A Modular Parametric Architecture for the TORCS Racing Engine

E. Onieva, D. A. Pelta, J. Alonso, V. Milanés, J. Pérez

***Abstract*— This paper presents our approach to TORCS Car Racing Competition 2009, it is based on a complete modular architecture capable of driving automatically a car along a track with or without oppents. The architecture is composed of five simple modules being each one responsible for a basic aspect of car driving. The modules control gear shiftings, steer movements and pedals positions by using of simple functions meanwhile the allowed speed in a certain track segment is managed by a simple TSK fuzzy system.**

Additionally, a module is in charge of modifying outputs' values in case of other cars presence in the environment, with the aim to overtake and avoid collisions with other cars.

As a first approach, we provide a "hand-tuned" version of the controllers that allow to achieve very good results, and excellent ones in some particular tracks when compared with last year's competitors.

The modules are highly intuitive and these preliminary results open the way to apply soft computing techniques to perform an automatic parameters adjustment for future competitions. Moreover, the modularity of the architecture will allow us to replace or add modules in future, as a way to enhance particular features of the controller.

I. Introduction

Simulated car racing competitions are becoming more popular in the field of computational intelligence since they provide an excellent framework to test learning, adaptability, evolution and reasoning features of algorithms under investigation.

The first competition was held in 2007 as part of the IEEE Congress on Evolutionary Computation (CEC) and Computational Intelligence and Games Symposium (CIG); these competitions used a graphically and mechanically simple game which bring about a good degree of participation. The organization, submitted entries and results of these competitions are published in [1]. Then, in 2008, in conjunction with the IEEE World Congress on Computational Intelligence (WCCI) a car racing competition based on TORCS (The Open Racing Car Simulator) was organized[2]. TORCS opened the possibility of simulate more complex racing simulations, specially due to the possibility of having many cars in the same track. Results of the WCCI competition are published in [3].

In this paper our contribution to the 2009 competition is presented. Its aim is 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.

Enrique Onieva, J. Alonso, V. Milanés and J. Pérez are with Industrial Computer Science Department, Instituto de Automática Industrial, Consejo Superior de Investigaciones Cientificas, La Poveda-Arganda del Rey, Madrid 28500, Spain (e-mail: {onieva, jalonso, vmilanes, jperez}@iai.csic.es). David Pelta is with Computer Science and Artificial Intelligence Department, Universidad de Granada, 18071 Granada, Spain (e-mail: dpelta@decsai.ugr.es)

The control architecture is composed by five basic modules: 1) gear control; 2) low level throttle and brake control to keep or reach certain target speed, 3) target speed determination. It is a TSK fuzzy controller [4] that infers the target speed value by means of a reduced set of simple fuzzy rules; 4) Steering wheel control; 5) Opponents modifier. This last module analyzes the opponents' positions and applies modifications over the steering, throttle and brake values in order to avoid collisions and overtake other drivers.

The main idea behind the architecture is to have a small set of simple and interpretable modules whose interactions lead to a good driving.

In order to fully present the architecture, the contribution is organized as follows:

Section 2 presents the racing simulator used to run the experiments, as well as the API provided by the organization to carry out experimentation in a car racing environment. Section 3 shows the architecture presented in this work, where details about the modules that compose it are explained. Section 4 presents comparative results with last year edition competitors, as well as results obtained in the frame of the *2009 Simulated Car Racing Championship* and its first competition held at IEEE Congress on Evolutionary Computation. Finally Section 5 provides concluding remarks about the results, as well as considerations about how to improve the current architecture in future works.

II. The TORCS Simulator

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

- It lies between an advanced simulator, like recent commercial car racing games, and a fully customizable environment, like the ones 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 as a free alternative to commercial racing games, but it was specifically devised to make it as easy as possible to develop your own controller.

In fact, controllers are implemented as separated software modules ,so it is easy to develop a new controller and to plug it into the game.

Within the 2007 competition, competitors were provided with a specific software interface developed on a client/server basis where the designed controllers run as external programs

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

978-1-4244-4815-9/09/$25.00 ©2009 IEEE

and communicate with a customized version of TORCS though UDP connections. The architecture is shown in Figure 1. The controller perceives the racing environment though a number of sensor readings which would reflect both the surrounding environment and the current game state and they could invoke basic driving commands to control the car. The complete list of sensors is reported in Table I.

