Evolving a Fuzzy Controller for a Car Racing Competition

Diego Perez, Gustavo Recio, Yago Saez, *Member, IEEE, Pedro Isasi, Member, IEEE*

***Abstract*— Computational intelligence competitions have recently gained a lot of interest. These contests motivate and encourage researchers to participate on them, and to apply their work areas to specific games. During the last two years, one of the most popular competitions held on Computational Intelligence in Games conferences is the Car Racing Competition. This competition combines the fun of driving to win and the challenge of obtaining autonomous driving, which is known as a very difficult problem and faced by a lot of researches from different perspectives. For this competition, we have developed a controller with fuzzy rules and fuzzy sets for input and output, which were evolved using a genetic algorithm in order to optimise lap times, damage taken and out of track time. The design of this controller is explained in detail in this article, as well as the results obtained at the end of the contest.**

I. Introduction

International game competitions have become a very important meeting point for computational intelligent researchers. In these events, competitors are highly motivated to win the contest and usually can apply their own research topics to face the competition challenge.

The controller explained in this article was proposed for the Computational Intelligence in Games 2008 Car Racing Competition, where the objective was to develop a controller able to win races, alone and with other cars on the track. In the next lines we explore the literature in order to determine the approaches followed recently in the world of autonomous driving.

The techniques and the technologies supporting Automatic Vehicle Guidance (AVG) are an important issue [1]. Automobile manufacturers see automatic driving as a very interesting product with motivating key features which allow improving car safety, reduction of emissions and fuel consumption or optimising of driver comfort during long journeys. This topic has been addressed by numerous researchers, e.g. [2], [3], [4], [5], [6], [7], as an engineering problem. Many different approaches have been studied in recent years and the most promising ones are being engineered on real prototypes, i.e., [5], [8], [9], [10], [11] or the well-known DARPA competition [12].

However, researchers have a long way to go before these intelligent vehicles, capable of driving in a fully automated way, are actually available. Most of the work mentioned before was engineered in real prototypes, involving two main problems: the costs of buying and modifying a car, and the time needed for each test. To overcome these constraints we propose to use car simulators. Today, car simulators are very close to reality, and they allow us to speed up the research by testing more techniques.

The authors are with Carlos III University of Madrid, Av. de la Universidad 30, Madrid, Spain (contact phone: +34-91-624-8456; email: yago.saez@uc3m.es).

A. Evolutionary computation techniques applied to automated driving

Our proposal for this task is studying how evolutionary computation techniques can help in automatic driving. For this proposal we have collected some work related to evolutionary computation techniques applied to automated driving.

One of the first approaches in this area was carried out at the Carnegie Mellon University by Sukthankar *et al.* in 1996, [6]. This work uses reasoning modules which combine high level task goals with low-level sensor constraints. Those modules are directly dependent on a large number of parameters, and the setting of these parameters must be done carefully. As this manual selection is tedious and error prone, a Population Based Incremental Learning (PBIL), which is a combination of Genetic Algorithms (GAs) and competitive learning, [13], is proposed for automatically setting each module's parameters.

Therefore, the algorithm will analise what the probabilities of using a set of rules are depending on each situation. The evaluation function takes into account different aspects, such as serious crashes, collisions, wrong exits, distance completed, etc. For the simulation, the system uses a program called SHIVA (Simulated Highways for Intelligent Vehicle Algorithms) which reproduces a microsimulation of vehicles moving and interacting in a user-defined roadway. The algorithm can influence the vehicle motion sending simulated commands (steering, accelerate and brake). In addition, the system provides a perception module responsible for obstacle detection, positioning, lane tracking, vehicle sensing, etc. Two years later the PBIL was compared to the GA in this same framework, [14]. The results of these works were quite motivating; at the end of experiments the vehicles automatically entered the test track, completed one lap and a half and finally took the exit. Although they were not capable of avoiding collisions with other vehicles, these experiments showed the potential for intelligence behavior in tactical driving.

Other interesting work carried out in 1996 by Pyeatt *et al.* from Colorado University dealt with simulated race car driving [15]. In this case a study on autonomous driving was developed based upon RARS simulator software (also known as TORCS). This simulator gives information about the vehicle position, vehicle speed, distance to the current track segment, curvature of the track, and relative position and velocity of nearby cars. It also offers the possibility of controlling the speed and steering of the vehicle.

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

This work applied neural networks for learning automatic driving through the use of inputs such as position, speed, distance measured till the end of the segment, angle between the vehicle and the road, etc. This neural network produced a set of rules which decided when to accelerate, brake or steer. The results showed that the RARS simulator was adequate for developing the test framework and that neural networks were competitive techniques for producing autonomous racing cars.

