J Intell Robot Syst (2010) 59:3–30
DOI 10.1007/s10846-009-9380-4

A Reinforcement Learning Adaptive Fuzzy Controller for Differential Games

Sidney N. Givigi Jr. · Howard M. Schwartz · Xiaosong Lu

Received: 4 June 2008 / Accepted: 28 September 2009 / Published online: 23 October 2009

Abstract In this paper we develop a reinforcement fuzzy learning scheme for robots playing a differential game. Differential games are games played in continuous time, with continuous states and actions. Fuzzy controllers are used to approximate the calculation of future reinforcements of the game due to actions taken at a specific time. If an immediate reinforcement reward function is defined, we may use a fuzzy system to tell what is the predicted reinforcement in a specified time ahead. This reinforcement is then used to adapt a fuzzy controller that stores the experience accumulated by the player. Simulations of a modified two car game are provided in order to show the potentiality of the technique. Experiments are performed in order to validate the method. Finally, it should be noted that although the game used as an example involves only two players, the technique may also be used in a multi-game environment.

Keywords Differential games · Learning · Pursuer-evader games · Intelligent systems · Reinforcement learning · Fuzzy control

S. N. Givigi Jr. (✉)
Department of Electrical and Computer Engineering, Royal Military College of Canada,
P.O. Box 17000 Station Forces, Kingston, Ontario K7K 7B4, Canada
e-mail: Sidney.Givigi@rmc.ca

H. M. Schwartz · X. Lu
Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive,
Ottawa, Ontario K1S 5B6, Canada

H. M. Schwartz
e-mail: schwartz@sce.carleton.ca

X. Lu
e-mail: luxiaos@sce.carleton.ca

1 Introduction

Learning in games has been largely studied in the last couple of decades. It already includes several books [7, 24] and recent research papers abound in the specialized literature [5, 13]. However, little attention has been given to learning in the differential game domain [12].

Game theory is basically the study of decision making [18] in order to solve conflicts. It was introduced by von Neumann and Morgenstern [19]. 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 [14] and is called *differential games*. Differential games investigate how decision making takes place over time [25] considering continuous domains, for example, 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.

When we approach a game from the point of view of learning, the question we want to answer is if the game *converges* to an equilibrium point. This point may be the *Nash* equilibrium or the *value* of the game, but not necessarily.

In the case of differential games, one of the most popular learning approaches has been the use of reinforcement learning and Q-learning [11, 12]. However, there is a disadvantage in this technique when we deal with continuous processes such as the ones considered in differential games. Since Q-learning is 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 infeasible for implementation in a microcontroller. Moreover, it is not easy to discretize the action space as well as the state space [6].

In order to avoid this problem, one could use a fuzzy controller. It is well known that a fuzzy system is a universal approximator [23]. 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 [23] and uncertainties [1].
- Reinforcement learning updates is an adequate way of updating the fuzzy rules on line [1].
- We could use a technique based on neural networks or genetic algorithms to extract rules from data and then apply the approach proposed in order to speed up the convergence of the controller.
- A fuzzy controller such as presented here could be easily implemented in a microcontroller.

We may think of the pursuit-evasion differential game in two different scenarios. The first is the typical one-on-one game, i.e., there is only one pursuer and one evader. This game has been extensively studied in the literature [14, 17, 21]. However, little attention has been given to how a player may learn how to play the game. The other more complicated scenario is the multiplayer pursuit-evasion differential

game, wherein there may be several pursuers and/or evaders. This is a less studied case, but interesting results have also been published [8, 9]. In these papers, a hierarchical structure is proposed which is based on numerical or analytical solution of the pursuit-evasion game. Our learning approach, however, does not dwell on the mathematical solution of the games. We are interested in the players to "learn" how to play a game as it has been done in traditional game theory [2, 16]. Therefore, our fuzzy controller obtained after training (or even during the execution of the task) would determine the behaviour of the robot. Moreover, it must be noted that the approach presented in the present article is not only applicable to pursuit-evasion games. Other dynamic games would have rigorously the same type of learning algorithm presented in here.

The paper is divided as follows. Section 2 introduces the notation of a differential game. In Section 3 we present the control structure for the system. Section 4 reviews the learning techniques used in games in general and introduces a fuzzy algorithm for learning in differential games. In Section 5, we describe the system that will be used in the simulations and experiments as well as how it relates to the notation in Section 2. The hardware and software framework will also be introduced in this section. Section 6 reports the identification of the robots' models. In Section 7, simulations of the system identified in the previous sections are presented. Section 8 presents the experimental results reached with the controller derived in the simulation section. And, finally, Section 9 presents our conclusions from the simulations and experiments and points to future work that will be done in the field.

2 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 [21]

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

subject to the state dynamics

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

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 $i \in N$.

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 are also called the *constraints* of the system and could be represented by inequalities [21]. When solving a game theoretically, one also assumes that one agent (or player) has access to the states of the other players involved in the game at all times as well as their cost functions. This is called the *perfect information* assumption. For the learning procedure presented in this work, though, we do not assume that each player knows the cost functions of all the others. Also, notice that the *perfect information* assumption is not necessary for the system to learn since we are using a fuzzy system that is intrinsically designed to deal with noise.

Our only assumption is that all the robots have the same dynamics. Lastly, one must notice that the game may have different solutions and the achievement of the optimal solution is not guaranteed for the general case.

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

$$\dot{\bar{x}}(s) = f\left(\bar{x}(s), \bar{\phi}(s), \bar{\psi}(s), s\right), \quad \bar{x}(t_0) = \bar{x}_0 \tag{3}$$

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

$$P\left(\bar{\phi}, \bar{\psi}\right) = q\left(t^*, \bar{x}(t^*)\right) + \int_{t_0}^{t^*} g\left(\bar{x}(s), \bar{\phi}, \bar{\psi}, s\right) ds \tag{4}$$

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 $\bar{\phi}$ wants to maximize the payoff $P(\cdot)$, whereas the player using strategy $\bar{\psi}$ wants to minimize it. Therefore, the objective of the game is to find control signals $\bar{\phi}^*$ and $\bar{\psi}^*$ such that [3]

$$P\left(\bar{\phi}^*, \bar{\psi}\right) \geq P\left(\bar{\phi}^*, \bar{\psi}^*\right) \geq P\left(\bar{\phi}, \bar{\psi}^*\right), \quad \forall \bar{\phi}, \ \bar{\psi} \tag{5}$$

In Section 5 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 3 and to establish in Section 4 how a robot could learn how to play a game using a fuzzy inference system.

3 Controller Structure

In Fig. 1 we show the proposed structure for the controller [4].

In this section, we are going to focus on the structure of each block, more specifically the *controller* and the *critic*. The *reinforcement* block is particular to applications and will be dealt with in Section 7. Also, the learning procedures will be postponed until the next section.

