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

# An Experimental Adaptive Fuzzy Controller for Differential Games

Sidney N. Givigi Jr., Howard M. Schwartz and Xiaosong Lu
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, Canada
Email: sgivigi@sce.carleton.ca

***Abstract*—In this paper a reinforcement fuzzy learning scheme for robots playing a differential game is derived. A differential game may be considered a Markov decision process in continuous time, with continuous states and actions. The robots receive reinforcements from the environment after they take an action; and this reinforcement is then used to adapt a fuzzy controller that stores the experience accumulated by the robot. Every calculation is done in a physical system based on microcontrollers to control the movement of the robots and sensors to measure their position and angle in a 2D-plane. Filters are also implemented to approximate the derivatives of the states. Experiments of a pursuer-evader game are provided in order to show the feasibility of the technique. It should be noted, though, that the technique may also be used in a multi-game environment.**

***Index Terms*—Differential games, learning, pursuer-evader games, intelligent systems.**

## I. Introduction

Game theory is the study of decision making [1] in order to solve conflicts. It was introduced by von Neumann and Morgenstern [2]. Each player is given a utility function (the reward or penalty it receives) of its own strategy and the strategies played by all the other players (or a subset of them). In the general approach, the game and the strategies are discrete, therefore, matrices with strategies and payoffs (the rewards or penalties) may be assembled. Another way of viewing a game was introduced by Isaacs [3] and is called *differential games*. Differential games investigates how decision making takes place over time [4] considering continuous domains, i.e., when there is not a small number of strategies at each player's disposal. In order to represent this game, we need to model the dynamic equations that are related to the process under investigation. These equations are typically differential or difference equations.

Learning in games has been largely studied in the last couple of decades. It already includes several books [5], [6] and recent research papers abound in the specialized literature [7], [8]. However, little attention has been given to learning in the differential game domain [9]. In the case of differential games, one of the most popular learning approaches has been the use of reinforcement learning and Q-learning [9], [10]. However, there is a disadvantage in this technique when we deal with continuous processes such as the ones considered in differential games. Since they are based on the construction of tables, several different actions must be considered coupled with states in order to describe the possible behaviours of the players. This could lead to a proliferation of updates that would make the approach unfeasible for implementation in a microcontroller. Moreover, it is not easy to discretize the action space as well as the state space [11].

In order to avoid this problem, one could use a fuzzy controller. It is well known that a fuzzy system is a universal approximator [12]. Therefore, we propose in this work a fuzzy controller that is updated by a reinforcement learning algorithm. The advantages of such approach are:

- A fuzzy controller can deal with noisy data [12] and uncertainties [13].
- Reinforcement learning updates is an adequate way of updating the fuzzy rules on line [13].
- A fuzzy controller such as presented here can be easily implemented in a microcontroller.

The paper is divided as follows. Section II briefly introduces the notation of a differential game. In section III we present the control structure for the system. Section IV reviews the learning techniques used in games in general and introduces a fuzzy algorithm for learning in differential games. In section V, we describe the system that will be used in the simulations as well as how it relates to the notation in section II. Section VI describes the experimental environment we built. In section VII, experiments are presented. And, finally, section VIII presents our conclusions.

## II. Differential Games

In the general, nonzero-sum, $N$-player differential game, a player $i$ tries to choose a control signal $u_i$ to minimize the cost equation [14]

$$J_i = q_i(\bar{x}(T)) + \int_{t_0}^{T} g_i(\bar{x}(s), u_1(s), \cdots, u_N(s), s)ds \qquad (1)$$

subject to the state dynamics

$$\dot{\bar{x}}(s) = f(\bar{x}(s), \bar{u}_1(s), \cdots, \bar{u}_N(s), s), \quad \bar{x}(t_0) = \bar{x}_0 \qquad (2)$$

where $\bar{x}(s) \in \mathbb{R}^m$ is the state vector of dimension $m$, $T$ is the terminating time (or the time where the terminal state is reached), $q_i(\cdot)$ is the payoff of the terminal state and $g_i(\cdot)$ is the integral payoff for player $i \in N$.

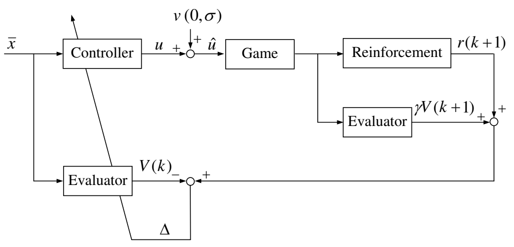

Fig. 1: Architecture of the control system

Functions $q_i(\cdot)$ and $g_i(\cdot)$ are chosen in order to achieve an objective. Function $f(\cdot)$ determines the dynamics of the system. They could be represented by inequalities [14]. We also assume that one agent (or player) has access to the states of the other players involved in the game at all times. This is called the *perfect information* assumption.

