An Evolutionary Tuned Driving System for Virtual Car Racing Games: The AUTOPIA Driver

E. Onieva,[1,*] D. A. Pelta,[2,‡] J. Godoy,[1,§] V. Milanés,[1,¶] J. Pérez[1,†]
[1]*AUTOPIA program, Centro de Automática y Robótica (UPM-CSIC), La Poveda-Arganda del Rey, 28500 Madrid, Spain*
[2]*Department of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain*

This work presents a driving system designed for virtual racing situations. It is based on a complete modular architecture capable of automatically driving a car along a track with or without opponents. The architecture is composed of intuitive modules, with each one being responsible for a basic aspect of car driving. Moreover, this modularity of the architecture will allow us to replace or add modules in the future as a way to enhance particular features of particular situations. In the present work, some of the modules are implemented by means of hand-designed driving heuristics, whereas modules responsible for adapting the speed and direction of the vehicle to the track's shape, both critical aspects of driving a vehicle, are optimized by means of a genetic algorithm that evaluates the performance of the controller in four different tracks to obtain the best controller in a large number of situations; the algorithm also penalizes controllers that go out of the track, lose control, or get damaged. The evaluation of the performance is done in two ways. First, in runs with and without adversaries over several tracks. And second, the architecture was submitted as a participant to the *2010 Simulated Car Racing Competition*, which in end won laurels.

1. INTRODUCTION

Games have long been seen as an ideal test-bed for the study of computational intelligence. Until recently, most academic work in the area had focused on traditional board and card games, the challenge being to beat expert human players.[1]

*Author to whom all correspondence should be addressed: e-mail: enrique.onieva@car.upm-csic.es.
†e-mail: oshue.perez@car.upm-csic.es.
‡e-mail: dpelta@decsai.ugr.es.
§e-mail: jorge.godoy@car.upm-csic.es.
¶e-mail: vicente.milanes@car.upm-csic.es.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 27, 217–241 (2012)

View this article online at wileyonlinelibrary.com. • DOI 10.1002/int.21512

Traditional games are constrained by discrete interactions on the game objects (the pieces or the cards). The rules of this kind of game specify the interaction with the objects, and fundamental combinatorics quickly produce enormous game trees. The result is that complex decision-making processes are required to play such games at a high level.[2]

The past few decades have seen a phenomenal increase in the quality, diversity, and pervasiveness of computer games. Worldwide, the computer games market is estimated to be worth around US$ 21 billion annually and is predicted to continue to grow rapidly.[3]

Car racing is a popular genre in computer games. The great realism implemented in recent car simulators has also led to its becoming an interesting test domain for methods of artificial and computational intelligence and an excellent test-bed for the application of autonomous driving techniques.

This realism suggests that computational intelligence can be used in different but complementary ways in racing games, and that there is unrealized potential for cross-fertilization between research in robotics and artificial intelligence for games.[4]

In racing games, the goal is to guide some sort of vehicle toward a goal efficiently. In its simplest form, the challenge for the player comes from controlling the dynamics of the vehicle. Additional challenges might be avoiding collisions, shifting gears, following traffic rules, or competing with other drivers.

In the past couple of years, *The Open Racing Car Simulator* (TORCS) has been widely used as test-bed for computational intelligence algorithms, as has been reflected in the literature. For instance, In Ref. 5 it is proposed a robust approach to learning the track models from partial sensory data, and a classifier is responsible for providing six types of track shapes easily recognizable by humans. The work presented in Ref. 6 is focused on blocking behaviors and how to deal with them during a race by means of the use of fuzzy logic. In Ref. 7 a player's enjoyment is modeled from physiological signals. In Ref. 8, reinforcement learning techniques are used to achieve two complex racing behaviors: overtaking a fast opponent on a straight and overtaking on a tight bend.

But the most obvious application of computational intelligence methods to racing games has been the generation and control of automatic characters (non-player characters, NPC). This gives researchers the opportunity of applying and testing their techniques in a continuous environment in which performance is usually evaluated in terms of accuracy in driving along a track. In recent years, there has been a notable emergence of competitions whose immediate objective is to control a car in simulated racing environments, but whose ultimate objective is to encourage researchers to apply their knowledge and experience to this topic.

One of the most popular of these competitions is the *Simulated Car Racing Competition*. During the past 3 years, this has been held in a broad series of international conferences. The first used a computationally simple car racing model in which the objective was to plan and visit a set of predefined target points before your opponent.[9] The last two competitions were based on TORCS, which allowed experimentation in more realistic racing environments.[10,11]

The entries submitted to these competitions represent a broad set of computational intelligence techniques and approaches—evolving driving rules, both

fuzzy[12] and crisp,[13] covariance matrix adaptation,[14] evolutionary strategies,[5] neural networks,[15] and genetic programming,[16] among others.

Genetic algorithms (GA) are search algorithms based on the mechanics of natural selection and genetics. They combine survival of the fittest among structures representing artificial individuals with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of individuals is created using pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, genetic algorithms are not simple random walks. They efficiently exploit historical information to speculate on new search points that have expected improved performance.[17]

The prime objective of the present work is twofold: First, to design, implement, and test a complete architecture enabling automatic driving in racing situations; and second, to gain insight into how to construct efficient and simple to understand controllers for car bots. In this sense, it is a continuation of the authors' earlier work[18] in which a driving architecture with a fuzzy-based module responsible for calculating the allowed speed achieved excellent results in the *2009 Simulated Car Racing Competition*.

The architecture is based on the conjunction of simple functional modules each responsible for some basic driving task. Some of them are heuristically designed, whereas two of them (those responsible for inferring the allowed speed and for moving the steering wheel) are adjusted by a GA. The GA evaluates the performance of the controller in four different tracks to obtain the best controller in a large number of situations; the algorithm also penalizes controllers that go out of the track, lose control, or get damaged, obtaining so, a good balance in the task of *driving as fast as possible with safe*.

The paper is structured as follows. Section 2 presents the racing environment used in this work (TORCS), together with the sensor information and available actuation variables. Section 3 describes in detail the driving architecture implemented, highlighting those modules that will be optimized. Section 4 presents the genetic optimization process implemented to obtain the *fastest controller*. Section 5 presents the experiments conducted to test and validate the resulting driver, both in laboratory conditions and participating in the *2010 Simulated Car Racing Championship*. Finally, Section 6 presents some concluding remarks and future work.

2. TORCS RACING ENVIRONMENT

TORCS[19] (the open racing car simulator) is one of the most popular car-racing simulators. It is written in C++ and is available under the GPL license from its Web page. For academic purposes, TORCS presents various advantages over other simulators, such as

- It lies between an advanced simulator, such as those implemented in recent commercial car-racing games, and a fully customizable environment, such as those typically used by computational intelligence researchers for benchmark purposes.

- It features a sophisticated physics engine (aerodynamics, fuel consumption, traction, and so on), as well as three-dimensional graphics for the visualization of the races.
- It was not conceived as an alternative to commercial racing games, but specifically devised to make it as easy as possible to develop one's own controller.

Each vehicle is controlled by an automatic *driver* or *bot*. At each control step, a bot can access the current game state (which includes information about the car, the track, and the opponents) and can then, after a decision process, manage the vehicle's actuators (pedals, gears, and steering). The distribution of the game includes several preprogrammed bots, which can be customized or extended to build new ones.

TORCS has been used in the past few years for several computational intelligence competitions; see WCCI-2008,[a] CIG-2008,[b] CEC-2009,[c] GECCO-2009,[d] and CIG-2009.[e]

For the competitions, the organizers provided competitors with a specific software interface developed on a client/server basis where the designed controllers run as external programs and communicate with a customized version of TORCS through User Datagram Protocol (UDP) connections. A detailed description of the information interchanged can be found in Ref. 20. The following subsections describe the information processing stage, which obtains input values to present to the controller and the available actuators.

2.1. Sensor Information

Sensor information is used to create the automatic driver that manages the vehicle. Some of this information is taken directly from the server, and some is inferred from other sensor information. For example, there is no *lapcounter*, but there is a *laptime* value. Therefore, in the step $laptime = 0$, we increase our own *lapcounter* value.

The main guiding information comes from a set of proximity sensors. These are of two classes: (i) measuring distance to the track borders and (ii) measuring the distance to an opponent. Both classes have a maximum measurement of 200 m. They differ in the number and angular distribution of the sensors.

There are 36 opponent sensors uniformly distributed every 10 deg in $[-180, 170]$. Henceforth, O_d will indicate the measurement of the proximity opponent sensor oriented at d deg. Figure 1 shows the distribution and measurements of this set of sensors.

There are 19 track sensors that measure distances to the track borders. They are distributed at orientations $\{90, 75, 60, 45, 30, 20, 15, 10, 5, 0\}$ deg (and the corresponding negative values). Henceforth, T_d will indicate the measurement of the proximity track sensor oriented at d deg. Figure 2 shows the track sensors and their orientations.