We assume that the *controller* in Fig. 1 is a fuzzy controller. More specifically, it is a fuzzy controller implemented by Takagi-Sugeno (TS) rules with constant

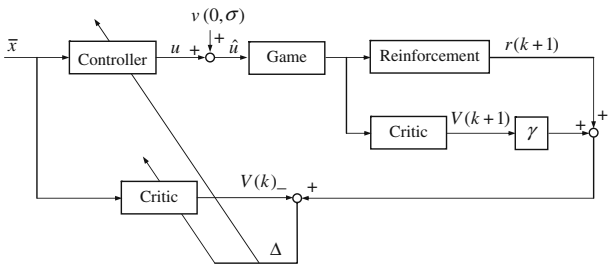

Fig. 1 Architecture of the control system

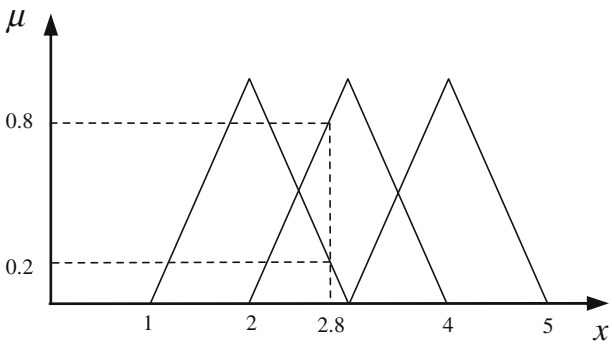

Fig. 2 Triangular membership functions used

consequents [22]. It consists of M rules with n fuzzy variables as inputs and one constant number as consequent. Therefore, each rule is of the form [6]

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

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

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 [23], t norm, singleton fuzzifier and center-average defuzzifier, the output of the system is [6]

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

where c_l in Eq. 7 is represented by χ_l for the controller.

Throughout the paper, the membership functions used are triangular ones such as the ones depicted in Fig. 2.

For the *critic*, we also assume a TS system with constant consequents [4]. However, it must be noted that this is not the only possible choice. A time-delay neural network (TDNN) could be used instead [6]. 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 Eq. 8, the output of the critic is [4] an approximation to the value of the state $V(\cdot)$

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

where c_l in Eq. 7 is represented by ζ_l.

4 Learning

In the context of Game Theory, learning may be understood as strategy adjustments [2]. These adjustments may be functions 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 [2] 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 Eqs. 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

$$\dot{\bar{\delta}}_1 = f_{\bar{\delta}_1}(\bar{x}, \bar{\delta}_1, \bar{\delta}_2) \tag{10}$$

$$\dot{\bar{\delta}}_2 = f_{\bar{\delta}_2}(\bar{x}, \bar{\delta}_1, \bar{\delta}_2) \tag{11}$$

Notice, also, that the functions $f_{\bar{\delta}_1}(\cdot)$ and $f_{\bar{\delta}_2}(\cdot)$ must take into consideration the cost Eq. 4 in such a way that the *saddle point* (for the case of a two-player game) in Eq. 5 is reached. Also, the strategy adjustments in Eqs. 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 strategies (the control signal) played by all the other players, although they do not know how they chose those strategies.

The problem of Eqs. 10 and 11 is that the strategies played by each player are continuous. This means that the strategy vectors $\bar{\delta}_1$ and $\bar{\delta}_2$ 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 easily done for most differential games. In order to avoid these potential problems, we use the architecture shown in Fig. 1.

In this figure we see the addition of two blocks called *critic*. This block is used to approximate the value function for reinforcement learning [6]. It could be implemented by a time-delay neural network [6] or a fuzzy system [4]. The learning is practically the same for both cases. The value function for approximation of the reinforcement rewards has the format

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

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 Eq. 12 in a recursive fashion as

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

With this approximation, we may compare it with the expected reward such that we generate a prediction error of the prediction $\hat{V}(k)$ [4] that is the output of the critic so that

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

as shown in Fig. 1. This difference error is then used to train the critic. Supposing it has parameters ζ to be adapted, the adaptation law would then be [6]

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

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

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

The controller in Fig. 1 is a fuzzy controller implemented by Takagi-Sugeno rules with constant consequents [22]. 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 [6]. With the noisy signal $u'(t)$, taking χ as the parameters to be adapted, we may establish the controller update law [4]

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

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 critic. 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 Eq. 17 is easily calculated to be from Eq. 8

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

Notice that the partial derivatives in Eqs. 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 Eqs. 15 and 17 are based on a "forecasted" improvement of the cost function (Eq. 4).

In the next section, we describe the game we will solve with the technique described so far. In Section 7 we present the particular case of the equations presented in this section to the problem under study.

5 Pursuer Evader Model

In this section we will introduce the model used in the coming simulations and experiments. The section will be divided in two subsections. In the first one, we will describe the general mathematical model for the robots and the game itself. The second subsection will describe the hardware involved in the experiments, consisting of the communication framework and how the robots were constructed.

5.1 Mathematical Model

Let us assume that we have a game where two robots are present. One of them is called the *pursuer* and tries to catch another one, called the *evader*. The model (kinematic equations) for the pursuer may be represented by

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

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

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

In Eqs. 19 and 20, x_i and y_i, $i = \{e, p\}$ are the positions of the robots; V_p is the speed of the pursuer, V_e is the speed of the evader, and $V_p > V_e$; θ_p is the orientation of the pursuer and $|\theta_p| < \pi$; θ_e is the orientation of the evader and $|\theta_e| < \pi$; R_p is the rate of turn for the pursuer, R_e is the rate of turning of the evader, and $R_p > R_e$ and they are such that $\frac{V_p}{R_p} < \frac{V_e}{R_e}$. Finally, $|\delta_p| \leq 1$ 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. This game is known as the "game of two cars" [14].

Let us now assume a coordinate frame centered in the pursuer with its y'-axis in the direction of the pursuer's velocity vector [3] as shown in Fig. 3. The pair (x', y') is the relative position of the evader in this coordinate frame [3].

If we define $\psi = \theta_e - \theta_p$, then the differential equations for this game are of the form

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

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

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

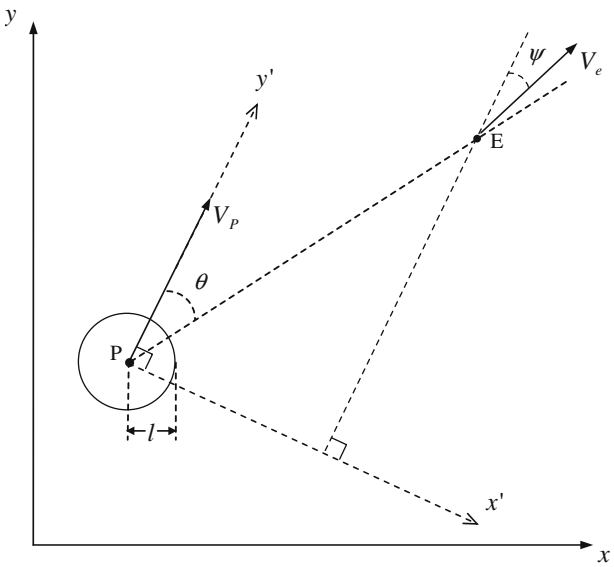

Fig. 3 Relative position for the evader with respect to the pursuer

Equations 21 to 23 describe how the vector from the point P in Fig. 3 to point E in the same figure behaves over time. Let us now define that interception happens when

$$x'^2 + y'^2 \leq l^2 \tag{24}$$

where l is an arbitrarily defined distance from the pursuer to the evader.

The cost equation (Eq. 4) in this particular case is such that

$$P(\delta_p, \delta_e) = \left(x'^2 + y'^2\right)|_{t=t_f} + \int_{t_0}^{t_f} 1 ds \tag{25}$$

Therefore, the functions $q(\cdot)$ and $g(\cdot)$ in Eq. 4 are

$$q\left(\bar{x'}(T)\right) = x'^2 + y'^2 \tag{26}$$

$$g\left(\bar{x'}(s), \delta_p(s), \delta_e(s), s\right) = 1 \tag{27}$$

The Hamiltonian is then found as [3]

$$H = \lambda_{x'}\dot{x'} + \lambda_{y'}\dot{y'} + \lambda_{\psi}\dot{\psi} + 1 \tag{28}$$

where $\lambda_{x'}$, $\lambda_{y'}$ and λ_{ψ} are lagrange multipliers.