If the game under study is (as it is for this paper) between only two players, the system dynamics may be written [15]

$$\dot{\bar{x}}(s) = f(\bar{x}(s), \bar{\delta}_p(s), \bar{\delta}_e(s), s), \quad \bar{x}(t_0) = \bar{x}_0 \tag{3}$$

where $\bar{\delta}_p$ and $\bar{\delta}_e$ are the strategies played by each player. The payoff, now represented as $P(\bar{\delta}_p, \bar{\delta}_e)$, is given in the form

$$P(\bar{\delta}_p, \bar{\delta}_e) = q(t^*, \bar{x}(t^*)) + \int_{t_0}^{t^*} g(\bar{x}(s), \bar{\delta}_p, \bar{\delta}_e, s)ds \tag{4}$$

where $t^*$ is the first time the states $\bar{x}(t)$ intersect a given final condition. In this case it is also assumed that the player who uses strategy $\bar{\delta}_p$ wants to minimize the payoff $P(\cdot)$, whereas the player using strategy $\bar{\delta}_e$ wants to maximize it. Therefore, the objective of the game is to find control signals $\bar{\delta}_p^*$ and $\bar{\delta}_e^*$ such that [16]

$$P(\bar{\delta}_p^*, \bar{\delta}_e) \leq P(\bar{\delta}_p^*, \bar{\delta}_e^*) \leq P(\bar{\delta}_p, \bar{\delta}_e^*), \;\; \forall \bar{\delta}_p, \, \bar{\delta}_e \tag{5}$$

In section V we are going to present a simple model that fits the equations presented above. However, before doing that, we need to describe the control structure in section III and to establish in section IV how a robot could learn how to play a game using a fuzzy inference system. Lastly, notice that in the simulations the positions of the robots are noisy, which presents no theoretical problem since fuzzy systems are intrinsically designed to deal with noise.

## III. System Structure

Figure 1 shows the proposed structure for the controller [17]. In this section, we are going to focus on the structure of each block, more specifically the *controller* and the *evaluator*.

We assume that the *controller* in figure 1 is a fuzzy controller. More specifically, it is a fuzzy controller implemented by Takagi-Sugeno (TS) rules with constant consequents [18]. It consists of $M$ rules with $n$ fuzzy variables as inputs and one constant number as consequent. Therefore, each rule is of the form [11]

$$R_l \quad : \quad IF\, x_1\, is\, F_1^l, \cdots, \, and\, x_n\, is F_n^l \tag{6}$$

$$THEN\, u = c_l \tag{7}$$

where $x_i$ are the values passed to the controller, $F_i^l$ is the fuzzy set related to the corresponding fuzzy variable, $u$ is the rule's output, and $c_l$ is a constant that describes the center of a fuzzy set.

Therefore, if we use the product inference for fuzzy implication [12], *t* norm, singleton fuzzifier and center-average defuzzifier, the output of the system is [11]

$$u(\bar{x}) = \frac{\sum_{l=1}^{M}((\prod_{i=1}^{n} \mu^{F_i^l}(x_i)) \cdot \omega_l)}{\sum_{l=1}^{M}(\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} \tag{8}$$

where $c_l$ in (7) is represented by $\omega_l$ for the controller. Throughout the paper, the membership functions used are triangular.

For the *evaluator*, we also assume a TS system with constant consequents [17]. However, it must be noted that this is not the only possible choice. A time-delay neural network (TDNN) could be used instead [11]. There are advantages and disadvantages in this choice for both cases. We chose the fuzzy system just for its simplicity. Therefore, just as in the case of (8), the output of the evaluator is [17] an approximation to the value of the state $V(\cdot)$

$$\hat{V}(\bar{X}) = \frac{\sum_{l=1}^{M}((\prod_{i=1}^{n} \mu^{F_i^l}(x_i)) \cdot \zeta_l)}{\sum_{l=1}^{M}(\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} \tag{9}$$

where $c_l$ in (7) is represented by $\zeta_l$.

## IV. Learning

In the context of Game Theory, learning may be understood as strategy adjustments [19]. These adjustments may be function of several different sets of variables.

