamma \in [0, 1)$ is a discount factor and $r_i$ is the immediate external reward that the learning agent gets from the learning environment.

The recursive form of Eq. (1) can be defined as follows [46, 54],

$$V_t(\mathbf{s}_t) = r_t + \gamma V_t(\mathbf{s}_{t+1}) \tag{2}$$

The temporal difference residual error, $\Delta_t$, and the mean square error, $E_t$, between the two sides of Eq. (2) are given as follows,

$$\Delta_t = [r_t + \gamma V_t(\mathbf{s}_{t+1})] - V_t(\mathbf{s}_t)] \tag{3}$$

$$E_t = \frac{1}{2}\Delta_t^2 \tag{4}$$

For a deterministic MDP, after transition from a state to another, the direct value iteration algorithm updates the weights of the function approximation system as follows [48],

$$\psi_{t+1} = \psi_t - \alpha \frac{\partial E_t}{\partial \psi_t} \tag{5}$$

where $\psi_t$ represents the input and output parameters of the function approximation system that needs to be tuned, $\alpha$ is a learning rate, and the term $\frac{\partial E_t}{\partial \psi_t}$ is defined as follows,

$$\frac{\partial E_t}{\partial \psi_t} = \Delta_t \frac{\partial \Delta_t}{\partial \psi_t} = -[r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t)].\frac{\partial}{\partial \psi_t} V_t(\mathbf{s}_t) \tag{6}$$

Thus,

$$\psi_{t+1} = \psi_t + \alpha[r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t)].\frac{\partial}{\partial \psi_t} V_t(\mathbf{s}_t) \tag{7}$$

The direct value iteration algorithm updates the derivative $\frac{\partial E_t}{\partial \psi_t}$ as in Eq. (6). This equation shows that the direct value iteration algorithm treats the value function $V_t(\mathbf{s}_{t+1})$ in the temporal error $\Delta_t$ as a constant. Therefore, the derivative $\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \psi_t}$ will be zero. However, the value function $V_t(\mathbf{s}_{t+1})$ is not a constant, and it is a function of the input and output parameters of the function approximation system, $\psi_t$. Therefore, the derivative $\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \psi_t}$ should not be assigned to zero during the tuning of the input and output parameters of the function approximation system.

### 2.1.2 The Residual Gradient Value Iteration Algorithm

The residual gradient value iteration algorithm updates the weights of the function approximation system as follows [48],

$$\psi_{t+1} = \psi_t - \alpha \frac{\partial E_t}{\partial \psi_t} \tag{8}$$

where

$$\frac{\partial E_t}{\partial \psi_t} = \Delta_t \frac{\partial \Delta_t}{\partial \psi_t} = [r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t)].\left[\frac{\partial}{\partial \psi_t}\gamma V_t(\mathbf{s}_{t+1}) - \frac{\partial}{\partial \psi_t} V_t(\mathbf{s}_t)\right] \tag{9}$$

Thus,

$$\psi_{t+1} = \psi_t - \alpha[r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t)].\left[\frac{\partial}{\partial \psi_t}\gamma V_t(\mathbf{s}_{t+1}) - \frac{\partial}{\partial \psi_t}V_t(\mathbf{s}_t)\right] \tag{10}$$

The residual gradient value iteration algorithm updates the derivative $\frac{\partial E_t}{\partial \psi_t}$ as in Eq. (9). In this equation, the residual gradient value iteration algorithm treats the value function $V_t(\mathbf{s}_{t+1})$ in the temporal error $\Delta_t$ as a function of the input and output parameters of the function approximation system, $\psi_t$. Therefore, the derivative $\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \psi_t}$ will not be assigned to zero during the tuning of the input and output parameters of the function approximation system. This will make the residual gradient value iteration algorithm performs better than the direct value iteration algorithm in terms of convergence [48].

## 2.2 Fuzzy Inference Systems

Fuzzy inference systems may be used as function approximation systems so that reinforcement learning approaches can be applied to real systems with continuous domains [10, 13, 43]. Among the most widely used FISs are the Mamdani FIS proposed in [55] and the Takagi–Sugeno–Kang (TSK) FIS proposed in [56, 57]. The FISs used in this work are zero-order TSK FISs with constant consequents. Each FIS consists of $L$ rules. The inputs of each rule are $n$ fuzzy variables, whereas the consequent of each rule is a constant number. Each rule $l$ $(l = 1, \ldots, L)$ has the following form,

$$R_l: \text{ IF } s_1 \text{ is } F_1^l, \ldots, \text{ and } s_n \text{ is } F_n^l \quad \text{THEN} \quad z_l = k_l \tag{11}$$

where $s_i$ $(i = 1, \ldots, n)$ is the $i$th input state variable of the fuzzy system, $n$ is the number of input state variables, and $F_i^l$ is the linguistic value of the input $s_i$ at the rule $l$. Each input $s_i$ has $h$ membership functions. The variable $z_l$ represents the output variable of the rule $l$, and $k_l$ is a constant that describes the consequent parameter of the rule $l$. In this work, Gaussian membership functions are used for the inputs and each membership function (MF) is defined as follows,

$$\mu^{F_i^l}(s_i) = \exp\left(-\left(\frac{s_i - m}{\sigma}\right)^2\right) \tag{12}$$

where $\sigma$ and $m$ are the standard deviation and the mean, respectively.

In each FIS used in this work, the total number of the standard deviations of the membership functions of its inputs is defined as $H$, where $H = n \times h$. In addition, the total number of the means of the membership functions of its inputs is $H$. Thus, for each FIS used in this work, the standard deviations and the means of the membership functions of the inputs are defined, respectively, as $\sigma_j$ and $m_j$, where $j = 1, \ldots, H$. We define the set of the parameters of the membership functions of each input, $\Omega(s_i)$, as follows,

$$\begin{gathered}
\Omega(s_1) = \{(\sigma_1, m_1), (\sigma_2, m_2), \ldots, (\sigma_h, m_h)\} \\
\Omega(s_2) = \{(\sigma_{h+1}, m_{h+1}), (\sigma_{h+2}, m_{h+2}), \ldots, (\sigma_{2h}, m_{2h})\} \\
\vdots \\
\Omega(s_n) = \{(\sigma_{(n-1)h+1}, m_{(n-1)h+1}), (\sigma_{(n-1)h+2}, m_{(n-1)h+2}), \ldots, (\sigma_H, m_H)\}
\end{gathered} \tag{13}$$

The output of the fuzzy system is given by the following equation when we use the product inference engine with singleton fuzzifier and center-average defuzzifier [10].

$$Z(\mathbf{s}_t) = \frac{\sum_{l=1}^{L}\left[\left(\prod_{i=1}^{n}\mu^{F_i^l}(s_i)\right)k_l\right]}{\sum_{l=1}^{L}\left(\prod_{i=1}^{n}\mu^{F_i^l}(s_i)\right)} = \sum_{l=1}^{L}\Phi_l(\mathbf{s}_t)k_l \tag{14}$$

where $\mathbf{s}_t = (s_1, \ldots, s_n)$ is the state vector, $\mu^{F_i^l}$ describes the membership value of the input state variable $s_i$ in the rule $l$, and $\Phi_l(\mathbf{s}_t)$ is the normalized activation degree (normalized firing strength) of the rule $l$ at the state $\mathbf{s}_t$ and is defined as follows:

$$\Phi_l(\mathbf{s}_t) = \frac{\prod_{i=1}^{n}\mu^{F_i^l}(s_i)}{\sum_{l=1}^{L}\left(\prod_{i=1}^{n}\mu^{F_i^l}(s_i)\right)} = \frac{\omega_l(\mathbf{s}_t)}{\sum_{l=1}^{L}\omega_l(\mathbf{s}_t)} \tag{15}$$

where $\omega_l(\mathbf{s}_t)$ is the firing strength of the rule $l$ at the state $\mathbf{s}_t$, and it is defined as follows,

$$\omega_l(\mathbf{s}_t) = \prod_{i=1}^{n}\mu^{F_i^l}(s_i) \tag{16}$$

We define the set of the parameters of each firing strength of each rule in each FIS, $\Omega(\omega_l)$, as follows,

$$\begin{gathered}
\Omega(\omega_1) = \{(\sigma_1, m_1), (\sigma_{h+1}, m_{h+1}), \ldots, (\sigma_{(n-1)h+1}, m_{(n-1)h+1})\} \\
\Omega(\omega_2) = \{(\sigma_1, m_1), (\sigma_{h+1}, m_{h+1}), \ldots, (\sigma_{(n-1)h+2}, m_{(n-1)h+2})\} \\
\vdots \\
\Omega(\omega_h) = \{(\sigma_1, m_1), (\sigma_{h+1}, m_{h+1}), \ldots, (\sigma_H, m_H)\} \\
\Omega(\omega_{h+1}) = \{(\sigma_1, m_1), (\sigma_{h+2}, m_{h+2}), \ldots, (\sigma_{(n-1)h+1}, m_{(n-1)h+1})\} \\
\vdots \\
\Omega(\omega_L) = \{(\sigma_h, m_h), (\sigma_{2h}, m_{2h}), \ldots, (\sigma_H, m_H)\}
\end{gathered} \tag{17}$$

## 3 The Q-Learning Fuzzy Inference System Algorithm

The QLFIS algorithm is a modified version of the algorithms presented in [46] and [26]. The QLFIS algorithm combines the Q-learning algorithm with a function approximation system and is applied directly to games with continuous state and action spaces. The structure of the QLFIS algorithm is shown in Fig. 1. The QLFIS algorithm uses a function approximation system (FIS) to estimate the state-action value functions $Q(\mathbf{s}_t, u)$. The QLFIS algorithm also uses a function approximation system (FLC) to generate the continuous action. The QLFIS algorithm tunes the input and output parameters of its function approximation systems [45]. The QLFIS algorithm uses the so-called direct algorithms described in [48] as a mechanism to tune the input and output parameters of its function approximation systems.

### 3.1 Update Rules for the Function Approximation System (FIS)

The input parameters of the FIS are the parameters of the Gaussian membership functions of its input: the standard deviations $\sigma_j$ and the means $m_j$. On the other hand, the output parameters of the FIS are the consequent (or conclusion) parts of the fuzzy rules, $k_l$. To simplify notations, we refer to the input and output parameters of the FIS as $\psi^Q$. Thus, the update rules of the input and output parameters for the FIS of the QLFIS algorithm are given as follows [45],

$$\psi^Q_{t+1} = \psi^Q_t + \rho\Delta_t \frac{\partial Q_t(\mathbf{s}_t, u_c)}{\partial \psi^Q_t} \tag{18}$$

where $\rho$ is a learning rate for the FIS parameters, and $\Delta_t$ is the temporal difference error that is defined as follows,

$$\Delta_t = r_t + \gamma Q_t(\mathbf{s}_{t+1}, u_c) - Q_t(\mathbf{s}_t, u_c) \tag{19}$$

where $r_t$ is the reward received at time $t$, $\gamma$ is a discount factor, and $Q_t(\mathbf{s}_t, u_c)$ is the estimated state-action value function at the state $\mathbf{s}_t$.