By solving the hamiltonian, we may find the optimal play. Another approach to find the best capture time is by solving the similar equation [17]

$$\begin{aligned}\min_{\phi}\max_{\psi}\left(\frac{dP}{dt}\right) = \min_{\phi}\max_{\psi}\Bigg[&P_{x'}\left(V_e\sin\psi - \frac{V_p}{R}y'\phi\right)\\ &P_{y'}\left(V_e\cos\psi - V_p + \frac{V_p}{R}x'\phi\right)\\ &P_{\psi}\left(-\frac{V_p}{R_p}\delta_p + \frac{V_e}{R_e}\delta_e\right)\Bigg]\\ &= -1 \end{aligned} \tag{29}$$

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

Solving either way, the optimal control for the pursuer and evader may be found (given some constraints) such that [3]

$$\delta_p = \operatorname{sgn}(\lambda_{x'}y' - \lambda_{y'}x' + \lambda_{\psi}) \tag{30}$$

$$\delta_e = \operatorname{sgn}(\lambda_{\psi}) \tag{31}$$

If we define some constraints and values for the parameters in Eqs. 19 and 20, the solution may be further simplified, but this is not our objective in here. The interested reader is referred to Isaacs [14] and Bryson and Ho [3] for more details.

However, for the experiment we are going to change the objectives of the game. The reason for that is due to the limitations of the experimental setup to be discussed in the next section. A game of two cars would tend to last longer and run farther than the range that our sensors would allow. Therefore, we are going to use a variation where there is a target that the evader tries to reach while the pursuer tries to intercept the evader. This is depicted in Fig. 4. The pursuer is also supposed to be farther to the target than the evader. The strategy that the evader follows is to get

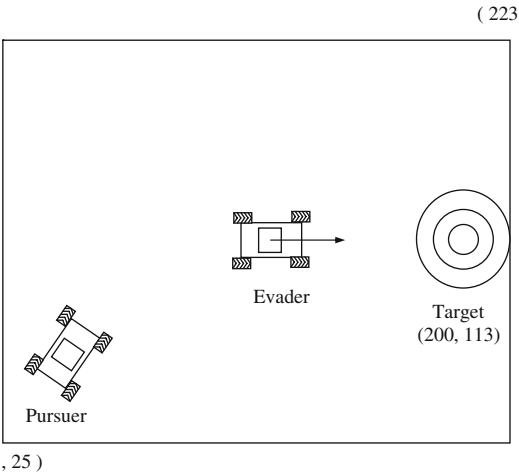

Fig. 4 Conceptualization frame where the game is played

the closest to the target as possible. In our simulations, the pursuer is assumed to not know the dynamics of the evader and its strategy.

Some remarks may be made with regard to Eqs. 19 and 20. First, notice that the transfer functions for θ_p and θ_e describe an integrator. This is not quite the case, but, as it will be discussed in Section 6, this is a good assumption. Moreover, in an actual system, V_p and V_e are not constant. There is some transient response that must be taken into account. Therefore, in Section 6 we will also identify equations for this transient response, such that we will have

$$\frac{V_p(s)}{u_p(s)} = \frac{b_p}{s + a_p} \tag{32}$$

$$\frac{V_e(s)}{u_e(s)} = \frac{b_e}{s + a_e} \tag{33}$$

Furthermore, we are actually going to identify the transfer function for the angular velocity instead of the one for the angle. This means that, for the pursuer, we will identify the transfer function

$$\frac{\omega_p(s)}{\delta(s)} = \frac{\frac{V_p}{R_p}}{s + a_{\omega_p}} \tag{34}$$

The same identification procedure will be performed for the evader, resulting in

$$\frac{\omega_e(s)}{\delta(s)} = \frac{\frac{V_e}{R_e}}{s + a_{\omega_e}} \tag{35}$$

However, as it will be seen in Section 6 the poles of Eqs. 34 and 35 will be very "fast" and the transfer functions will be similar to an integrator, which makes the approximations in Eqs. 19 and 20 valid.

5.2 Experimental System

We want to implement the mathematical model of the previous subsection in an experimental set up. In order to do that, we built some robots in our lab. Figure 5 depicts one individual robot that may be used as pursuer or evader.

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.

Fig. 5 An individual robot

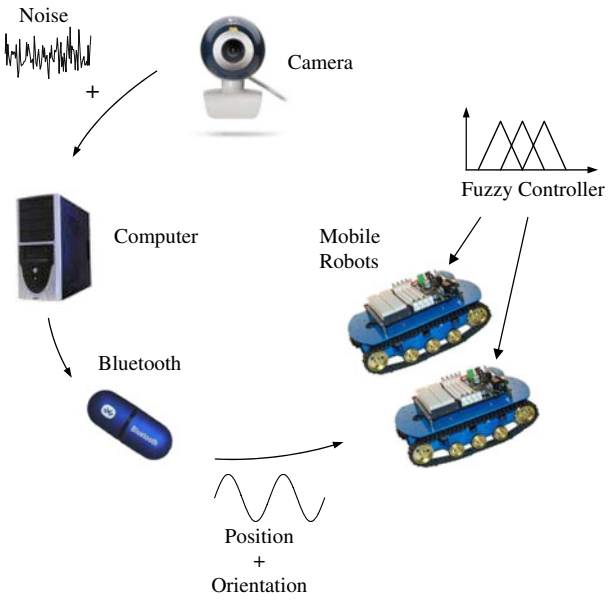

Fig. 6 Experimental system used for implementing the differential game

The whole system for the experiment is shown in Fig. 6 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.

The first thing that must be decided is the information that is required to be measured. 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

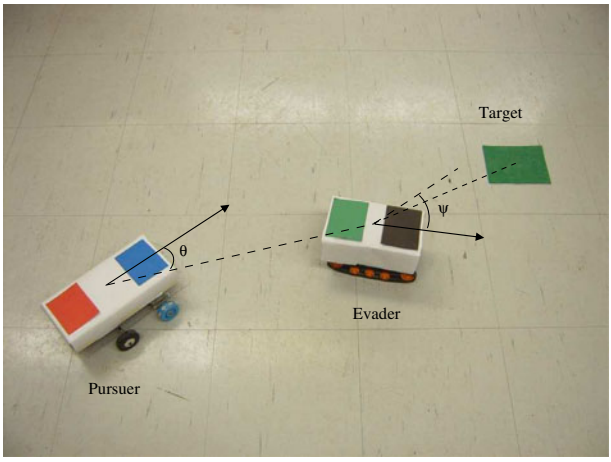

Fig. 7 An evader and a pursuer robots

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 Fig. 7. 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. However, it is supposed that the controllers may deal with such noise. Also, we assume that such noise is gaussian white. Moreover, the camera is not synchronous and the time elapsed between measurements is not constant. These are supposed to be nonlinear effects (or noise) and we suppose the controllers have to deal with it 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 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 Section 3. The evader, however, runs a simpler control law that only guides it towards a fixed target as shown in Fig. 7. Notice also that the measurements of the angles depicted in Fig. 7 are also noisy.

In order to simulate the Eqs. 19 and 20, we have to first identify the transfer functions for the angle variation and the speed of each robot. This will be done in the next section.