In 1998, Bernard *et al.* from Iowa State University illustrated the power of GAs to model driver/vehicle behavior. In fact, their work determined how fast and safe a given vehicle model could be driven through a short course without failure from hitting a cone or lifting the wheels. In this work they used a simulator that was developed by them. The GA represents the vehicle movements with the starting and ending points of the path, and a measure of the position with the first and second derivatives of the path in a point near the middle set of cones. The results were very good but they only reached a near optimal solution due to the constraints of the genetic representation they used.

In 2004, Floreano *et al.*, [16], imitating strategies observed in simple insects, used a GA to tune up a neural network which visually recognises edges, corners and height. This active vision system acts as an artificial retina, moving and focusing on important features. It was tested with the open source simulator Car-World (http://carworld.sourceforge.net) and the best evolved individuals performed equal to or better than well-trained human drivers tested on the same circuits.

In 2005, Sun *et al.* [17], used a GA to optimise the parameters of a set of Gabor filters in the context of vehicle detection from images. They tested the proposed framework on real data with success and improved the performance of on-road vehicle detection. In the same year another interesting approach was proposed for automated evolutionary design of driving agents, [18] [19]. This work showed how GAs can help in the task of designing an agent able to remotely operate a scale racing car. The agent perceives the environment from a camera mounted overhead (position, orientation, velocity, approach angle, distance to the apex and outside/inside slow down zone). With these perceptions the agent sends commands to the remote controlled car (forward, neutral, reverse, left, straight or right). The comparative analysis established that on long runs the agent's operated car was 5% slower than the human operated one.

Working with the evolving weights of a neural network, Julian Togelius *et al.* compared, with their own simulator, that simulated cars with evolved neural network controllers (in first-person and third-person) [20], [21]. They extended their work to a more complex case of two cars competing against each other in the same track at the same time, [22], using evolutionary strategies to solve the problem of the co-evolution. Finally, an interesting study which compares neuroevolution and genetic programming in the same environment can be found in [23].

B. Fuzzy logic applied to autonomous driving

The usage of fuzzy systems to manage controllers for certain devices is widely spread in the literature. The characteristics of them allow a fuzzy perception of the world, what it is very useful for designing controllers for robots, machines or vehicles. Usually, a robot may use parameters as position or distance to move ahead or to raise something up form the environment, and depends on the exact measure of its sensors to continue working. Fuzzy logic allows us to establish non-crispy values to these parameters, so we can say that the box to take is "on the right", or that the next bend is "close" enough to start turning.

An interesting description of how fuzzy logic can be used in autonomous robot navigation can be found in the works done by Alessandro Saffiotti in 1993 [24] and 2004 [25]. Furthermore, implementations of these systems have been done in simulators, as in the work performed by Astudillo *et al.* in 2006 [26], where stabilisation and trajectory tracking were optimised for an unicycle mobile robot, or by Baltes and Otte in 1999 [27], where a simple heuristic helps the design of the fuzzy system (rules, input and output sets) in car-like mobile robots. In this last research the evaluation was performed both in simulated and real world, stating that the results are significantly better that the ones obtained with traditional controls.

Additionally, some research can be found about using evolutionary computation within fuzzy logic, as Takagi *et al.* did in 1993 [28] and Y. Lee *et al.* in 1994 [29], where a genetic algorithm determines the shape and position of the membership functions. Also in the work of Andrea, in 1996 [30], a population of fuzzy rules is evolved through competition and cooperation, in order to obtain a sub-optimal fuzzy logic controller for the classical cart-pole problem.

Several works have used both fuzzy logic and evolutionary computation for the problem of autonomous navigation. For instance, Tunstel in 1996 [31] used genetic programming for generating rules for a fuzzy logic controller, that manages the steering of a mobile robot, or Freeman et al., in 1997 [32], where a genetic algorithm learns and optimises the definitions of linguistic variables used in the rule system that controls the spacecraft rendezvous process. Furthermore, some research can be found about using evolutionary computation and fuzzy logic on autonomous driving, such Hoffmann in 1994 [33], that describes an implementation for a hierarchical fuzzy controller evolved with a genetic algorithm, and Huang in 1999 [34], where a neural fuzzy network is used, which weights are obtained through a genetic algorithm.