Different types of continuous-time models of learning have been proposed for games. In the case of games in extensive form, Arslan and Shamma [19] describe some models discussed in the literature, including the well known *replicator dynamics*, where one player tries to approximate the way the other (or others) will play in order to decide which strategy is more profitable for it to play. However, in the special case of differential games, the learning techniques presented in the literature are very different. Most of them are based on reinforcement learning [20].

Let us consider a game between two players as described in (3) and (4). Notice that the approach presented here may be used for any number of players, but, for simplicity sake, we present the technique for just two-player games. In this case, the learning dynamics may be described by

$$\dot{\bar{\delta}}_p = f_{\bar{\delta}_p}(\bar{x}, \bar{\delta}_p, \bar{\delta}_e) \tag{10}$$

$$\dot{\bar{\delta}}_e = f_{\bar{\delta}_e}(\bar{x}, \bar{\delta}_p, \bar{\delta}_e) \tag{11}$$

Notice, also, that the functions $f(\cdot)$ must take into consideration the cost (4) in such a way that the *saddle point* in (5) is reached. Also, the strategy adjustments in (10) and (11) will be functions of the strategies played by *both* players at a given time as well as their states. Therefore, we assume that the players play their strategies synchronously. Also, we assume that they know the noisy state of the other player as well as its own state.

The problem of (10) and (11) is that the strategies played by each player are continuous. This means that the strategy vectors $\bar{\delta}_p$ and $\bar{\delta}_e$ should have dimensions of infinity. Another approach would be to try to discretize the action space and then use a Q-learning algorithm to calculate the strategy vectors. Although, there is another problem. We would have to discretize the state space as well and this is not very easily done for most of the differential games. In order to avoid these potential problems, we use the architecture shown figure 1.

In this figure we see the addition of two blocks called *evaluator*. This block is used to approximate the value function for reinforcement learning [11]. It could be implemented by a time-delay neural network [11] or a fuzzy system [17]. The learning is practically the same for both cases. The value function for approximation of the reinforcement rewards has the format

$$V(k) = E\{\sum_{i=k}^{\infty} \gamma^{i-k} r(i+1)\} \tag{12}$$

where $\gamma \in [0,1)$ is known as the forgetting factor and $r(\cdot)$ is the immediate external reward from the environment. Notice that we can rewrite (12) in a recursive fashion as

$$V(k) = r(k+1) + \gamma V(k+1) \tag{13}$$

With this approximation in hands, we may compare it with the expected reward such that we generate a prediction error of the prediction $\hat{V}(k)$ [17] that is the output of the evaluator so that

$$\Delta = [r(k+1) + \gamma \hat{V}(k+1)] - \hat{V}(k) \tag{14}$$

as shown in figure 1. This difference error is then used to train the evaluator. Supposing it has parameters $\zeta$ in (9) are to be adapted, the adaptation law would then be [11]

$$\zeta(k+1) = \zeta(k) + \alpha \Delta \frac{\partial \hat{V}(k)}{\partial \zeta} \tag{15}$$

where $\alpha \in (0,1)$ is the learning rate for the adaptation. Observe that we do not want $\alpha$ to be too big in order to avoid instability in the generated system. Also the partial derivative in (15) is easily calculated to be from (9)

$$\frac{\partial \hat{V}(k)}{\partial \zeta} = \frac{\prod_{i=1}^{n} \mu^{F_i^l}(x_i)}{\sum_{l=1}^{M} (\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} \tag{16}$$

The controller in figure 1 is a fuzzy controller implemented by Takagi-Sugeno rules with constant consequents [18]. Observe that to its generated control signal $u(k)$ is added a random white noise $v(0,\sigma)$. This is done in order to promote exploration of the action space [11]. With the noisy signal $u'(t)$, taking $\omega$ in (8) as the parameters to be adapted, we may establish the controller update law [17]

$$\omega(k+1) = \omega(k) + \beta \Delta [\frac{u'(k) - u(k)}{\sigma}] \frac{\partial u(k)}{\partial \omega} \tag{17}$$

where $\beta \in (0,1)$ is the learning rate for the controller adaptation. Note that we want $\beta < \alpha$, meaning that we want the controller to converge slower than the evaluator. This is done in order to avoid instability in the controller. Also notice that the initial fuzzy controller can give a bad performance for the player. The partial derivative in (17) is easily calculated to be from (8)

$$\frac{\partial u(k)}{\partial \omega} = \frac{\prod_{i=1}^{n} \mu^{F_i^l}(x_i)}{\sum_{l=1}^{M} (\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} \tag{18}$$

Notice that the partial derivatives in (16) and (18) are rigorously the same and need to be calculated just once. Also notice that at no time we have a "desired" trajectory and, therefore, we have no error signal. The update laws in (15) and (17) are based on a "forecasted" improvement of the cost function (equation (4)).