6 Identification

In order to simulate the system of equations presented in Section 5.1, we need to identify the parameters of the equations. Namely, we need to know what are the

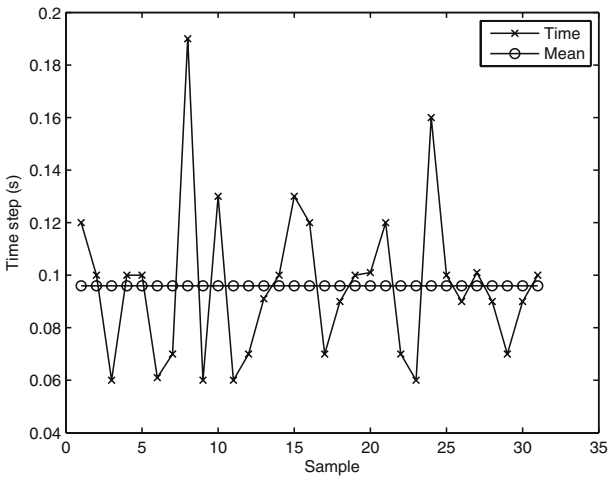

Fig. 8 Time in one arbitrary run of a robot

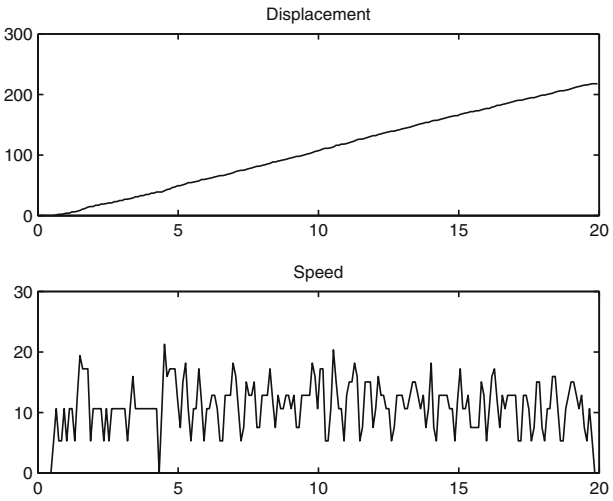

Fig. 9 Displacement for one arbitrary run of a robot

velocities V_p in Eq. 19 and V_e in Eq. 20. Moreover, we need to determine the differential equations for θ_p in Eq. 19 and θ_e in Eq. 20.

The necessity of having a model is that we are going to pre-train the controller offline in order to have some useful control signal for our experiments. Clearly, the modeling is only approximate. This is not a big problem, for a fuzzy controller should be able to deal with such uncertainties [23]. Moreover, we assume that the dynamics of the robots change slowly and smoothly over time. Therefore, the online adaptation would take care of these small variations.

For each robot, we are going to identify two difference equations that determine how the robot behaves. The first one is how the forward velocity of the robot changes when a nominal signal $u(t)$ is applied to both motors. The second one is related to the angular velocity when a specific differential signal δ is applied to the motors.

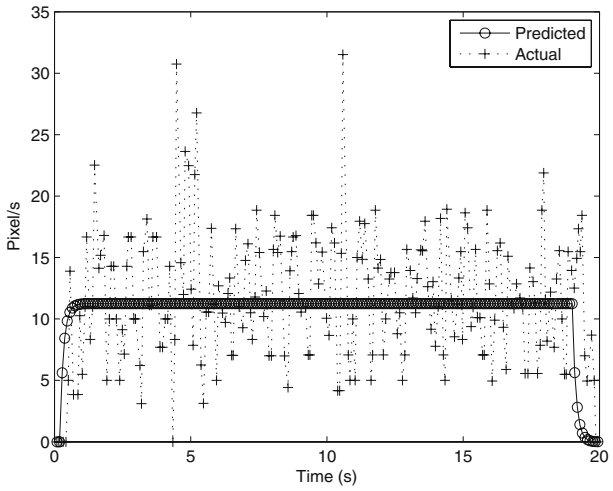

Fig. 10 Identified system and actual collected data

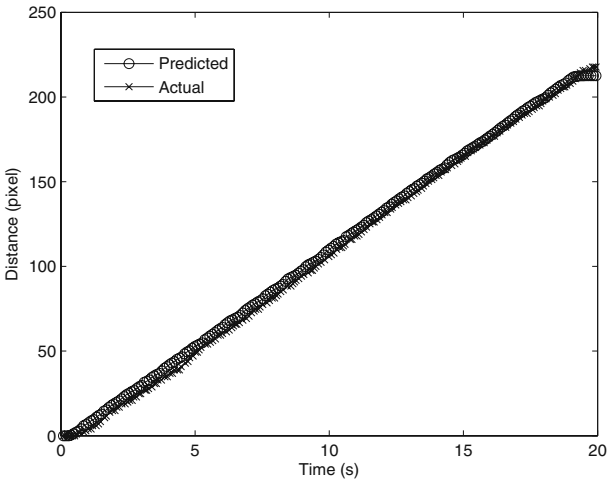

Fig. 11 Simulation of the identified system

The relationship between the signals $u(t)$ and $\delta(t)$ is as follows. Let us suppose that we decide to drive the robot around a nominal input of $u(t) = 50$, where 50 is related to the duty cycle of the PWM. The forward speed of the robot increases until it reaches a steady state. Then we decide to turn the robot to the right. In order to do this, we increase the duty cycle of the left motor by $\delta(t) = 25$, where 25 is an experimentally determined value, and decrease the duty cycle in the right motor by the same value, such that, if $u_R(\cdot)$ is the signal for the right motor and $u_L(\cdot)$ is the signal for the left one. Mathematically, the signals to each motor are represented by

$$u_R(t) = u(t) - \delta(t) \tag{36}$$

$$u_L(t) = u(t) + \delta(t) \tag{37}$$

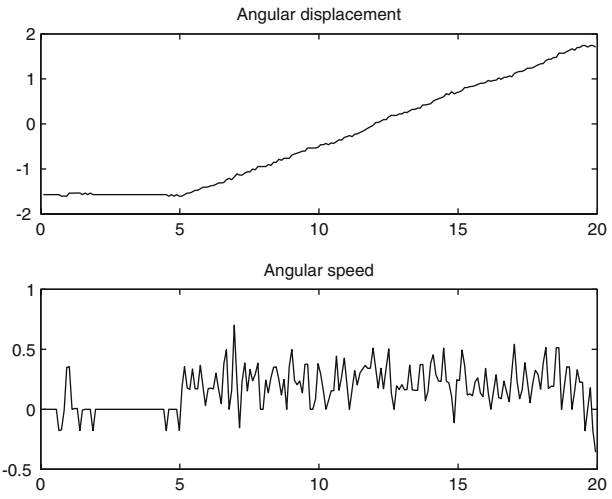

Fig. 12 Angular displacement for one arbitrary run of a robot

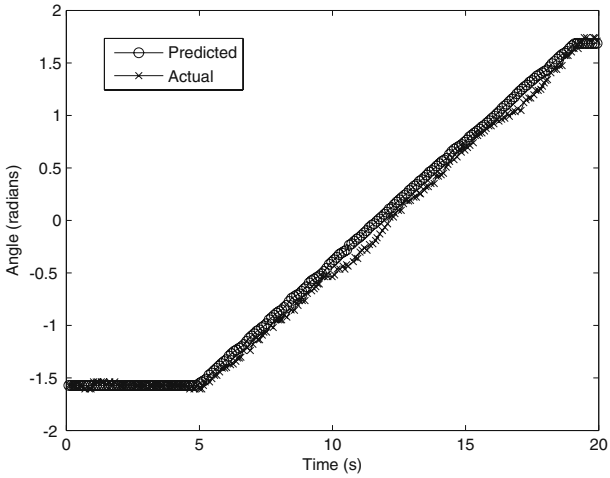

Fig. 13 Simulation of the identified system

Observe that the forward speed is unchanged by the changes of the signals of the motors, since the average input is still the same.

After some tests, we determined that a first order transfer function was enough to represent the behaviours listed above. For the forward velocity, we want to identify the parameters a_v and b_v of the difference equation

$$v[k+1] = a_v v[k] + b_v u[k] \tag{38}$$

Observe that the velocities V_p and V_e in Eqs. 19 and 20 are the steady state values of Eq. 38 for each one of the robots.