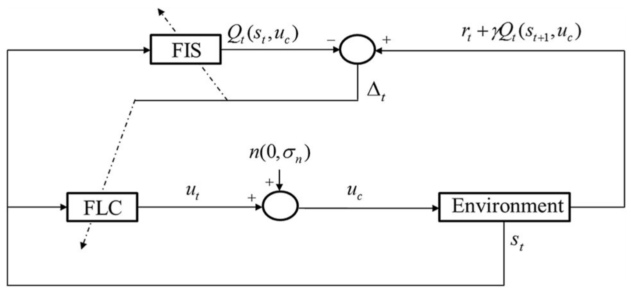


**Fig. 1** The QLFIS technique [45]

The term $\frac{\partial Q_t(\mathbf{s}_t,u_c)}{\partial \psi^Q_t}$ in Eq. (18) is computed as in [45] as follows,

$$\frac{\partial Q_t(\mathbf{s}_t, u_c)}{\partial k_l} = \Phi_l(\mathbf{s}_t) \tag{20}$$

$$\frac{\partial Q_t(\mathbf{s}_t, u_c)}{\partial \sigma_j} = \frac{k_l - Q_t(\mathbf{s}_t, u_c)}{\sum_l \omega_l(\mathbf{s}_t)} \quad \omega_l(\mathbf{s}_t) \quad \frac{2(s_i - m_j)^2}{(\sigma_j)^3} \tag{21}$$

$$\frac{\partial Q_t(\mathbf{s}_t, u_c)}{\partial m_j} = \frac{k_l - Q_t(\mathbf{s}_t, u_c)}{\sum_l \omega_l(\mathbf{s}_t)} \quad \omega_l(\mathbf{s}_t) \quad \frac{2(s_i - m_j)}{(\sigma_j)^2} \tag{22}$$

### 3.2 Update Rules for the Function Approximation System (FLC)

Likewise the FIS, the input parameters of the FLC are the parameters of the Gaussian membership functions of its input: the standard deviations $\sigma_j$ and the means $m_j$. On the other hand, the output parameters of the FLC are the consequent (or conclusion) parts of the fuzzy rules, $k_l$. We refer to the input and output parameters of the FLC as $\psi^u$. Thus, the update rules of the input and output parameters for the FLC of the QLFIS algorithm are given as follows [45],

$$\psi^u_{t+1} = \psi^u_t + \tau\Delta_t \frac{\partial u_t}{\partial \psi_{u_t}} \left[\frac{u_c - u_t}{\sigma_n}\right] \tag{23}$$

where $\tau$ is a learning rate for the FLC parameters, $u_c$ is the output of the FLC with a random Gaussian noise. The term $\frac{\partial u_t}{\partial \psi^u_t}$ in Eq. (23) can be calculated by replacing $Q_t(\mathbf{s}_t, u_c)$ with $u_t$ in Eqs. (20), (21) and (22) as follows,

$$\frac{\partial u_t}{\partial k_l} = \Phi_l(\mathbf{s}_t) \tag{24}$$

$$\frac{\partial u_t}{\partial \sigma_j} = \frac{k_l - u_t}{\sum_l \omega_l(\mathbf{s}_t)} \quad \omega_l(\mathbf{s}_t) \quad \frac{2(s_i - m_j)^2}{(\sigma_j)^3} \tag{25}$$

$$\frac{\partial u_t}{\partial m_j} = \frac{k_l - u_t}{\sum_l \omega_l(\mathbf{s}_t)} \quad \omega_l(\mathbf{s}_t) \quad \frac{2(s_i - m_j)}{(\sigma_j)^2} \tag{26}$$

## 4 The Pursuit–Evasion Game

The pursuit–evasion game is defined as a differential game [58]. In this game, the pursuer's objective is to capture the evader, whereas the evader's objective is to escape from the pursuer or at least prolong the capture time. Figure 2 shows the model of the pursuit–evasion differential game. The equations of motion of the pursuer and the evader robots can be described by the following equations [59, 60],

$$\begin{aligned} \dot{x}_\kappa &= V_\kappa \cos(\theta_\kappa) \\ \dot{y}_\kappa &= V_\kappa \sin(\theta_\kappa) \\ \dot{\theta}_\kappa &= \frac{V_\kappa}{L_\kappa} \tan(u_\kappa) \end{aligned} \tag{27}$$

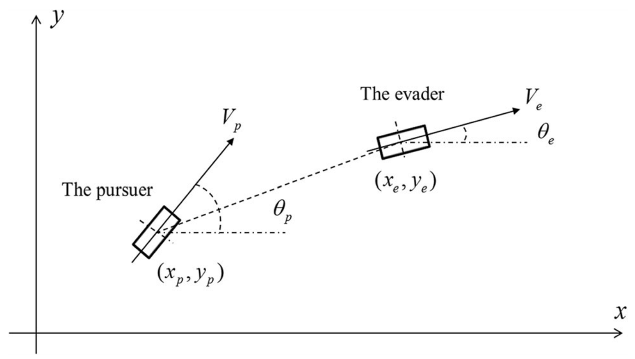


**Fig. 2** Pursuit–evasion model

where $\kappa$ represents both the pursuer "$p$" and the evader "$e$", $V_\kappa$ represents robot $\kappa$'s speed, $\theta_\kappa$ is the orientation of robot $\kappa$, $(x_\kappa, y_\kappa)$ is the position of robot $\kappa$, $L_\kappa$ represents the wheelbase of robot $\kappa$, and $u_\kappa$ represents robot $\kappa$'s steering angle, where $u_\kappa \in [-u_{\kappa_{\max}}, u_{\kappa_{\max}}]$.

In this work, we assume that the pursuer is faster than the evader by making $V_p > V_e$. It is also assumed that the evader is more maneuverable than the pursuer by making $u_{e_{\max}} > u_{p_{\max}}$. A simple classical control strategy that can be used to define the control strategies of the pursuer and the evader, in a pursuit–evasion game, can be given as follows,

$$u_\kappa = \begin{cases} -u_{\kappa_{\max}}: & \delta_\kappa - u_{\kappa_{\max}} \\ \delta_\kappa: & -u_{\kappa_{\max}} \leq \delta_\kappa \leq u_{\kappa_{\max}} \\ u_{\kappa_{\max}}: & \delta_\kappa u_{\kappa_{\max}} \end{cases} \tag{28}$$

and,

$$\delta_\kappa = \tan^{-1}\left(\frac{y_e - y_p}{x_e - x_p}\right) - \theta_\kappa \tag{29}$$

where $\delta_\kappa$ represents the angle difference between the direction of robot $\kappa$ and the line-of-sight (LoS) to the other robot.

To capture the evader in a pursuit–evasion game when the pursuer uses the simple control strategy described by Eqs. (28) and (29), the angle difference $\delta_p$ has to be driven to zero by the pursuer. Thus, the control strategy of the pursuer in this case is to drive this angle difference to zero. On the other hand, the control strategy of the evader is to escape from the pursuer and keep the distance between the evader and the pursuer as large as possible. The evader can do so by following the intelligent control strategy described by the following two rules [11, 45, 60, 61]:

1. If the distance between the evader and the pursuer is greater than a specific distance $\bar{d}$, the control strategy of the evader is defined as follows,

$$u_e = \tan^{-1}\left(\frac{y_e - y_p}{x_e - x_p}\right) - \theta_e \tag{30}$$

2. If the distance between the evader and the pursuer is less than or equal to the distance $\bar{d}$, the evader exploits its higher maneuverability, and the control strategy for the evader in this case is given as follows,

$$u_e = (\theta_p + \pi) - \theta_e \tag{31}$$

The distance $\bar{d}$ is defined as follows,

$$\bar{d} = \frac{L_p}{\tan(u_{p_{\max}})} \tag{32}$$

The pursuer succeeds to capture the evader if the distance between them is less than the capture radius $d_c$. The distance between the pursuer and the evader is defined by $d$ and is given as follows,

$$d = \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} \tag{33}$$

## 5 The Proposed Algorithm

In this work, we propose a new fuzzy reinforcement learning algorithm for differential games that have continuous state and action spaces. The proposed algorithm FISs as function approximation systems: an actor (fuzzy logic controller, FLC) and a critic. The critic is used to estimate the value functions $V_t(\mathbf{s}_t)$ and $V_t(\mathbf{s}_{t+1})$ of the learning agent at two different states $\mathbf{s}_t$ and $\mathbf{s}_{t+1}$, respectively. The values of $V_t(\mathbf{s}_t)$ and $V_t(\mathbf{s}_{t+1})$ will depend on the input and output parameters of the critic. Unlike the algorithms proposed in [16, 26, 43, 45–47] which use the direct algorithms to tune the parameters of their function approximation systems, the proposed algorithm uses the residual gradient value iteration algorithm described in [48] to tune the input and output parameters of its function approximation systems. It has been shown in [48–52] that the direct algorithms may not converge to an answer in some cases, while the residual gradient algorithms are always guaranteed to converge. The proposed algorithm is called the RGFACL algorithm. The structure of the proposed RGFACL algorithm is shown in Fig. 3. The input parameters of the critic are the parameters of the MFs of its inputs, $\sigma_j$ and $m_j$ (where $j = 1, \ldots, H$). The output parameters of the critic are the consequent parameters of its rules, $k_l$ (where $l = 1, \ldots, L$). To simplify notations, we refer to the input and output parameters of the critic as $\psi^C$. Similarly, the input parameters of the actor are the parameters of the MFs of its input $\sigma_j$ and $m_j$, and the

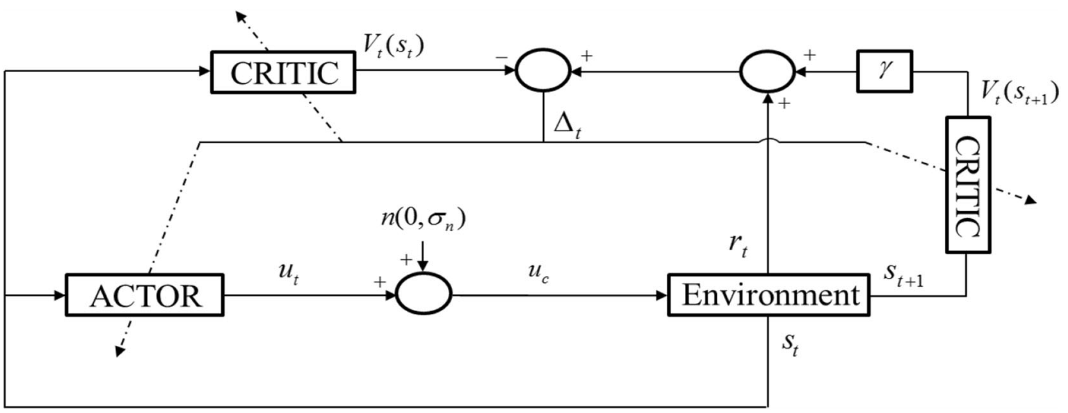


**Fig. 3** The proposed RGFACL algorithm

output parameters of the actor are the consequent parameters of its rules $k_l$. To simplify notations, we refer to the input and output parameters of the actor as $\psi^A$. The temporal difference residual error, $\Delta_t$, is defined as follows,

$$\Delta_t = r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t) \tag{34}$$

## 5.1 Adaptation Rules for the Critic

In this subsection, we derive the adaptation rules that the proposed algorithm uses to tune the input and output parameters of the critic. For the sake of completeness, we are going to present the complete derivation of the partial derivatives that are needed by the proposed algorithm.