Name	Range (unit)	Description
angle	[-π,+π] (rad)	Angle between the car direction and the direction of the track axis.
curLapTime	[0,-] (s)	Time elapsed during current lap.
damage	[0,-] (point)	Current damage of the car.
distFromStart	[0,-] (m)	Distance of the car from the start line along the track line.
distRaced	[0,-] (m)	Distance covered from the beginning of the race.
fuel	[0,-] (l)	Current fuel level.
gear	{-1,0,1,2,...,6}	Current gear: -1 is reverse, 0 is neutral and the gear from 1 to 6.
lastLapTime	[0,-] (s)	Time to complete the last lap
opponents	[0, 100] (m)	Vector of 36 sensors that detects the opponent distance in meters within a specific 10 degrees sector: each sensor covers 10 degrees, from -π to π in front of the car.
racePos	1, 2, ...	Position in the race with to respect to other cars.
rpm	[2000 - 7000] (rpm)	Number of rotation per minute of the car engine.
speedX	- (km/h)	Speed of the car along the longitudinal axis of the car.
speedY	- (km/h)	Speed of the car along the transverse axis of the car.
track	[0,100] (m)	Vector of 19 range finder sensors: each sensors represents the distance between the track edge and the car. Sensors are oriented every 10 degrees from -π/2 and +π/2. Distance are in meters. When the car is outside of the track, these values are not reliable!
trackPos	-	Distance between the car and the track axis. The value is normalized w.r.t to the track width: it is 0 when 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 of the car. Values greater than 1 or smaller than -1 means that the car is outside of the track.
wheelSpinVel	[0,-] (rad/s)	Vector of 4 sensors representing the rotation speed of wheels.

TABLE I

DESCRIPTION OF AVAILABLE SENSORS. RANGE IS REPORTED WITH THE CORRESPONDING UNIT MEASURE (WHERE IT IS POSSIBLE). SYMBOL - MEANS THAT RANGE IS NOT LIMITED.

Once the controller obtain the sensors values from the server, it will infer output commands to carry out the driving of the simulated car by means of the effectors of the car. Effectors represent what action the client bot can perform and, therefore, how it can affect the current game state. Table II shows a detailed description of the effectors available. While `accel`, `brake`, `gear`, and `steer` effectors represents properly some

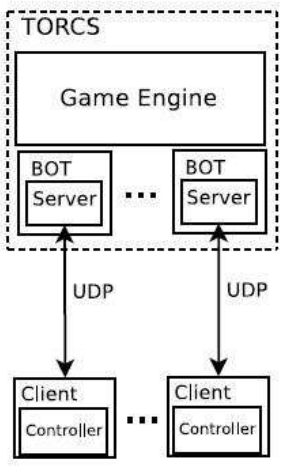

Fig. 1. The architecture of the API developed for the WCCI 2008 competition.

control actions on the car, the effector `meta` is used to affect the status of the game and was introduced with the purpose of restarting the current race. See [5] for additional details.

Name	Range	Description
accel	[0,1]	Virtual gas pedal (0 means no gas, 1 full gas).
brake	[0,1]	Virtual brake pedal (0 means no brake, 1 full brake).
gear	-1,0,1,...,6	Gear value.
steer	[-1,1]	Steering value: -1 and +1 means respectively full right and left, that corresponds to an angle of 0.785398 rad.
meta	0, 1	This is meta-control command: 0 do nothing, 1 ask competition server to restart the race.

TABLE II

DESCRIPTION OF AVAILABLE EFFECTORS

III. PARAMETRIZED CONTROL ARCHITECTURE

The methodology and techniques used to design a suitable controller are explained in this section. The global control architecture is showed in Figure 2, where the modular design is shown.

The basic architecture is composed by 5 modules. The motivation of this work is to create simple controllers for each one of the modules of the architecture, looking for the best configuration possible. The basic functionality of each module is explained below:

1) The Gear Control module is in charge of shifting between gears.
2) The Desired Speed Module assigns the allowed speed for a certain track segment.
3) The Low Level Gas & Brake Control Module is in charge of following the indications of the desired speed module. That is, to reach or keep a certain speed by means of the use of gas and brake pedals.