For the angular velocity, we want to identify the parameters a_ω and b_ω of the difference equation

$$\omega[k+1] = a_\omega \omega[k] + b_\omega \delta[k] \tag{39}$$

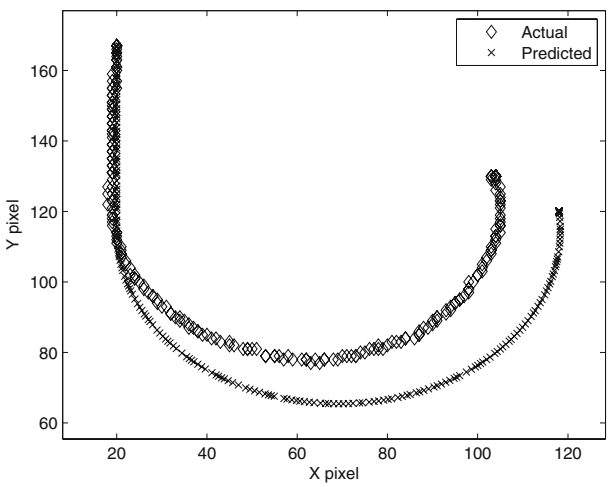

Fig. 14 Simulation of the identified system

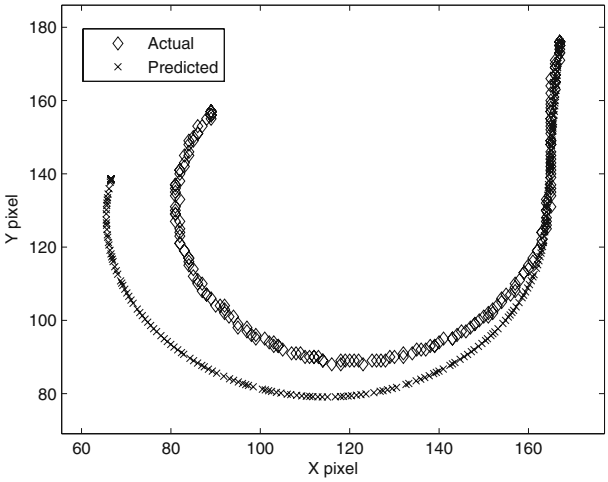

Fig. 15 Prediction of the position of the robot when turning right

The data collection is done through successive runs of the robots for different values of the signal $\delta(t)$. The nominal value for the motors $u(t)$ is the same for all the runs. Since, as discussed in the previous section, the time step for the measurements is not constant, the first step is to determine the mean value for this time. This is shown in Fig. 8. Notice that the time steps vary a lot. Any difference around the mean value depicted in this figure is considered as nonlinear noise. We assume that this noise will be small if compared to the linear behaviour of the system.

The data representing the forward displacement and the numerically calculated forward speed for one arbitrary run of a robot is shown in Fig. 9.

Then we use a least squares algorithm in order to identify the system. A figure with the predicted (identified) system and actual data is shown in Fig. 10.

We then simulate the identified system in order to compare its result with the actual data collected. Numerically calculating Eq. 38, and comparing it to the actual values read in the experiment, we have the results shown in Fig. 11.

The same steps have to be taken for the identification of Eq. 39. Let us take as an example a robot turning to the left. The data representing the angle over time is shown in Fig. 12.

After taking the derivative of the data and identifying the system with a least squares algorithm, we numerically integrate Eq. 39 (Eqs. 19 and 20 use the turning rate and not the angular speed) and get the data shown in Fig. 13. One may notice that the agreement is very good.

After doing the identification, one may numerically integrate Eqs. 19 and 20 and end up with the simulation shown in Fig. 14.

If we apply a signal for the robot to turn to the other side, we can predict its position as shown in Fig. 15.

Table 1 Difference equations for the pursuer

Variable	Difference equation
Angular speed	$\omega_e[k+1] = 0.3624\omega_e[k] + 0.1569\delta_p[k]$
Forward speed	$v_e[k+1] = 0.5380v_e[k] + 9.1477u[k]$

Table 2 Difference equations for the pursuer

Variable	Difference equation
Angular speed	$\omega_e[k+1] = 0.2243\omega_e[k] + 0.1775\delta_e[k]$
Forward speed	$v_e[k+1] = 0.5156v_e[k] + 5.4471u[k]$

Observe that in all the graphs, we use only the predicted value and do not use actual data to update the estimation. This means that if $y[k]$ is the actual data and $\hat{y}[k]$ its prediction, Figs. 11, 13 and 14 use the following relationship

$$\hat{y}[k+1] = a\hat{y}[k] + b\,u[k] \tag{40}$$

and not

$$\hat{y}[k+1] = ay[k] + b\,u[k] \tag{41}$$

As such, the difference seen in Figs. 14 and 15 represent the integration of a very small modeling error.

Therefore, the agreement is indeed very good and the equations found may be used to simulate the system to a very good degree of accuracy.

Table 1 summarizes the difference equations found for the pursuer robot using the procedure discussed in this section. Table 2 summarizes the same information for the evader robot. In both cases the sampling period is 0.09 s. In the next section we are going to simulate the system found in order to train a fuzzy controller that will drive the pursuer robot to catch the evader robot.

7 Simulation

In this section we are going to assess how the structure presented in Sections 3 and 4 may be used to solve the problem described in Section 5. For this evaluation, we suppose that the evader plays the optimal control to get to the target disregarding the position of the pursuer. In fact, this may be shown to be the optimal solution of the game given the constraints on the initial conditions of the game to be described. Since just one player adapts its strategy while the other plays its optimal strategy, it is expected that eventually the one who is learning will improve its response for capture time in Eq. 29.

Let us start by assigning values to the parameters of the model presented in Section 5 according to the identification done in Section 6. Calculating the steady state value for the speed of the difference equations presented in Tables 1 and 2 for a unit step input (i.e., $u[k] = 1$), we find that V_e and V_p in Eqs. 20 and 19 are $V_e = 11.235$ and $V_p = 19.800$. In the same way, finding the steady state values for ω_e and ω_p for unit step inputs ($\delta_p[k] = \delta_e[k] = 1$), we find that $\frac{V_e}{R_e} = 0.2285$ and $\frac{V_p}{R_p} = 0.1969$. One may calculate then 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. Moreover, we remind the reader that the control signal for the pursuer and the evader are bounded and dependent on some internal dynamics; however, the pursuer is faster than the evader, who, on the other hand, can turn sharper than the pursuer. Also, the pursuer knows the orientation of the evader by checking its evolution over time. In other words, it may very well be noisy. The

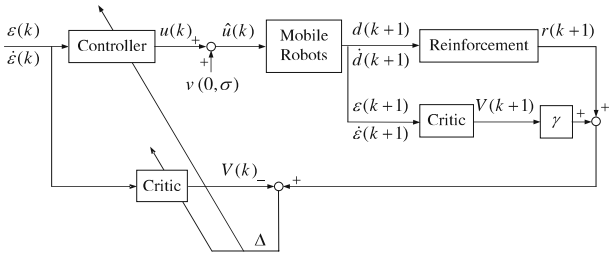

Fig. 16 Architecture of the pursuer-evader model system

evader, on the other hand, has perfect information on the states of the pursuer. Of course, this last statement is only true for the simulation and not for the experiments.

