Soft Comput (2011) 15:1617–1629
DOI 10.1007/s00500-011-0691-6

ORIGINAL PAPER

# A fuzzy-rule-based driving architecture for non-player characters in a car racing game

**Enrique Onieva · David A. Pelta · Vicente Milanés · Joshue Pérez**

Published online: 15 January 2011


**Abstract** Videogame-based competitions have been the target of considerable interest among researchers over the past few years since they provide an ideal framework in which to apply soft computing techniques. One of the most popular competitions is the *Simulated Car Racing Competition* which, thanks to the realism implemented by recent car simulators, provides an excellent test bed for the application of autonomous driving techniques. The present work describes the design and implementation of a car controller able to deal with competitive racing situations. The complete driving architecture consists of six simple modules, each one responsible for a basic aspect of car driving. Three modules use simple functions to control gear shifting, steering movements, and pedal positions. A fourth manages speed control by means of a simple fuzzy system. The other two modules are in charge of (i) adapting the driving behaviour to the presence of other cars, and (ii) implementing a basic 'inter-lap' learning mechanism in order to remember key track segments and adapt the speed accordingly in future laps. The controller was evaluated in two ways. First, in runs without adversaries over several track designs, our controller allowed some of the longest distances to be covered in a set time in comparison with data from other previous controllers, and second, as a participant in the *2009 Simulated Car Racing Competition* which it ended up winning.

**Keywords** Fuzzy control · Car racing · Videogames

E. Onieva (✉) · V. Milanés · J. Pérez
AUTOPIA program of the Center for Automation and Robotics, Universidad Politécnica de Madrid–Consejo Superior de Investigaciones Científicas, La Poveda-Arganda del Rey, 28500 Madrid, Spain
e-mail: enrique.onieva@car.upm-csic.es

V. Milanés
e-mail: vicente.milanes@car.upm-csic.es

J. Pérez
e-mail: joshue.perez@car.upm-csic.es

D. A. Pelta
Department of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain
e-mail: dpelta@decsai.ugr.es

## 1 Introduction

Games have long been a popular area of artificial intelligence (AI) research. They are challenging yet easy to formalize, making it possible to develop new AI methods, measure how well they are working, and demonstrate that machines are capable of impressive behaviour generally thought to require intelligence, without putting human lives or property at risk.

Until recently, most of the academic work in this area had focused on traditional board games and card games, the challenge being to beat expert human players. Following the release of Pong in the early 1970s, the subsequent decades have seen a huge increase in the quality, diversity, and pervasiveness of videogames (Lucas 2009). This presents academic researchers (Laird 2002) and game developers with the challenge of developing next generation game AI.

Deep Blue for Chess (Hsu 2002) and Chinook for Checkers (Schaeffer et al. 2007; Schaeffer 2009) are examples of applications of AI techniques to game-tree search applications (Hong et al. 2002), i.e., games where the accuracy of a player is measured by moving tiles on a board or dealing with cards. These games are based on discrete time movements, and game-tree search methods

have demonstrated that, with enough calculating power, it is in many cases possible to find an optimum strategy.

In recent years a large number of application programming interfaces (APIs) have emerged, giving researchers the opportunity to apply learning, control, and planning techniques to continuous environments. Many of the resulting simulated worlds are quite similar to the real world. They have provided researchers and developers with an opportunity to take on the new generation of games, applying their techniques to the automatic creation of content, adaptive lighting, intelligent camera control, and, as the most obvious example, the control of automatic players (non-player characters, NPC). Applications of artificial intelligence to NPC control in videogames can be found in several classes of game (strategy games, puzzles, and continuous games). Examples are Tetris (Siegel and Chaffee 1996), Pac Man (Rosca 1996), and Quake (Schloman and Blackfordm 1951).

Car simulators in general, and racing games in particular, have been the object of burgeoning academic interest in recent years for two main reasons: (i) they have evolved to the point that they can implement physical dynamics that is very close to the real world, enabling the application of techniques and methods previously reserved for the field of Autonomous Vehicle Guidance (Arroyabe et al. 2000); and (ii) the emergence of competitions with the objective of controlling a car in simulated racing environments has encouraged computational intelligence or soft computing researchers to apply their knowledge and experience to this topic.

The prime objective of the present work was twofold: first, to design, implement, and test a complete architecture enabling automatic driving in racing situations; and second, to better understand how to construct efficient and simple to understand controllers for car bots. An additional aim was to show how very well known ideas from the field of soft computing can be applied to a new, fun, and challenging research area.

The main idea behind the architecture is to have a small set of simple and interpretable modules whose interactions lead to good driving. Hence, each module would manage one basic action that has to be considered when managing a car in racing situations. Another requirement imposed in the design stage was that each module had to be parametrized, thus allowing for future improvement by means of automatic parameter adjustment.

We considered, of course, that one of the key factors in racing situations is the determination of the optimal speed that a given segment of track allows. An incorrect calculation could end in such undesirable situations as going too slowly when the car could safely go faster and hence achieve a shorter *lap time*, or going too fast and hence losing control, crashing, or driving off the track around a curve.

Due to this perceived importance of proper speed management, Fuzzy Logic (Zadeh 1965) was used for the corresponding module. In particular, a fuzzy controller was designed with the aim of simplifying in so far as possible the task of determining an appropriate racing speed in every situation.

The work is structured as follows: Sect. 2 presents the racing simulator used, and the sensor information and action commands available to interact with the car. Section 3 describes the architecture, first explaining it from a general perspective and then passing to more specific descriptions detailing the reasoning behind each module. Section 4 presents the two sets of comparative results which served to evaluate the architecture: first, the laboratory experiments which showed us the reliability of the proposed architecture with or without opponents; and then the results of the participation of the controller in the *2009 Simulated Car Racing Championship*. Section 5 gives some concluding remarks about these results and about controlling simulated vehicles in racing environments, and some reflections on how the architecture may be improved in future work.

## 2 The TORCS racing environment

The Open Racing Car Simulator[1] (TORCS) is one of the most popular car racing simulators. It is written in C++ and is available under the GPL license from its Web site. TORCS presents several advantages for its use for academic purposes, such as

- It lies between the extremes of an advanced simulator such as recent commercial car racing games and a fully customizable environment such as is typically used by computational intelligence researchers for benchmark purposes.
- It features a sophisticated physics engine (aerodynamics, fuel consumption, traction, ...) as well as a 3D graphics engine for the visualization of the races.
- It was not conceived to be a free alternative to commercial racing games, but was specifically devised to make it as easy as possible for users to develop their own controllers.

Indeed, the controllers can be implemented as separate software modules, so that it is easy to develop a new controller and plug it into the game. Because of that, TORCS has been receiving increasing academic attention for its use as a platform for computational intelligence competitions.

[1] http://www.torcs.sourceforge.net/

The first car racing competitions were held in 2007 as part of the IEEE Congress on Evolutionary Computation (CEC07) and the Computational Intelligence and Games Symposium (CIG07). These first competitions used a graphically and mechanically simple game which attracted a good number of participants (Togelius et al. 2008).

Then, in 2008, a car racing competition based on TORCS was organized in conjunction with the IEEE World Congress on Computational Intelligence (WCCI08) (Loiacono et al. 2008b). The use of TORCS opened up the possibility of simulating more complex racing conditions, especially since it made it possible for there to be many cars on the same track. A similar competition was held at the 2008 IEEE Computational Intelligence and Games Symposium (CIG08).

Competitors in 2008 were provided with a specific software interface developed on a client/server basis in which the designed controllers run as external programs and communicate with a customized version of TORCS through UDP connections (Loiacono et al. 2008a).