In the next section, we describe the game we will solve with the technique described so far.

## V. Pursuer Evader Model

Let us focus on one of the models of pursuit-evasion generated in the work by Isaacs [3] also known as the "game of two cars" model [21]. Let us consider the models for the pursuer and evader to be

$$\begin{aligned} \dot{x}_p &= V_p \cos(\theta_p) \\ \dot{y}_p &= V_p \sin(\theta_p) \\ \dot{\theta}_p &= \frac{V_p}{R_p} \delta_p \end{aligned} \tag{19}$$

In the same way, the model for the evader may be described as

$$\begin{aligned} \dot{x}_e &= V_e \cos(\theta_e) \\ \dot{y}_e &= V_e \sin(\theta_e) \\ \dot{\theta}_e &= \frac{V_e}{R_e} \delta_e \end{aligned} \tag{20}$$

where $V_p$ is the pursuer's speed, $V_e$ is the evader's speed and $V_p > V_e$; $R_p$ is the radius of turning of the pursuer, $R_e$ is the radius of turning of the evader and $R_p > R_e$; $\theta_p$ is the orientation of the pursuer and $\theta_e$ is the orientation of the evader; $|\delta_p| \leq 1$ is the control signal for the pursuer and $|\delta_e| \leq 1$ is the control signal for the evader. In words, the pursuer is faster, but the evader can make sharper turns.

Now, we assume a coordinate frame centered in the pursuer with its $y'$-axis in the direction of the pursuer's velocity vector [16] as shown in figure 2. The pair $(x', y')$ is the relative

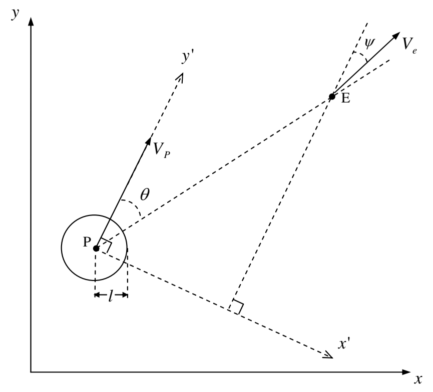

Fig. 2: Relative position for the evader with respect to the pursuer

position of the evader in this coordinate frame. One then may find that the differential equations for this game are of the form

$$\dot{x}' = V_e \sin\psi - \frac{V_p}{R_p} y' \delta_p \tag{21}$$

$$\dot{y}' = V_e \cos\psi - V_p + \frac{V_p}{R_p} x' \delta_p \tag{22}$$

$$\dot{\psi} = -\frac{V_p}{R_p}\delta_p + \frac{V_e}{R_e}\delta_e \tag{23}$$

where $\psi = \theta_e - \theta_p$.

The game finishes when (and if) the evader is captured. Under optimal play, capture time is then found by solving the equation [21]

$$\begin{aligned}\min_{\delta_p}\max_{\delta_e}\left(\frac{dP}{dt}\right) &= \min_{\delta_p}\max_{\delta_e}[P_{x'}(V_e\sin\psi - \frac{V_p}{R}y'\delta_p) + \\ &\quad P_{y'}(V_e\cos\psi - V_p + \frac{V_p}{R}x'\delta_p) + \\ &\quad P_{\psi}(-\frac{V_p}{R_p}\delta_p + \frac{V_e}{R_e}\delta_e)] \\ &= -1 \end{aligned} \tag{24}$$

where $P$ is the cost function of the game and $P_{x'}$ and $P_{y'}$ are, respectively, its partial derivatives with respect to $x'$ and $y'$.

The optimal control for the pursuer and evader is found such that [16]

$$\delta_p = \mathrm{sgn}(\lambda_{x'}y' - \lambda_{y'}x' + \lambda_\psi) \tag{25}$$

$$\delta_e = \mathrm{sgn}(\lambda_\psi) \tag{26}$$

where $\lambda_{x'}$, $\lambda_{y'}$ and $\lambda_\psi$ are lagrange multipliers. As noted by Merz [21], there are more than 20 qualitatively different forms of solution for such problems given different values of the parameters $V_p$, $V_e$, $R_p$ and $R_e$. Since we are interested in learning and not in the different solutions of (24) given the control signals in (25) and (26), in next section we are going to set the parameters to fixed values and assess how the control signal for the pursuer adapts.