[a] http://www2.mae.cuhk.edu.hk/ wcci2008/.
[b] http://www.csse.uwa.edu.au/cig08/.
[c] http://www.cec-2009.org/.
[d] http://www.sigevo.org/gecco-2009/.
[e] http://www.ieee-cig.org/2009/.

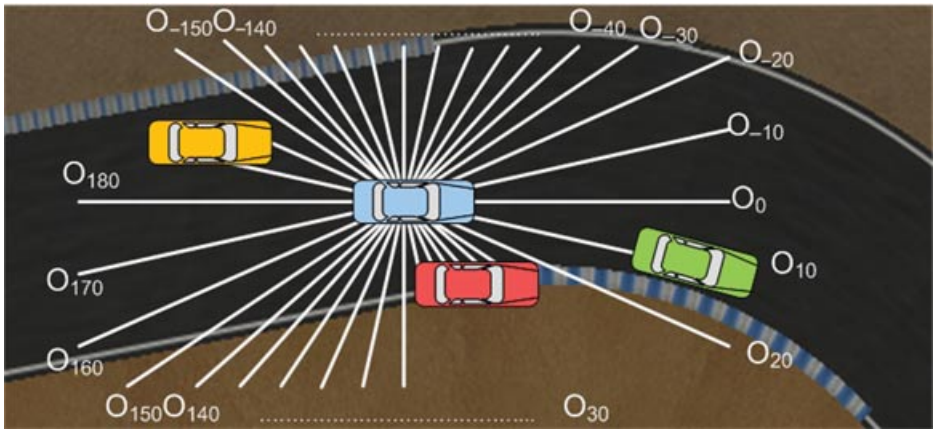


Figure 1. Distribution and measurements of the opponent sensors. When no opponent is detected, the maximum measurement (200 m) is returned.

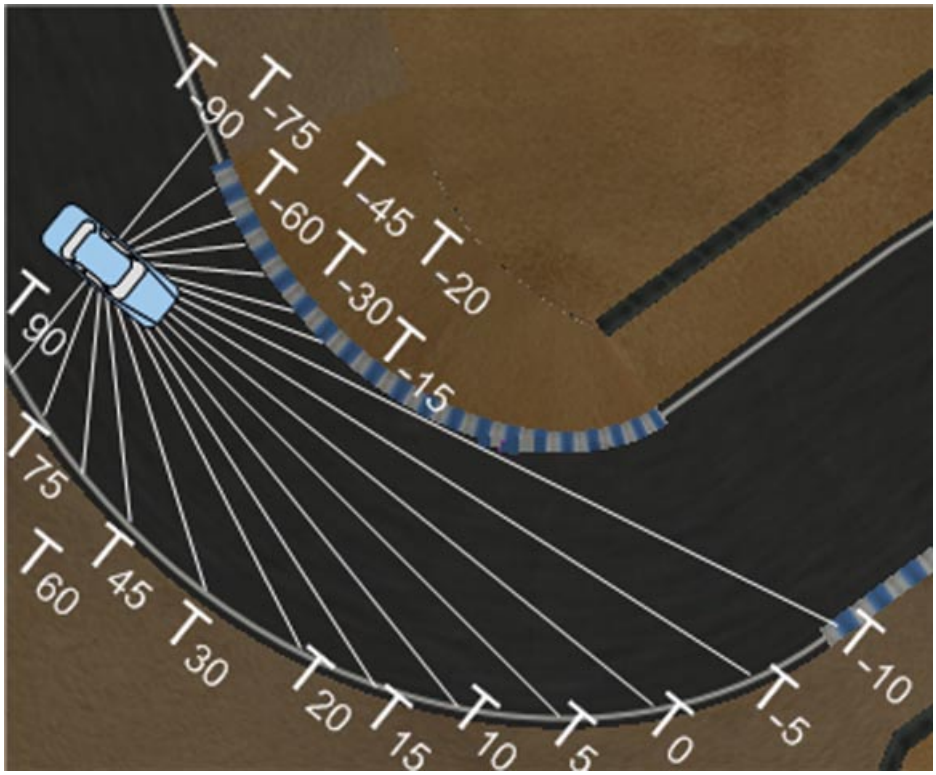


Figure 2. Distribution and measurements of the track sensors.

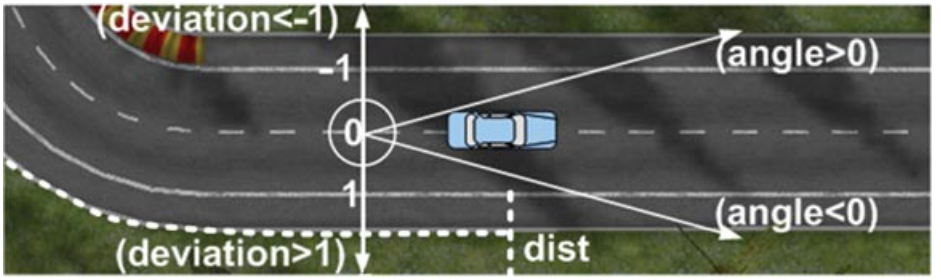


Figure 3. Representation of *deviation*, *angle*, and *dist*, and their signs.

As noted above, these proximity sensors are the main source of information used to drive the vehicle. Also, information referring to the car's position is provided:

- *Deviation* denotes the normalized lateral deviation of the vehicle from the center of the track; $deviation = 0$, when the vehicle is in the center of the track, $deviation < 0$ when the vehicle has deviated to the left, and vice versa, $deviation < -1$ or $deviation > 1$ denote values off the track.
- *Angle* measures the angle (in radians) of the vehicle with the track axis; $angle < 0$ means that the vehicle is oriented to the right and vice versa.
- In addition, a sensor called *dist* gives the distance from the starting line to the current vehicle position along the center line of the track.

The variables *deviation*, *angle*, and *dist* are shown graphically in Figure 3.

There is additional information about the current *gear*, *speed* (in km/h), and engine *rpm* of the vehicle, and the *trackwidth* and *tracklength* (in meters) of the current track. The *TotalTime* and *LastLapTime* in seconds, and the *lapcounter* or *position* provide information about the development of the race.

The vehicle's *damage* measured as a percentage is also provided: (i) The vehicle is slower and reacts more poorly with greater damage and (ii) If *damage* reaches 100%, the vehicle is retired from the race, finishing in last position. Because of this, it is important to achieve a good trade-off between speed and caution to avoid crashes. Some features of the architecture presented are oriented to slowing the vehicle down or make it more cautious when the damage is high.

2.2. Control Actions

Once the decision layer has processed the information, four outputs must be generated:

- *Gear* that can take a value in $\{-1,0,1,2,\ldots,6\}$, where -1 means reverse gear, 0, neutral, and 1–6 are respective forward gears.
- *Steer* defines the steering value in $[-1, 1]$, in which the extremes mean, respectively, full right and full left.
- *Throttle* and *Brake* are defined in [0, 1] and denote the pressure to apply on the corresponding pedal. In this work, we use a variable *Pedal*, which codifies both pedals in one value in $[-1, 1]$.

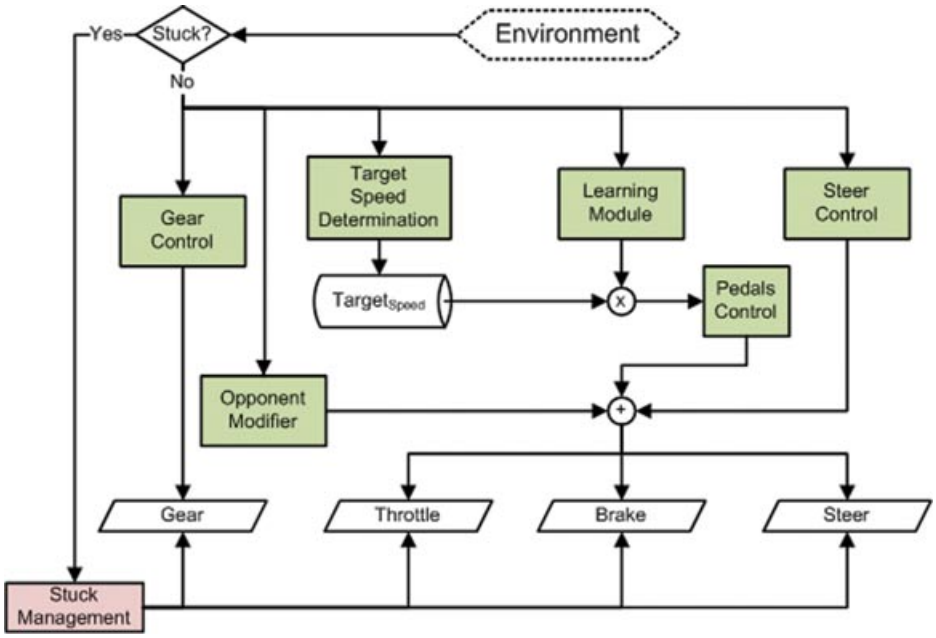


Figure 4. Overview of the implemented architecture.