The architecture is shown in Fig. 1. The controller perceives the racing environment through a number of sensor readings reflecting both the surrounding environment and the current game state and can invoke driving commands to control the car. The complete list of sensors is given in Table 1. The present work is based on this UDP architecture to communicate with the server so as to receive sensor information and send control actions.

Henceforth, the notation $track_d$ and $opponents_d, d = \{-180, -170, \ldots, 0, 10, \ldots, 170, 180\}$, will be used to denote the values of the *track* and *opponents* laser sensors oriented at $d$ degrees, respectively. Figure 2 shows two illustrative examples of plots of the *track* sensor readings in which the vehicle is situated at (0,0). A graphical representation of the input variables *position* and *angle* is shown in Fig. 3, in which the meaning of the sign of the value is also represented.

Once the controller has obtained the sensor values from the server, it will generate output commands to drive the simulated car by means of the effectors. The effectors represent which action the driver can perform and thus affect the game state. Table 2 gives a detailed description of the available effectors.

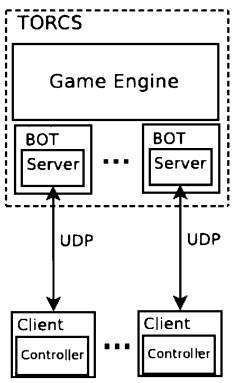


**Fig. 1** The architecture of the API developed for the WCCI 2008 competition

## 3 Proposed architecture

The methodological approach and the techniques used to design a suitable control architecture are explained in this section. The overall control architecture is shown in Fig. 4, in which one can appreciate the modular design.

The basic architecture consists of six simple modules. The goal of the present work was to create and test simple controllers in each of these modules. Their basic functions are as follows:

1. *Gear Control* is mainly responsible for shifting through the first to sixth gears. Its functionality is complemented with the task of detecting situations of blockage ("stuck") so as to apply reverse gear.
2. *Target Speed* assigns the maximum speed allowed by a certain track segment.
3. *Speed Control* follows the indications of the previous module, i.e., to reach or maintain a certain speed by managing the throttle and brake pedals. Additionally, two anti-skid filters are implemented: a traction control (TCL) filter and an anti-lock brake system (ABS) filter will, if necessary, reduce actions on the throttle and brake, respectively.
4. *Steering Control* manages the car's direction by acting on its steering wheel.
5. *Learning* detects segments of the circuit where the target speed can be increased or has to be decreased on the basis of information about the previous laps, such as long straight segments or segments where the vehicle left the track.
6. *Opponents Modifier* is activated when opponents are close, modifying the steering, throttle, and brake outputs to adapt driving actions. Its main objectives are avoiding collisions and overtaking opponents.

One notes therefore that modules 1–4 are sufficient to implement a car controller able to drive one lap alone on a track. The fifth module allows the system to remember actions taken during the current lap so as to improve the performance on subsequent laps. The sixth module adds the controlling features required to deal with the presence of opponents. The rest of the section describes the details of each module.

### 3.1 Gear control module

This module has three basic functions: (i) shifting through first to sixth gears; (ii) detecting blocked situations so as to

**Table 1** Description of the available sensors

| Name | Range (unit) | Description |
|---|---|---|
| *angle* | $[-\pi, +\pi]$ (rad) | Angle between the car's direction and the direction of the track |
| *curLapTime* | [0,−] (s) | Time elapsed during current lap |
| *damage* | [0,−] (point) | Current damage of the car |
| *distFromStart* | [0,−] (m) | Distance from the start line along the track |
| *distRaced* | [0,−] (m) | Distance covered from the beginning of the race |
| *fuel* | [0,−] (l) | Current fuel level |
| *gear* | $\{-1, 0, 1, 2, \ldots, 6\}$ | Current gear: −1 is reverse, 0 is neutral, and 1–6 |
| *lastLapTime* | [0,−] (s) | The last lap time |
| *opponents* | [0,100] (m) | Vector of 36 sensors that detects opponent distances in metres within a specific 10° sector, from −180° to 180° |
| *racePos* | $1, 2, \ldots$ | Position (placing) in the race with respect to other cars |
| *rpm* | [2,000, 10,000] (rpm) | Revolutions per minute of the car's engine |
| *speed* | − (km/h) | Speed of the car along the longitudinal axis |
| *speedY* | − (km/h) | Speed of the car along the transverse axis |
| *track* | [0,100] (m) | Vector of 19 range-finding sensors: Each sensor gives the distance to the track edge. Sensors are oriented every 10° between −90° and 90° |
| *position* | − | Distance between the car and the track borders. The value is normalized w.r.t. the track width: it is 0 when the car is on the axis, −1 when the car is on the right edge of the track, and +1 when it is on the left edge. Values greater than 1 or less than −1 mean that the car is off the track |
| *wheelVel* | $[0, -]$ (rad/s) | Vector of 4 sensors representing the rotation speed of the wheels |

The ranges are reported with the corresponding measurement unit (where this is applicable). The symbol − means unlimited range

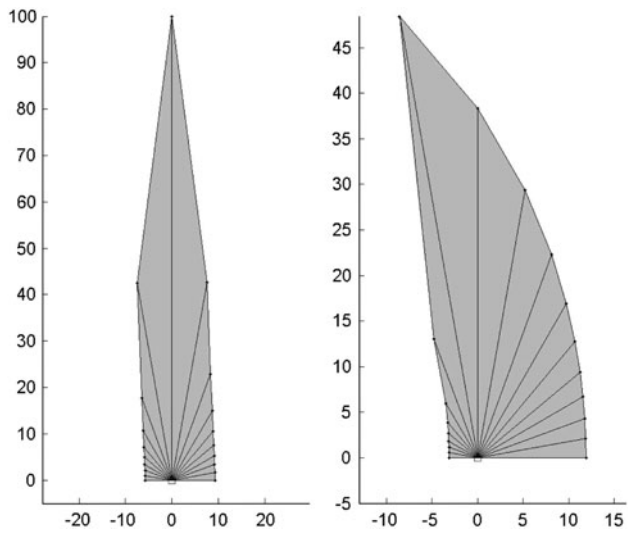


**Fig. 2** Graphical example of *track* sensor readings: *left* in a straight segment; *right* in a curved segment. *Gray area* denotes track. The value of the maximum allowed measurement distance is 100 m

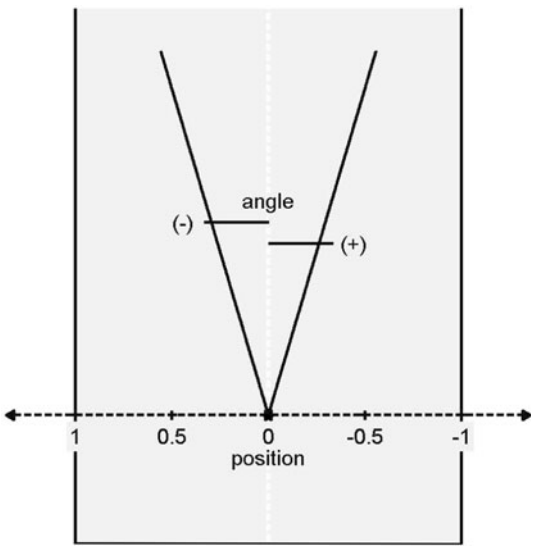


**Fig. 3** Graphical representation of *position* and *angle* sensor values

apply the reverse gear; and (iii) determining the end of such blocked situations so as to apply first gear.