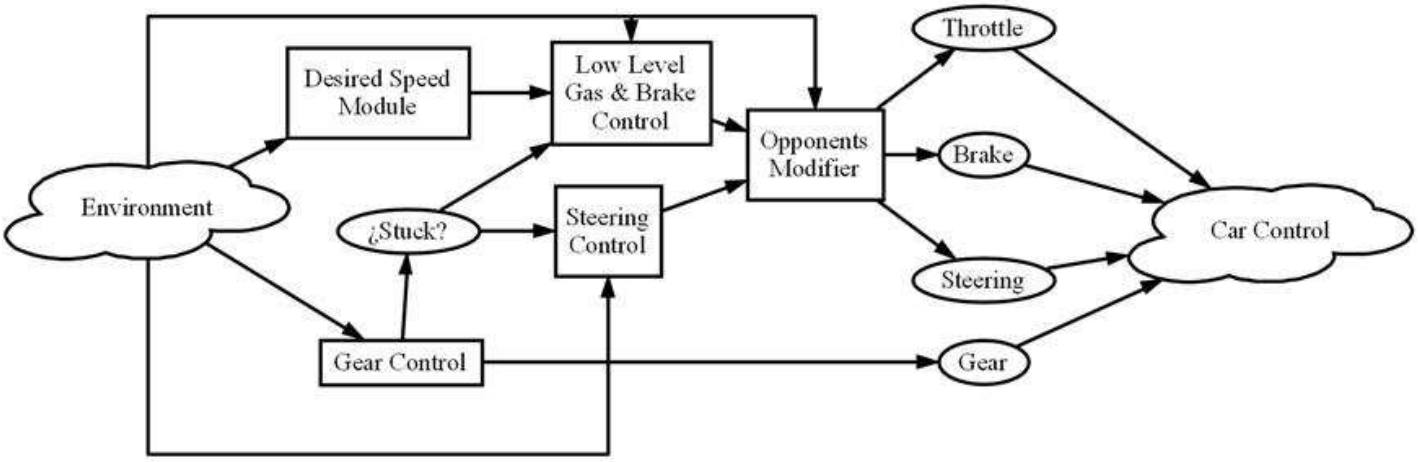

Fig. 2. Scheme of the Control Architecture proposed.

4) The Steering Wheel Control is in charge of controlling the car direction.
5) The Opponents Modifier Module works when opponents are present. It will apply changes over the steering, gas and brake values to adapt the driving in presence of other cars, trying to avoid collisions and overtake other opponents.

This architecture will allow us to change easily each module and test the suitability of the interactions between all of them to carry out the handling of the vehicle and measuring its performance based in two features: *distRaced* and *damage* (see Table I). It is important to note that modules 1 to 4 are enough to implement a car control able to drive alone in a track; the fifth module adds features to deal with opponents.

Each module will be describen in following sections.

A. Gear Control

The Gear Control increases the current gear if the rpm value is higher than a certain value and decreases it if the *rpm* value is lower than a certain value. A special case is the use of reverse gear and the change from reverse to first gear again because these changes are not realized by observing the *rpm* value but observing the environment of the car (if the car is or not stuck).

$GI_i, i = 1,2,3,4,5$ and $GD_i, i = 2,3,4,5,6$ represent the rpm value used to increase or decrease the current gear respectively. Additionally, to avoid shift problems, a gear must to be maintained during, at least 20 time steps. Values assigned to GI and GD are showed in Table III.

i	1	2	3	4	5	6
GI_i	8000	8000	8000	8000	8000	
GD_i		2500	3000	3000	3500	3500

TABLE III
GEAR CHANGING CONSTANTS BETWEEN FIRST AND SIXTH GEAR.

The use of reverse gear needs a particular attention because it has to be defined when the car is stuck. We define a counter $Stuck_{time}$ to record the number of consecutive game turns while the condition $(|angle| > \pi/6)$ is satisfied. When $Stuck_{time} = 25$, it is considered that the car is stuck and reverse gear is applied until $(angle \times trackPos > 0)$ or $(front > 10$ and $|angle| < \pi/2)$ is satisfied; in this moment the gear change from reverse to first gear.

B. Desired Speed Module

This module is based on the so-called fuzzy rule based systems and its main aim is to calculate the $Target_{Speed}$ value at a certain time step.

Fuzzy rule-based systems are considered one of the most important applications of the fuzzy set theory suggested by Zadeh [6]. They represent an extension of classical rule-based systems because they deal with fuzzy rules instead of classical control rules. The method of fuzzy inference proposed by Takagi, Sugeno and Kang [7], [4], which is known as the TSK model in fuzzy systems literature, has been one of the major topics in theoretical studies and practical applications of fuzzy modeling and control.

The controller is based on a computational model of a fuzzy coprocessor named ORBEX (Spanish acronym for Fuzzy Experimental Computer) [8]. ORBEX implements fuzzy system with singleton shapes in output variables. Sugeno et al. [9] proved that a fuzzy system modeled with singleton consequents is a special case of a fuzzy system modeled with trapezoidal consequents and can do almost everything the latter can. Singletons are very commonly used in practical control system applications; and that is the reason because ORBEX and our present work use singletons as consequents of its fuzzy rules.