3. FULL DRIVING ARCHITECTURE

The aim of the architecture is to separate out each of the basic driving tasks involved in a racing situation. Six such tasks are considered: (1) gear control, (2) pedal control, (3) steering control, (4) blockage ("stuck") situation manager, (5) target speed determination, (6) opponent modifier, and (7) learning module.

Modules 1–4 are assumed to be part of a low-level control layer since they will be ultimately responsible for acting on the vehicle's controls. Modules 5–7 represent a higher layer since they provide objectives for the previous low-level modules.

An overview of the modules conforming the architecture is shown in Figure 4. The architecture works as follows: A check is made for the existence of a stuck situation. If there is one then the stuck situation manager is given the responsibility for controlling the vehicle (by using reverse gear and special pedal/steering wheel behaviors). Otherwise, the steering and gear modules control their corresponding actuators, the target speed module determines the allowed speed and sends it to the pedal control to infer the throttle and brake outputs. In the case that the learning module *remembers* a special situation in which the speed must be reduced (e.g., *the vehicle crashed near the current position on a previous lap*), the allowed speed is multiplied by some factor. At the end, the opponent modifier checks for the presence of opponents near the vehicle and modifies (if necessary) one or more of the values assigned to the steering, throttle, or brake (e.g., *if there is an opponent near then reduce the throttle action or increase the brake action*).

The following subsections explain in detail the functionality implemented by each of the modules.

3.1. Gear Control

This module is responsible for shifting up or down among the forward gears (from first to sixth) as a function of the current *rpm* value.

This control is carried out by means of the 10 rules summarized in (1). They are evaluated each second (50 time steps) with the aim of avoiding too rapid shifting.

$$Gear = \begin{cases} gear + 1, & \text{if} \quad (rpm > GI_{gear}) \\ gear - 1, & \text{if} \quad (rpm < GD_{gear}) \end{cases} \tag{1}$$

where, for each $i = 1 \ldots 6$, GI_i and GD_i represent, respectively, the minimum or maximum *rpm* value required to increase or decrease the ith gear. $GI_6 = \infty$ and $GD_1 = 0$ since there is no seventh gear and first gear cannot (or should not) be decreased.

The values used are $GI_{i=1\cdots5} = \{9500\ 9500\ 9500\ 9500\ 9000\}$ and $GD_{i=2\cdots6} = \{4000\ 6300\ 7000\ 7300\ 7300\}$ since values quite similar to these were found to give good results, and there was no clear improvement observed when the values were changed slightly.

3.2. Pedal Control

The pedal (*throttle* and *brake*) signals are codified as a single *pedal* signal to avoid nonsense values. Positive values represent actions on the throttle, maintaining the brake at 0. Negative values represent actions on the brake, maintaining the throttle at 0.

If the vehicle is off the track then $pedal = 0.3$. Otherwise pedal actions will be oriented to following the $Target_{speed}$ imposed by the corresponding module. To this end, the pedal value is calculated as

$$pedal = \frac{2}{1 + e^{speed\text{-}Target_{speed}}} - 1 \tag{2}$$

Figure 5 shows the pedal action in function of the difference between the current speed of the vehicle and the desired one ($Target_{speed}$), coming from the module dedicated to calculate this value.

In addition, a simple Antilock Brake System (ABS) filter has been implemented to avoid the vehicle skidding and losing control when braking. Two constraints are applied to the brake signal: (i) in intensity (no greater than 75%) and (ii) in time (to brake for 5 time steps and not to brake for five time steps).

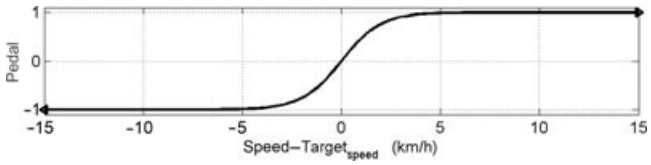


Figure 5. Plot of pedal actions.

3.3. Steering Control

The steering control manages the car's steering wheel with the aim of following the track without leaving it. There are two cases that have to be considered: (1) when the vehicle is driving within the track and (2) when it is driving in the zone outside the track. This second case has to be considered separately since the track sensors (T_d in Section 2.1) do not return reliable values when the vehicle is off the track.

During normal driving, i.e., within the track, Equation (3) is used to guide the vehicle. The objective is to orient the vehicle through the track proximity sensors that have the greatest measurement values.

$$\begin{aligned} steering = ST_1 \cdot \alpha + ST_2 \frac{(\alpha_{-1} \cdot t_{-1}) + (\alpha_{+1} \cdot t_{+1})}{t_{-1} + t_{+1}} \\ + ST_3 \frac{(\alpha_{-2} \cdot t_{-2}) + (\alpha_{+2} \cdot t_{+2})}{t_{-2} + t_{+2}} \\ + ST_4 \frac{(\alpha_{-3} \cdot t_{-3}) + (\alpha_{+3} \cdot t_{+3})}{t_{-3} + t_{+3}} \end{aligned} \tag{3}$$

where α represents the angle (in radians) of the track sensor with the greatest measurement; $\alpha_{-1,-2,-3}$ represent the angle in radians of the first, second, and third neighboring track sensors to the left of the one with the greatest measurement, and mirroring to the right for $\alpha_{1,2,3}$. t_i represents the measurement (in meters) of the track sensor oriented at α_i. Figure 6 is a graphical representation of the control variables used to manage the steering.

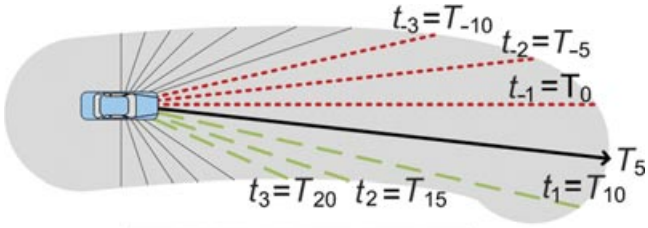
The objective of Equation 3 is to move the steering proportionally in the direction of the largest sensor measurement and, to smooth out the actions, with the addition of terms proportional to the values reported for neighboring sensors.

A fifth parameter (ST_5) is added for driving along straight segments. A segment is considered straight if the following condition is satisfied:

$$(|deviation| < 0.75)\ AND\ ((\alpha = 0)\ OR\ (T_0 > 190)\ OR\ (T_0 > ST_5 \cdot speed)),$$

with the speed in meters/second. In that case, the steering will be assigned the value $0.5 \cdot angle$, to maintain the vehicle oriented along the straight segment.

The five parameters that will define the driver's steering behavior are ST_i. Their optimization will be presented in Section 4.



α = 0.0873 (5° in radians)	
$\alpha_{-1} = 0$	$\alpha_1 = 0.1745$
$\alpha_{-2} = -0.0873$	$\alpha_2 = 0.2618$
$\alpha_{-3} = -0.1745$	$\alpha_3 = 0.3491$

Figure 6. Graphical representation of α and t values used in the steering control.

When the vehicle is off the track, Equation 4 is used to calculate the steering action.

$$steering = \frac{angle - deviation \times 0.5}{steer_{lock}} \tag{4}$$

where $steer_{lock} = 0.7853$ represents the angle in radians of the car when a full steering lock is applied according to the car's description. Figure 7 shows the control

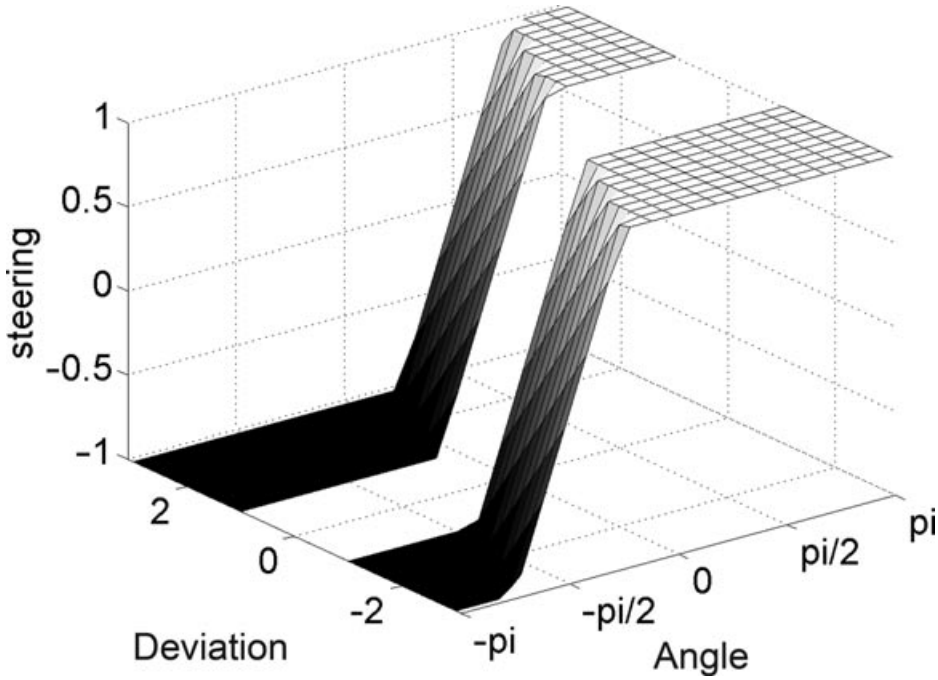


Figure 7. Graphical representation of steering actions when the vehicle is driving off the track.

surface corresponding to this equation. The central zone (*deviation* $\in [-1, 1]$) is not considered because in that case the vehicle is within the track, so that the previous method is applied.