Gears are shifted up or down according to the value of *rpm*. A maximum and minimum limit is established for each gear, so that when the *rpm* value goes beyond this range then the next gear up or down will be selected. Table 3 lists these maximum and minimum values.

A blocked situation is taken to be when *angle* is greater than $\frac{\pi}{6}$ radians (30°) and *position* is greater than 0.5 (half of the track width), which means that the car is going off the track. Also, if the speed is less than 10 km/h, the car may have crashed (with a barrier or with a stationary opponent) and cannot continue. The blocked situation is detected with the condition

**Table 2** Description of available effectors

| Name | Range | Description |
|---|---|---|
| accel | [0,1] | Virtual throttle pedal (0 means no throttle, 1 full throttle) |
| brake | [0,1] | Virtual brake pedal (0 means no brake, 1 full brake) |
| gear | $-1, 0, 1, \ldots, 6$ | Gear value |
| steer | [−1,1] | Steering value: −1 and +1 mean respectively full right and full left |

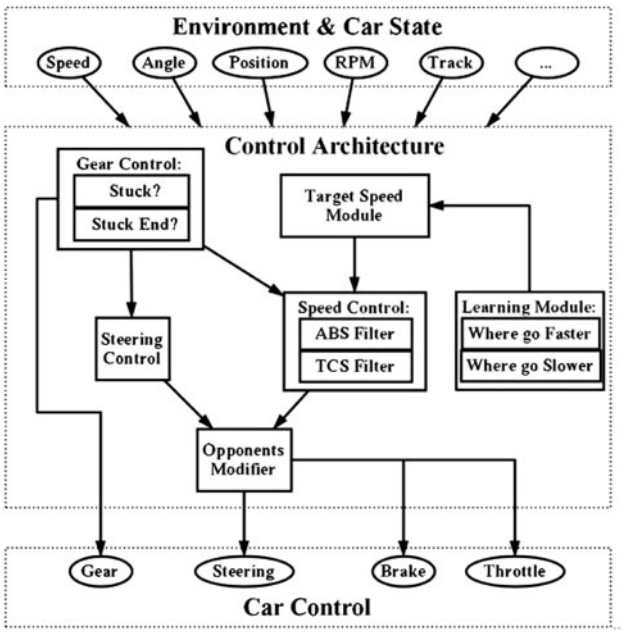


**Fig. 4** Schema of the control architecture

**Table 3** Forward gear change thresholds

| Current gear | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Shift up | ≥9,000 | ≥9,000 | ≥9,000 | ≥8,000 | ≥8,000 | |
| Shift down | | ≤3,000 | ≤3,000 | ≤3,000 | ≤3,500 | ≤3,500 |

$$\left(|angle| \geq \frac{\pi}{6}\ AND\ |position| \geq 0.5\right)\ OR\ (speed < 10) \quad (1)$$

Reverse gear is applied by the module when condition 1 has been maintained for 2 s.

The end of the blocked situation is detected when, being in a such a situation, *angle* and *position* have the same sign. This means that the car is now oriented appropriately with respect to its position, i.e., the car is displaced to the left and oriented to the right (or vice versa).

### 3.2 Target speed module

The main aim of this module is to calculate at each time step the value $Target_{speed}$ of the speed at which the car should drive along a track segment. Two cases must be considered since the *track* sensor values are not valid when the car is off the track.

When the car is off the track, the module increments the current speed by 5 km/h, maintaining it within the [30,150] km/h range, while the ABS and TCS filters will be responsible for avoiding skidding or loss of traction.

When the car is on the track, a fuzzy rule-based system is used. The system is based on a computational model of a fuzzy coprocessor named ORBEX (Spanish acronym for Fuzzy Experimental Computer) (García and de Pedro 1998). ORBEX implements a fuzzy system with trapezoidal membership functions for the input variables, t-norm minimum and t-conorm maximum, singleton-shaped output variables, and centre-of-mass defuzzification.

The fuzzy system's input variables are taken from three of the 19 available *track* sensors:

1. $front = track_0$
2. $max_{10} = max(track_{-10}, track_{10})$
3. $max_{20} = max(track_{-20}, track_{20})$

Each input variable is codified by three membership functions denominated *Low*, *Medium*, and *High* representing the level of available or free distance in the vehicle's forward direction. The membership functions used in the fuzzification process are plotted in Fig. 5.

The output value ($Target_{speed}$) is codified by seven singletons as shown in Fig. 6, from a maximum of 200 km/h to a minimum of 50 km/h.

The rule base is designed under the premise that, if the free distance ahead is maximal, then $Target_{speed}$ has to be maximal, and this value has to decline with declining free distance ahead. In particular, the fuzzy rules used to infer the $Target_{speed}$ value are the following:

1. If *Front* is *High* Then $Target_{speed}$ is $TS_1$
2. If *Front* is *Medium* Then $Target_{speed}$ is $TS_2$
3. If *Front* is *Low* and $Max_{10}$ is *High* Then $Target_{speed}$ is $TS_3$
4. If *Front* is *Low* and $Max_{10}$ is *Medium* Then $Target_{speed}$ is $TS_4$
5. If *Front* is *Low* and $Max_{10}$ is *Low* and $Max_{20}$ is *High* Then $Target_{speed}$ is $TS_5$
6. If *Front* is *Low* and $Max_{10}$ is *Low* and $Max_{20}$ is *Medium* Then $Target_{speed}$ is $TS_6$

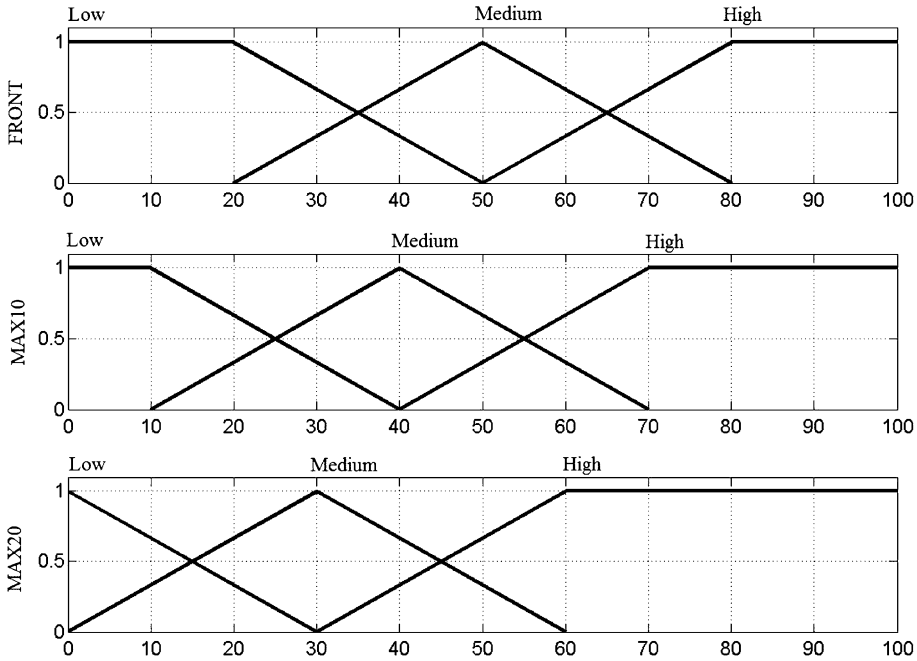


**Fig. 5** Membership functions for each input variable

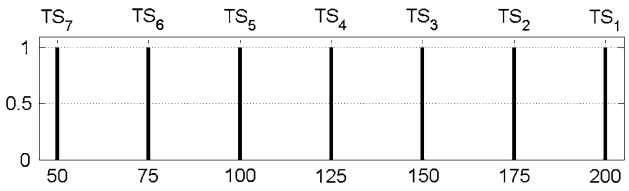


**Fig. 6** Output singletons

7. If *Front* is *Low* and $Max_{10}$ is *Low* and $Max_{20}$ is *Low* Then $Target_{speed}$ is $TS_7$

A crisp rule is added to the rule base to obtain a maximum $Target_{speed}$ value when one of the three input variables has the maximum possible measurement value of 100 m:

If $Front = 100$ or $Max_{10} = 100$ or $Max_{20} = 100$ Then $Target_{speed} = 300$

This crisp rule will always be activated when the condition is true, i.e., if one of the module's inputs ($Front, Max_{10}$, or $Max_2 0$) attains the maximum value, then $Target_{speed}$ will be set to 300 km/h.