The mean squared error, $E$, of the temporal difference residual error, $\Delta_t$, is defined as follows,

$$E = \frac{1}{2}\Delta_t^2 \tag{35}$$

The input and output parameters of the critic are updated based on the residual gradient method described in [48] as follows,

$$\psi_{t+1}^C = \psi_t^C - \alpha \frac{\partial E}{\partial \psi_t^C} \tag{36}$$

where $\psi_t^C$ represents the input and output parameters of the critic at time $t$, and $\alpha$ is a learning rate for the parameters of the critic.

The QLFIS algorithm proposed in [45] uses the so-called direct algorithms described in [48] to define the term $\frac{\partial E}{\partial \psi_t^C}$. On the other hand, the proposed algorithm defines the term $\frac{\partial E}{\partial \psi_t^C}$ based on the residual gradient value iteration algorithm which is also described in [48] as follows,

$$\frac{\partial E}{\partial \psi_t^C} = \Delta_t \left[ \gamma \frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \psi_t^C} - \frac{\partial V_t(\mathbf{s}_t)}{\partial \psi_t^C} \right] \tag{37}$$

The proposed algorithm treats the value function $V_t(\mathbf{s}_{t+1})$ in the temporal difference residual error $\Delta_t$ as a function of the input and output parameters of its function approximation system (the critic), $\psi_t^C$. Unlike the QLFIS algorithm, the proposed algorithm will not assign a value of zero to the derivative $\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \psi_t^C}$ during the tuning of the input and output parameters of the critic. This is because the value function $V_t(\mathbf{s}_{t+1})$ is a function of the input and output parameters of the critic $\psi_t^C$, and its derivative $\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \psi_t^C}$ should not be assigned to zero all the time.

From Eq. (37), Eq. (36) can be rewritten as follows,

$$\psi_{t+1}^C = \psi_t^C - \alpha[r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t)].\ \left[\gamma \frac{\partial}{\partial \psi_t^C} V_t(\mathbf{s}_{t+1}) - \frac{\partial}{\partial \psi_t^C} V_t(\mathbf{s}_t)\right] \tag{38}$$

The derivatives of the state value functions, $V_t(\mathbf{s}_t)$ and $V_t(\mathbf{s}_{t+1})$, with respect to the output parameters of the critic, $k_l$, are calculated from Eq. (14) as follows,

$$\frac{\partial V_t(\mathbf{s}_t)}{\partial k_l} = \Phi_l(\mathbf{s}_t) \tag{39}$$

where $l = 1, \ldots, L$.

Similarly,

$$\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial k_l} = \Phi_l(\mathbf{s}_{t+1}) \tag{40}$$

where $\Phi_l(\mathbf{s}_t)$ and $\Phi_l(\mathbf{s}_{t+1})$ are calculated by using Eq. (15) at the states $\mathbf{s}_t$ and $\mathbf{s}_{t+1}$, respectively.

We use the chain rule to calculate the derivatives $\frac{\partial V_t(\mathbf{s}_t)}{\partial \sigma_j}$ and $\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \sigma_j}$. We start with the derivative $\frac{\partial V_t(\mathbf{s}_t)}{\partial \sigma_j}$ which is calculated as follows,

$$\frac{\partial V_t(\mathbf{s}_t)}{\partial \sigma_j} = \frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_l(\mathbf{s}_t)} \cdot \frac{\partial \omega_l(\mathbf{s}_t)}{\partial \sigma_j} \tag{41}$$

where $j = 1, \ldots, H$.

The term $\frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_l(\mathbf{s}_t)}$ is calculated as follows,

$$\frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_l(\mathbf{s}_t)} = \left[\frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_1(\mathbf{s}_t)} \quad \frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_2(\mathbf{s}_t)} \cdots \frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_L(\mathbf{s}_t)}\right] \tag{42}$$

We first calculate the term $\frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_1(\mathbf{s}_t)}$ as follows,

$$\frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_1(\mathbf{s}_t)} = \frac{\partial}{\partial \omega_1(\mathbf{s}_t)}\left[\frac{\sum_l \omega_l(\mathbf{s}_t) k_l}{\sum_l \omega_l(\mathbf{s}_t)}\right] = \frac{k_1 - V_t(\mathbf{s}_t)}{\sum_l \omega_l(\mathbf{s}_t)} \tag{43}$$

Similarly, we can calculate the terms $\frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_2(\mathbf{s}_t)}$, ..., and $\frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_L(\mathbf{s}_t)}$. Thus, Eq. (42) can be rewritten as follows,

$$\frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_l(\mathbf{s}_t)} = \left[\frac{k_1 - V_t(\mathbf{s}_t)}{\sum_l \omega_l(\mathbf{s}_t)} \quad \frac{k_2 - V_t(\mathbf{s}_t)}{\sum_l \omega_l(\mathbf{s}_t)} \cdots \frac{k_L - V_t(\mathbf{s}_t)}{\sum_l \omega_l(\mathbf{s}_t)}\right] \tag{44}$$

On the other hand, the term $\frac{\partial \omega_l(\mathbf{s}_t)}{\partial \sigma_j}$ in Eq. (41) is calculated as follows,

$$\frac{\partial \omega_l(\mathbf{s}_t)}{\partial \sigma_j} = \left[\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_j} \quad \frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_j} \cdots \frac{\partial \omega_L(\mathbf{s}_t)}{\partial \sigma_j}\right]^T \tag{45}$$

The derivative $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_j}$ can be calculated based on the definition of $\omega_1(\mathbf{s}_t)$ given in Eq. (16) as follows,

$$\begin{aligned}\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_j} &= \frac{\partial}{\partial \sigma_j}\left[\prod_{i=1}^{n} \mu^{F_i^1}(s_i)\right] \\ &= \frac{\partial}{\partial \sigma_j}\left[\mu^{F_1^1}(s_1) \times \mu^{F_2^1}(s_2) \times \cdots \times \mu^{F_n^1}(s_n)\right]\end{aligned} \tag{46}$$

We then substitute Eq. (12) into Eq. (46) as follows,

$$\begin{aligned}\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_j} = \frac{\partial}{\partial \sigma_j}\Bigg[ &\exp\left(-\left(\frac{s_1 - m_1}{\sigma_1}\right)^2\right) \\ &\times \exp\left(-\left(\frac{s_2 - m_{h+1}}{\sigma_{h+1}}\right)^2\right) \times \cdots \\ &\times \exp\left(-\left(\frac{s_n - m_{(n-1)h+1}}{\sigma_{(n-1)h+1}}\right)^2\right)\Bigg]\end{aligned} \tag{47}$$

Thus, the derivatives $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_1}$, $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_2}$, ..., and $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_H}$ are calculated as follows,

$$\begin{aligned}&\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_1} = \frac{2(s_1 - m_1)^2}{\sigma_1^3}\omega_1(\mathbf{s}_t) \\ &\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_2} = 0 \\ &\quad\vdots \\ &\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_{h+1}} = \frac{2(s_2 - m_{h+1})^2}{\sigma_{h+1}^3}\omega_1(\mathbf{s}_t) \\ &\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_{h+2}} = 0 \\ &\quad\vdots \\ &\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_{(n-1)h+1}} = \frac{2(s_n - m_{(n-1)h+1})^2}{\sigma_{(n-1)h+1}^3}\omega_1(\mathbf{s}_t) \\ &\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_{(n-1)h+2}} = 0 \\ &\quad\vdots \\ &\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_H} = 0\end{aligned} \tag{48}$$

Thus, from Eq. (48), the derivative $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_j}$ $(j = 1, \ldots, H)$ can be rewritten as follows,

$$\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_j} = \begin{cases}\frac{2(s_i - m_j)^2}{\sigma_j^3}\omega_1(\mathbf{s}_t) & \text{if} \quad (\sigma_j, m_j) \in \Omega(\omega_1) \\ 0 & \text{if} \quad (\sigma_j, m_j) \notin \Omega(\omega_1)\end{cases} \tag{49}$$

where the term $s_i$ is the $i$th input state variable of the state vector $\mathbf{s}_t$ and is defined as follows,

$$s_i = \begin{cases} s_1 & \text{if} \quad (\sigma_j, m_j) \in \Omega(s_1) \\ s_2 & \text{if} \quad (\sigma_j, m_j) \in \Omega(s_2) \\ \quad\vdots \\ s_n & \text{if} \quad (\sigma_j, m_j) \in \Omega(s_n)\end{cases} \tag{50}$$

We can rewrite Eq. (49) as follows,

$$\frac{\partial \omega_1(\mathbf{s}_t)}{\partial \sigma_j} = \xi_{j,1}\frac{2(s_i - m_j)^2}{\sigma_j^3}\omega_1(\mathbf{s}_t) \tag{51}$$

where

$$\xi_{j,1} = \begin{cases}1 & \text{if} \quad (\sigma_j, m_j) \in \Omega(\omega_1) \\ 0 & \text{if} \quad (\sigma_j, m_j) \notin \Omega(\omega_1)\end{cases} \tag{52}$$

Similarly, we can calculate the derivatives $\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_j}$ as follows,

$$\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_j} = \frac{\partial}{\partial \sigma_j}\left[\prod_{i=1}^{n} \mu^{F_i^2}(s_i)\right] = \frac{\partial}{\partial \sigma_j}\left[\mu^{F_1^2}(s_1) \times \mu^{F_2^2}(s_2) \times \cdots \times \mu^{F_n^2}(s_n)\right] \tag{53}$$

$$\begin{aligned}\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_j} = \frac{\partial}{\partial \sigma_j}\Bigg[ & \exp\left(-\left(\frac{s_1 - m_1}{\sigma_1}\right)^2\right) \\ & \times \exp\left(-\left(\frac{s_2 - m_{h+1}}{\sigma_{h+1}}\right)^2\right) \times \cdots \\ & \times \exp\left(-\left(\frac{s_n - m_{(n-1)h+2}}{\sigma_{(n-1)h+2}}\right)^2\right)\Bigg]\end{aligned} \tag{54}$$