In this module, the $Track$ information coming from the server is used to determine the $Target_{Speed}$ value when the car is inside the track axis, $TrackPos$ and $Angle$ variables are used when it is outside.

When the car is inside of the track, the fuzzy controller uses three of the 19 track sensors available as inputs:

- $Front$: track sensor that measures the front distance to the track axis.

- Max_{10}: maximum value between the track sensors at ± 10 degrees.
- Max_{20}: maximum value between the track sensors at ± 20 degrees.

All the inputs are codified with three trapezoidal membership functions called $\{Low, Medium, High\}$. Their shapes are represented by 3 real values for each input $\{l_f, m_f, h_f\}$, $\{l_{10}, m_{10}, h_{10}\}$ and $\{l_{20}, m_{20}, h_{20}\}$. In this work it will be use the conditions $(l_f = l_{10} = l_{20})$, $(m_f = m_{10} = m_{20})$ and $(h_f = h_{10} = h_{20})$ in the way showed in figure 3.

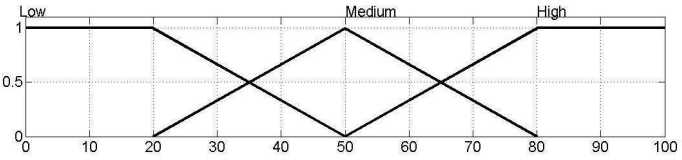

Fig. 3. Membership functions $\{Low, Medium, High\}$ for each input.

When the car is inside the track, the $Target_{Speed}$ is calculated using the following seven rules:

- R_1: If $Front$ is $High$ Then $Target_{Speed}$ is $TS1$
- R_2: If $Front$ is $Medium$ Then $Target_{Speed}$ is TS_2
- R_3: If $Front$ is Low and Max_{10} is $High$ Then $Target_{Speed}$ is TS_3
- R_4: If $Front$ is Low and Max_{10} is $Medium$ Then $Target_{Speed}$ is TS_4
- R_5: If $Front$ is Low and Max_{10} is Low and Max_{20} is $High$ Then $Target_{Speed}$ is TS_5
- R_6: If $Front$ is Low and Max_{10} is Low and Max_{20} is $Medium$ Then $Target_{Speed}$ is TS_6
- R_7: If $Front$ is Low and Max_{10} is Low and Max_{20} is Low Then $Target_{Speed}$ is TS_7

, where $TS_i = \{200, 175, 150, 125, 100, 75, 50\}$, $i = 1...7$ represent a discrete value (singleton shape) associated as consequent of the *ith* rule. An exception to this fuzzy system is applied when $Front = 100$ (the maximum posible value) where $Target_{Speed} = 300$ is used.

When the car is outside of the track axis, the $Target_{Speed}$ value is calculated using the $TrackPos$ and $Angle$ values obtained from the server. Only a value is used to codify the membership functions for implementing this controller, since $TrackPos$ is codified like a bi-valued function (it is $left$ when $TrackPos < -1$ and $rigth$ when $TrackPos > 1$) and $Angle$ is represented as shown in Figure 4.

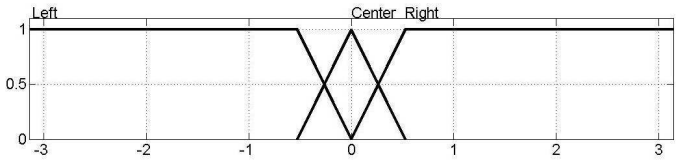

Fig. 4. Membership functions $\{Left, Center, Right\}$.

The rule base is composed by five additional rules

- R_8: If $Angle$ is $Center$ Then $Target_{Speed}$ is TS_8
- R_9: If $Angle$ is $Left$ and $TrackPos$ is $Left$ Then $Target_{Speed}$ is TS_9
- R_{10}: If $Angle$ is $Left$ and $TrackPos$ is $Right$ Then $Target_{Speed}$ is TS_{10}
- R_{11}: If $Angle$ is $Right$ and $TrackPos$ is $Left$ Then $Target_{Speed}$ is TS_{10}
- R_{12}: If $Angle$ is $Right$ and $TrackPos$ is $Right$ Then $Target_{Speed}$ is TS_9

As before $TS_i = \{125, 100, 50\}$, $i = 8, \ldots, 10$ represents the discrete value associated with the consequent's singleton of each rule; rules (9 and 12) and (10 and 11) must have associated the same value because of symmetry.