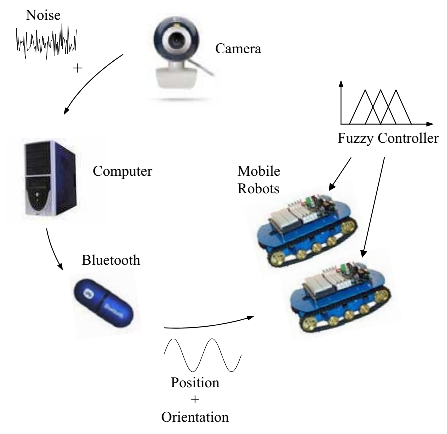

Fig. 3: Experimental system used for implementing the differential game

## VI. Experimental System

In order to implement the mathematical model of section V in an experimental set up we built some robots (depicted in figure 3 in our lab). This robot is equipped with a Motorola 68HC11 microprocessor and two motors that may drive the robot forward or backward. By changing the PWM duty cycle in each motor, it is also possible to turn the robot to the right and to the left.

Figure 3 actually depicts the whole system for the experiment and it may be divided in the following modules:

- Sensory - a camera and the filtering system coupled to it;
- Communication - after receiving the information from the sensors, it has to be sent to the robots;
- Controllers - implemented directly in the robots.

Equations (19) and (20) require that we know the positions and orientations of each one of the robots. Therefore, the camera has to measure the states of all the robots. So, a webcamera is used to read the positions (in pixels) and the orientations (in radians) of each robot. The positions and orientations are measured by the use of a colour code. Each robot has two rectangles over it as shown in figure 4. A system implemented in the computer station runs filters that locate the center of such rectangles. By making use of this data, the position and the orientation may be acquired. Obviously, the data is intrinsically noisy (gaussian white). However, the fuzzy controllers are able to deal with such noise. Moreover, the camera is not synchronous and the time elapsed between measurements is not constant. These are nonlinear effects (or noise) and the controllers have to deal with them as well.

The message is then sent from the camera to a computer. The computer, on its turn, assembles a packet with the positions and orientations of each robot. This message is finally sent to the mobile robots through a bluetooth link. Notice that both robots receive the same set of data. However, each one of them will then handle the information differently. There is no protocol that guarantees the delivery of the messages and

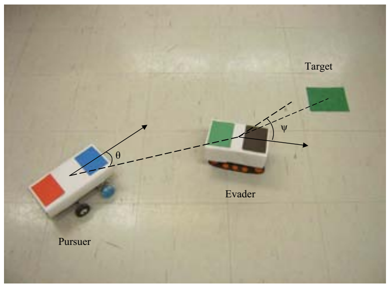

Fig. 4: An evader and a pursuer robots

some packets may not be received correctly. Indeed, this is the case in our experiments.

Both robots have embedded controllers that will determine the strategy that each one of them will execute. The pursuer is supposed to run the controller structure defined in Sect. III. The evader, however, runs a simpler control law that only guides it towards a fixed target as shown in figure 4.

The constants $V_e$, $V_p$, $R_e$ and $R_p$ in equations (19) and (20) are found by using a least squares algorithm and then are supposed to be constant for the rest of the experiment.

## VII. Experiments

In this section we are going to assess how the structure presented in sections III and IV may be used to solve the problem described in section V and VI. Again, we assume that just the pursuer adapts its strategy while the evader plays a *hardwired* optimal strategy. It is expected that eventually the one who is learning will improve its response for capture time in (24).

Let us start by finding the values of the parameters of the model presented in section V according to the identification of the system. Calculating the steady state value for the speed of the difference equations presented in (19) and (20) for a unit step input, we find that $V_e = 11.235$ and $V_p = 19.800$. In the same way, finding the steady state values for $\omega_e$ and $\omega_p$ for unit step inputs, we find that $R_e = 49.17$ and $R_p = 100.56$. Therefore, the pursuer is almost twice as fast as the evader; and the radius of turning of the pursuer is twice that of the evader. Notice that each one knows the (noisy) orientation of the other by checking its evolution over time.