Thus, the derivatives $\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_1}$, ..., and $\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_H}$ are calculated as follows,

$$\begin{aligned}
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_1} &= \frac{2(s_1 - m_1)^2}{\sigma_1^3}\omega_2(\mathbf{s}_t) \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_2} &= 0 \\
&\vdots \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_{h+1}} &= \frac{2(s_2 - m_{h+1})^2}{\sigma_{h+1}^3}\omega_2(\mathbf{s}_t) \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_{h+2}} &= 0 \\
&\vdots \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_{(n-1)h+1}} &= 0 \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_{(n-1)h+2}} &= \frac{2(s_n - m_{(n-1)h+2})^2}{\sigma_{(n-1)h+2}^3}\omega_2(\mathbf{s}_t) \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_{(n-1)h+3}} &= 0 \\
&\vdots \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_H} &= 0
\end{aligned} \tag{55}$$

Thus, from Eq. (55), the derivatives $\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_j}$ $(j = 1, \ldots, H)$ can be rewritten as follows,

$$\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_j} = \begin{cases} \frac{2(s_i - m_j)^2}{\sigma_j^3}\omega_2(\mathbf{s}_t) & \text{if} \quad (\sigma_j, m_j) \in \Omega(\omega_2) \\ 0 & \text{if} \quad (\sigma_j, m_j) \notin \Omega(\omega_2) \end{cases} \tag{56}$$

We can rewrite Eq. (56) as follows,

$$\frac{\partial \omega_2(\mathbf{s}_t)}{\partial \sigma_j} = \xi_{j,2}\frac{2(s_i - m_j)^2}{\sigma_j^3}\omega_2(\mathbf{s}_t) \tag{57}$$

where

$$\xi_{j,2} = \begin{cases} 1 & \text{if} \quad (\sigma_j, m_j) \in \Omega(\omega_2) \\ 0 & \text{if} \quad (\sigma_j, m_j) \notin \Omega(\omega_2) \end{cases} \tag{58}$$

Similarly, we can calculate the derivatives $\frac{\partial \omega_3(\mathbf{s}_t)}{\partial \sigma_j}$, $\frac{\partial \omega_4(\mathbf{s}_t)}{\partial \sigma_j}$, ..., and $\frac{\partial \omega_L(\mathbf{s}_t)}{\partial \sigma_j}$. Thus, Eq. (45) can be rewritten as follows,

$$\frac{\partial \omega_l(\mathbf{s}_t)}{\partial \sigma_j} = \left[\xi_{j,1}\frac{2(s_i - m_j)^2}{\sigma_j^3}\omega_1(\mathbf{s}_t) \quad \xi_{j,2}\frac{2(s_i - m_j)^2}{\sigma_j^3}\omega_2(\mathbf{s}_t) \quad \ldots \xi_{j,L}\frac{2(s_i - m_j)^2}{\sigma_j^3}\omega_L(\mathbf{s}_t)\right]^T \tag{59}$$

where

$$\xi_{j,l} = \begin{cases} 1 & \text{if} \quad (\sigma_j, m_j) \in \Omega(\omega_l) \\ 0 & \text{if} \quad (\sigma_j, m_j) \notin \Omega(\omega_l) \end{cases} \tag{60}$$

Hence, from Eqs. (44) and (59), the derivative $\frac{\partial V_t(\mathbf{s}_t)}{\partial \sigma_j}$ $(j = 1, \ldots, H)$ in Eq. (41) is then calculated as follows,

$$\frac{\partial V_t(\mathbf{s}_t)}{\partial \sigma_j} = \frac{2(s_i - m_j)^2}{\sigma_j^3} \times \sum_{l=1}^{L} \xi_{j,l}\frac{k_l - V_t(\mathbf{s}_t)}{\sum_l \omega_l(\mathbf{s}_t)}\omega_l(\mathbf{s}_t) \tag{61}$$

Similarly, we calculate the derivative $\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \sigma_j}$ as follows,

$$\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \sigma_j} = \frac{2(s'_i - m_j)^2}{\sigma_j^3} \times \sum_{l=1}^{L} \xi_{j,l}\frac{k_l - V_t(\mathbf{s}_{t+1})}{\sum_l \omega_l(\mathbf{s}_{t+1})}\omega_l(\mathbf{s}_{t+1}) \tag{62}$$

where

$$s'_i = \begin{cases} s'_1 & \text{if} \quad (\sigma_j, m_j) \in \Omega(s_1) \\ s'_2 & \text{if} \quad (\sigma_j, m_j) \in \Omega(s_2) \\ \vdots \\ s'_n & \text{if} \quad (\sigma_j, m_j) \in \Omega(s_n) \end{cases} \tag{63}$$

where $s'_i$ is the $i$th input state variable of the state vector $\mathbf{s}_{t+1}$.

We also use the chain rule to calculate the derivatives $\frac{\partial V_t(\mathbf{s}_t)}{\partial m_j}$ and $\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial m_j}$. We start with the derivative $\frac{\partial V_t(\mathbf{s}_t)}{\partial m_j}$ which is calculated as follows,

$$\frac{\partial V_t(\mathbf{s}_t)}{\partial m_j} = \frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_l(\mathbf{s}_t)} \cdot \frac{\partial \omega_l(\mathbf{s}_t)}{\partial m_j} \tag{64}$$

The term $\frac{\partial V_t(\mathbf{s}_t)}{\partial \omega_l(\mathbf{s}_t)}$ is calculated as in Eq. (44), and the term $\frac{\partial \omega_l(\mathbf{s}_t)}{\partial m_j}$ is calculated as follows,

$$\frac{\partial \omega_l(\mathbf{s}_t)}{\partial m_j} = \left[\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_j} \quad \frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_j} \cdots \frac{\partial \omega_L(\mathbf{s}_t)}{\partial m_j}\right]^T \tag{65}$$

We first calculate the term $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_j}$ by using Eq. (16) as follows,

$$\begin{aligned}\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_j} &= \frac{\partial}{\partial m_j}\left[\prod_{i=1}^{n} \mu^{F_i^1}(s_i)\right] \\ &= \frac{\partial}{\partial m_j}\left[\mu^{F_1^1}(s_1) \times \mu^{F_2^1}(s_2) \times \cdots \times \mu^{F_n^1}(s_n)\right]\end{aligned} \tag{66}$$

We then substitute Eq. (12) into Eq. (66) as follows,

$$\begin{aligned}\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_j} = \frac{\partial}{\partial m_j}\Bigg[ &\exp\left(-\left(\frac{s_1 - m_1}{\sigma_1}\right)^2\right) \\ &\times \exp\left(-\left(\frac{s_2 - m_{h+1}}{\sigma_{h+1}}\right)^2\right) \times \cdots \\ &\times \exp\left(-\left(\frac{s_n - m_{(n-1)h+1}}{\sigma_{(n-1)h+1}}\right)^2\right)\Bigg]\end{aligned} \tag{67}$$

Thus, the derivatives $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_1}$, $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_2}$, ..., and $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_H}$ are calculated as follows,

$$\begin{aligned}
\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_1} &= \frac{2(s_1 - m_1)}{\sigma_1^2}\omega_1(\mathbf{s}_t) \\
\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_2} &= 0 \\
&\vdots \\
\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_{h+1}} &= \frac{2(s_2 - m_{h+1})}{\sigma_{h+1}^2}\omega_1(\mathbf{s}_t) \\
\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_{h+2}} &= 0 \\
&\vdots \\
\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_{(n-1)h+1}} &= \frac{2(s_n - m_{(n-1)h+1})}{\sigma_{(n-1)h+1}^2}\omega_1(\mathbf{s}_t) \\
\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_{(n-1)h+2}} &= 0 \\
&\vdots \\
\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_H} &= 0
\end{aligned} \tag{68}$$

Thus, from Eq. (68), the derivative $\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_j}$ $(j = 1, ..., H)$ can be rewritten as follows,

$$\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_j} = \begin{cases} \frac{2(s_i - m_j)}{\sigma_j^2}\omega_1(\mathbf{s}_t) & \text{if } (\sigma_j, m_j) \in \Omega(\omega_1) \\ 0 & \text{if } (\sigma_j, m_j) \notin \Omega(\omega_1) \end{cases} \tag{69}$$

We can rewrite Eq. (69) as follows,

$$\frac{\partial \omega_1(\mathbf{s}_t)}{\partial m_j} = \xi_{j,1}\frac{2(s_i - m_j)}{\sigma_j^2}\omega_1(\mathbf{s}_t) \tag{70}$$

where $s_i$ is defined as in Eq. (50), and $\xi_{j,1}$ is defined as in Eq. (52).

Similarly, we can calculate the term $\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_j}$ as follows,

$$\begin{aligned}\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_j} &= \frac{\partial}{\partial m_j}\left[\prod_{i=1}^{n} \mu^{F_i^2}(s_i)\right] \\ &= \frac{\partial}{\partial m_j}\left[\mu^{F_1^2}(s_1) \times \mu^{F_2^2}(s_2) \times \cdots \times \mu^{F_n^2}(s_n)\right]\end{aligned} \tag{71}$$

$$\begin{aligned}\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_j} = \frac{\partial}{\partial m_j}\Bigg[ &\exp\left(-\left(\frac{s_1 - m_1}{\sigma_1}\right)^2\right) \\ &\times \exp\left(-\left(\frac{s_2 - m_{h+1}}{\sigma_{h+1}}\right)^2\right) \times \cdots \\ &\times \exp\left(-\left(\frac{s_n - m_{(n-1)h+2}}{\sigma_{(n-1)h+2}}\right)^2\right)\Bigg]\end{aligned} \tag{72}$$

Thus, the derivatives $\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_1}$, $\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_2}$, ..., and $\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_H}$ are calculated as follows,

$$\begin{aligned}
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_1} &= \frac{2(s_1 - m_1)}{\sigma_1^2}\omega_2(\mathbf{s}_t) \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_2} &= 0 \\
&\vdots \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_{h+1}} &= \frac{2(s_2 - m_{h+1})}{\sigma_{h+1}^2}\omega_2(\mathbf{s}_t) \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_{h+2}} &= 0 \\
&\vdots \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_{(n-1)h+1}} &= 0 \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_{(n-1)h+2}} &= \frac{2(s_n - m_{(n-1)h+2})}{\sigma_{(n-1)h+2}^2}\omega_2(\mathbf{s}_t) \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_{(n-1)h+3}} &= 0 \\
&\vdots \\
\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_H} &= 0
\end{aligned} \tag{73}$$

Thus, from Eq. (73), the derivative $\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_j}$ $(j = 1, \ldots, H)$ can be rewritten as follows,

$$\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_j} = \begin{cases} \frac{2(s_i - m_j)}{\sigma_j^2}\omega_2(\mathbf{s}_t) & \text{if } (\sigma_j, m_j) \in \Omega(\omega_2) \\ 0 & \text{if } (\sigma_j, m_j) \notin \Omega(\omega_2) \end{cases} \tag{74}$$

We can rewrite Eq. (74) as follows,

$$\frac{\partial \omega_2(\mathbf{s}_t)}{\partial m_j} = \xi_{j,2}\frac{2(s_i - m_j)}{\sigma_j^2}\omega_2(\mathbf{s}_t) \tag{75}$$

where $s_i$ is defined as in Eq. (50), and $\xi_{j,2}$ is defined as in Eq. (58).