3.4. Target Speed

This module provides the pedal control with the desired speed to follow along each track segment. This module is used only when the car is in a normal driving situation (not off the track or in a blockage situation). The desired speed at a given instant is calculated from the expression of Equation 5 using the track sensors oriented at $\{0, \pm 10, \pm 20\}$ deg.

$$\begin{aligned} Target_{speed} = {} & TS_1 \cdot t_0 + TS_2 \cdot \max(T_{10}, T_{-10}) \\ & + TS_3 \cdot \min(T_{10}, T_{5-10}) + TS_4 \cdot \max(T_{20}, T_{-20}) \\ & + TS_5 \cdot \min(T_{20}, T_{-20}) \end{aligned} \tag{5}$$

where TS_i are parameters to optimize. The equation infers a target speed value proportional to the measurements reported by the track sensors covering from -20 to $+20$ degrees in front of the vehicle.

In addition, to allow the vehicle to go more slowly when it has more damage, a simple *caution when damaged* strategy is implemented as shown in Equation 6 to reduce the speed by up to 20% (when $damage \approx 100\%$).

$$Target'_{speed} = Target_{speed} \cdot (1 - 0.002 \cdot damage) \tag{6}$$

3.5. Opponent Modifier

The objective of this module is to slightly modify the *driving in normal situation* values of the steering and pedals to overtake and avoid collisions with other opponents.

Opponents can only be detected by the proximity sensors that surround the vehicle at each 10 deg (O_{deg}). The parameters ΔO_{deg} represent the variations of these values in time.

Steering modifications are oriented at overtaking the opponents and are implemented by the heuristic rule set presented below. The aim of these rules is to modify the output of *Steer Control* in the case that an opponent is detected by the proximity sensors. Figure 8 is an illustrative case.

1. If any $O_{\pm 20, \pm 10}$ detects an opponent with a lateral displacement of less than 10 m, or $O_0 < 50$, then move the steering by $trackwidth/100$ in the direction with more free distance.
2. If any $O_{-10, -20, -30, -40}$ detects an opponent with a lateral displacement of less than 10 m, then move the steering by $trackwidth/50$ to the right.

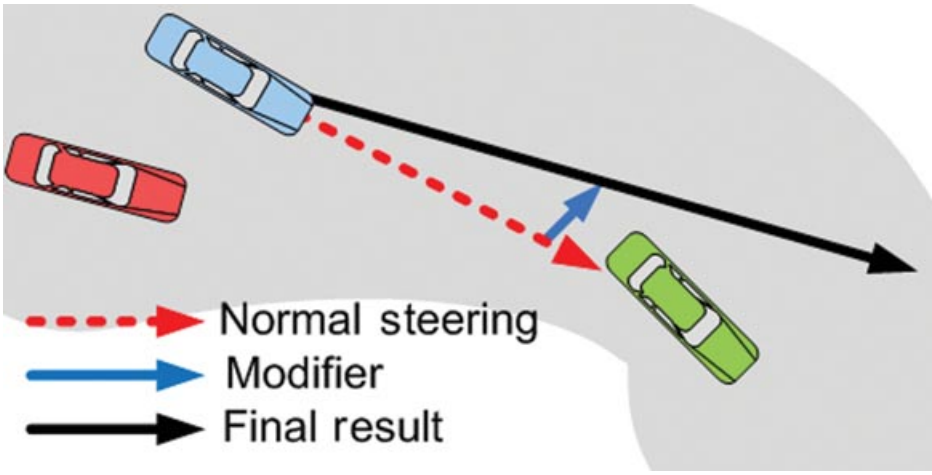


Figure 8. Effect of the opponent modifier on the normal steering output.

3. If any $O_{-50,-60,-70} < 15$, then move the steering by $trackwidth/50$ to the right.
4. If any $O_{-80,-90,\ldots,-140} < 15$, then move the steering by $trackwidth/50$ to the right.

Rules 2, 3, and 4 have their corresponding mirror rules, with positive sensor angles and steering movements to the left. Steering modifications are limited to 0.35 so as not to be so strong as to drive the vehicle off the track.

The pedal modifications have the purpose of slowing the vehicle down when there is an opponent in front of it with the risk of collision. Another heuristic rule set is defined:

1. If $speed > 100$ and $\Delta O_0 < 0$ and $O_0 < 20$, then pedal $= -0.1$ (brake 10%).
2. If $speed > 100$ and any $\Delta O_{\pm 20,\pm 10} < 0$ and any $O_{\pm 20,\pm 10}$ detects an opponent with lateral displacement less than 4 m, then pedal = -0.1.
3. If $speed > 50$ and $O_0 < 15$, then pedal $= -0.1$.
4. If $speed > 50$ and any $O_{\pm 10}$ detects an opponent with lateral displacement less than 2 m, then pedal $= -0.1$.

In addition, with the objective of driving more cautiously when the vehicle suffers damage, the emergency braking action will be maintained for $(2 + 0.3 \cdot damage)$ steps.

3.6. "Stuck" Management

The stuck manager is responsible for detecting when reverse gear must be applied and to determine when the stuck situation has ended so as to shift to first gear again.

A stuck situation is detected by means of a variable that measures the time during which the *angle* of the vehicle is greater than $\pi/6$ radians and the *deviation*

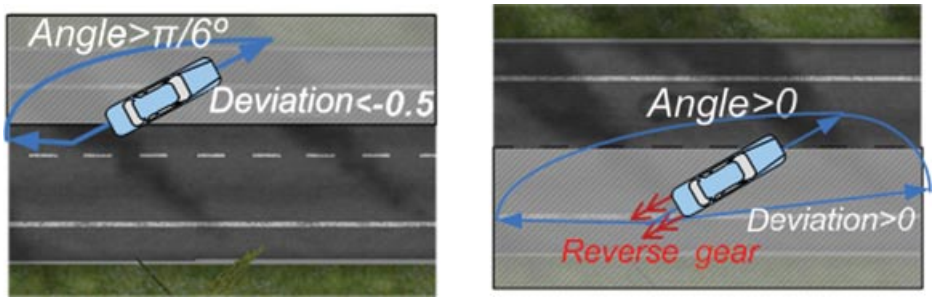


Figure 9. Representation of situations in which the vehicle needs to apply reverse gear (left) and then first gear must be applied again (right).

is greater than 0.5 (represented in Figure 9 left) or the speed is less than 10 km/h. If this time is greater than 1 s then reverse gear is applied to guide the car back to the track.

The deblocking maneuvre is done in three steps. (i) Once in reverse gear, the vehicle has to stop if it is still moving forward, with the brake at 50%. (ii) Then, $steering = -\text{angle}/\text{Steer}_{lock}$ while maximum throttle is applied, to get the vehicle oriented. Finally, (iii) the stuck situation is considered terminated when the frontal free distance is greater than 15 m or the vehicle is displacing itself from the center of the track ($angle \cdot deviation > 0$), represented in Figure 9 right. Then first gear is applied, and the vehicle continues racing normally.

3.7. Learning Module

This module is responsible for acquiring information about the track's features during a first phase of racing alone during which the vehicle must *learn* the track. This stage has a certain known duration ($Time_{train}$).

A quite simple learning module is implemented. The vehicle stores a vector with elements map_i, and with length equal to *tracklength* in meters; the vector's elements are initiated to 1 and represent a multiplying factor for the $Target_{speed}$ value during the corresponding segment of the track:

$$Target'_{speed} = Target_{speed} \cdot map_{dist} \tag{7}$$

where *dist* represents the distance to the start point along the track (Section 2.1).

The vector is updated when at a certain point (termed *accident*) the vehicle gets stuck, suffers damage, or goes off the track. The speed must then be reduced before reaching *accident*. This reduction is applied to the 250 m before reaching the *accident* point. The reduction at each corresponding map_i is governed by three rules:

1. If $map_i > 1$ then $map_i = 1$.
2. Else, if $Time_{train} - TotalTime > 5 \cdot LastLapTime$, then map_i is multiplied by 0.95.
3. Else map_i is multiplied by 0.9.

Rules 2 and 3 are aimed at reducing the speed less harshly if the vehicle calculates that it can still run for five more laps without using up the amount of time of the learning stage.

In addition, map_i is multiplied by 0.75 for the 15 m before a *jump* is detected. This is not a usual situation, but needs to be considered.

Once the second lap has been reached, and if the vehicle can run for five more laps, all the map_i that have as yet not been reduced are multiplied by 1.25.

4. GENETIC OPTIMIZATION PROGRESS

The preceding section presented the model of the architecture that defines the driver. Most of the modules involved, such as *stuck management* or *learning module*, were hand coded since they are used in only a few cases. Others, such as *opponent modifier*, *gear control*, or *pedal control*, did not give rise to major performance differences when they were slightly changed.