The $Target_{speed}$ value obtained represents the desirable speed at which to drive along the current track segment. That speed will be sent to the speed control module (Sect. 3.4) to manage the pedals appropriately. The learning module (Sect. 3.6) multiplies the $Target_{speed}$ value by a factor (unity on the first lap) in order to increase or decrease it on a certain track segment depending on previous experience. Finally, the opponent modifier (Sect. 3.5) can use an emergency braking policy to avoid collisions by reducing the value of the target speed.

Given these circumstances, it should be clear that this module is the core of the driver since it affects and can be affected by most of the other modules. Its proper design is therefore a key to obtaining a good driving strategy.

### 3.3 Speed control module

This module is responsible for managing the pedals to bring the speed up or down to the inferred $Target_{speed}$ value. The throttle and brake effectors are codified as a single common output value (*Gas* in $[-1,1]$) in order to avoid nonsensical actions such as acting on both pedals at the same time. Positive values represent actions on the throttle, maintaining the brake at 0, and negative values represent the mirror situation.

This module is implemented by the function of the difference between *speed* and $Target_{speed}$ given in Eq. 2. This function is plotted in Fig. 7.

$$Gas(speed - Target_{speed}) = -1 + \frac{2}{1 + e^{speed - Target_{speed}}} \quad (2)$$

Two filters are implemented: (i) the TCL filter to reduce the throttle action when $Gas > 0$, and (ii) the ABS filter to

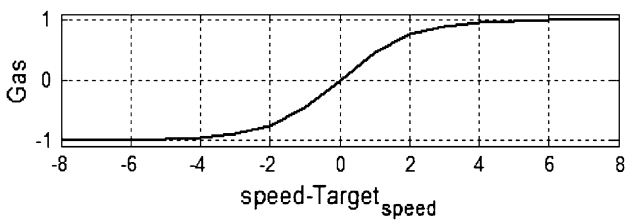


**Fig. 7** Graphical representation of the action on the pedals

reduce the brake action when $Gas < 0$. They allow the vehicle to adapt the pedal outputs to the roadway conditions, thus avoiding skidding.

The filters are based on the difference between the value of *speed* and the car's speed as computed with the angular speed of the wheels (*wheelVel*). When these two speed values differ, the vehicle has lost traction. They are activated if that speed difference is greater than 1.5 km/h and will modify the current value of *Gas* according to

$$Gas = Gas \pm \frac{speed_{wheels} - speed - 1.5}{5} \quad (3)$$

with the minus sign used when $Gas > 0$ (TCL), and the plus sign when $Gas < 0$ (ABS).

### 3.4 Steering control module

Three cases are taken into account for the control of the steering wheel: (i) the car is on the track, (ii) the car is off the track, and (iii) the car is in reverse gear.

For case (i), when the car is on the track, we define $M$ as the orientation (in degrees) of the *track* sensor with the greatest value. If two sensors have the same value, that with orientation closest to zero will be taken. Then $track_M$ will represent the measurement (in metres) of the sensor oriented at $M$ degrees, and $track_{M\pm10}$ will represent the measurement of two adjacent track sensors (oriented at $M \pm 10$).

If the greatest distance is measured by sensors oriented at $\pm 40$ (or more) degrees, then maximum steering actions are applied. In the case that $M \leq -40$, then *Steer* is set to $+1$, while in the case that $M \geq 40$, then *Steer* is set to $-1$.

Otherwise, *Steer* is computed using the distances measured by the track sensor with maximum value and its two adjacent sensors (subindices $M \pm 10$) as follows:

$$Steer = S_M + \frac{track_{M-10} \times |S_M - S_{M-10}| - track_{M+10} \times |S_M - S_{M+10}|}{track_M} \quad (4)$$

where $S_{deg}$ represents a vector of actions on the steering wheel depending on the *track* sensor with maximum value. $S_{-30} = 1$, $S_{-20} = 0.75$, $S_{-10} = 0.5$, $S_0 = 0$, and mirror values for $S_{10,20,30}$. Equation 4 allows the car to be steered in the direction of the greatest free distance.

For case (ii), when the car is off the track, the *track* sensors are unreliable. Then, Eq. 5 is used to correct the car's lateral and angular errors and bring it back onto the track.

$$Steer = \frac{angle - 0.5 \times position}{Steer_{lock}} \quad (5)$$

$Steer_{lock}$ is a constant taken from the car's description available in TORCS. Its value is 0.785.

For case (iii), when the car is in reverse gear, the main objective of steering management is to get the car oriented with the track's centre line in order to continue racing. Equation 6 is used in this case.

$$Steer = \frac{-angle}{Steer_{lock}} \quad (6)$$

### 3.5 Opponents modifier module

This module will adapt the driving behaviour to the proximity of opponents, by modifying or replacing the outputs of the steering, throttle, and brake controllers. The modification is performed mainly on the basis of information coming from *opponents* sensors. Three kinds of modification are made: (i) on the steering output in order to avoid opponents while going past them (overtaking), (ii) again on the steering, but with a sharp movement when there exists a risk of collision (collision avoidance), and (iii) on the brake output when there is an opponent in front of the vehicle at a dangerous distance (emergency braking). The two actions on the steering are performed simultaneously.

In order to overtake opponents, the steering output is modified using a set of rules each of which has the following structure:

$$If \left( \frac{opponent_{deg}}{speed} < T^o_{deg} \right) then\ Steer = Steer + inc^o_{deg} \quad (7)$$

where opponent sensors with $deg = \{-90, -80, \ldots 0, \ldots 80, 90\}$ are considered. Table 4 summarizes the values of the variables in this case. The sensor oriented at 0° is treated as a special case: $T^o_0 = 1$ and $inc^o_0 = \pm 0.3$, with the sign of the increment being the same as the sign of the steering output in order to reinforce actions on the steering in case there is an opponent just in front of the car.

To avoid collisions or simply to avoid (or "dodge" around) opponents, stronger steering actions are required as well as smaller threshold values. The rules in this case are defined as

**Table 4** Thresholds and increments to overtake opponents

| $deg$ | $inc^o_{deg}$ | $T^o_{deg}$ | $deg$ | $inc^o_{deg}$ | $T^o_{deg}$ |
|---|---|---|---|---|---|
| $\leq -60°$ | +0.1 | 0.3 | $\geq 60°$ | −0.1 | 0.3 |
| −50° | +0.12 | 0.5 | 50° | −0.12 | 0.5 |
| −40° | +0.12 | 0.5 | 40° | −0.12 | 0.5 |
| −30° | +0.13 | 0.75 | 30° | −0.13 | 0.75 |
| −20° | +0.14 | 0.75 | 20° | −0.14 | 0.75 |
| −10° | +0.15 | 1 | 10° | −0.15 | 1 |

$$If\ \left(opponent_{deg} < T^d_{deg}\right)\ then\ Steer = Steer + inc^d_{deg} \quad (8)$$

Now, only sensors with $deg = \{-20, -10, \ldots 0, \ldots 10, 20\}$ are taken into account, so that opponents just in front of the car are considered. Table 5 summarizes the values of the variables in this case. A special case is $T^d_0 = 15$ and $inc^d_0 = \pm 0.3$ with sign equal to that of the steering output.