II. Objectives

A. Competition Objectives

This competition is a new edition of the contests taken place in 2007 [35] and 2008 [36]. The first one, was organized as part of the IEEE Congress on Evolutionary Computation (CEC) and Computational Intelligence and Games Symposium (CIG). The second, was hold during

the celebration of the WCCI (World Congress of Computational Intelligence), where we participated with another controller [37]. This time, it was held the same year in the IEEE Symposium on Computational Intelligence and Games (CIG 2008). The new competition was very similar to the previous one, where the objective was to design and develop a controller able to race in a simulator. Some considerations have to be taken into account, as the fact that the circuits where the cars will run are, a priori, unknown by the competitors, and that the races will be performed both with and without other racers on the track.

B. TORCS Simulator

As we said before, the races will be run in a simulated environment. As the organizers did in the previous contest, the software used to hold the competition is TORCS (The Open Racing Car Simulator). This simulator, formerly known as RARS, is one of the most used simulators by different developers and gamers. The following characteristics make it a good choice for those interested in simulated car racing or autonomous driving:

- This simulator is written in C++ and can be downloaded under GPL license from its web page.
- It is available for several platforms: Source code and executables are ready for Microsoft Windows and Linux, while binaries can be obtained for MacOS.
- There is a big community of users and competitors that help to maintain the software updated and bug free.
- It provides a high level of realism, and a very reliable physics system.
- A large quantity of vehicles, tracks and controllers, so many different race configurations can be prepared.

Nevertheless, some problems have been found when using this simulators. One of them, maybe the most important one, is a memory leak that happens every time the race is restarted. Although it is not a huge quantity of memory lost each time, the usage of some learning or evolutionary algorithms that require a significant number of evaluations (and consequently, race restarts) make this memory leak become higher and higher until the simulator process crashes.

C. TORCS in Simulated Car Racing Competition

The organizers of the contests (Daniele Loiacono, Julian Togelius and Pier Luca Lanzi) provide the competitors with some modules to develop their controllers. The *server* module is a component for TORCS that provides the communication to the remote controller, implementing and supplying an API for the sensors and actuators models. In other words, the server module provides the drivers with a representation of the current game state and a protocol to interact with it. On the other hand, two *client* modules are provided, one written in C++, the other in Java. These components implement a basic communication module with the server, and allow the competitors to choose the programming language they are more comfortable with.

The communication between client and server modules is performed through UDP packages. When the information comes from the server module, reading of several car sensors is provided. If the information goes from the client to the server, it must contain the actuators of the controller, such acceleration and steering.

The competitors receive information from 16 different sensors. Among them, car status (angle of the car on track, damage suffered, fuel, current gear, speed, etc.), car information related to the track (distance raced from start line, last lap time, distance to track edges, etc.) and car information related to other cars (distance to opponents, race position) can be obtained. The effectors that can be used by the side of the clients are the steering wheel value, both pedals usage (throttle and brake) and gearing change.

A full description of all the sensors provided can be seen in the bases of the competition (documentation, sources and examples are available at http://cig.dei.polimi.it/?page_id=67).

D. Competition Rules

Every submitted controller is tested alone in three different tracks, and the score of the evaluation is the distance raced by the car during $10,000$ game tics (about 3 minutes and 20 seconds). When this first stage is over, the best few controllers compete among them on a different set of tracks, in order to determine how they behave in presence of other drivers.

Competition rules, objectives and sensors are described in the competition manual (http://cig.dei.polimi.it/wp-content/uploads/2008/10/manual_cig2008_v1.pdf).

III. Controller Design

The controller we prepared for this competition is based on two main pillars: as we said before, the usage of fuzzy logic has been proposed in several works in the field autonomous driving, so the first one is a fuzzy representation of the world. The second pillar is the evolution of this fuzzy representation through evolutionary computation.

A. Fuzzy representation

1) Membership functions: The environment of the car has been discretised to a set of *fuzzy entries*, each one of them referred to a sensor or an effector that the server provides to the controller. These propositions are composed by a group of *fuzzy sets* that represent the state of each one of these sensors.

All the fuzzy sets are defined by smooth trapezoidal membership functions. Its general expression is defined by the equation 1 and the shape depicted in the figure 1.

$$\mathsf{STrap}(x;a,b,c,d) = \left\{ \begin{array}{ll} 0 & ,x \leq a \\ \frac{1}{2} + \frac{1}{2}\cos(\frac{x-b}{b-a}\pi) & ,a \leq x \leq b \\ 1 & ,b \leq x \leq c \\ \frac{1}{2} + \frac{1}{2}\cos(\frac{x-c}{d-c}\pi) & ,c \leq x \leq d \\ 0 & ,d \leq x \end{array} \right\}, x \in \mathbb{R} \quad (1)$$

This function provides truth values for a given proposition, which crisp value is taken from one of the available sensors. Its shape is defined by four values (a,b,c,d). Furthermore,