Similarly, we can calculate the terms $\frac{\partial \omega_3(\mathbf{s}_t)}{\partial m_j}$, $\frac{\partial \omega_4(\mathbf{s}_t)}{\partial m_j}$, ..., and $\frac{\partial \omega_L(\mathbf{s}_t)}{\partial m_j}$. Thus, Eq. (65) can be rewritten as follows,

$$\frac{\partial \omega_l(\mathbf{s}_t)}{\partial m_j} = \left[\xi_{j,1}\frac{2(s_i - m_j)}{\sigma_j^2}\omega_1(\mathbf{s}_t) \quad \xi_{j,2}\frac{2(s_i - m_j)}{\sigma_j^2}\omega_2(\mathbf{s}_t) \cdots \xi_{j,L}\frac{2(s_i - m_j)}{\sigma_j^2}\omega_L(\mathbf{s}_t)\right]^T \tag{76}$$

where $\xi_{j,l}$ is defined as in Eq. (60).

From Eqs. (44) and (76), the derivative $\frac{\partial V_t(\mathbf{s}_t)}{\partial m_j}$ in Eq. (64) is calculated as follows,

$$\frac{\partial V_t(\mathbf{s}_t)}{\partial m_j} = \frac{2(s_i - m_j)}{\sigma_j^2} \times \sum_{l=1}^{L} \xi_{j,l}\frac{k_l - V_t(\mathbf{s}_t)}{\sum_l \omega_l(\mathbf{s}_t)}\omega_l(\mathbf{s}_t) \tag{77}$$

Similarly, we can calculate the derivative $\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial m_j}$ as follows,

$$\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial m_j} = \frac{2(s'_i - m_j)}{\sigma_j^2} \times \sum_{l=1}^{L} \xi_{j,l}\frac{k_l - V_t(\mathbf{s}_{t+1})}{\sum_l \omega_l(\mathbf{s}_{t+1})}\omega_l(\mathbf{s}_{t+1}) \tag{78}$$

Hence, From Eqs. (38), (61), (62), (77) and (78), the input parameters $m_j$ and $\sigma_j$ of the critic are updated at each time step as follows,

$$\sigma_{j,t+1} = \sigma_{j,t} - \alpha[r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t)].\left[\gamma\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial \sigma_{j,t}} - \frac{\partial V_t(\mathbf{s}_t)}{\partial \sigma_{j,t}}\right] \tag{79}$$

$$m_{j,t+1} = m_{j,t} - \alpha[r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t)].\left[\gamma\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial m_{j,t}} - \frac{\partial V_t(\mathbf{s}_t)}{\partial m_{j,t}}\right] \tag{80}$$

Similarly, From Eqs. (38), (39), (40), the output parameters $k_l$ of the critic are updated at each time step as follows,

$$k_{l,t+1} = k_{l,t} - \alpha[r_t + \gamma V_t(\mathbf{s}_{t+1}) - V_t(\mathbf{s}_t)].\left[\gamma\frac{\partial V_t(\mathbf{s}_{t+1})}{\partial k_{l,t}} - \frac{\partial V_t(\mathbf{s}_t)}{\partial k_{l,t}}\right] \tag{81}$$

## 5.2 Adaptation Rules for the Actor

The input and output parameters of the actor, $\psi^A$, are updated as follows, [45],

$$\psi^A_{t+1} = \psi^A_t + \beta\Delta_t\frac{\partial u_t}{\partial \psi^A_t}\left[\frac{u_c - u_t}{\sigma_n}\right] \tag{82}$$

where $\beta$ is a learning rate for the actor parameters, $u_c$ is the output of the actor with a random Gaussian noise. The derivatives of the output of the FLC (the actor), $u_t$, with respect to the input and output parameters of the FLC can be calculated by replacing $V_t(\mathbf{s}_t)$ with $u_t$ in Eq. (39), Eqs. (61) and (77) as follows,

$$\frac{\partial u_t}{\partial k_l} = \Phi_l(\mathbf{s}_t) \tag{83}$$

$$\frac{\partial u_t}{\partial \sigma_j} = \frac{2(s_i - m_j)^2}{\sigma_j^3} \times \sum_{l=1}^{L} \xi_{j,l}\frac{k_l - u_t}{\sum_l \omega_l(\mathbf{s}_t)}\omega_l(\mathbf{s}_t) \tag{84}$$

$$\frac{\partial u_t}{\partial m_j} = \frac{2(s_i - m_j)}{\sigma_j^2} \times \sum_{l=1}^{L} \xi_{j,l}\frac{k_l - u_t}{\sum_l \omega_l(\mathbf{s}_t)}\omega_l(\mathbf{s}_t) \tag{85}$$

Hence, From Eqs. (82), (84) and (85), the input parameters $\sigma_j$ and $m_j$ of the actor (FLC) are updated at each time step as follows,

$$\sigma_{j,t+1} = \sigma_{j,t} + \beta\Delta_t\frac{\partial u_t}{\partial \sigma_{j,t}}\left[\frac{u_c - u_t}{\sigma_n}\right] \tag{86}$$

$$m_{j,t+1} = m_{j,t} + \beta\Delta_t\frac{\partial u_t}{\partial m_{j,t}}\left[\frac{u_c - u_t}{\sigma_n}\right] \tag{87}$$

Similarly, From Eqs. (82) and (83), the output parameters $k_l$ of the actor (FLC) are updated at each time step as follows,

$$k_{l,t+1} = k_{l,t} + \beta\Delta_t\frac{\partial u_t}{\partial k_{l,t}}\left[\frac{u_c - u_t}{\sigma_n}\right] \tag{88}$$

The proposed RGFACL algorithm is given in Algorithm 1.

**Algorithm 1** The Proposed Residual Gradient Fuzzy Actor Critic Learning Algorithm:

```
(1) Initialize:
    (a) the input and output parameters of the critic, $\psi^C$.
    (b) the input and output parameters of the actor, $\psi^A$.
(2) For each EPISODE do:
(3) Update the learning rates $\alpha$ and $\beta$ of the critic and actor, respectively.
(4) Initialize the position of the pursuer at $(x_p, y_p) = 0$ and the position of the evader randomly at $(x_e, y_e)$, and then calculate the initial state $\mathbf{s}_t$.
(5) For each ITERATION do:
(6) Calculate the output of the actor, $u_t$, at the state $\mathbf{s}_t$ by using Eq. (14) and then calculate the output $u_c = u_t + n(0, \sigma_n)$.
(7) Calculate the output of the critic, $V_t(\mathbf{s}_t)$, at the state $\mathbf{s}_t$ by using Eq. (14).
(8) Perform the action $u_c$ and observe the next state $\mathbf{s}_{t+1}$ and the reward $r_t$.
(9) Calculate the output of the critic, $V_t(\mathbf{s}_{t+1})$, at the next state $\mathbf{s}_{t+1}$ by using Eq. (14).
(10) Calculate the temporal difference error, $\Delta_t$, by using Eq. (34).
(11) Update the input and output parameters of the critic, $\psi^C$, by using Eq. (79), Eq. (80) and Eq. (81).
(12) Update the input and output parameters of the actor, $\psi^A$, based on Eq. (86), Eq. (87) and Eq. (88).
(13) Set $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$.
(14) Check Termination Condition.
(15) end for loop (ITERATION).
(16) end for loop (EPISODE).
```

Example This example illustrates how to use the equations associated with the proposed algorithm to tune the input and output parameters of the FISs (actor and critic) of a learning agent. We assume that the actor of the learning agent has two inputs, $\mathbf{s}_t = (s_1, s_2)$. That is, $n = 2$. We assume that the output of the actor is a crisp output. The critic of the learning agent has the same inputs as the actor, $\mathbf{s}_t = (s_1, s_2)$, and with a crisp output. Thus, the fuzzy rules of the actor and the critic can be described as in Eq. (11), as follows,

$$R_l(\text{actor}):\ \text{IF } s_1 \text{ is } A_1^l \text{ and } s_2 \text{ is } A_2^l \text{ THEN } z_l = a_l \tag{89}$$

$$R_l(\text{critic}):\ \text{IF } s_1 \text{ is } C_1^l \text{ and } s_2 \text{ is } C_2^l \text{ THEN } z_l = c_l \tag{90}$$

The linguistic values of the actor's inputs, $A_1^l$ and $A_2^l$, are functions of the means and the standard deviations of the MFs of the actor's inputs, and $a_l$ is a constant that represents the consequent part of the actor's fuzzy rule $R_l$. On the other hand, the linguistic values of the critic's inputs, $C_1^l$ and $C_2^l$, are functions of the means and the standard deviations of the MFs of the critic's inputs, and $c_l$ represents the consequent part of the critic's fuzzy rule $R_l$.

We assume that each input of the two inputs to the FISs (actor and critic) has three Gaussian MFs, $h = 3$. That is, the input parameters of each FIS are $\sigma_j$ and $m_j$, where $j = 1, \ldots, H$ and $H = n \times h = 6$. In addition, each FIS has nine rules, $L = h^n = 9$. That is, the output parameters of each FIS are nine parameters ($a_l$ for the actor and $c_l$ for the critic), where $l = 1, \ldots, 9$. The parameters of the MFs of each input $\Omega(s_i)$, $(i = 1, 2)$, defined by Eq. (13) can be given as follows,

$$\begin{aligned}\Omega(s_1) &= \{(\sigma_1, m_1), (\sigma_2, m_2), (\sigma_3, m_3)\}\\ \Omega(s_2) &= \{(\sigma_4, m_4), (\sigma_5, m_5), (\sigma_6, m_6)\}\end{aligned} \tag{91}$$

On the other hand, the set of the parameters of each firing strength of each rule in each FIS, $\Omega(\omega_l)$, defined by Eq. (17) is given as follows,

$$\begin{aligned}\Omega(\omega_1) &= \{(\sigma_1, m_1), (\sigma_4, m_4)\}\\ \Omega(\omega_2) &= \{(\sigma_1, m_1), (\sigma_5, m_5)\}\\ \Omega(\omega_3) &= \{(\sigma_1, m_1), (\sigma_6, m_6)\}\\ \Omega(\omega_4) &= \{(\sigma_2, m_2), (\sigma_4, m_4)\}\\ \Omega(\omega_5) &= \{(\sigma_2, m_2), (\sigma_5, m_5)\}\\ \Omega(\omega_6) &= \{(\sigma_2, m_2), (\sigma_6, m_6)\}\\ \Omega(\omega_7) &= \{(\sigma_3, m_3), (\sigma_4, m_4)\}\\ \Omega(\omega_8) &= \{(\sigma_3, m_3), (\sigma_5, m_5)\}\\ \Omega(\omega_9) &= \{(\sigma_3, m_3), (\sigma_6, m_6)\}\end{aligned} \tag{92}$$

The term $\xi_{j,l}$ defined by Eq. (60) can be calculated based on the following matrix,

$$\xi_{j,l} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1\\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0\\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}_{6\times 9} \tag{93}$$

To tune the input and output parameters of the actor and critic, we follow the procedure described in Algorithm (1). After initializing the values of the input and output parameters of the actor and critic, learning rates, and the inputs, the output $u_t$ of the actor at the current state $\mathbf{s}_t$ is calculated by using Eq. (14) as follows,

$$u_t = \frac{\sum_{l=1}^{9}\left[\left(\prod_{i=1}^{2}\mu^{F_i^l}(s_i)\right)a_l\right]}{\sum_{l=1}^{9}\left(\prod_{i=1}^{2}\mu^{F_i^l}(s_i)\right)} \tag{94}$$

To solve the exploration/exploitation dilemma, a random noise $n(0,\sigma_n)$ with a zero mean and a standard deviation $\sigma_n$ should be added to the actor's output. Thus, the new output (action) $u_c$ will be defined as $u_c = u_t + n(0,\sigma_n)$.