The reinforcement function $r(\cdot)$ in figure 1 returns how well the control actuated so that the game came closer (or farther) to a solution. Since the objective of the game for the pursuer is to minimize the time to capture and capture is related to distance, it is clear that the reinforcement should be related to the variation of distance. Let us then define a function distance

$$D(\bar{x'}) = \sqrt{x'^2 + y'^2} \qquad (27)$$

and based on this function, let us define a variation of the distance

$$\Delta D(\bar{x'}(k+1)) \approx D(\bar{x'}(k+1)) - D(\bar{x'}(k)) \qquad (28)$$

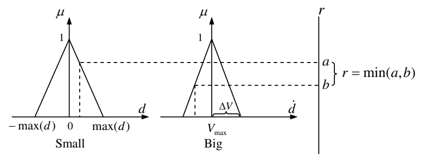

Fig. 5: Sets for calculation of the reinforcement signal

Observe that we can define, based on (28), an approximation of the derivative of the distance when the step size $\Delta t$ is small (which is our case) such that

$$\dot{D}(\bar{x'}(k+1)) = \frac{\Delta D(\bar{x'}(k+1))}{\Delta t} \qquad (29)$$

The *reinforcement* is then the output of a fuzzy system with only one rule [17]

$$r(k+1) = \min[\mu_{small}(\Delta D(k+1)), \mu_{big}(\dot{D}(k+1))] \quad (30)$$

As previously, the membership functions for these two cases are triangular. An example of such functions is depicted in Fig. 5. The center of the derivative of the distance is the maximum approach speed, calculated as the difference $V_e - V_p$ (equations (19) and (20)).

The reinforcement signal in (30) means that if the pursuer is getting closer to the evader at the maximum possible speed, the reinforcement is improved. Notice that if the pursuer is approaching the evader at $V_{max}$ in figure 5, the reinforcement signal is determined only by the distance. Also, notice that the reinforcement is not simply an error signal as in [11] and [17]. The distance and its derivative alone cannot represent error. Also, usually one requires errors to go to zero and in the case presented in (30), this is not what we want, since we require the approach speed to be the maximum. Furthermore, the pursuer does not have a "desired" path to follow, just a "desired" behaviour (catching the evader).

The input variables for the controller are the angle difference $\epsilon = \theta - \psi$ (both angles defined in Fig. 2) and its derivative, i.e., $\dot{\epsilon}$. In order to define the fuzzy controller, $\epsilon$ and $\dot{\epsilon}$ are set to be our *fuzzy variables*. Notice that this defines a "PD-like" type of control. Each one of the fuzzy variables has five fuzzy sets labeled: negative big (NB), negative small (NS), zero (ZE), positive small (PS) and positive big (PB). In order to avoid too much noise in the calculation of the derivative $\dot{\epsilon}$, a low-pass filter was implemented for the camera. Since the time step is not constant, the noise is amplified and outliers may become common, making it more difficult for the fuzzy controller to be efficient. The filter implemented is a simple discrete running average low-pass filter.

The evaluator has the same inputs as the controller. The output is the predicted reinforcement for the game that the evaluator seeks to approximate. In order to avoid too much time for the controller to converge due to errors in the evaluator, it is advisable to perform an off line training for a previous convergence of the evaluator [17]. This is just an introductory learning phase and we simulate 100 instances

TABLE I: Initial values for the control signal

| | angle rate | | | | |
|---|---|---|---|---|---|
| **angle** | NB | NS | ZE | PS | PB |
| NB | 1.5000 | 1.0000 | 0.5000 | 0.2500 | 0.2500 |
| NS | 1.0000 | 0.5000 | 0.2500 | 0.2500 | 0.1250 |
| ZE | 1.5000 | 1.0000 | 0 | -1.0000 | -1.5000 |
| PS | -0.1250 | -0.2500 | -0.2500 | -0.5000 | -1.0000 |
| PB | -0.2500 | -0.2500 | -0.5000 | -1.0000 | -1.5000 |

TABLE II: Final values for the control signal

| | angle rate | | | | |
|---|---|---|---|---|---|
| **angle** | NB | NS | ZE | PS | PB |
| NB | 1.3592 | 0.9442 | 0.6582 | 0.7440 | 0.2500 |
| NS | 0.9783 | 0.4072 | 2.2847 | 2.2105 | 0.1250 |
| ZE | 1.4988 | 0.4933 | -0.3137 | -0.4743 | -1.4989 |
| PS | -0.1250 | -1.7160 | -1.9857 | -0.4134 | -0.9737 |
| PB | -0.2500 | -0.7555 | -0.6608 | -0.9753 | -1.3260 |

of the game for this to take place. During this phase, the learning rate $\alpha$ in (17) is set to $0.1$ for a fast adaptation. Also, throughout the simulations we use $\gamma$ in (14) as $0.95$ that is a small forgetting factor, meaning the evaluator uses only around $20$ past signals for adaptation.

The initial control surface is shown in Fig. 6. This figure represents the fuzzy table shown in table I.