Let us now redefine the architecture defined in Fig. 1 in order to make it particular to the problem of Section 5. The new structure is shown in Fig. 16. First let us consider the reinforcement function $r(\cdot)$. This is the function that 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

$$D(\bar{x}') = \sqrt{x'^2 + y'^2} \tag{42}$$

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

$$\Delta D\left(\bar{x}'\right) \approx D\left(\bar{x}'(k+1)\right) - D\left(\bar{x}'(k)\right) \tag{43}$$

Observe that we can define, based on Eq. 43, an approximation of the derivative of the distance when the step size Δt is small (which is our case) such that

$$\dot{D}\left(\bar{x}'\,(k+1)\right) = \frac{\Delta D\left(\bar{x}'\,(k+1)\right)}{\Delta t} \tag{44}$$

Therefore, $\Delta D(\bar{x}')$ and $\dot{D}(\bar{x}')$ are used in order to find the solution of Eq. 25. $\Delta D(\bar{x}')$ is related to function $q(\cdot)$ in Eq. 26, while $\dot{D}(\bar{x}')$ relates to function $g(\cdot)$ in Eq. 27. Therefore, if the approach speed is big, then the capture time tends to decrease.

The *reinforcement* is then the output of a fuzzy system with only one rule [4]

$$r(k+1) = \min\left[\mu_{small}\left(\Delta D\left(\bar{x}'\,(k+1)\right)\right), \mu_{big}\left(\dot{D}\left(\bar{x}'\,(k+1)\right)\right)\right] \tag{45}$$

As previously, the membership functions for these two cases are triangular. An example of such functions is depicted in Fig. 17. The center of the derivative of

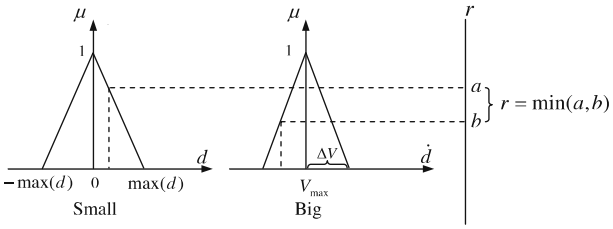

Fig. 17 Sets for calculation of the reinforcement signal

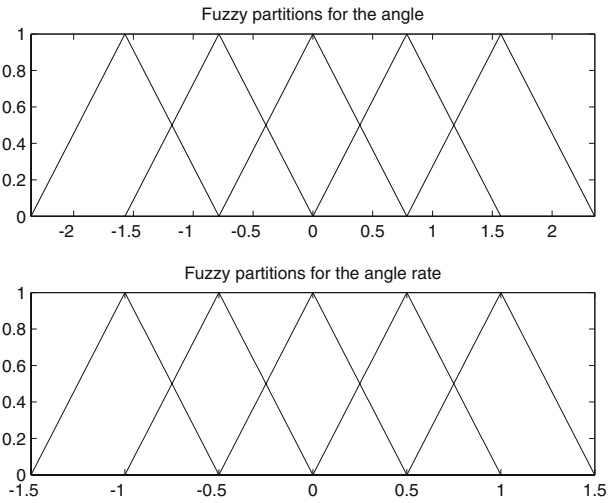

Fig. 18 Membership functions for the antecedent of the fuzzy rules

the distance is the maximum approach speed, calculated as the difference $V_e - V_p$ (introduced in Section 5).

The reinforcement signal in Eq. 45 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 Fig. 17, the reinforcement signal is determined only by the distance. Also, notice that the reinforcement is not simply an error signal as in Dai et al. [6] and Buijtenen et al. [4]. The distance and its derivative alone cannot represent error. Also, usually one requires errors to go to zero and in the case presented in Eq. 45, 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) and it is very difficult to define error based on behaviours [10].

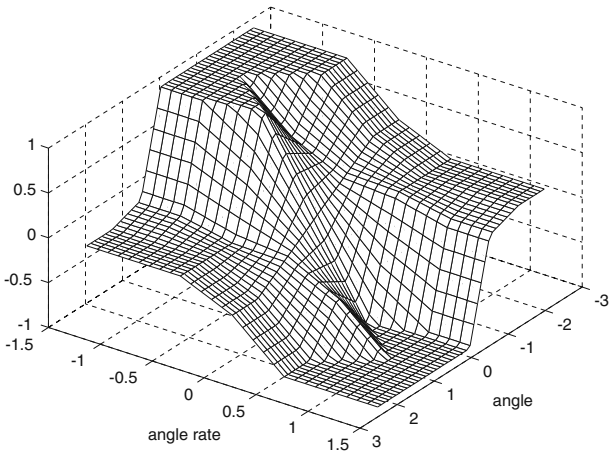

Fig. 19 Initial control surface of the fuzzy controller

Table 3 Initial values for the control signal

Angle	Angle rate				
	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

The input variables for the controller are the angle difference $\epsilon = \theta - \psi$ (both angles defined in Fig. 3) and its derivative, i.e., $\dot{\epsilon}$. In order to define the fuzzy controller, ϵ 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). The fuzzy sets for each of these fuzzy variables are depicted in Fig. 18.

The game takes place in a rectangle with the same dimensions as the camera range. The rectangle has its bottom-left point at $(11, 25)$ and its top-right point at $(193, 202)$. The target is assumed to be at the position $(0, 0)$. This set up information is represented in Fig. 4.

The critic has the same inputs as the controller. The output is the predicted reinforcement for the game that the critic seeks to approximate. In order to avoid too much time for the controller to converge due to errors in the critic, it is advisable to perform an off line training for a previous convergence of the critic [4]. This is just an introductory learning phase and we play only 100 instances of the game for this to take place. During this phase, the learning rate α in Eq. 15 is set to 0.1 for a fast adaptation. Also, throughout the simulations we use γ in Eq. 14 as 0.95 that is a small forgetting factor, meaning the critic uses only around 20 past signals for adaptation.

The initial control surface is shown in Fig. 19. This figure represents the fuzzy table shown in Table 3.

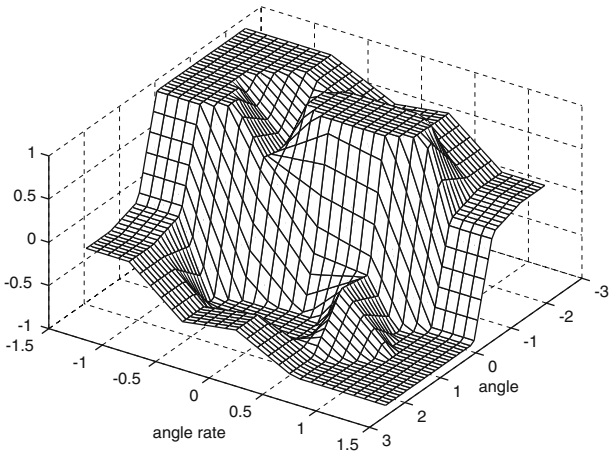

Fig. 20 Control surface of the fuzzy controller after learning

Table 4 Final values for the control signal

Angle	Angle rate				
	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

We then run the adaptation law in Eq. 17 for 1000 instances of the game 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 θ in Fig. 3 is supposed to be in the interval $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$;
- in the initial positions, if the optimal solution is played, the pursuer is able to catch the evader before it reaches the target.

The learning rate β in Eq. 17 is set to 0.01. At the same time, the critic continues to adapt at a learning rate of 0.1. After learning takes place, the control surface changes to the one shown in Fig. 20 that corresponds to the Table 4.