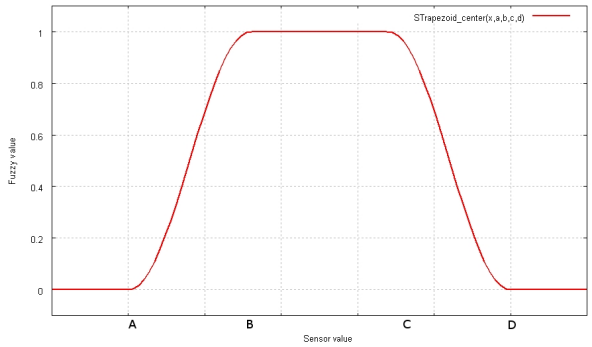

Fig. 1. General membership function shape

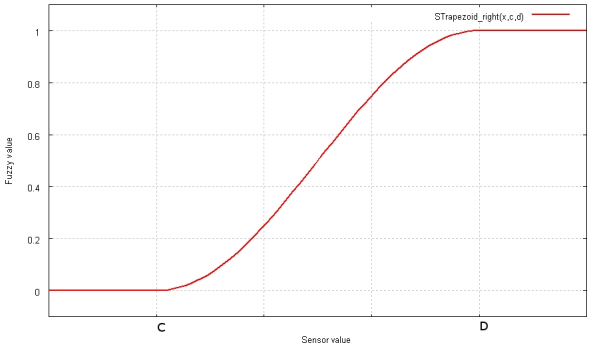

Fig. 3. Right membership function shape

two more membership functions are used to represent all the fuzzy propositions needed for the whole set of sensors and actuators used (see equations 2, 3,and figures 2, 3)

$$\text{SLTrap}(x;a,b) = \left\{ \begin{array}{ll} 0 & ,x \leq a \\ \frac{1}{2} + \frac{1}{2}\cos(\frac{x-b}{b-a}\pi) & ,a \leq x \leq b \\ 1 & ,b \leq x \end{array} \right\}, x \in \mathbb{R} \quad (2)$$

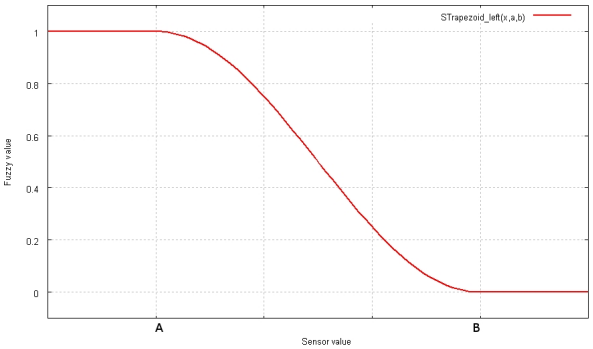

Fig. 2. Left membership function shape

$$\text{SRTrap}(x;c,d) = \left\{ \begin{array}{ll} 1 & ,x \leq c \\ \frac{1}{2} + \frac{1}{2}\cos(\frac{x-c}{d-c}\pi) & ,c \leq x \leq d \\ 0 & ,d \leq x \end{array} \right\}, x \in \mathbb{R} \quad (3)$$

The sensor *track position* determines the distance between the car and the track axis. This value is normalised with respect to the track width, where -1 means the right edge, 0 the center of the track and 1 the left side. Values far from -1 and 1 represent the car outside the track on each side. If we take, for instance, this sensor for a fuzzy entry, we could define the sets *left side*, *centered* and *right side* to represent the track position of the car in a fuzzy way.

This entry is defined then by three different sets, each one of them described with the three different membership functions depicted above. The *left side* set is used to determine when the car is in the left part of the track, and it only needs two variables ("a" and "b") to define its membership function, because of the semantic of the sets and the possible values of the sensor. The car will be on the left part of the track if this sensor value is between -1 and another value greater than -1. In this case, the best way to determine the membership of this proposition is defining the values for which the car is not on the left part, that is, we need to define the limits of "being on the left part" of the track. Therefore, the membership function that fits better in this set is the *left trapezoidal membership function*.

The same happens with the set *right side*, where 1 is the maximum value. However, to determine the value of the *centered* set, we can not make the assumption of where to put the limits for the car to be centered on the track. This is because we use the general trapezoidal membership function to define this set. The figure 4 represents an example for this fuzzy entry, with its three sets depicted together.