We then run the game with the robots with random initial positions that satisfy the constraints of the game. Namely,

- the distance from the evader to the target is smaller than the distance from the pursuer to the target;
- the evader is supposed to be in front of the pursuer, i.e., angle $\theta$ in Fig. 2 is supposed to be in the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$;
- in the initial positions, if the optimal solution is played, the pursuer is able to catch the evader before it reaches the target.

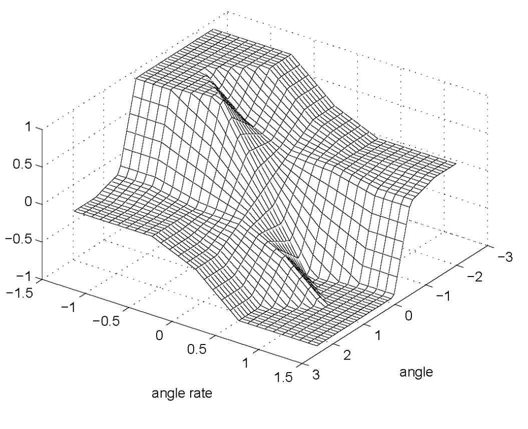

Fig. 6: Initial control surface of the fuzzy controller

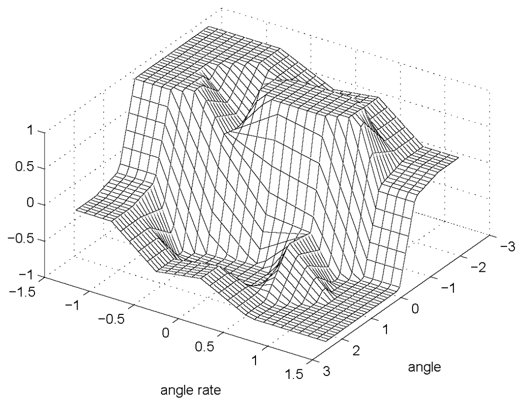

Fig. 7: Control surface of the fuzzy controller after learning

The learning rate $\beta$ in (17) is set to $0.01$. At the same time, the evaluator continues to adapt at a learning rate of $0.1$. After learning takes place, the control surface changes to the one shown in Fig. 7 that corresponds to the table II.

The surface shown in Fig. 7 changes in a very important way if compared to the initial control surface in Fig. 6. First of all, the area where the control signal is on the bounds, i.e., $\delta_p = \{-1, 1\}$, is significantly increased. Since we know from (25) that the optimal solution is in the set $\{-1, 0, 1\}$, this is very suggestive. (Although, notice that we do not use this information in our training). Moreover, the inclination of the curve changes considerably. It is much sharper in Fig. 7. Also, the absolute value of the control signal increases or remains the same at every point of the surface, thus making the pursuer catch the evader more quickly.

Let us now see how effective the learning is. Let us define initial conditions for the evader ($x_e = 50$, $y_e = 110$ and $\theta_e = 0^o$) and for the pursuer ($x_p = 15$, $y_p = 30$ and $\theta_p = 72^o$) that were never presented during the training phase. Figure 8 shows that with the initial controller of table I the pursuer does not intercept the evader before the evader reaches the target. In fact, the pursuer "overshoots" the evader due to its higher speed. As it is not able to turn sharply, it misses the evader. In Fig. 9, however, we show the pursuer catching the evader after learning takes place and it uses the learnt values of table II.

One advantage of this method compared to genetic algorithms, for example, is that the convergence happens in a much faster way. Learning takes, in total, only 50 experiments with

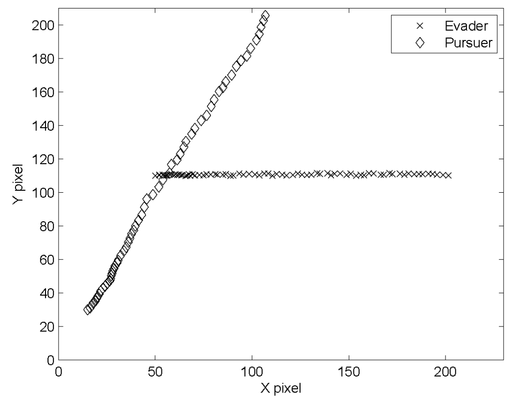

Fig. 8: Pursuer not able to catch the evader before learning

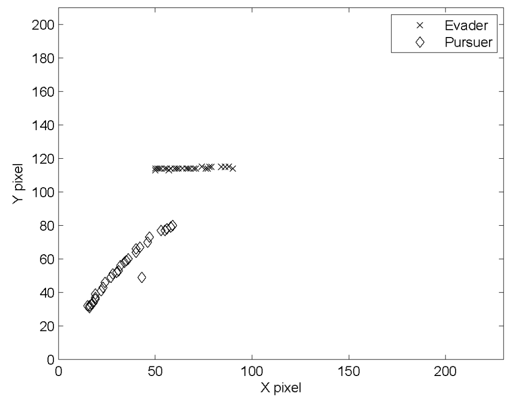

Fig. 9: Pursuer catching the evader after learning

random initial states.