The surface shown in Fig. 20 changes in a very important way if compared to the initial control surface in Fig. 19. 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 Eq. 30 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. 20. 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 ($x_p = 15$, $y_p = 30$

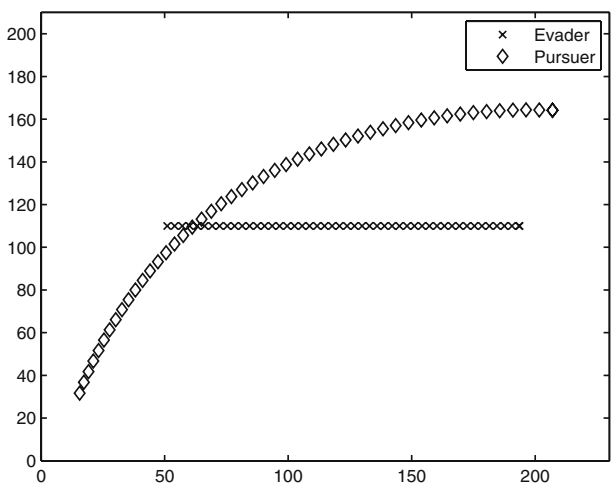

Fig. 21 Pursuer not able to catch the evader before learning

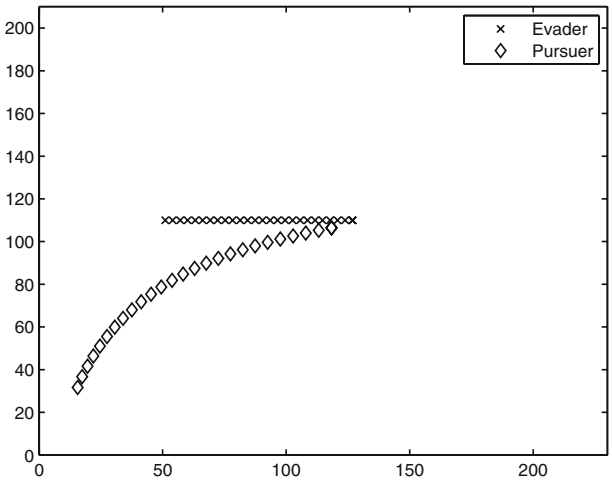

Fig. 22 Pursuer catching the evader after learning

and $\theta_p = 72^o$) that were never presented during the training phase. Figure 21 shows that with the initial controller of Table 3 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. 22, however, we show the pursuer catching the evader after learning takes place and it uses the learnt values of Table 4. The point of interception (evader's position) is (129, 110). For the matter of comparison, using the optimal controller, the interception occurs at (127, 110).

In order to prove the resilience of the controller, noise was added to the measurements for the simulations depicted in Fig. 22. Another 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 1100 epochs, which take only few minutes to run in a medium level computer.

The complete algorithm for the simulation discussed in this section, including the adaptation of the fuzzy controller and the fuzzy critic is shown in the following algorithm.

8 Experiments

Experiments are necessary in order to guarantee that the controller found in the previous section works in a real application.

As mentioned in Section 5.2, the measurements are sent to the robots by a computer through a Bluetooth link. The packets are assembled with the positions of the robots (the evader and the pursuer) and their orientations, the position of the target and the time of the reading, as shown in Fig. 23.

Fig. 23 Packet sent to the robots

...	x_e	y_e	θ_e	x_p	y_p	θ_p	x_T	y_T	t	...

Algorithm 1 Adaptation algorithm

1: Initialize the controller with the desired structure. The values for the antecedents of the rules are the ones shown in Fig. 18. The initial values for the consequent of the controller are the ones shown in Table 3.
2: Initialize the critic with the same antecedents as the controller. As for the consequents, it is all zeros. The output of the critic is not a control signal, but the expected reinforcement.
3: Initialize the Reinforcement Block with a fuzzy system that will return the reward for some specific state. This block is initialized with the fuzzy sets in Fig. 17 and the output is as in Eq. 45.
4: $\gamma \leftarrow 0.95$ (Eq. 13)
5: $\alpha \leftarrow 0.1$ (Eq. 15)
6: $\beta \leftarrow 0.01$ (Eq. 17)
7: **for** $i = 1$ to 1000 **do**
8: **while** The game does not finish **do**
9: Get the state of the system

$$\bar{x}[k] = [\epsilon, \dot{\epsilon}]^T$$

10: Calculate the output of the controller

$$u(k) \leftarrow \frac{\sum_{l=1}^{M} \left(\left(\prod_{i=1}^{n} \mu^{F_i^l}(x_i[k]) \right) \cdot \chi_l \right)}{\sum_{l=1}^{M} \left(\prod_{i=1}^{n} \mu^{F_i^l}(x_i[k]) \right)}$$

11: Calculate the output of the critic

$$\hat{V}(k) \leftarrow \frac{\sum_{l=1}^{M} \left(\left(\prod_{i=1}^{n} \mu^{F_i^l}(x_i[k]) \right) \cdot \zeta_l \right)}{\sum_{l=1}^{M} \left(\prod_{i=1}^{n} \mu^{F_i^l}(x_i[k]) \right)}$$

12: Run the game for the current time step
13: Get the states for the reinforcement

$$\bar{y}[k+1] = [d, \dot{d}]^T$$

14: Calculate the reinforcement signal

$$r(k+1) \leftarrow \min \left[\mu_{small}(\Delta D(k+1)), \mu_{big}\left(\dot{D}(k+1)\right) \right]$$

15: Get the new states of the game

$$\bar{x}[k+1] = [\epsilon, \dot{\epsilon}]^T$$

Algorithm 1 (continued)

16: Calculate the output of the critic with the new states

$$\hat{V}(k+1) \leftarrow \frac{\sum_{l=1}^{M}\left(\left(\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k+1])\right)\cdot\zeta_l\right)}{\sum_{l=1}^{M}\left(\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k+1])\right)}$$

17: Calculate the delta signal

$$\Delta \leftarrow \left[r(k+1)+\gamma\hat{V}(k+1)\right]-\hat{V}(k)$$

18: Calculate the critic gradient

$$\frac{\partial\hat{V}(k)}{\partial\zeta_j} \leftarrow \frac{\prod_{i=1}^{n}\mu^{F_i^j}(x_i[k])}{\sum_{l=1}^{M}\left(\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k])\right)}$$

19: Train the critic

$$\zeta_j(k+1) \leftarrow \zeta_j(k)+\alpha\Delta\frac{\partial\hat{V}(k)}{\partial\zeta_j}$$

20: Calculate the controller gradient

$$\frac{\partial u(k)}{\partial\chi_j} \leftarrow \frac{\prod_{i=1}^{n}\mu^{F_i^j}(x_i[k])}{\sum_{l=1}^{M}\left(\prod_{i=1}^{n}\mu^{F_i^l}(x_i[k])\right)}$$

21: Train the controller

$$\chi_j(k+1) \leftarrow \chi_j(k)+\beta\Delta\left[\frac{u'(k)-u(k)}{\sigma}\right]\frac{\partial u(k)}{\partial\chi_j}$$

22: **end while**

23: **end for**

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.

We also mentioned in Section 5.2 that our robots are driven by motors controlled by a Motorola 68HC11 microprocessor built in a Handyboard. The message handling for both robots and the fuzzy controller for the pursuer were implemented in

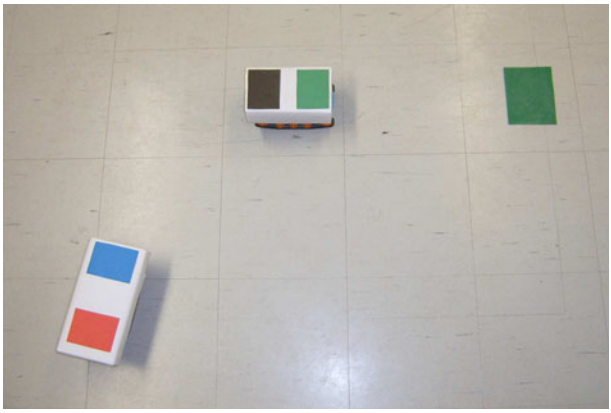

Fig. 24 Initial conditions for the experiment

the microprocessor using C. Care had to be taken in order to guarantee that the calculations were made in an efficient way. The signals to the motors must be calculated quickly in order to avoid nonlinearities in the robots' behaviours.