The output of the critic at the current state $\mathbf{s}_t$ is calculated by using Eq. (14) as follows,

$$V_t(\mathbf{s}_t) = \frac{\sum_{l=1}^{9}\left[\left(\prod_{i=1}^{2}\mu^{F_i^l}(s_i)\right)c_l\right]}{\sum_{l=1}^{9}\left(\prod_{i=1}^{2}\mu^{F_i^l}(s_i)\right)} \tag{95}$$

The learning agent performs the action $u_c$ and observes the next state $\mathbf{s}_{t+1}$ and the immediate reward $r_t$. The output of the critic at the next state $\mathbf{s}_{t+1}$ is then calculated by using Eq. (14), which is in turn used to calculate the temporal error $\Delta_t$ by using Eq. (34). Then, the input and output parameters of the actor can be updated by using Eqs. (86), (87) and (88). On the other hand, the input and output parameters of the critic can be updated by using Eqs. (79), (80) and (81).

## 6 Simulation and Results

We evaluate the proposed RGFACL algorithm, the FACL algorithm and the QLFIS algorithm on three different pursuit–evasion games. In the first game, the evader is following a simple control strategy, whereas the pursuer is learning its control strategy to capture the evader in minimum time. In the second game, it is also only the pursuer that is learning. However, the evader in this game is following an intelligent control strategy that exploits the advantage of the maneuverability of the evader. In the third game, we make both the pursuer and the evader learn their control strategies. In multi-robot learning systems, each robot will try to learn its control strategy by interacting with the other robot which is also learning at the same time. Therefore, the complexity of the system will increase as the learning in a multi-robot system is considered as a problem of a "moving target" [53]. In the problem of a moving target, the best-response policy of each learning robot may keep changing during learning until each learning robot adopts an equilibrium policy. It is important to mention here that the pursuer, in all games, is assumed to not know the dynamics of the evader nor its control strategy.

We use the same learning and exploration rates for all algorithms when they are applied to the same game. Those rates are chosen to be similar to those used in [45]. We define the angle difference between the direction of the pursuer and the line-of-sight (LoS) vector of the pursuer to the evader by $\delta_p$. In all games, we define the state $\mathbf{s}_t$ for the pursuer by the two input variables which are the pursuer angle difference $\delta_p$ and its derivative $\dot{\delta}_p$. In the third game, we define the state $\mathbf{s}_t$ for the evader by the two input variables which are the evader angle difference $\delta_e$ and its derivative $\dot{\delta}_e$. Three Gaussian membership functions (MFs) are used to define the fuzzy sets of each input.

In all games, we assume that the pursuer is faster than the evader, and the evader is more maneuverable than the pursuer. In addition, the pursuer is assumed to not know the dynamics of the evader nor its control strategy. The only information the pursuer knows about the evader is the position (location) of the evader. The parameters of the pursuer are set as follows, $V_p = 2.0$m/s, $L_p = 0.3$m and $u_p \in [-0.5, 0.5]$. The pursuer starts its motion from the position $(x_p, y_p) = (0, 0)$ with an initial orientation $\theta_p = 0$. On the other hand, the parameters of the evader are set up as follows, $V_e = 1$m/s, $L_e = 0.3$m and $u_e \in [-1.0, 1.0]$. The evader starts its motion from a random position at each episode with an initial orientation $\theta_e = 0$. The sampling time is defined as $T = 0.05$s, whereas the capture radius is defined as $d_c = 0.1$m.

### 6.1 Pursuit–Evasion Game 1

In this game, the evader is following a simple control strategy defined by Eq. (30). On the other hand, the pursuer is learning its control strategy with the proposed RGFACL algorithm. We compare our results with the results obtained when the pursuer is following the classical control strategy defined by Eqs. (28) and (29). We also compare our results with the results obtained when the pursuer is learning its control strategy by the FACL and the QLFIS algorithms. We define the number of episodes in this game as 200 and the number of steps (in each episode) as 600. For each algorithm (the FACL, the QLFIS and the proposed RGFACL algorithms), we ran this game 20 times and we averaged the capture time of the evader over this number of trials.

Table 2 shows the time that the pursuer takes to capture the evader when the evader is following a simple control strategy and starts its motion from different initial positions. The table shows the capture time of the evader when the pursuer is following the classical control strategy and when the pursuer is learning its control strategy by the FACL algorithm, the QLFIS algorithm and the proposed RGFACL algorithm. From Table 2, we can see that the capture time of the evader when the pursuer learns its control strategy by the proposed RGFACL algorithm is

**Table 2** The time that the pursuer trained by each algorithm takes to capture an evader that follows a simple control strategy. The number of episodes here is 200

| Algorithm | Evader | | | |
|---|---|---|---|---|
| | (−5, 9) | (−10, −6) | (7, 4) | (5, −10) |
| Classical strategy | 10.70 | 12.40 | 8.05 | 11.20 |
| The proposed RGFACL | 10.80 | 12.45 | 8.05 | 11.25 |
| QLFIS | 10.95 | 12.50 | 8.05 | 11.35 |
| FACL | 11.50 | 13.50 | 8.65 | 12.25 |

very close to the capture time of the evader when the pursuer follows the classical control strategy. This shows that the proposed RGFACL algorithm achieves the performance of the classical control.

## 6.2 Pursuit–Evasion Game 2

In this game, the evader is following the control strategy defined by Eqs. (30) and (31) with the advantage of using its higher maneuverability. On the other hand, the pursuer in this game is learning its control strategy with the proposed RGFACL algorithm. Similar to game 1, we compare our results obtained when the pursuer is learning by the proposed RGFACL algorithm with the results obtained when the pursuer is following the classical control strategy defined by Eqs. (28) and (29). We also compare our results with the results obtained when the pursuer is learning its control strategy by the FACL and the QLFIS algorithms. In [45], it is assumed that the velocity of the pursuer and evader are governed by their steering angles so that the pursuer and evader can avoid slips during turning. This constraint will make the evader slow down its speed whenever the evader makes a turn. This will make it easy for the pursuer to capture the evader. Our objective is to see how the proposed algorithm and the other studied algorithms will behave when the evader makes use of the advantage of the maneuverability without any velocity constraints. Thus, in this work, we take this velocity constraints out so that both the pursuer and the evader can make fast turns without any velocity constraints. In this game, we use two different numbers for the episodes (200 and 1000), whereas the number of steps (in each episode) is set as 3000. For each algorithm (the FACL, the QLFIS and the proposed RGFACL algorithms), we ran this game 20 times and, then, averaged the capture time of the evader over this number of trials.

Tables 3 and 4 show the time that the pursuer takes to capture the evader when the evader is following the control strategy defined by Eqs. (30) and (31) with the advantage of using its higher maneuverability. The number of episodes used here is 200 for Table 3 and 1000 for Table 4. The tables show that the pursuer fails to capture the evader when the pursuer is following the classical control strategy and when learning by the FACL algorithm. Table 3 shows that the pursuer succeeds to capture the evader in all 20 trials only when the pursuer is learning by the proposed RGFACL algorithm. When learning by the QLFIS algorithm, the pursuer succeeds to capture the evader only in 20% of the 20 trials. On the other hand, Table 4 shows that the pursuer always succeeds to capture the evader only when the pursuer is learning with the proposed RGFACL algorithm. However, when learning with the QLFIS algorithm, the pursuer succeeds to capture the evader only in 50% of the 20 trials. Tables 3 and 4 show that the proposed RGFACL algorithm outperforms the FACL and the QLFIS algorithms. This is because the pursuer using the proposed RGFACL algorithm to learn its control strategy always succeeds to capture the evader in less time as well as in a less number of episodes.

## 6.3 Pursuit–Evasion Game 3

Unlike game 1 and game 2, both the evader and the pursuer are learning their control strategies in this game. In multi-robot learning systems, each robot will try to learn its control strategy by interacting with the other robot which is also learning its control strategy at the same time. Thus, the complexity of the system will increase in this game as the learning in a multi-robot system is considered as a problem of a “moving target” [53]. We compare the results obtained by the proposed algorithm with the results obtained by the FACL and QLFIS algorithms. Unlike the first two pursuit–evasion games, we do not use the capture time of the evader as a criterion in our comparison in this game. This is because both the pursuer and the evader are learning. That is, a small capture time by the pursuer

**Table 3** The time that the pursuer trained by each algorithm takes to capture an evader that follows an intelligent control strategy. The number of episodes here is 200

| Algorithm | Evader | | | |
|---|---|---|---|---|
| | (−9, 7) | (−7, −10) | (6, 9) | (3, −9) |
| Classical strategy | No capture | No capture | No capture | No capture |
| The proposed RGFACL | 13.05 100% | 14.30 100% | 11.65 100% | 11.15 100% |
| QLFIS | 23.20 20% | 23.55 20% | 25.45 20% | 21.35 20% |
| FACL | No capture | No capture | No capture | No capture |

**Table 4** The time that the pursuer trained by each algorithm takes to capture an evader that follows an intelligent control strategy. The number of episodes here is 1000

| Algorithm | Evader | | | |
|---|---|---|---|---|
| | (−9, 7) | (−7, −10) | (6, 9) | (3, −9) |
| Classical strategy | No capture | No capture | No capture | No capture |
| The proposed RGFACL | 12.70 100% | 13.15 100% | 11.30 100% | 10.90 100% |
| QLFIS | 20.25 50% | 21.60 50% | 19.60 50% | 19.20 50% |
| FACL | No capture | No capture | No capture | No capture |

learning its control strategy by one of the learning algorithms can have two different indications; the first one is that the learning algorithm is working well as the pursuer succeeds to capture the evader quickly. The second indication, on the other hand, is that the learning algorithm is not working properly as the evader does not learn how to escape from the pursuer. Therefore, we compare the paths of the pursuer and the evader learning their control strategies by the learning algorithms with the paths of the pursuer and the evader following the classical control strategy defined by Eqs. (28) and (29).