C. Low Level Gas & Brake Control

This control is in charge to return a single $AccelBrake$ signal. It combines both pedals' signals in a common output, in order to avoid non-sense values. The control distinguishes two cases:

- When reverse gear is applied, $AccelBrake = AB_{reverse} = 1$. Applying so, a constant value.
- Otherwise $AccelBrake = f_{AB}(speed - Target_{speed})$. Using as output, a function of the difference between the current speed and the target speed obtained from the higher level module. The formula used to implement this simple speed control is defined in (1).

$$f_{AB}(Diff) = \frac{2}{1 + e^{Diff}} \tag{1}$$

Additionally, an ABS filter function is implemented in order to avoid slips of the car when a brake signal is applied. The filter is implemented by reducing the brake signal (if AccelBrake<0) as showed in (2).

$$AccelBrake = AccelBrake - \frac{speed - speed_{wheels} - 1.5}{5}, \quad if\,(speed - speed_{wheels}) > 1.5 \tag{2}$$

, where $speed_{wheels}$ is the averaged speed of the four wheels of the car.

D. Low Level Steering Control

Three cases are considered to control of the steering wheel.

The first one is when the car is inside the track axis and is not using the reverse gear. In this situation 9 $Track$ sensors that go from -40 to 40 degrees are used. The output value is calculated as a weighted average from all the sensors' values, aditionally, a correction factor F_0 which will depend on the value of the frontal sensor is used. This factor will be useful, for example, to reduce the steering action in straight segments. We define $F_0 = 0.2$ if $T_0 = 100$ and $F_0 = 1$ in other cases.

Equation 3 shows the calculation, where $w_t, t = -40, ..., 40$ represents the weight to apply to each sensor value, it is used $w_t = t/20$.

$$F^{inside}_{Straight}(T_{-40}, ..., T_{40}) = F_0 \times \frac{\sum T_t \times w_t}{\sum T_t} \tag{3}$$

The second case is used when reverse gear is applied, and the third case is applied when the car is outside of the track

axis. Equations 4 and 5 describe the calculations for both scenarios respectively.

$$F_{Reverse}(angle) = \frac{-angle}{steer_{lock}} \quad (4)$$

$$F^{outside}_{Straight}(angle, trackPos) = \frac{angle - trackPos \times 0.5}{steer_{lock}} \quad (5)$$

, where $Steer_{lock} = 0.785398$ in both cases since this number represents the angle in radians of the car when a full steering turn is applied [5].

E. Opponents Modifier

This module is in charge of adapting the driving behavior when opponents are present. It will modify or replace outputs obtained from the steering, gas and brake controllers.

The modification of the gas & brake control output will be performed using the information coming from $Opponents$ sensors in the range of -40 to 40 degrees. Braking with a constant value of -0.5 if the current speed is higher than 70km/h and a security distance is violated. More precisely, the controller checks if $(Opponents_{\pm degrees} < Tol^{brake}_{\pm degrees})$ is true for $degrees = \{40, 30, 10, 0\}$. In other words, a certain minimum distance must be maintained from the opponents in every direction tested. The subindex represents the orientation of the opponent sensor being consider or its respective tolerance. The tolerance values are shown in Table IV.

Parameter	Description	Value
$Tol^{brake}_{\pm 40}$	Tolerance for opponents sensors at ± 40	6 m
$Tol^{brake}_{\pm 30}$	Tolerance for opponents sensors at ± 30	6.5 m
$Tol^{brake}_{\pm 20}$	Tolerance for opponents sensors at ± 20	7 m
$Tol^{brake}_{\pm 10}$	Tolerance for opponents sensors at ± 10	7.5 m
Tol^{brake}_{0}	Tolerance for opponents sensors at 0	8 m

TABLE IV
OPPONENTS MODIFIER OVER THE GAS & BRAKE ACTION.

In order to perform the overtaking manoeuver, the steering value should be modified. In this case, we consider $Opponents$ sensors from -100 to 100 and each one applies an increment of $\pm Inc^{overtake}_{degrees}$ over the steering output when the corresponding sensor's value is lower that certain tolerance $(Opponents_{\pm degrees} < Tol^{overtake}_{\pm degrees})$. Due to simmetry, $degrees = \{100, 90, ..., 0\}$, tolerance and increments values are sumarized in Tables V and VI

Parameter	Description	Value
$Tol^{overtake}_{-10,0,10}$	Tolerance for opponents sensors at ± 10 and 0	20 m
$Tol^{overtake}_{\pm 20}$	Tolerance for opponents sensors at ± 20	18 m
$Tol^{overtake}_{\pm 30}$	Tolerance for opponents sensors at ± 30	16 m
$Tol^{overtake}_{\pm 40}$	Tolerance for opponents sensors at ± 40	14 m
$Tol^{overtake}_{\pm 50}$	Tolerance for opponents sensors at ± 50	12 m
$Tol^{overtake}_{\pm d}$	Tolerance for opponents sensors at d>50	10 m

TABLE V
OPPONENTS SENSORS TOLERANCES FOR OVERTAKING.