The initial positions of the robots are the same as the initial positions for the simulations shown in Figs. 21 and 22. This is done for the comparison with the simulation to be meaningful. The real positions of the robots are shown in Fig. 24.

With the controller of Table 4 implemented in the microprocessor of the pursuer and the same control strategy as the one presented in the simulation implemented for the evader robot, the experiment yielded the behaviour shown in Fig. 26. Figure 25 shows the simulation considering the real size of the robots. In other words, capture occurs when the pursuer and the evader robots physically touch.

As it may be seen from a comparison of Figs. 25 and 26, the experiment and the simulation agree very well. This shows that the identification and derivation of the

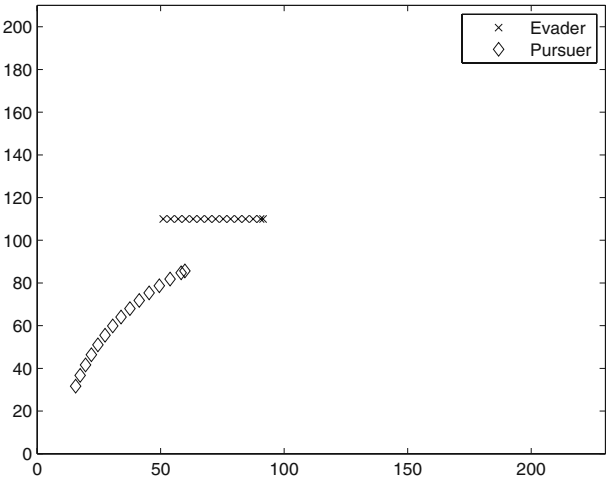

Fig. 25 Simulation results

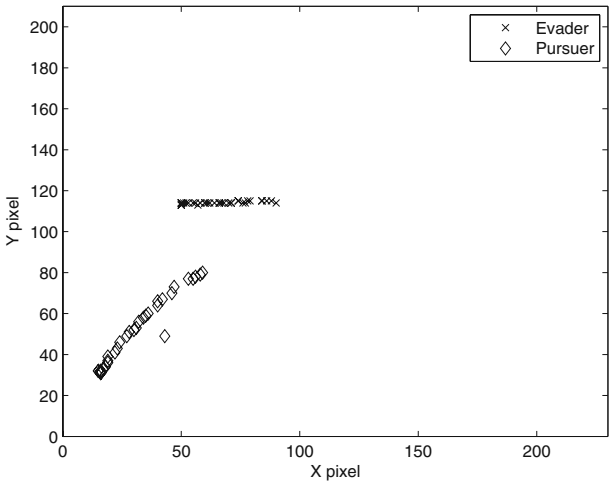

Fig. 26 Experimental results

controller were successful. Moreover, the learning was efficient in finding the best controller for an actual application.

Several other instances of the game were executed and the results invariably met the ones reached in the simulation. This shows that the controller found is suitable for the experimental set up.

One last remark must be stated regarding the experiment. The learning engine was turned off during the experiment. The reason for that is the limited range of our sensors. The games finish too soon for a reasonable learning to take place. In order to take advantage of the learning, we would have to have a larger range for the game. We are working on this and in the future we intend to have a fully experimental environment where no simulation is required for the learning to take place.

9 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 Dai et al. [6] and Buijtenen et al. [4] is shown to be suitable for learning in a continuous environment.

A simulation 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 adapted using the scheme described in Section 4 is superior to the initial controller. Also, we show that the controller is resilient to noise.

We then implemented the derived control system in actual robots. Results show that the quality of the response is similar to the simulation, illustrating that our approach is reasonable. Moreover, adaptation may continue in the robots in order for them to deal with changes in their dynamics.

References

1. Andrecut, M., Ali, M.K.: Fuzzy reinforcement learning. Int. J. Mod. Phys. C **13**(5), 659–674 (2002)
2. Arslan, G., Shamma, J.S.: Anticipatory learning in general evolutionary games. In: Proceedings of the 45th IEEE Conference on Decision and Control, pp. 6289–6294 (2006)
3. Bryson, A.E., Ho, Y. (eds.): Applied Optimal Control: Optimization, Estimation, and Control. Taylor & Francis, Levittown (1975) (Rev. printing)
4. Buijtenen, W.M., Schram, G., an H. B. Verbruggen, R.B.: Adaptive fuzzy control of satellite attitude by reinforcement learning. IEEE Trans. Fuzzy Syst. **6**(2), 185–194 (1998)
5. Conlisk, J.: Adaptation in games: two solutions to the crawford puzzle. J. Econ. Behav. Organ. **22**, 25–50 (1993)
6. 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)
7. Fudenberg, D., Levine, D.K.: The Theory of Learning in Games. MIT, Cambridge (1998)
8. Ge, J., Tang, L., Reimann, J., Vachtsevanos, G.: Hierarchical decomposition approach for pursuit-evasion differential game with multiple players, p. 7 (2006)
9. Ge, J., Tang, L., Reimann, J., Vachtsevanos, G.: Suboptimal approaches to multiplayer pursuit-evasion differential games, pp. 5272–5278 (2006)
10. Givigi, S.N., Schwartz, H.M.: Swarm robot systems based on the evolution of personality traits. Turkish Journal of Electrical Engineering & Computer Sciences (Elektrik): Special Issue on Swarm Robotics **15**(2), 257–282 (2007)
11. Harmon, M.E., III, L.C.B.: Residual advantage learning applied to a differential game. In: Proceedings of the International Conference on Neural Networks, pp. 1–6 (1996)
12. Harmon, M.E., III, Baird, L.C., Klopf, A.H.: Reinforcement learning applied to a differential game. Adapt. Behav. **4**(1), 3–28 (1995)
13. Hofbauer, J., Sigmund, K.: Evolutionary game dynamics. Bull. Am. Math. Soc. **40**(4), 479–519 (2003)
14. Isaacs, R.: Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization. Wiley, New York (1965)
15. Ishibuchi, H., Sakamoto, R., Nakashima, T.: Learning fuzzy rules from iterative execution of games. Fuzzy Sets Syst. **135**, 213–240 (2003)
16. Mannor, S., Shamma, J.: Multi-agent learning for engineers. Artificial Intelligence, Special issue on "Foundations of Multi-Agent Learning", pp. 417–422 (2007)
17. Merz, A.W.: The homicidal chauffeur. AIAA J. **12**(3), 259–260 (1974)
18. Myerson, R.B.: Game Theory: Analysis of Conflict. Harvard University Press, Cambridge (1991)
19. von Neumann, J., Morgenstern, O.: The Theory of Games and Economic Behavior, 2nd edn. Princeton University Press, Princeton (1947)
20. Sheppard, J.W.: Colearning in differential games. Mach. Learn. **33**, 201–233 (1998)
21. Starr, A.W., Ho, Y.C.: Nonzero-sum differential games. J. Optim. Theory Appl. **3**(3), 184–206 (1969)
22. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its application to modeling and control. IEEE Trans. Syst. Man Cybern. **15**, 116–132 (1985)
23. Wang, L.X.: A Course in Fuzzy Systems and Control. Prentice Hall, Englewood Cliffs (1997)
24. Weibull, J.W.: Evolutionary Game Theory. MIT, Cambridge (1995)
25. Yeung, D.W.K., Petrosyan, L.A.: Cooperative Stochastic Differential Games. Springer, New York (2006)