Case (iii) is the modification of the pedal actions when there is an opponent near and in front of the vehicle. In this situation, $Target_{speed}$ is reduced by 25% if any of the values of the *opponents* sensors oriented at $-10$, 0, or 10° is less than 10 m.

### 3.6 Learning module

The objective of this module is to identify track segments where the vehicle should go faster or slower than on previous laps. Using the *distFromStart* value provided by the server, the vehicle knows its position, as well as the track length (in metres).

A vector $Speed_{factor}$ with size equal to the track length in metres and initialized to 1 is defined. $Speed_{factor}$ represents a factor which increases or decreases the $Target_{speed}$ value provided by the target speed module described in Sect. 3.2.

Throughout the race, the module will check for situations where the car must reduce or increase its speed. When these are detected, a modification of $Speed_{factor}$ is applied to positions prior to the point where this situation was detected. The effect is that the vehicle will remember special locations along the track.

Three types of location are considered by the module: (i) positions where the vehicle has left the track, (ii) positions where the vehicle has been damaged by crashing with the barriers, and (iii) very long straight (or smoothly curved) segments.

To consider the 'off the track' situations, a variable *out* is defined as the value of *distFromStart* at the point where the car went off the track (without opponents within a radius of 15 m); $out < 0$ is used when the vehicle is on the track. $Speed_{factor}$ for the following laps will be reduced by 10% in the segment from 200 to 100 m before the *out* point (Eq. 9) and by 20% in the segment from 100 m to the *out* point (Eq. 10).

$$Speed_{factor}(i) = 0.9 \times Speed_{factor}(i),\quad i = \{out - 200, \ldots, out - 100\} \quad (9)$$

$$Speed_{factor}(i) = 0.8 \times Speed_{factor}(i),\quad i = \{out - 100, \ldots, out\} \quad (10)$$

To handle 'crashes with barriers' situations, we define *crash* as the *distFromStart* at the point where the crash occurred (without opponents within a radius of 15 m). Then, $Speed_{factor}$ for the following laps will be reduced by 10% in segment from 150 to 75 m before the *crash* point (Eq. 9) and by 20% in the segment from 75 m to the *crash* point (Eq.10).

$$Speed_{factor}(i) = 0.9 \times Speed_{factor}(i),\quad i = \{crash - 150, \ldots, crash - 75\} \quad (11)$$

$$Speed_{factor}(i) = 0.8 \times Speed_{factor}(i),\quad i = \{crash - 75, \ldots, crash\} \quad (12)$$

The recognition of 'long straights' is performed during the second and following laps, with the critical points being identified on the first lap. A vector $Speed_{previous}$ saves the speed at which the car drove on the last lap along the corresponding track segment.

If $Speed_{previous}(distFromStart, \ldots, distFromStart + end) > 180$ and $end > 25$ is true, this means that there has been a segment with length greater than 25 m in which *speed* was always greater than 180 km/h. It is thus considered a safe segment on which to drive faster on the next lap. As a consequence, the $Target_{speed}$ value is increased on the following laps by 50% in the first 75 m of the safe segment and by 25% in the next 50 m.

$$Speed_{factor}(i) = 1.25,\quad i = \{distFromStart + 75, \ldots, distFromStart + end - 25\} \quad (13)$$

$$Speed_{factor}(i) = 1.5,\quad i = \{distFromStart, \ldots, distFromStart + 75\} \quad (14)$$

Thus, given the $Speed_{factor}(current)$ (with $current = distFromStart$) and the $Target_{speed}$ value returned by the target speed module of Sect. 3.2, a new target speed is calculated as follows:

$$Target'_{speed} = Target_{speed} \times Speed_{factor}(current) \quad (15)$$

**Table 5** Thresholds and increments for manouvres to avoid opponents

| $deg$ | $inc^d_{deg}$ | $T^d_{deg}$ | $deg$ | $inc^d_{deg}$ | $T^d_{deg}$ |
|---|---|---|---|---|---|
| −30° | +0.25 | 10 | 30° | −0.25 | 10 |
| −20° | +0.25 | 10 | 20° | −0.25 | 10 |
| −10° | +0.25 | 10 | 10° | −0.25 | 10 |

## 4 Experimentation and results

Computational experimentation was carried out in order to test the controller's performance in as many situations as

possible. The experiments were divided into two stages: (i) 'laboratory' tests, and (ii) the *2009 Simulated Car Racing Competition*.

In the laboratory tests, two kinds of experiment were conducted. First, the controller was run alone on a set of tracks in order to evaluate its performance in several curve combinations. The results of these "racing alone" tests were compared with the results obtained by controllers in the *2008 Simulated Car Racing Competition* and with the previous version of the controller that is described in detail in (Onieva et al. 2009b). Second, several races were performed against controllers available within the TORCS environment.

In the *2009 Simulated Car Racing Competition*, the controller competed against 13 entries from distinct research groups. The following subsections describe each experimentation phase in some detail.

### 4.1 Laboratory experiments

A set of tracks used in the 2008 competitions was taken for our experiments. These were the *Ruudskogen*, *Street*-1, and *D-Speedway* tracks from the WCCI08 leg, and the *CG Speedway* 1, *E-Track*3, and *B-Speedway* from CEC07. The six tracks are shown in Fig. 8.

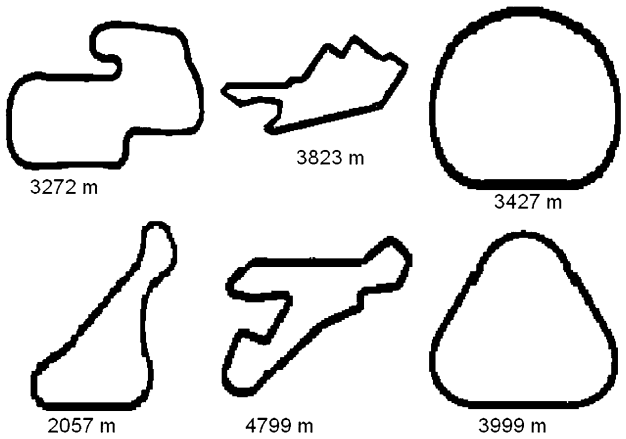


**Fig. 8** Tracks used in the WCCI2008 (*upper*) and CIG2008 (*lower*) competitions

**Table 6** Comparative results of the distance covered (metres) in 200 s of simulated time

| Track | 2008 Winner reference | Controller (Onieva et al. 2009b) | Current controller |
|---|---|---|---|
| *Ruudskogen* | 6,716.7 | 8,735.4 | 9,562.5 |
| *Street*-1 | 6,477.8 | 7,091.8 | 8,797.8 |
| *D-Speedway* | 14,406.9 | 15,612.3 | 16,241.9 |
| *CG Speedway* 1 | 7,131.7 | 8,970.4 | 10,005.1 |
| *E-Track* 3 | 7,651.1 | 9,117.4 | 9,987.3 |
| *B-Speedway* | 12,598.8 | 15,550.4 | 16,254.6 |

For each track, the longest distance raced by any of the cars in a simulation time of 200 s was taken (values from the 2008 competition). Then our controller was run on the same track for the same time. The results are given in Table 6, with the values corresponding to the distance covered (in metres). It can be readily observed that our proposal was the best alternative for every track.