We shall therefore consider the optimization of the *steering control* and the *target speed* modules since they are used most of the time, and their good functioning forms the basis for obtaining an accurate driver. The two modules were explained in detail in Sections 3.3 and 3.4, and each is defined by five parameters:

- $ST_{i=1\ldots5}$ define the steering control (Equation 3).
- $TS_{i=1\ldots5}$ define the equation used to calculate the target speed (Equation 5).

To optimize these values, a GA is defined,[21] following the generational GA schema. This creates new offspring from the members of an old population using the genetic operators and places these individuals in a new population, which becomes the old population once the whole new population has been created.[17]

Each potential solution or individual is represented in the form of a vector $([ST_1, \ldots, ST_5, TS_1, \ldots, TS_5])$. A set of $N = 20$ individuals forms a population. The population is initialized randomly in the interval $[-5, 5]$. The population changes or *evolves* under the operation of the genetic operators (*selection*, *crossover*, *mutation*, and *replacement*),[22] which will be explained below.

The process starts with a population of $N = 20$ individuals and lasts for $G = 100$ generations. The application of the operators in each generation is represented graphically in Figure 10. The steps followed are select a set of nine pairs of individuals (P_i); apply the crossover operator to these pairs of individuals to generate 18 offspring (O_i); then apply the mutation operator to these offspring (M_i). The new population (N_i) is formed by the 18 individuals generated in addition to the two best of the previous population and will replace the old one.

The *selection* process is based on the principle of *survival of the fittest*. Some individuals are chosen from the population based on their fitness value. The selection is done by binary tournament, i.e., randomly choose two individuals of the population, compare their fitness, and select as parent the one with the greater value.

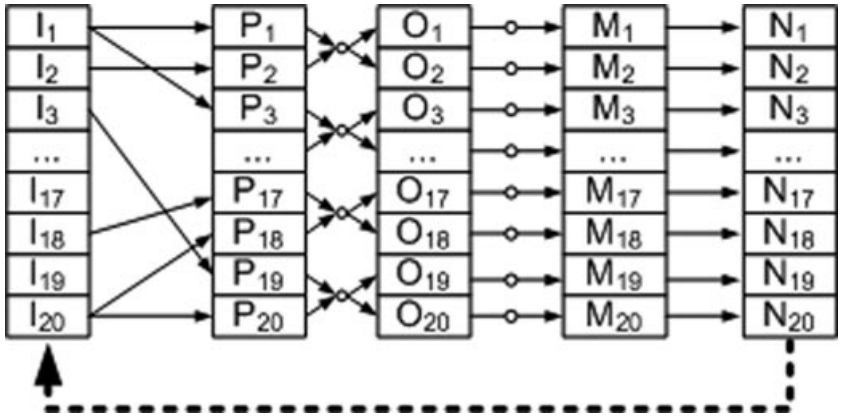


Figure 10. Schema of the generational genetic algorithm.

The *crossover* between two individuals is implemented as a $BLX-0.5$ crossover.[23] Assume that $C_1 = (c_1^1 \ldots c_{10}^1)$ and $C_2 = (c_1^2 \ldots c_{10}^2)$ are two chromosomes that have been selected on which to apply the crossover operator. Each position in the generated offspring $h_{i=\{1,\ldots 10\}}$ is a randomly (uniformly) chosen number from the interval $[C_{\min} - I \cdot 0.5, C_{\max} + I \cdot 0.5]$, where $C_{\min} = \min(c_i^1, c_i^2)$, $C_{\max} = \max(c_i^1, c_i^2)$ and $I = (C_{\max} - C_{\min})$.

Each position $h_{i=\{1,\ldots 10\}}$ of each chromosome undergoes a random *mutation* in $[h_i \pm 2]$ according to a probability defined by a mutation rate—the mutation probability, $p_m = 0.1$. Values are not truncated in case they fall out the initial interval (± 5).

To evaluate each individual, four different *oval tracks* taken from the TORCS distribution are used. They will henceforth be denoted $Oval_{\{1,2,3,4\}}$. Their shapes and basic characteristics are shown in Figure 11. Each individual drives for 80 s on each of the tracks, and the fitness is calculated as the sum of the distances traveled on the four tracks in meters. In addition, the controllers are penalized with -2000 m if the vehicle gets stuck, or gets more than 1% damage. The genetic optimization process was executed once due to the computational cost of the evaluation process.

The plot shown in Figure 12 represents the evolution of the fitness of the controller on the training circuits, compared with a *base* hand-chosen configuration [0.75, 0.75, 0, 0, 1.5, 2, 2, 0, 0, 0], which has been obtained by trial and error until find an acceptable behavior.

Figure 13 shows a box-and-whisker plot of the individuals in the last generation of the process together with the values for the base individual and the best one found. One observes that this last generation maintains a reasonable diversity of the population, with the exception of variable TS_4. In general, and particularly so for the TS_i values, the values of the best and the base individuals are fairly close.

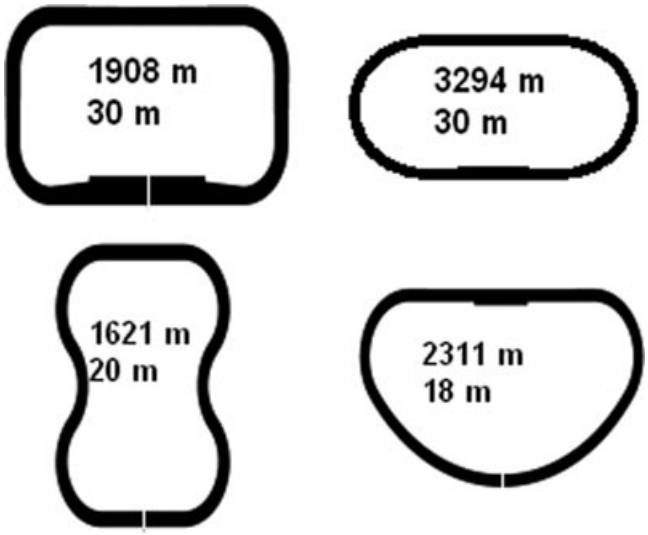


Figure 11. Tracks used in the optimization of the steering and target speed modules (track set *Oval*). The values are, respectively, the track's length and width.

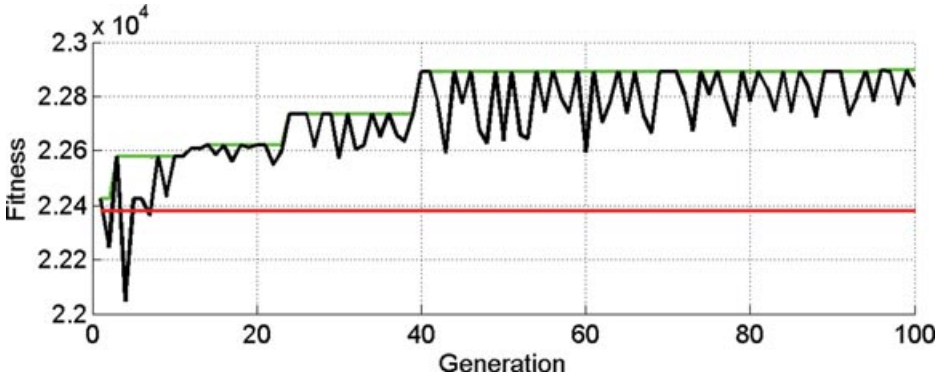


Figure 12. Genetic algorithm: evolution of the fitness function during the process. Red line: base configuration. Black line: best individual in current generation. Green line: best individual found.

Taking the best individuals, Equations 3 (steering control) and (5) (target speed) particularize to Equations 8 and 9.

$$\begin{aligned} steering = {} & 0.21 \cdot \alpha + 1.56 \frac{(\alpha_{-1} \cdot t_{-1}) + (\alpha_{+1} \cdot t_{+1})}{t_{-1} + t_{+1}} \\ & + 0.68 \frac{(\alpha_{-2} \cdot t_{-2}) + (\alpha_{+2} \cdot t_{+2})}{t_{-2} + t_{+2}} \\ & + 0.53 \frac{(\alpha_{-3} \cdot t_{-3}) + (\alpha_{+3} \cdot t_{+3})}{t_{-3} + t_{+3}} \end{aligned} \tag{8}$$