Parameter	Description	Value
$Inc^{overtake}_{-10,0,10}$	Increment for opponents sensors at ± 10 and 0	± 0.20
$Inc^{overtake}_{\pm 20}$	Tolerance for opponents sensors at ± 20	± 0.18
$Inc^{overtake}_{\pm 30}$	Tolerance for opponents sensors at ± 30	± 0.16
$Inc^{overtake}_{\pm 40}$	Tolerance for opponents sensors at ± 40	± 0.14
$Inc^{overtake}_{\pm 50}$	Tolerance for opponents sensors at ± 50	± 0.12
$Inc^{overtake}_{\pm d}$	Tolerance for opponents sensors at d>50	± 0.10

TABLE VI
OPPONENTS SENSORS INCREMENTS FOR OVERTAKING.

IV. EXPERIMENTATION AND RESULTS

In order to show the benefits of our proposal, we will present two different validations. First one is in terms of distance raced in comparison with 2008 competition results, while the second one shows the results of the first race of the 2009 competition.

A. Distance Raced

Two different sets of tracks will be used in this phase of the experimentation, the first one shows results from six tracks available in TORCS that were used in 2008 competitions: the *Ruudskogen*, *Street-1* and *D-Speedway* were used at *IEEE World Congress on Computational Intelligence* [2]; and *CG Speedway 1*, *E-Track3* and *B-Speedway* were used at *Computational Intelligence and Games Symposium* [3]. All the tracks used in this experimentation phase are showed in figure 5.

From each track, the longest distance raced by all of the car's bot in a simulation time of 200s are taken. Then, our controller is executed in the same tracks for the same time and the results are displayed Table VII. Values shows the mean distance raced (in meters) by controllers in 10 executions over the track.

Results show a good behavior in the drive-alone tests carried out in comparison with the controllers submitted in last year competition. The cooperative-competitive behavior showed in presence of opponents has been tested with several bots of the TORCS simulator, showing reasonable responses both in overtaking and avoiding collisions.

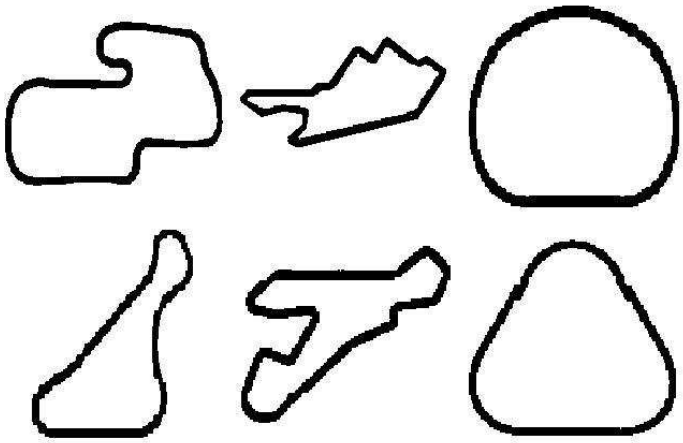

Fig. 5. Tracks used to evaluate the controller

Track	Last's year reference (m)	Proposed controller (m)	Improvement obtained (%)
Ruudskogen	6716.7	8735.4	+30.1
Street-1	6477.8	7091.8	+9.5
D-Speedway	14406.9	15612.3	+8.4
CG Speedway 1	7131.7	8970.4	+25.8
E-Track 3	7651.1	9117.4	+19.2
B-Speedway	12598.8	15550.4	+23.4

TABLE VII
COMPARATIVE RESULTS IN DISTANCE RACED IN 200S OF SIMULATED TIME

These results encouraged us to participate in a simulated car racing championship. In the next section, we report our results in the first race.

B. First Full Race

The second test carried out in the context of the *2009 Simulated Car Racing Championship,* composed by three simulated car racing competitions held at: *IEEE Congress on Evolutionary Computation* (CEC[2]), *Genetic and Evolutionary Computation Conference* (GECCO[3]) and *Computational Intelligence and Games Symposium* (CIG[4]).