In order to study the impact of the learning module on the lap times, 20 laps were now run over the six tracks of Fig. 8 with the learning module either activated or deactivated. The results are presented in Table 7, giving the time per lap with the learning module deactivated, and the reduction resulting from activating the module. Since the architecture presented is deterministic, and the learned knowledge is used from the second lap onwards, the improvement in $Lap_1$ is necessarily zero in all cases. With respect to the stability of the module, the time differences between laps 2 or 3 and lap 20 are minimal, so that stability is guaranteed. On *easy* tracks such as *D-Speedway* and *B-Speedway*, there was no improvement at all with the use of the learning module. On the rest of the tracks, the improvement was between 1 and 4 s in the best cases, i.e., an improvement of between about 1 and 4%. The value seemed to be related to the difficulty of the track (more difficult, greater improvement).

Given the excellent results in 'race alone' scenarios, the performance on many of the tracks provided by the TORCS engine was also evaluated in order to evaluate the controller's behaviour in critical curve combinations.

The opponents management module was also a critical aspect, since racing involves many unknown factors. For example, since the strategy of an opponent is unknown, the decision to brake or overtake could be a matter of "trusting

**Table 7** Results using the learning module

| | $Lap_1$ | $Lap_2$ | $Lap_3$ | $Lap_4$ | $Lap_5$ | ... | $Lap_{20}$ | Total |
|---|---|---|---|---|---|---|---|---|
| *Ruudskogen* | 75.9 | 70.1 | 69.8 | 70.0 | 69.9 | ... | 69.8 | 1,404.0 |
| *Learned* | 0.0 | −1.1 | −0.7 | −1.2 | −1.1 | ... | −0.8 | −18.3 |
| *Street*-1 | 95.8 | 91.0 | 91.5 | 91.7 | 91.2 | ... | 89.8 | 1,814.5 |
| *Learned* | 0.0 | −2.9 | −4.1 | −4.4 | −3.4 | ... | −2.4 | −53.8 |
| *D-Speedway* | 51.7 | 41.8 | 41.8 | 41.8 | 41.8 | ... | 41.8 | 846.0 |
| *Learned* | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| *CG-Speedway*1 | 48.2 | 42.6 | 42.6 | 42.6 | 42.5 | ... | 42.5 | 857.1 |
| *Learned* | 0.0 | −1.1 | −1.2 | −1.0 | −1.1 | ... | −1.2 | −21.2 |
| *E-Track*3 | 103.3 | 96.0 | 95.8 | 95.6 | 95.6 | ... | 95.4 | 1,910.6 |
| *Learned* | 0.0 | −2.6 | −3.7 | −2.4 | −2.2 | ... | −3.0 | −49.0 |
| *B-Speedway* | 58.6 | 48.8 | 48.8 | 48.7 | 48.8 | ... | 48.8 | 985.0 |
| *Learned* | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |

to luck", because if the opponent tries to avoid being overtaken it would cause a collision.

Full races were carried out confronting our controller with ten instances of the driver *Berniw Hist* available in the TORCS distribution. This driver has full knowledge of the track and the rest of the cars, while our approach is based only on local information, so the idea of beating these instances was in principle very challenging. The *Berniw Hist* model was chosen because its lap times were slightly slower than our approach, so that they were the perfect candidates to be overtaken several times.

Seventeen races were carried out on the same number of tracks. Our controller always started in the last position, with the ten *Berniw Hist* instances in front of it. It was able to finish in the first position in 14 races, in second position in 2, and in third in 1. The average damage incurred was 25%, with a maximum (worst case) of 59%.

The preliminary conclusions were therefore that excellent results were being obtained, since the final classification in full races was always between first and third positions despite starting in last position. The damage incurred after several overtaking manoeuvres and confronting opponents several times during the race was also reasonable.

But, as was noted earlier, the real experiment and comparison with other techniques was carried out in the context of the *2009 Simulated Car Racing Championship* since this allowed the behaviour of the controller to be tested in an objective comparison with other approaches. The following subsection describes the results provided by participation in the championship. Henceforth, in the tables of results our controller will be denominated *Onieva&Pelta*.

### 4.2 The 2009 Simulated Car Racing Championship

In 2009, The Simulated Car Racing Championship (Daniele et al. 2009) consisted of three legs held in congresses:

- IEEE Congress on Evolutionary Computation (CEC09), in Trondheim (Norway), in May 2009.
- Genetic and Evolutionary Computation Conference (GECCO09), Montreal (Canada), in July 2009.
- IEEE Congress on Computational Intelligence and Games (CIG09), Milan (Italy), in September 2009.

The model described here was presented without major modifications to the GECCO-2009 and CIG-2009 competitions after improvements had been made to a preliminary architecture presented to the CEC-2009 competition (Onieva et al. 2009b). In particular, it was clearly necessary to include a Learning Module after observing that the car went off the track at the same point in every lap. The TCL filter was added because, when the car went off the track, it sometimes skidded and lost control (even also getting blocked). Several other, but simple, modifications were made to the rest of the modules in order to improve the performance.

Each competition consisted of two stages: a warm-up and the actual race. In the warm-up, each driver races alone for 200 s. The eight drivers that cover the greatest distances qualify for the second stage in which they compete together in a series of ten five-lap races from a random starting grid.

Drivers were scored using the following scale: 10 points for first place, 8 points for second, 6 for third, 5 for fourth, 4 for fifth, 3 for sixth, 2 for seventh, and 1 for eighth. Two extra points were awarded to the driver performing the fastest lap in the race and to the driver completing the race with the least amount of damage. The final score for each driver was computed as the median of the 10 scores collected in the series of races.

Six, eleven, and thirteen entries were submitted respectively to CEC2009, GECCO2009, and CIG2009. In each case, the organizers included the champion of the 2008 competition (denominated Champ2008) as one of the entries. The tracks used in the races of the three legs of the championship are illustrated in Fig. 9. The *Michigan*, *Alpine*-2, and *Corkscrew* tracks were used at CEC2009; *Dirt*-3, *Alpine*-1, and *E-Road* at GECCO2009; and *Migrants*, *Buzzard* and *Forza* at CIG2009.

The following is a brief description of three of the entries presented to the competition. *Cobostar* (Butz and Lonneker 2009) maps sensory information onto actual

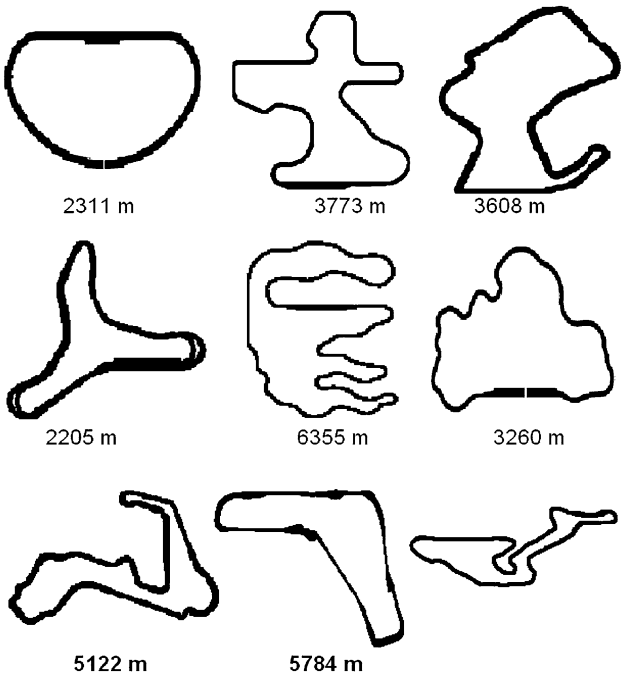


**Fig. 9** Tracks used in the CEC2009 (*top*), GECCO2009 (*middle*), and CIG2009 (*bottom*) competitions

motor behaviour, optimizing the mappings for on-track and off-track behaviour. *Perez & Saez* (Perez et al. 2009) implements a fuzzy controller evolved using a genetic algorithm to minimize lap times and damage incurred. *Jorge Muñoz* (Munoz et al. 2009) uses a feed-forward neural network and back propagation learning algorithm to imitate both a human driver and the winner of the 2008 competition. Most of the entries are summarized in (Loiacono et al. 2010), together with a detailed analysis of the full competition. It is important to note that exactly the same information is made available for all the bots. Also, all the entries had the opportunity of being improved and modified after each leg. Therefore, although the names of the teams were maintained throughout the competition, their corresponding bots had evolved.