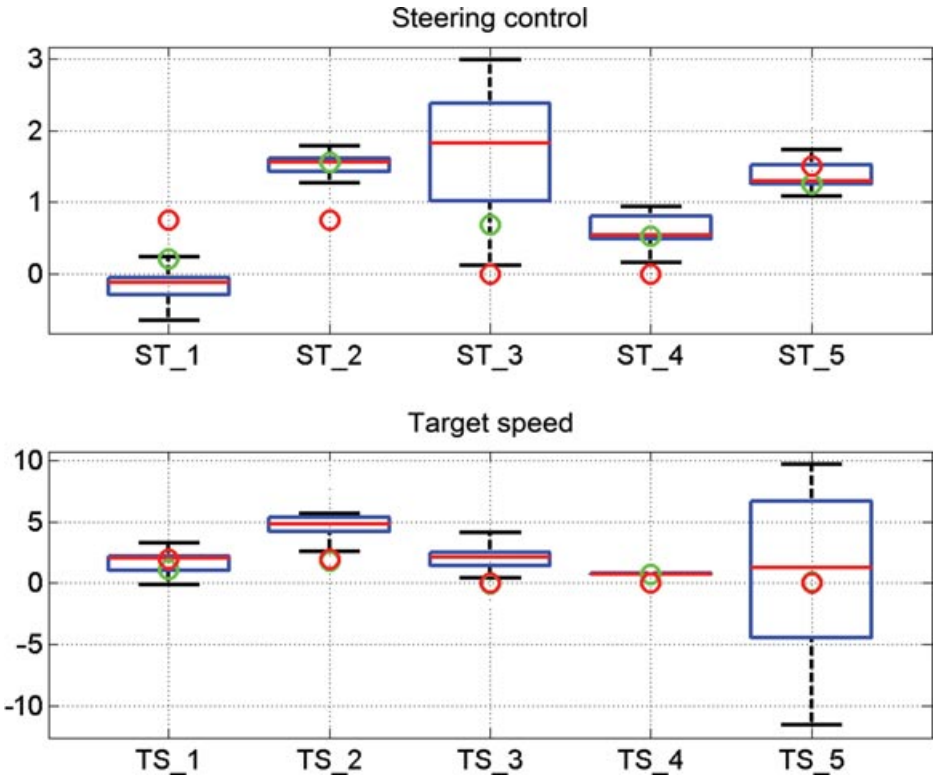


Figure 13. Box-and-whisker plot of the last generation. Steering control values (top) and target speed calculations (bottom). Green and red circles represent, respectively, the values of the best and base individuals.

with the steering being set to straight-ahead when $|deviation| < 0.75$ *AND* ($\alpha = 0$ *OR* $T_0 > 190$ *OR* $T_0 > 1.25 \cdot speed$).

$$\begin{aligned} Target_{speed} = {} & 1.08 \cdot t_0 + 1.79 \cdot \max(T_{10}, T_{-10}) \\ & - 0.02 \cdot \min(T_{10}, T_{-10}) + 0.75 \cdot \max(T_{20}, T_{-20}) \\ & + 0.13 \cdot \min(T_{20}, T_{-20}) \end{aligned} \tag{9}$$

Finally, the base and the optimized controllers are compared by driving alone for ten laps over 12 different tracks included in TORCS. These were of three types:

1. Four tracks classified as *Road Tracks*, i.e., tracks that emulate real racing circuits, with a wide variety of turns, lengths, and widths.
2. Four *Dirt Tracks*, quite similar to the Road Tracks but with a sand roadway, so that the vehicles can skid and crash easily. The *Road* and *Dirt* tracks are shown in Figure 14.
3. Finally, the four *Oval Tracks* used to train the low-level modules (Figure 11).

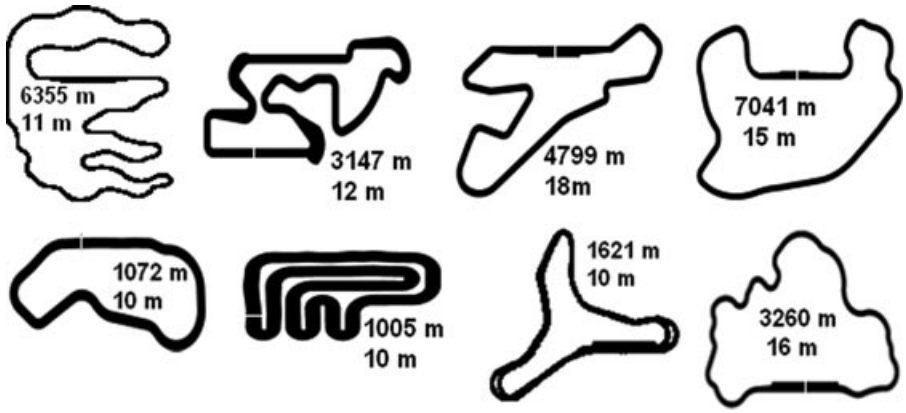


Figure 14. Additional track set used for the validation. *Road Tracks* (top) and *Dirt Tracks* (bottom).

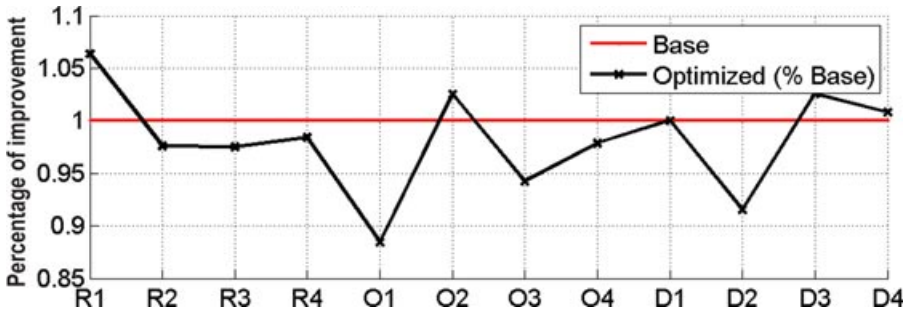


Figure 15. Comparison of results between the base configuration (red) and the optimized configuration (black). *Rx*, *Ox*, *and Dx* represent, respectively the *Road*, *Oval*, and *Dirt* track set.

Figure 15 shows the results of the comparison between the base and the optimized controllers over the 12 tracks used. The results are presented as fractions of the time obtained by the base controller on each track. It can be seen that, in general, the optimized controller improves the base controller's results, the improvement with the genetic optimization reaching 10% in mean lap time. Only in at least three cases did the optimized controller obtain poorer results. It shows the complexity of trying to solve the problem of building a driver able to *drive faster than other in every situation or track shape*.

5. EXPERIMENTATION AND RESULTS

Computational experimentation was carried out to check the controller's performance in as many situations as possible. The experiments were divided into two

phases: (1) *laboratory* tests and (2) the *2010 Simulated Car Racing Competition*. They will both be presented and analyzed in the following subsections.

5.1. Laboratory Experimentation

The objective of the laboratory test was to check the controller's performance in real racing situations, i.e. to compete against opponents on a *learned* track. First, measurements were made of tests driving alone, then races were recreated in order to evaluate the controller against opponents.

In this subsection, the races were performed on the twelve tracks described in the preceding section (Figures 11 and 14). The present driver competed with seven bots included in the TORCS version. These were 7 of the 10 controllers in the *Berniw Hist* group. They were created by Bernhard Wymann, current leader of the TORCS project, who wrote one of the first and most competitive of the bots included in the distribution. This bot has been used as their base by an extensive set of driver developers. The 10 controllers in this group have different characteristics, behaviors, and vehicles that they use. We shall henceforth refer to Bot_i, with $i = 1, 2, \ldots 7$, for the corresponding *Berniw Hist i* bot.

It is important to remark that the bots created for the TORCS distribution have a full view of the surrounding environment, i.e. they have available a full description of the track's shape, and their calculations can hence be made more accurately. The present controller was designed with a limited set of sensors that does not allow the complete view of the environment.

Table I lists the times taken to complete a 10-lap race alone over each track by each of the drivers. The three best controllers for each track are in boldface. It can be seen that the present controller clearly outperforms all the drivers in both the *Oval* and the *Road* set, even though it had not been trained in this latter set. With respect to the *Dirt* tracks, our controller was the slowest on the $Dirt_2$ track, and the third fastest on $Dirt_3$.

Table I. Time (in seconds) elapsed for 10 laps racing alone.

Track	Optimized	Bot_1	Bot_2	Bot_3	Bot_4	Bot_5	Bot_6	Bot_7
$Road_1$	**1420.2**	1687.7	**1490.5**	**1515.8**	1919.2	2222.2	1797.4	1669.1
$Road_2$	**815.1**	949.7	**832.3**	**861.9**	1060.6	1224.6	1000.5	929.9
$Road_3$	**882.7**	1121.8	**953.4**	**987.9**	1239.8	1421.0	1171.7	1076.1
$Road_4$	**1080.6**	1395.1	**1338.8**	**1249.7**	1605.7	1820.9	1512.7	1367.7
$Oval_1$	**276.3**	406.9	**351.8**	**357.6**	460.9	536.4	431.6	401.4
$Oval_2$	**423.8**	541.7	**457.0**	**449.8**	626.2	729.1	596.5	533.8
$Oval_3$	**271.6**	408.2	**368.9**	**372.1**	473.6	549.5	444.7	412.4
$Oval_4$	**352.3**	475.6	**404.6**	**417.6**	537.6	616.8	501.8	459.3
$Dirt_1$	**339.3**	395.7	**371.6**	**374.3**	454.7	516.8	419.0	397.9
$Dirt_2$	963.3	**627.1**	**622.4**	**605.4**	728.1	836.2	671.1	645.5
$Dirt_3$	**742.4**	746.9	**697.3**	**707.6**	859.1	957.8	786.4	745.8
$Dirt_4$	**796.5**	937.3	**859.5**	**865.6**	1107.1	1266.3	1006.4	937.3
Total	**8364.1**	9693.7	**8748.1**	**8765.3**	11072.6	12697.6	10339.8	9576.2

Table II. Results obtained in the full race scenarios.