Teams will be allowed to update their driver during championship, by submitting a different driver to each leg. Each competition will consist of two stages: the warm-up and the actual race. In the warm-up, each driver will race alone for 10000 game tics, approximatively 3 minutes and 20 seconds of actual game time. The eight drivers that will cover the more distance will qualify for the second stage. In the second stage, the eight drivers will race together. Each race consists of ten trials. The goal of each trial is to complete five laps from a randomly generated starting grid. At the end of each trial, the drivers will be scored using the F1 system: 10 points to the first controller that completed the three laps, 8 points to the second one, 6 to the third one, 5 to the fourth, 4 to the fifth one, 3 to the sixth, 2 to the seventh, and 1 to the eighth. The driver performing the fastest lap in the race will get two additional points. The driver completing the race with the smallest amount of damage will receive two extra points. The final score for each driver in the Grand Prix will be computed as the median of the 10 scores collected during the trials.

For this reason, the distance raced has been considered as main measure to evaluate the performance of the parameter assignment, also taking into account the damage obtained. It is important to note that an equilibrium between both is necessary because, according with the last year's competition rules, two different stages were used to evaluate the controllers; measuring the distance raced by the car along a track in a simulation time of 200s and a tournament of competitive races with several controllers in direct competence where the time of the race is unknown. So it has to be considered that a controller able to run a certain distance in 200s with a certain total damage maybe the controller will not be able to finish a longer race because the damage obtained in more race time reach the maximum and the car would be retired from the race (finishing in the last position).

The first leg of the championship was held at CEC on May 18-21, 2009. Including us, 6 racing teams participated. One of them is the champion of the previous edition at CIG. It has been developed by Luigi Cardamone and it is called *Champ CIG2008*.

In the warm-up stage, the controllers were tested in three different unknown tracks. Results obtained by all the teams are shown in Figures 6, 7 and 8, where our developed controller (called *Onieva & Pelta*) classified in third, second and fourth position, respectively.

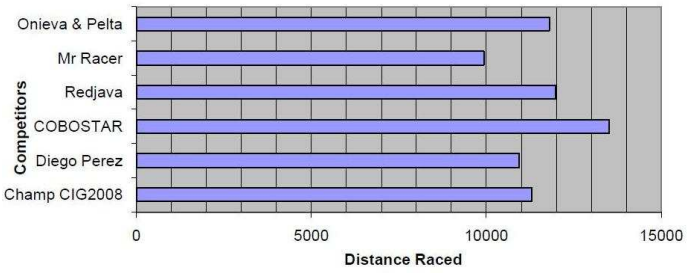

Fig. 6. First track: results of the warm-up stage

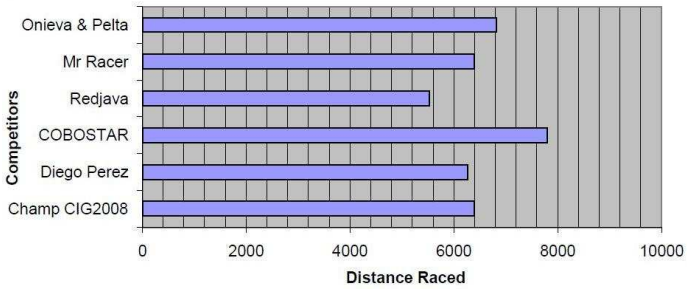

Fig. 7. Second track: results of the warm-up stage

The race itself, which evaluates the behavior of the controllers in presence of opponents was ran in tracks *Michigan*,

[2]http://www.cec-2009.org/
[3]http://www.sigevo.org/gecco-2009/
[4]http://www.ieee-cig.org/

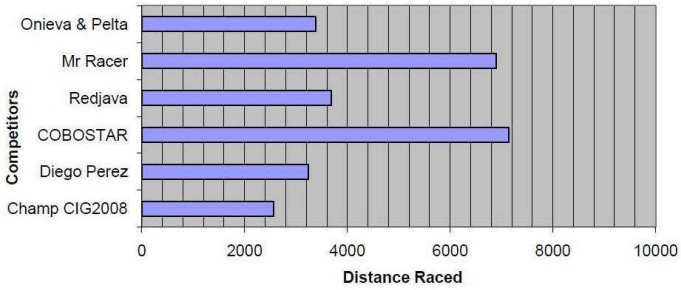

Fig. 8. Third track: results of the warm-up stage

Alpine 2 and *Corkscrew*, showed in Figure 9. The results (in points) are summarized in Table VIII. Our proposal obtained the second place, being this our first participation in a competition of this kind.