Our entry's warm-up stage results are summarized in Table 8. In this stage, the controllers' performances are measured in terms of distance covered in 200 s driving along the track without opponents. Our controller classified for the full race stage in second place (taking into account the three tracks) in all three legs.

**Table 8** The controller's results in the warm-up stages

| Track | Distance (m) | Position | Number of entries |
|---|---|---|---|
| CEC2009 | | | |
| Michigan | 11,797.7 | 3 | 6 |
| Alpine-2 | 6,826.4 | 2 | 6 |
| Corkscrew | 3,377.7 | 4 | 6 |
| GECCO2009 | | | |
| Dirt-3 | 9,301.1 | 1 | 11 |
| Alpine-1 | 6,659.8 | 2 | 11 |
| E-Road | 8,386.5 | 2 | 11 |
| CIG2009 | | | |
| Migrants | 11,997.2 | 1 | 13 |
| Buzzard | 9,255.9 | 2 | 13 |
| Forza | 9,140.9 | 2 | 13 |

After the warm-up stage, the eight best controllers (all six entries in the case of CEC2009) were put together on the same track to measure their performance in a real racing environment with opponents. Figures 10, 11, and 12 summarize the results obtained in the respective competitions.

In CEC2009 (Fig. 10), the *Corkscrew* track had two critical points where our car left the track on each lap. These points were the reason for the poor results on this track in both the warm-up and full race stages. As noted earlier, in light of these results the architecture was modified significantly for the following two legs of the competition. In particular, we observed that inclusion of the learning module could have reduced lap times by up to 20 s (12%) on this track.

In GECCO2009 (Fig. 11), our controller won the races on the three tracks. It is important to note that in the warm-up stage first place was only achieved on one track. Intuitively, this suggests that it is more important in racing situations to have a good strategy (represented here by the opponents management module) than a fast controller. While the controller presented in this leg was not the fastest (as seen in the warm-up stage, Table 8), it clearly showed the best performance in this leg of the competition.

In addition, the points difference relative to COBOSTAR, the winner of the CEC2009 leg, was much greater than in that previous leg (−6.5 points in CEC2009; +15.5 in GECCO). The good results obtained on the *Dirt*-3 track were because it comprises a variety of roadway surfaces, so that skidding was very common. The ABS and TCL filters helped the controller in this situation, and the

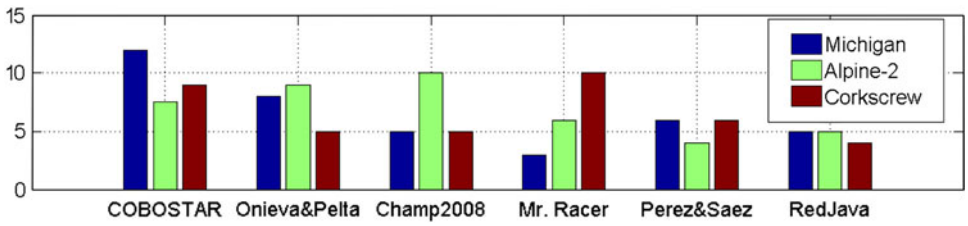


**Fig. 10** Final results in the CEC2009 leg of the competition

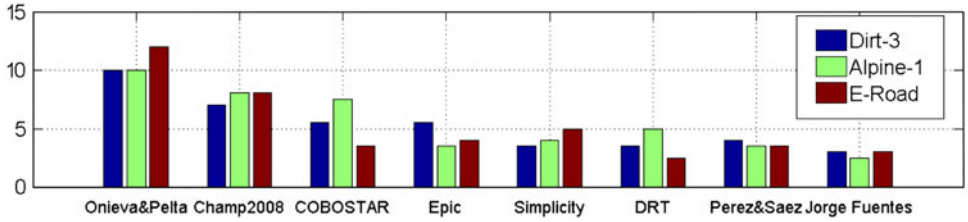


**Fig. 11** Final results in the GECCO2009 leg of the competition

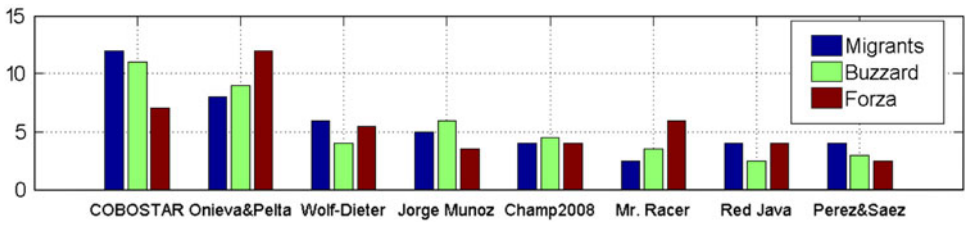


**Fig. 12** Final results in the CIG2009 leg of the competition

**Table 9** Final results of the 2009 Simulated Car Racing Competition. *NP* not presented, *NC* not classified

| | CEC2009 | GECCO2009 | CIG2009 | Total |
|---|---|---|---|---|
| Onieva&Pelta | 22 | 32 | 29 | 83 |
| COBOSTAR | 28.5 | 16.5 | 30 | 75 |
| Champ2008 | 20 | 23 | 12.5 | 55.5 |
| Perez&Saez | 16 | 11 | 12.5 | 36.5 |
| Mr. Racer | 19 | NC | 12 | 31 |
| Red Java | 14 | NC | 10.5 | 24.5 |
| Jorge Muñoz | NP | 8.5 | 14.5 | 23 |
| Wolf-Dieter | NP | NP | 15.5 | 15.5 |
| Epic | NP | 13 | NC | 13 |
| Simplicity | NP | 12.5 | NC | 12.5 |
| DRT | NP | 11 | NC | 11 |
| Witold | NP | NC | NC | 0 |
| Ebner&Tiede | NP | NP | NC | 0 |

critical points were remembered between laps by the learning module.

In CIG2009 (Fig. 12) the final results of our controller were similar to those obtained in the warm-up stage. Second place was achieved with only a single point behind COBOSTAR. The differences between COBOSTAR or Onieva&Pelta and the rest of the entries were greater than in the other two legs of the competition.

Table 9 summarizes the final results of the *2009 Simulated Car Racing Competition*, which are simply the sums of each leg's results. Our proposal won the championship. One notes that only the two top-placed controllers obtained higher scores than the winner of the 2008 competition.