	F_P	O	D/O (%)
$Road_1$	1*st*	16	0.13
$Road_2$	1*st*	16	0.57
$Road_3$	1*st*	18	1.43
$Road_4$	1*st*	18	5.97
$Oval_1$	1*st*	27	0.99
$Oval_2$	1*st*	20	1.38
$Oval_3$	1*st*	19	2.81
$Oval_4$	1*st*	20	3.79
$Dirt_1$	1*st*	13	1.36
$Dirt_2$	8*th*	0	–
$Dirt_3$	4*th*	3	6.74%
$Dirt_4$	1*st*	15	2.35%

The results in Table I thus show how good is the adjustment made to the controller because it is able to outperform the others even on tracks where the driver had not been trained. The controller obtains the best time in 10 of 12 tracks used; just in $Dirt_2$ and $Dirt_3$ it obtains the eighth (the worst) and third position. Bot_2 and Bot_3 get the runner ups positions with big distance from the rest of the bots.

Now the eight bots ($Bot_{1...7}$ and the present proposal) were run together in 10-lap races. The starting order was the inverse of the total time taken to finish all the tracks in the racing alone mode for $Bot_{1...7}$, and the present proposed bot started in last position. Thus the starting grid was Bot_5, Bot_4, Bot_6, Bot_1, Bot_7, Bot_3, Bot_2, and the present proposal in the last position, for all the tracks tested. This was done with the aim of providing a scenario with more chances for several overtaking maneuvres during the race.

Table II gives the final results of these full races, showing, by columns, the track, the final position of the presented controller (F_P), the number of successful overtakes (O), and the average damage sustained per overtake, i.e. the total amount of damage accrued divided by the number of overtaking maneuvres carried out.

Overtaking maneuvres are counted in absolute terms, that is, do not measuring situations where the controller overtakes an opponent and then the opponent overtakes the controller and so on. To measure the number of overtakes, it is counted the number of complete laps ran by opponents at the moment in what the controller finishes the race. An opponent with 10 or more laps has not been overtaken (it has finished before the controller), an opponent with nine laps have been overtaken once, and opponent with eight laps have been overtaken twice, and so on.

On most of the tracks, the present proposal attained first place. This means that the driver has at least overtaken the rest of the drivers (it started from last position). The even larger number of overtakes shows that the driver has been able to *lap* some (or all) opponents. The amount of damage obtained per maneuvre was not significantly high. On only two of the *Dirt* tracks were the results not so good.

5.2. The 2010 Simulated Car Racing Championship

In 2010, The Simulated Car Racing Championship[f] consisted of three simulated car racing competitions held at:

- Genetic and Evolutionary Computation Conference (GECCO-2010), Portland (USA), in July 2010.
- IEEE World Congress on Computational Intelligence (WCCI-2010), Barcelona (Spain), in July 2010.
- IEEE Conference on Computational Intelligence and Games (CIG-2010), Copenhagen (Denmark), in August 2010.

The goal of the championship was to design a controller for a racing car that will compete on a set of unknown tracks both alone (against the clock) and against other drivers.

The controller perceives the environment through the information explained in Section 2. Each competition consisted of three races (a total of nine races) over three tracks. Each race was divided into three stages:

1. The *warm-up*, where each driver races alone for 30 min on each track to collect information about the tracks (learning).
2. During the *qualifying* stage, each driver races alone for 300 s. The eight controllers that cover the greatest distances qualify for the final races.
3. During the *final* races, these best eight drivers race together. The race consists of eight runs on each of the three tracks, varying the starting grid. Drivers are scored using the following system: 10 points to the first, 8 points to the second, 6, 5, 4, 3, 2, and 1 to the rest. In addition, the driver performing the fastest lap and the one completing the race with least amount of damage receive two additional points.

The organization established that each leg of the championship would focus on one of the three kinds of track available in TORCS (Road, Oval, and Dirt), and also that the final races would vary in number of laps depending on the kind of track. Thus, *Oval* tracks and 50 laps were used in *GECCO*, *Road* tracks and 15 laps in *WCCI*, and *Dirt* tracks and 25 laps in *CIG*. The shapes of the tracks used in each leg of the championship are shown in Figure 16.

The controller described in this work was presented without major changes to the three legs of the competition. Just two minor changes were made to slightly adapt and improve its behavior. These improvements were as follows: the adaptation of the $Target_{speed}$ value to the damage that the vehicle has suffered (Equation 6) was added after *GECCO-2010*; and the *jump* detector in the learning module (in Section 3.7) was added after *WCCI-2010*.

Four, five, and six competitors participated, respectively, in the *GECCO*, *WCCI*, and *CIG* legs of the championship. In addition, the organization included two additional competitors developed by the organizers: *Cobostar*[14] and *Polimi*,[24,25] both of them participants from the 2009 competition, which ended that competition with remarkable standings.[11] The names of the rest of the controllers are: *Muńoz*,[26] *Mr. Racer*,[5] *Alford*, *Neil*, and *Joseph*.

[f] http://cig.ws.dei.polimi.it/?page_id=134.

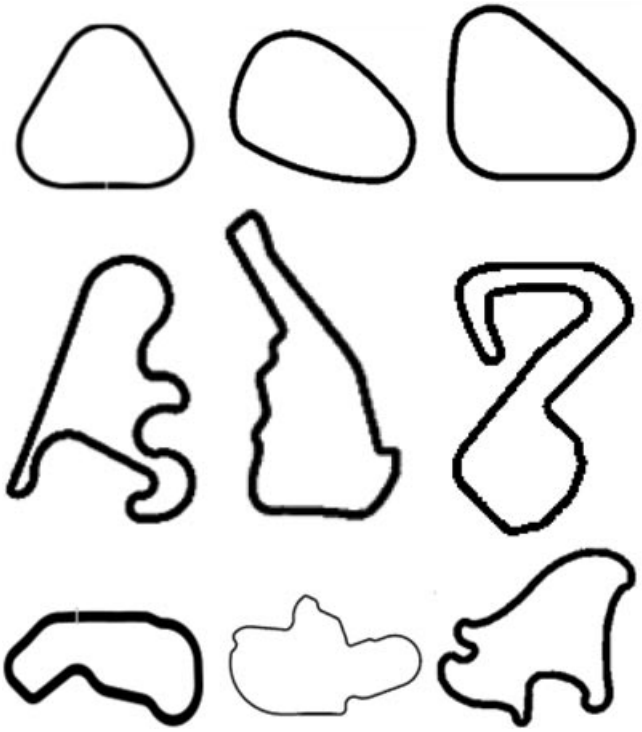


Figure 16. Tracks used during the *2010 Simulated Car Racing Championship*. *Oval* tracks used in *GECCO* (top), *Road* tracks used in *WCCI*, and *Dirt* tracks used in *CIG*.

To summarize therefore, the presented controller (henceforth *AUTOPIA*) had to compete against six, seven, and eight competitors. In this case, the qualifying stages did not disqualify any controller and all of the proposals reached the final races. Nonetheless, these qualifying stages give an idea of the controllers' speeds.

The qualifying results reported by the organization are shown in Figure 17. As can be seen, only *Cobostar* shows a generally better driving alone behavior than *AUTOPIA*. It is interesting to note that, despite the relatively poor behavior shown on the *Dirt* tracks, it was precisely on this kind of track (*CIG*) that *AUTOPIA* had its best results in comparison with the rest of the entries.

AUTOPIA was able to run faster than four of the controllers on all of the tracks, while only three of them ran faster than it on one of the nine tracks. *Joseph* was reported by the organization as *Not Classified* in the *CIG* competition due to the vehicle crashing with the borders.

With respect to the final races, Table III lists the median values of the scores obtained in the eight races run on each of the tracks, the subtotal scores obtained for each leg, and the final scores for the full championship.

One observes in the table that *Autopia* got excellent results on the *Oval* tracks, obtaining first place in the *GECCO* leg. In the *WCCI* leg (*Road* tracks), it finished in third place, with four points of difference from the two tied for first. In the *CIG*

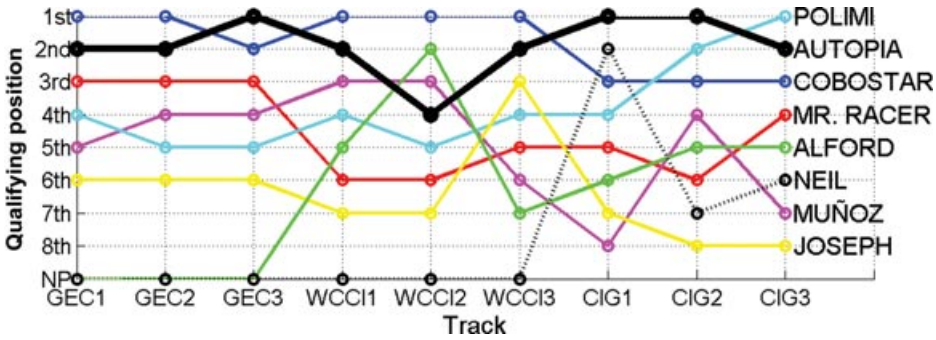


Figure 17. Positions obtained by the controllers in the qualifying stages. *ALFORD* did not participate in *GECCO*, and *NEIL* only participated in *CIG*.