In game 3, the pursuer starts its motion from the position $(x_p, y_p) = (0, 0)$ with an initial orientation $\theta_p = 0$. On the other hand, the evader starts its motion from a random position at each episode with an initial orientation $\theta_e = 0$. We run game 3 twice. In the first run, we set the number of the episodes in this game to 200, whereas the number of steps in each episode is set to 3000. The results of the first run are shown in Figs. 4, 5 and 6 when the evader starts its motion from the position $(x_e, y_e) = (-10, -10)$. These figures show the paths of the pursuer and the evader (starred lines) when learning by the FACL, the QLFIS, and the proposed RGFACL algorithms, respectively. The paths of the pursuer and the evader following the classical control strategy are also shown in the figures (dotted lines). The results of the first run show that the proposed RGFACL algorithm outperforms the FACL and the QLFIS

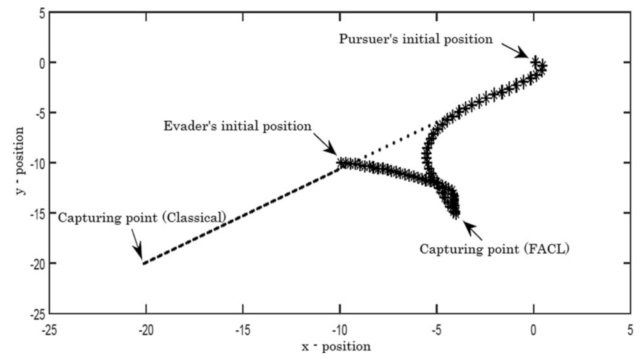


**Fig. 4** The paths of the pursuer and the evader when learning by the FACL algorithm proposed in [43] (*starred lines*) against the paths of the pursuer and the evader when following the classical strategy defined in Eqs. (28) and (29) (*dotted lines*). The number of episodes used here is 200

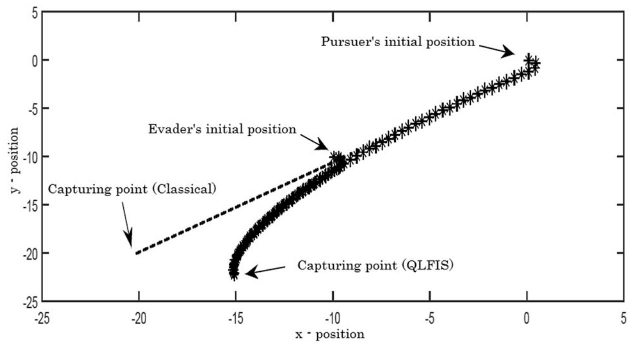


**Fig. 5** The paths of the pursuer and the evader when learning by the QLFIS algorithm proposed in [45] (*starred lines*) against the paths of the pursuer and the evader when following the classical strategy defined in Eqs. (28) and (29) (*dotted lines*). The number of episodes used here is 200

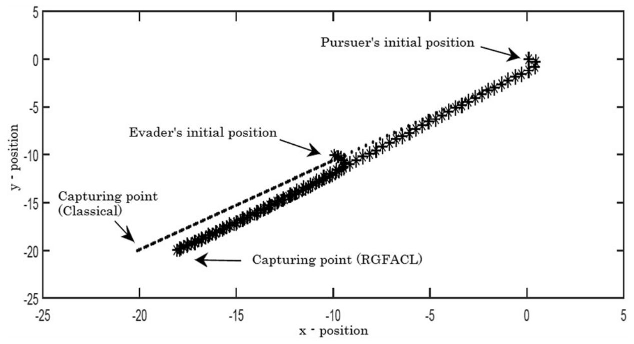


**Fig. 6** The paths of the pursuer and the evader when learning by the proposed RGFACL algorithm (*starred lines*) against the paths of the pursuer and the evader when following the classical strategy defined in Eqs. (28) and (29) (*dotted lines*). The number of episodes used here is 200

algorithms as the performance of the proposed algorithm is close to the performance of the classical control strategy. In the second run of game 3, we set the number of the episodes in this game to 500, whereas the number of steps in each episode is set to 3000. The results of the second run are shown in Figs. 7, 8 and 9 when the evader starts its motion from the position $(x_e, y_e) = (-10, -10)$. The figures show that the performance of the proposed RGFACL algorithm and the performance of the QLFIS algorithm are close to the performance of the classical control strategy and both algorithms outperform the FACL algorithm.

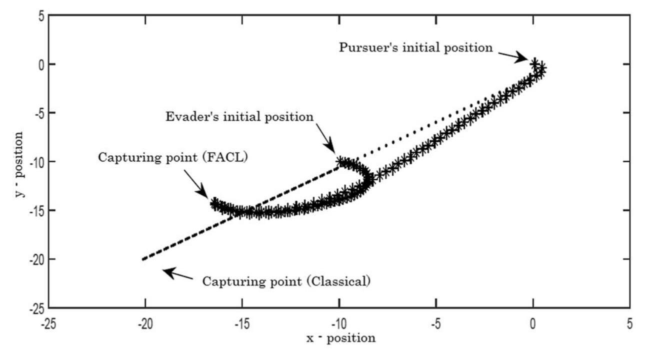


**Fig. 7** The paths of the pursuer and the evader when learning by the FACL algorithm proposed in [43] (*starred lines*) against the paths of the pursuer and the evader when following the classical strategy defined in Eqs. (28) and (29) (*dotted lines*). The number of episodes used here is 500

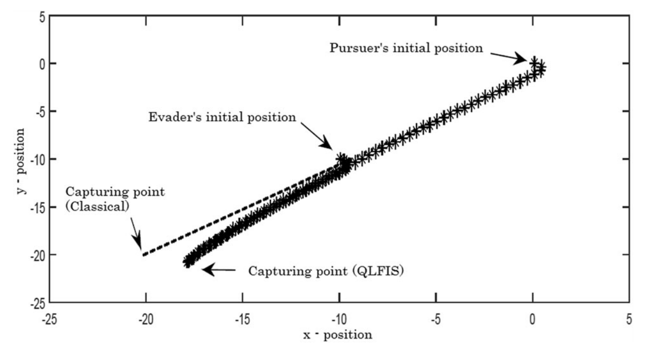


**Fig. 8** The paths of the pursuer and the evader when learning by the QLFIS algorithm proposed in [45] (*starred lines*) against the paths of the pursuer and the evader when following the classical strategy defined in Eqs. (28) and (29) (*dotted lines*). The number of episodes used here is 500

## 7 Conclusion

In this work, we propose a new fuzzy reinforcement learning algorithm for differential games that have continuous state and action spaces. The proposed algorithm uses FISs as function approximation systems: an actor (fuzzy logic controller, FLC) and a critic. The proposed algorithm tunes the input and output parameters of its function approximation systems (the actor and the critic) differently from the tuning mechanisms used in the algorithms proposed in the literature. The proposed algorithm uses the residual gradient value iteration algorithm as a mechanism in tuning the parameters of its function approximation systems, whereas the algorithms proposed in the literature use the direct algorithms in their mechanisms to tune the parameters of their function approximation systems. The proposed algorithm is called the RGFACL algorithm. It has been shown in the literature that the residual gradient algorithms are superior to the direct algorithms as the residual gradient algorithms are always guaranteed to

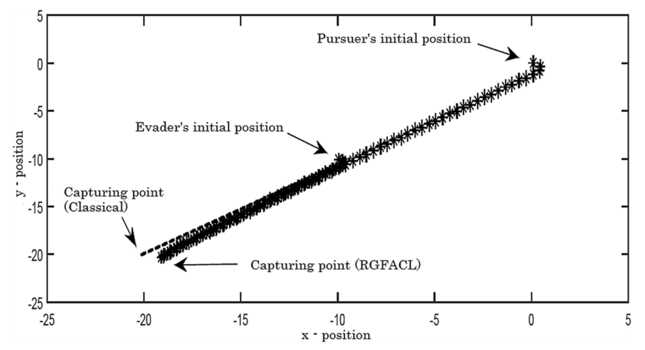


**Fig. 9** The paths of the pursuer and the evader when learning by the proposed RGFACL algorithm (*starred lines*) against the paths of the pursuer and the evader when following the classical strategy defined in Eqs. (28) and (29) (*dotted lines*). The number of episodes used here is 500

converge, whereas the direct algorithms may not converge to an answer in some cases. For ease of implementation, the complete derivation of the partial derivatives that are needed by the proposed algorithm is presented in this work. The proposed algorithm is used to learn three different pursuit–evasion games. We start with the game where the pursuer learns its control strategy and the evader follows a simple control strategy. In the second game, the pursuer learns its control strategy and the evader follows an intelligent control strategy that exploits the advantage of higher maneuverability. In the third game, we increase the complexity of the system by making both the pursuer and the evader learn their control strategies. Simulation results show that the proposed RGFACL algorithm outperforms the FACL and the QLFIS algorithms in terms of performance and the learning time when they all are used to learn the pursuit–evasion games considered in this work.