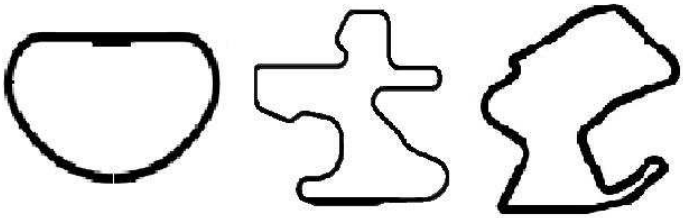

Fig. 9. Tracks used to evaluate the controller at CEC-2009

	Michigan	Alpine 2	Corkscrew	Total
COBOSTAR	12	7.5	9	28.5
Onieva & Pelta	8	9	5	22
Champ CIG-2008	5	10	5	20
Mr. Racer	3	6	10	19
Perez & Saez	6	4	6	16
Redjava	5	5	4	14

TABLE VIII
FINAL RESULTS OBTAINED AT CEC-2009

V. CONCLUSIONS AND FUTURE WORK

This work presented a complete modular architecture oriented, at first, to driving a car along a track. In addition, a module to avoid collisions and overtake possible opponents has been implemented. One of the main advantages is that each module is in charge of managing one basic aspect of the driving.

Basic modules manage the car's actuators (accelerator, brake, gear and steering) while two more complex modules carry out additional functionality. A simple TSK fuzzy controller is in charge to determine the target speed to drive along a certain track segment. Modifications over the actuators are applied with the aim to make the controller able to compete with other ones, doing it able to overtake and brake in case of an imminent collision.

The results obtained in both tests performed greatly exceed our expectations, moreover if we consider that many of the features / parameters are hand-tuned.

Several lines of research are now opened.

The modularity of the architecture is suitable to make incremental steps towards the improvement of the controller. For example, we detected that the car skids and lose the control when it go out of the track axis, where usually the ground is sand. It could be solved by adding a filter over the throttle signal to avoid this situation. Also, many parameters (as increments applied in presence of other cars) are constants, but it would not be hard to make them change as a function of the current speed. This may lead to the construction of "driving maps" oriented to, for example, reduce consumption, being prudent due to high damage, etc.

The application of computational intelligence techniques for learning/adapting the parameters is a clear venue to explore. At first, we consider that the TSK controller is a suitable module to be improved by evolutionary techniques as it is done in the area of genetic fuzzy systems [10], [11].

ACKNOWLEDGMENTS

This work was supported by the Spanish Ministry of Fomento project GUIADE (P9/08), the CENIT project MARTA 2007/06, and the TRANSITO project TRA2008-06602-C03-01, and the Spanish Ministry of Science and Innovation through project TIN2008/01948, and Junta de Andalucia, through project P07-TIC02970.

REFERENCES

[1] J. Togelius, S. Lucas, H. Duc Thang, J. M. Garibaldi, T. Nakashima, C. Hiong Tan, I. lhanany, S. Berant, P. Hingston, R. M. MacCallum, T. Haferlach, A. Gowrisankar, and P. Burrow, "The 2007 ieee cec simulated car racing competition," *Genetic Programming and Evolvable Machines*, vol. 9, pp. 295–329, 2008.

[2] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. Saez, "The wcci 2008 simulated car racing competition," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

[3] W. of Politecnico di Milano, "Car racing @ wcci 2008. http://cig.dei.polimi.it/."

[4] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. Syst., Man, Cybern*, vol. 15, pp. 116–132, 1985.

[5] "Software manual of the car racing competition." [Online]. Available: http://cig.dei.polimi.it/wpcontent/ uploads/2008/04/manual v03.pdf.

[6] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[7] M. Sugeno and G. T. Kang, "Structure identification of fuzzy model," *Fuzzy Sets and Systems*, vol. 28, pp. 15–33, 1988.

[8] R. García and T. de Pedro, "Modeling a fuzzy coprocessor and its programming language," *Mathware and Soft Computing*, vol. 5(2-3), pp. 167–174, 1998.

[9] M. Sugeno, "On stability of fuzzy systems expressed by fuzzy rules," *IEEE Trans. Fuzzy Syst.*, vol. 7(2), pp. 201–224, 1999.

[10] O. Cordón, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena, "Ten years of genetic fuzzy systems: Current framework and new trends," *Fuzzy Sets and Systems*, vol. 141, no. 1, pp. 5–31, 2004.

[11] O. Cordón, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena, "Genetic fuzzy systems: Taxonomy, current research trends and prospects," *Evolutionary Intelligence*, vol. 1, pp. 27–46, 2008.