Table III. Results obtained in the Championship's final races.

Race	AUTOPIA	Muñoz	Mr. Racer	Polimi	Joseph	Cobostar	Alford	Neil
$GECCO_1$	12	5.5	9	6	4	4	*	*
$GECCO_2$	12	8	4	4.5	5	4.5	*	*
$GECCO_3$	10	9	3	5.5	6.5	5.5	*	*
GECCO	**34**	**22.5**	**16**	**16**	**15.5**	**14**	**NP**	**NP**
$WCCI_1$	10	10	4	5	2	8	4	*
$WCCI_2$	8	10	3	5	4	10	3	*
$WCCI_3$	6	8	2	6	5	10	3	*
WCCI	**24**	**28**	**9**	**16**	**11**	**28**	**10**	**NP**
CIG_1	8	4	3	10	*	8	4	7
CIG_2	12	2	4	6	*	8	6	5
CIG_3	5	6	8	8	*	8	3	4
CIG	**25**	**12**	**15**	**24**	**NC**	**24**	**13**	**16**
Total	**83**	**62.5**	**40**	**56**	**26.5**	**66**	**23**	**16**

NP = not presented, NC = not classified.

leg (*Dirt* tracks), it again achieved first place, with just a small difference from its two immediate followers.

The final result was therefore that the *Autopia* driver got two first places and one third in the three legs of the championship, winning the championship overall with 17 points of difference from the second classified (*Cobostar*). *Autopia* was also the only controller able to beat those presented by the organizers of the championship.

6. CONCLUSIONS AND FUTURE WORK

We have described the design, implementation, and testing of a driving architecture for a virtual car in the context of a car-racing simulation game. To develop the driver, modularity of the system was maintained to separate tasks in such a way that each maneuvre could be changed or improved.

The main modules (*steering management* and *allowed speed determination*) were optimized by means of a GA by running the solutions on four different tracks and reporting the distance raced during a certain amount of time as the fitness function. Crashes and track leaving were penalized in the process of selecting the chromosomes of the genetic process to obtain a controller capable of dealing with different track shapes at the maximum speed possible.

Low-level modules such as *gear control* or *pedal control* (to reach the determined allowed speed) were implemented straightforwardly since their modification showed no significant improvement in the overall system.

More complex behavior, such as *managing reverse gear* or *remembering special track segments*, were implemented by means of heuristic systems designed on the basis of human knowledge and showed qualitatively good results. The authors did not consider their optimization necessary as these modules will presumably be of infrequent application—only for some specific and special tracks and situations.

The module responsible for *dealing with opponents* was implemented as a subsystem capable of slightly modifying the actions on the pedals and steering in the case of the presence of an opponent. It was designed as an intuitive rule system in the following form: *If there is an opponent nearby on the left then modify the steering command to the right*.

From the perspective of online adaptation, much work remains to be done. In addition to the learning module provided by the architecture, several adaptations need to be made to the speed and braking action when an opponent is near as a function of the actual damage that the vehicle has undergone. We hypothesize that 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.

Acknowledgments

This work was supported by the Plan Nacional, under the project Tránsito (TRA2008-06602-C03-01) and by the Comisión Interministerial de Ciencia y Tecnología under the project GUIADE (Ministerio de Fomento T9/08).

D. Pelta acknowledges the support by projects TIN2008-01948 from the Spanish Ministry of Science and Innovation, 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 2010 Simulated Car Racing Competition for their effort and dedication in organizing the event.

References

1. Lucas S. Computational intelligence and ai in games: a new ieee transactions. IEEE Trans Computat Intell Artif Intell Games; 2009;1(1):1–3.
2. Russell S, Wefald E. On optimal game-tree search using rational metareasoning. In: Proc 11th Int Joint Conf on Artificial Intelligence; 1989. pp 334–340.
3. Lucas S. Computational intelligence and games: challenges and opportunities. Int J Autom Comput, 2008,5(1):45–57.
4. Togelius J, Lucas S, Nardi R. Computational intelligence in racing games. In: Advanced Intelligent Paradigms in Computer Games, Springer; 2007. pp. 39–69.

5. Quadflieg J, Preuss M, Kramer O, Rudolph G. Learning the track and planning ahead in a car racing controller. In: IEEE Symp Computational Intelligence and Games; 2010. pp 395–402.
6. Onieva E, Cardamone L, Loiacono D, Lanzi P. Overtaking opponents with blocking strategies using fuzzy logic. In: IEEE Symp Computational Intelligence and Games; 2010. pp 123–130.
7. Tognetti S, Garbarino M, Bonarini A, Matteucci M. Modeling enjoyment preference from physiological responses in a car racing game. In: IEEE Symp Computational Intelligence and Games; 2010. pp 321–328.
8. Loiacono D, Prete A, Lanzi P, Cardamone L. Learning to overtake in torcs using simple reinforcement learning. In: IEEE Congress on Evolutionary Computation: 2010. pp 1–8.
9. Togelius J, Lucas S, Thang H, Garibaldi J, Nakashima T, Tan C, Elhanany I, Berant S, Hingston P, MacCallum R, Haferlach T, Gowrisankar A, Burrow P. The 2007 ieee cec simulated car racing competition. Genetic Programming and Evolvable Machines, 2008, 9:295–329.
10. Loiacono D, Togelius J, Lanzi P, Kinnaird-Heether L, Lucas S, Simmerson M, Perez D, Reynolds R, Saez Y. The WCCI 2008 simulated car racing competition. In: IEEE Symp on Computational Intelligence and Games; 2008. pp 119–126.
11. Loiacono D, Lanzi P, Togelius J, Onieva E, Pelta D, Butz M, Lonneker T, Cardamone L, Perez D, Saez Y, Preuss M, Quadflieg J. The 2009 simulated car racing championship. IEEE Trans Comput Intell Games 2010;2(2):131–147.
12. Perez D, Recio G, Saez Y, Isasi P. Evolving a fuzzy controller for a car racing competition. In: IEEE Symp Computational Intelligence and Games; 2009. pp. 263–270.
13. Perez D, Saez Y, Recio G, Isasi P. Evolving a rule system controller for automatic driving in a car racing competition. In: IEEE Symp Computational Intelligence and Games; 2009. pp. 336–342.
14. Butz M, Lonneker T. Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In: IEEE Symp Computational Intelligence and Games; 2009. pp 317–324.
15. Muñoz J, Gutierrez G, Sanchis A. Controller for torcs created by imitation. In: IEEE Symp Computational Intelligence and Games; 2009. pp 271–278.
16. Ebner M, Tiede T. Evolving driving controllers using genetic programming. In: IEEE Symp Computational Intelligence and Games; 2009. pp 279–286.
17. Goldberg D. Genetic algorithms in search, optimization, and machine learning. New York: Addison Wesley; 1989.
18. Onieva E, Pelta DA, Alonso J, Milanés V, Pérez J. A modular parametric architecture for the torcs racing engine. In: IEEE Symp Computational Intelligence and Games; 2009. pp 256–262.
19. The Open Racing Car Simulator Website, 2008. http://torcs.sourceforge.net/. Accessed Dec 2, 2011.
20. Simulated Car Racing Championship 2010 Competition Software Manual [online]. http://kent.dl.sourceforge.net/project/cig/Championship%202010%20Manual/1.0/manual.pdf. Accessed Dec 2, 2011.
21. Holland J. Adaptation in natural and artificial systems. Cambridge, MA: MIT Press; 1992.
22. Herrera F, Lozano M, Verdegay J. Tackling real-coded genetic algorithms: operators and tools for behavioural analysis. Artif Intell Rev, 1998;12(4):265–319.
23. Eshelman L, Schaffer J. Real coded genetic algorithms and interval schemata. foundation of genetic algorithms 2. San Francisco, CA: Morgan Kaufmann; 1993.
24. Cardamone L, Loiacono D, Lanzi P. Learning to drive in the open racing car simulator using online neuroevolution. IEEE Trans Comput Intell Games, 2010;2(3):176–190.
25. Cardamone L, Loiacono D, Lanzi PL. Evolving competitive car controllers for racing games with neuroevolution. In: Proc Conf Genetic and Evolutionary Computation; 2009. pp 1179–1186.
26. Muńoz J, Gutierrez G, Sanchis A. A human-like torcs controller for the simulated car racing championship. In: IEEE Symp Computational Intelligence and Games; 2010. pp 473–480.