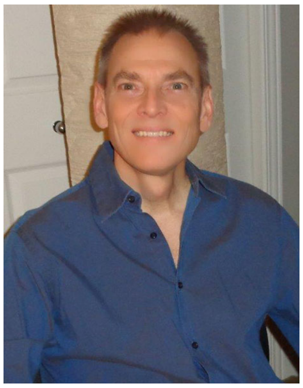
## References

1. Passino, K.M., Yurkovich, S.: Fuzzy control. Addison Wesley Longman, Inc., Menlo Park (1998)
2. Marin, N., Ruiz, M.D., Sanchez, D.: Fuzzy frameworks for mining data associations: fuzzy association rules and beyond. Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. **6**(2), 50–69 (2016)
3. Micera, S., Sabatini, A.M., Dario, P.: Adaptive fuzzy control of electrically stimulated muscles for arm movements. Med. Biol. Eng. Comput. **37**(6), 680–685 (1999)
4. Daldaban, F., Ustkoyuncu, N., Guney, K.: Phase inductance estimation for switched reluctance motor using adaptive neuro-fuzzy inference system. Energy Convers. Manag. **47**(5), 485–493 (2005)
5. Jang, J.S.R., Sun, C.T., Mizutani, E.: Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence. Prentice Hall, Upper Saddle River (1997)
6. Labiod, S., Guerra, T.M.: Adaptive fuzzy control of a class of SISO nonaffine nonlinear systems. Fuzzy Sets Syst. **158**(10), 1126–1137 (2007)

7. Lam, H.K., Leung, F.H.F.: Fuzzy controller with stability and performance rules for nonlinear systems. Fuzzy Sets Syst. **158**(2), 147–163 (2007)
8. Hagras, H., Callaghan, V., Colley, M.: Learning and adaptation of an intelligent mobile robot navigator operating in unstructured environment based on a novel online Fuzzy-Genetic system. Fuzzy Sets Syst. **141**(1), 107–160 (2004)
9. Mucientes, M., Moreno, D.L., Bugarn, A., Barro, S.: Design of a fuzzy controller in mobile robotics using genetic algorithms. Appl. Soft Comput. **7**(2), 540–546 (2007)
10. Wang, L.X.: A Course in Fuzzy Systems and Control. Prentice Hall, Upper Saddle River (1997)
11. Desouky, S.F., Schwartz, H.M.: Self-learning fuzzy logic controllers for pursuit-evasion differential games. Robot. Auton. Syst. **59**, 22–33 (2011)
12. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans. Syst. Man Cybern. **5**, 834–846 (1983)
13. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 1.1. MIT press, Cambridge (1998)
14. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996)
15. Awheda, M.D., Schwartz, H.M.: The residual gradient FACL algorithm for differential games. IN: IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 1006–1011 (2015)
16. Hinojosa, W., Nefti, S., Kaymak, U.: Systems control with generalized probabilistic fuzzy-reinforcement learning. IEEE Trans. Fuzzy Syst. **19**(1), 51–64 (2011)
17. Rodríguez, M., Iglesias, R., Regueiro, C.V., Correa, J., Barro, S.: Autonomous and fast robot learning through motivation. In: Robotics and Autonomous Systems, vol. 55.9, pp. 735–740. Elsevier (2007)
18. Schwartz, H.M.: Multi-agent machine learning: a reinforcement approach. Wiley, New York (2014)
19. Awheda, M.D., Schwartz, H.M.: A decentralized fuzzy learning algorithm for pursuit-evasion differential games with superior evaders. J. Intell. Robot. Syst. **83**(1), 35–53 (2016)
20. Awheda, M.D., Schwartz, H.M.: A fuzzy learning algorithm for multi-player pursuit-evasion differential games with superior evaders. In: Proceedings of the 2016 IEEE International Systems Conference, Orlando, Florida (2016)
21. Awheda, M.D., Schwartz, H.M.: A fuzzy reinforcement learning algorithm using a predictor for pursuit-evasion games. In: Proceedings of the 2016 IEEE International Systems Conference, Orlando, Florida (2016)
22. Smart, W.D., Kaelbling, L.P.: Effective reinforcement learning for mobile robots. In: IEEE International Conference on Robotics and Automation, Proceedings ICRA'02, 4 (2002)
23. Ye, C., Yung, N.H.C., Wang, D.: A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance. IEEE Trans. Syst. Man Cybern. Part B: Cybern. **33**(1), 17–27 (2003)
24. Kondo, T., Ito, K.: A reinforcement learning with revolutionary state recruitment strategy for autonomous mobile robots control. Robot. Auton. Syst. **46**, 111–124 (2004)
25. Gutnisky, D.A., Zanutto, B.S.: Learning obstacle avoidance with an operant behavior model. Artif. Life **10**(1), 65–81 (2004)
26. Dai, X., Li, C., Rad, A.B.: An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control. IEEE Trans. Intell. Transp. Syst. **6**(3), 285–293 (2005)
27. Luo, B., Wu, H.N., Huang, T.: Off-policy reinforcement learning for $H_\infty$ control design. IEEE Trans. Cybern. **45.1**, 65–76 (2015)
28. Luo, B., Wu, H.N., Li, H.X.: Adaptive optimal control of highly dissipative nonlinear spatially distributed processes with neuro-dynamic programming. IEEE Trans. Neural Netw. Learn. Syst. **26.4**, 684–696 (2015)
29. Luo, B., Wu, H.N., Huang, T., Liu, D.: Reinforcement learning solution for HJB equation arising in constrained optimal control problem. In: Neural Networks, vol. 71, pp. 150–158. Elsevier (2015)
30. Modares, H., Lewis, F.L., Naghibi-Sistani, M.B.: Integral reinforcement learning and experience replay for adaptive optimal control of partially-unknown constrained-input continuous-time systems. In: Automatica, vol. 50.1, pp. 193–202. Elsevier (2014)
31. Dixon, W.: Optimal adaptive control and differential games by reinforcement learning principles, J. Guid. Control Dyn. **37.3**, 1048–1049 (2014)
32. Luo, B., Wu, H.N., Li, H.X.: Data-based suboptimal neuro-control design with reinforcement learning for dissipative spatially distributed processes, Ind. Eng. Chem. Res. **53.19**, 8106–8119 (2014)
33. Wu, H.N., Luo, B.: Neural network based online simultaneous policy update algorithm for solving the HJI equation in nonlinear control. IEEE Trans. Neural Netw. Learn. Syst. **23.12**, 1884–1895 (2012)
34. Xia, Z., Zhao, D.: Online reinforcement learning control by Bayesian inference. IET Control Theory Appl. **10**(12), 1331–1338 (2016)
35. Liu, Y.J., Gao, Y., Tong, S., Li, Y.: Fuzzy approximation-based adaptive backstepping optimal control for a class of nonlinear discrete-time systems with dead-zone. IEEE Trans. Fuzzy Syst. **24**(1), 16–28 (2016)
36. Zhu, Y., Zhao, D., Li, X.: Using reinforcement learning techniques to solve continuous-time non-linear optimal tracking problem without system dynamics. IET Control Theory Appl. **10**(12), 1339–1347 (2016)
37. Kamalapurkar, R., Walters, P., Dixon, W.E.: Model-based reinforcement learning for approximate optimal regulation. Automatica **64**, 94–104 (2016)
38. Jiang, H., Zhang, H., Luo, Y., Wang, J.: Optimal tracking control for completely unknown nonlinear discrete-time Markov jump systems using data-based reinforcement learning method. Neurocomputing **194**, 176–182 (2016)
39. Sutton, R.S.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**(1), 9–44 (1988)
40. Dayan, P., Sejnowski, T.J.: TD($\lambda$) converges with probability 1. Mach. Learn. **14**, 295–301 (1994)
41. Dayan, P.: The convergence of TD($\lambda$) for general $\lambda$. Mach. Learn. **8**(3–4), 341–362 (1992)
42. Jakkola, T., Jordan, M., Singh, S.: On the convergence of stochastic iterative dynamic programming. Neural Comput. **6**, 1185–1201 (1993)
43. Jouffe, L.: Fuzzy inference system learning by reinforcement methods. IEEE Trans. Syst. Man Cybern. C **28.3**, 338–355 (1998)
44. Bonarini, A., Lazaric, A., Montrone, F., Restelli, M.: Reinforcement distribution in fuzzy Q-learning. Fuzzy Sets Syst. **160**(10), 1420–1443 (2009)
45. Desouky, S.F., Schwartz, H.M.: Q ($\lambda$)-learning adaptive fuzzy logic controllers for pursuit–evasion differential games. Int. J. Adapt. Control Signal Process. **25**(10), 910–927 (2011)
46. Givigi Jr., S.N., Schwartz, H.M., Lu, X.: A reinforcement learning adaptive fuzzy controller for differential games. J. Intell. Robot. Syst. **59**, 3–30 (2010)
47. Wang, X.S., Cheng, Y.H., Yi, J.Q.: A fuzzy Actor–Critic reinforcement learning network. Inf. Sci. **177**(18), 3764–3781 (2007)
48. Baird, L.: Residual algorithms: reinforcement learning with function approximation. In: ICML, pp. 30–37 (1995)
49. Boyan, J., Moore, A.W.: Generalization in reinforcement learning: safely approximating the value function. In: Advances in

Neural Information Processing Systems, vol. 7, pp. 369–376. Cambridge, MA, The MIT Press (1995)
50. Gordon, G.J.: Reinforcement learning with function approximation converges to a region. In: Advances in Neural Information Processing Systems, vol. 13, pp. 1040–1046. MIT Press (2001)
51. Schoknecht, R., Merke, A.: TD(0) converges provably faster than the residual gradient algorithm. In: ICML (2003)
52. Tsitsiklis, J.N., Roy, B.V.: An analysis of temporal-difference learning with function approximation. IEEE Trans. Autom. Control **42**(5), 674–690 (1997)
53. Bowling, M., Veloso, M.: Multiagent learning using a variable learning rate. Artif. Intell. **136**(2), 215–250 (2002)
54. Van Buijtenen, W.M., Schram, G., Babuska, R., Verbruggen, H.B.: Adaptive fuzzy control of satellite attitude by reinforcement learning. IEEE Trans. Fuzzy Syst. **6**(2), 185–194 (1998)
55. Mamdani, E.H., Assilian, S.: An experiment in linguistic synthesis with a fuzzy logic controller. Int. J. Man Mach. Stud. **7.1**, 113 (1975)
56. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modelling and control. IEEE Trans. Syst. Man Cybern. SMC **15**(1), 116–132 (1985)
57. Sugeno, M., Kang, G.: Structure identification of fuzzy model. Fuzzy Sets Syst. **28**, 15–33 (1988)
58. Isaacs, R.: Differential Games. Wiley, New York (1965)
59. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
60. Lim, S.H., Furukawa, T., Dissanayake, G., Whyte, H.F.D.: A time-optimal control strategy for pursuit-evasion games problems, In: International Conference on Robotics and Automation, New Orleans, LA (2004)
61. Desouky, S.F., Schwartz, H.M.: Different hybrid intelligent systems applied for the pursuit–evasion game. In: 2009 IEEE International Conference on Systems, Man, and Cybernetics, pp. 2677–2682 (2009)

**Mostafa D. Awheda** received his B.Sc. degree in control engineering from the College of Electronic Technology, Bani Walid, Libya. Mr. Awheda received his M.Sc. in control engineering from Lakehead University, Thunder Bay, Canada. Right now, Mr. Awheda is pursuing his Ph.D. in electrical engineering at Carleton University, Ottawa, Canada. Mr. Awheda's research interests include machine learning, fuzzy control, and intelligent control systems.



**Howard M. Schwartz** (S'85-M'87-SM'11) received the B.Eng. degree in Civil Engineering from McGill University, Montreal, QC, Canada in 1981, and the M.S. in Aeronautics and Astronautics in 1982 and the Ph.D. degree in Mechanical Engineering in 1987 from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. He is currently a Professor with the Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada. His research interests include adaptive and intelligent systems, reinforcement learning, robotics, system modeling, and system identification. His most recent research is in multi-agent learning with applications to teams of mobile robots.