## 5 Conclusions and future work

We have described our design, implementation, and testing of a driving architecture for a virtual car in the context of a car racing simulation game. The architecture is modular, with each module dealing with a specific task. Low-level modules manage the pedals (throttle, brake), gear shifting, and steering wheel. Higher-level modules are responsible for determining the target speed allowed by the track (straights or curves), for sending modifications to lower level modules to adapt their behaviour to the proximity of opponents (overtaking and avoidance), and for remembering situations between laps in order to run the following lap faster than the previous one.

The computational experiments allowed us to conclude that the proposed controller provided fast driving performance (better than the controllers that had competed in the 2008 conferences), and that it was also very efficient in racing situations—the car won the 2009 Simulated Car Racing Championship.

Our experience with this controller shows that one of the key aspects for success is the determination of the target speed on every sector of the track. An appropriate target speed can not only allow the car to drive faster around the track, but can also avoid losses of control in which the car may drive off the track, crash, or get into a blocked situation. A fuzzy controller was designed for this task. It implements three input variables, each with three trapezoidal membership functions and an output variable codified by singletons. This schema endows the controller with the short processing time needed to be able to quickly infer output values in a real-time environment such as handling simulated vehicle control.

The other important aspect is a good opponent management policy in order to avoid crashing with other cars. The rather simple set of crisp rules we implemented in the design seemed sufficient to provide a reasonably good policy. However, in terms of the development of non-player characters, such crispness is not a good feature. We therefore plan to manage opponents in a fuzzy (not crisp) manner in order to smooth out the corresponding control actions.

In the current implementation, the parameters of the modules were hand-tuned. Clearly, an avenue to explore is the application of soft computing techniques to the learning and adaptation of the architecture's parameters. We made some preliminary trials attempting to improve the parameters of the fuzzy system using genetic algorithms (Herrera et al. 1995; Homaifar and McCormick 1995; Onieva et al. 2009a), but achieved no clear overall benefits. Nevertheless, we did observe a kind of over-fitting to a particular track, thus opening the way to considering genetic tuning for specific tracks. One drawback that we noted was that the initial feature of interpretability of the fuzzy rule base was almost lost.

From the perspective of "online" adaptation, much work remains to be done. Our architecture at present provides a learning module that detects safe or dangerous track segments where speeds should be increased or decreased, respectively. We hypothesize that temporal fuzzy logic will allow human racing knowledge to be included in the form of, for instance, *if this is one of the last laps and you are in a low-placed position, then drive more aggressively*. The strategies developed in this way should lead to better results.

To conclude, we should just like to remark that one of the most interesting lessons we have learnt from this experience is that videogames (such as car racing games and others) constitute excellent examples of complex, dynamic, and uncertain environments—precisely the environments in which soft computing techniques should play a key role.

**Acknowledgments** This work was supported by the Spanish Ministry of Science and Innovation under the projects Tránsito (TRA2008-06602-C03-01), CityElec (PS-370000-2009-4), and TIN2008-01948, by the Ministerio de Fomento under the project GUIADE (T9/08), and by the Consejería de Innovación, Ciencia y Empresa, Junta de Andalucía, under Project P07-TIC-02970. The authors wish to thank the organizers of the 2009 Simulated Car Racing Competition for their effort in organizing the event and for allowing our team to participate without having submitted associated publications.

## References

Arroyabe JL, Aranguren G, Nozal LAL, Martin JL (2000) Autonomous vehicle guidance with fuzzy algorithm. In: Proceedings of 26th annual conference of the IEEE Industrial Electronics Society IECON 2000, vol 3, pp 1503–1508

Butz MV, Lonneker TD (2009) Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In: IEEE symposium on computational intelligence and games, 2009 (CIG 2009), 7–10 September 2009, pp 317–324

García R, de Pedro T (1998) Modeling a fuzzy coprocessor and its programming language. Mathw Soft Comput 5(2–3):167–174

Herrera F, Lozano M, Verdegay JL (1995) Tuning fuzzy logic controllers by genetic algorithms. Int J Approx Reason 12:299–315

Homaifar A, McCormick E (1995) Simultaneous design of membership functions and rule sets for fuzzy controller using genetic algorithms. IEEE Trans Fuzzy Syst 3(2):129–139

Hong TP, Huang KY, Lin WY (2002) Applying genetic algorithms to game search trees. Soft Comput: A Fusion Found Methodol Appl 6(3):277–283

Hsu F (2002) Behind deep blue. PhD thesis, Princeton University Press

Laird JE (2002) Research in human-level AI using computer games. Commun ACM 3(8):32–35

Loiacono D, Togelius J, Lanzi PL (2008a) Car racing competition WCCI2008. Software manual

Loiacono D, Togelius J, Lanzi PL, Kinnaird-Heether L, Lucas SM, Simmerson M, Perez D, Reynolds RG, Saez Y (2008b) The WCCI 2008 simulated car racing competition. In: Proceedings of the IEEE symposium on computational intelligence and games

Loiacono D, Togelius J, Lanzi PL (2009) Simulated car racing. In: Proceedings of IEEE symposium on computational intelligence and games CIG 2009, 7–10 September 2009, p 1

Loiacono D, Lanzi PL, Togelius J, Onieva E, Pelta DA, Butz MV, Lonneker TD, Cardamone L, Perez D, Saez Y, Preuss M, Quadflieg J (2010) The 2009 simulated car racing championship. IEEE Trans Comput Intell AI Games 2(2):131–147

Lucas SM (2009) Computational intelligence and AI in games: a new IEEE transactions. IEEE Trans Comput Intell AI Games 1(1):1–3

Munoz J, Gutierrez G, Sanchis A (2009) Controller for torcs created by imitation. In: IEEE symposium on computational intelligence and games, 2009 (CIG 2009), 7–10 September 2009, pp 271–278

Onieva E, Alonso J, Pérez J, Milanés V, de Pedro T (2009a) Autonomous car fuzzy control modeled by iterative genetic algorithms. In: Proc 2009 IEEE international conference on fuzzy systems

Onieva E, Pelta DA, Alonso J, Milanes V, Perez J (2009b) A modular parametric architecture for the torcs racing engine. In: Proc IEEE symposium on computational intelligence and games (CIG 2009), 7–10 September 2009, pp 256–262

Perez D, Recio G, Saez Y, Isasi P (2009) Evolving a fuzzy controller for a car racing competition. In: IEEE symposium on computational intelligence and games, 2009 (CIG 2009), 7–10 September 2009, pp 263–270

Rosca JP (1996) Generality versus size in genetic programming. In: Genetic programming 1996: proceedings of the first annual conference. Stanford University, MIT Press, Cambridge, pp 381–387

Schaeffer J (2009) One jump ahead: computer perfection at checkers. Springer-Verlag, New York

Schaeffer J, Burch N, Yngvi Björnsson AK, Mueller M, Lake R, Lu P, Sutphen S (2007) Checkers is solved. Sci Agric 317:1518–1522

Schloman J, Blackfordm B (2007) Genetic programming evolves a human competitive player for a complex, on-line, interactive, multiplayer game of strategy. In: Proceedings of the genetic and evolutionary computation conference, pp 1951–1958

Siegel EV, Chaffee DA (1996) Genetically optimizing the speed of programs evolved to play tetris. In: Advances in genetic programming, vol 2. MIT Press, Cambridge, pp 279–298

Togelius J, Lucas S, Duc Thang H, Garibaldi JM, Nakashima T, Hiong Tan C, lhanany I, Berant S, Hingston P, MacCallum RM, Haferlach T, Gowrisankar A, Burrow P (2008) The 2007 IEEE CEC simulated car racing competition. Genet Program Evol Mach 9:295–329

Zadeh LA (1965) Fuzzy sets. Inf Control 8:338–353