## VIII. Conclusion

In this paper we showed a method for learning in differential games. A fuzzy controller adapted by reinforcement learning is presented. An architecture derived from [11] and [17] is shown to be suitable for learning in a continuous environment.

An experiment of a modified version of the *game of two cars* is described. We suppose that only one of the players (the pursuer) adapts its behaviour. The evader is supposed to play its optimal strategy. Results show that the pursuer learns to catch the evader in an effective way. The controller implemented in actual robots using the scheme described in section IV is superior to the initial controller. Also, we show that the controller is resilient to noise. Moreover, adaptation may continue in order to deal with changes in the robots' dynamics.

## References

[1] R. B. Myerson, *Game Theory: Analysis of Conflict*. Cambridge, Massachusetts: Harvard University Press, 1991.
[2] J. von Neumann and O. Morgenstern, *The Theory of Games and Economic Behavior*, 2nd ed. Princeton: Princeton University Press, 1947.
[3] R. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. New York, New York: John Wiley and Sons, Inc., 1965.
[4] D. W. K. Yeung and L. A. Petrosyan, *Cooperative Stochastic Differential Games*. New York, NY: Springer Science+Business Media, Inc., 2006.
[5] D. Fudenberg and D. K. Levine, *The Theory of Learning in Games*. Cambridge, Massachusetts: The MIT Press, 1998.
[6] J. W. Weibull, *Evolutionary Game Theory*. Cambridge, Massachusetts: MIT Press, 1995.
[7] J. Hofbauer and K. Sigmund, "Evolutionary game dynamics," *Bulletin of the American Mathematical Society*, vol. 40, no. 4, pp. 479–519, 2003.
[8] J. Conlisk, "Adaptation in games: two solutions to the crawford puzzle," *Journal of Economic Behaviour and Organization*, vol. 22, pp. 25–50, 1993.
[9] M. E. Harmon, L. C. B. III, and A. H. Klopf, "Reinforcement learning applied to a differential game," *Adaptive Behavior*, vol. 4, no. 1, pp. 3–28, 1995.
[10] M. E. Harmon and L. C. B. III, "Residual advantage learning applied to a differential game," in *Proceedings of the International Conference on Neural Networks*, 1996, pp. 1–6.
[11] X. Dai, C. Li, and A. B. Rad, "An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 3, pp. 285–293, 2005.
[12] L. X. Wang, *A Course in Fuzzy Systems and Control*. Englewood Cliffs, NJ: Prentice Hall, 1997.
[13] M. Andrecut and M. K. Ali, "Fuzzy reinforcement learning," *International Journal of Modern Physics C*, vol. 13, no. 5, pp. 659–674, 2002.
[14] A. W. Starr and Y. C. Ho, "Nonzero-sum differential games," *Journal of Optimization Theory and Applications*, vol. 3, no. 3, pp. 184–206, 1969.
[15] H. Ishibuchi, R. Sakamoto, and T. Nakashima, "Learning fuzzy rules from iterative execution of games," *Fuzzy Sets and Systems*, vol. 135, pp. 213–240, 2003.
[16] A. E. Bryson and Y. Ho, Eds., *Applied Optimal Control: Optimization, Estimation, and Control*. Levittown, PA: Taylor & Francis, 1975, rev. printing.
[17] W. M. Buijtenen, G. Schram, and R. B. an H. B. Verbruggen, "Adaptive fuzzy control of satellite attitude by reinforcement learning," *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 2, pp. 185–194, 1998.
[18] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," *IEEE Transactions on Systems, man, Cybernetics*, vol. 15, pp. 116–132, 1985.
[19] G. Arslan and J. S. Shamma, "Anticipatory learning in general evolutionary games," in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006, pp. 6289–6294.
[20] J. W. Sheppard, "Colearning in differential games," *Machine Learning*, vol. 33, pp. 201–233, 1998.
[21] A. W. Merz, "The homicidal chauffeur," *AIAA Journal*, vol. 12, no. 3, pp. 259–260, 1974.