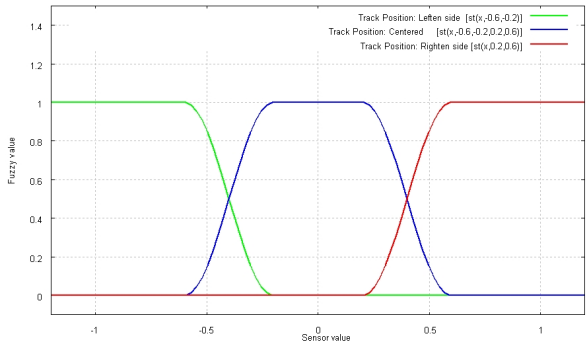

Fig. 4. Right membership function shape

To define this entry, we need eight different values for the functions: Two for the *left side* set (-0.6 and -0.2), four for *centered* set (-0.6, -0.2, 0.2, 0.6) and two more for *right side* shape (0.2 and 0.6).

2) Fuzzy sets: The tables I and II summarise all the sensors and actuators used for this controller, their fuzzy entries associated and all the fuzzy sets defined for each entry.

TABLE I

FUZZY SETS TABLE (INPUT)

Sensors (Input)			
Name	**Fuzzy entry**	**Fuzzy Set**	**Trapezoidal Shape**
TrackPos: [-1, 1]	Track Position	Left side	Left membership function
		Centered	General membership function
		Right side	Right membership function
Angle: [-π, π]	Angle	Oriented hard left	Left membership function
		Oriented soft left	General membership function
		Centered	General membership function
		Oriented soft right	General membership function
		Oriented hard right	Right membership function
Speed: [0, N] (km/h)	Speed	Slow	Left membership function
		Medium	General membership function
		Fast	Right membership function
Track: [0, 100]	Track	Turn on left	Left membership function
		Turn on right	Right membership function
Time to turn: [0, 100]	Time to turn	Time to turn	Left membership function
Overhead position: [0,100]	Car ahead on	Left	Left membership function
		Center	Left membership function
		Right	Left membership function

TABLE II

FUZZY SETS TABLE (OUTPUT)

Actuators (Output)			
Name	**Fuzzy entry**	**Fuzzy Set**	**Trapezoidal Shape**
Steer: [-1, 1]	Steer	Hard turn left	Left membership function
		Soft turn left	General membership function
		Slight turn left	General membership function
		Center	General membership function
		Slight turn right	General membership function
		Soft turn right	General membership function
		Hard turn right	Right membership function
Accel: [0, 1]	Acceleration	Full brake	Left membership function
		Medium brake	General membership function
		Maintain speed	General membership function
		Medium accel	General membership function
		Full accel	Right membership function

As we can see, both sensors and actuators are related to fuzzy sets. An important aspect to be taken into account is that we need a crisp value for the effector sets. For instance, if we want to apply *full acceleration* for the acceleration actuator, we need to obtain a concrete value to use. For all the actuator fuzzy sets, a Center Of Gravity (COG) function is used to retrieve the crisp value. The next formula is used as an approximation for a fuzzy set *f(x)*, as shown in equation 4.

$$\mathrm{COG}\ f(x) = \frac{\sum_{x=\min}^{\max} x * f(x)}{\sum_{x=\min}^{\max} f(x)} \tag{4}$$

3) Fuzzy rules and fuzzy state: Once all the fuzzy sets for the controller have been defined, the next step is to determine fuzzy rules that use these sets. In this controller, the definition of all the fuzzy rules has been done by hand. Each one of them is composed by a certain number of fuzzy sets as an entry (that correspond to sensors) and two fuzzy sets as output (acceleration and steering actuators). In a general way, these rules can be represented as shown in equation 5, where A to Z represent a subset of sensor fuzzy sets, and α and β stand for actuator ones. An example of these rules could be as the one in the figure 5.

$$R_i = A_i \otimes B_i \otimes C_i \ldots Z_i \Rightarrow \alpha_i, \beta_i \tag{5}$$

R_1 = Car on center ⊗ Angle Centered ⊗ Track Position centered ⊗ Speed fast
⇒
Maintain accel, Soft turn left

Fig. 5. Fuzzy rule example

In this rule, our car is centered on the track on high speed, its angle is parallel with the track axis and it has another vehicle ahead. The actions recommended in this case is to maintain acceleration and steering to the left to overhead the opponent.

We define *fuzzy state* as the collection of the condition part of all the rules. Internally, this rules are stored in an array, where each position corresponds to an index rule and its value is performed by multiplying the truth values of each fuzzy set in the condition. What we obtain then is an fuzzy state of the car, where each rule has its own truth value.

Each simulation cycle, this array is updated calculating the truth value of each fuzzy set and applying the multiplications needed. The next step is to find the highest truth value for a rule and apply its right part (this is, the actuators), using center of gravity function for the crisp values. This algorithm is depicted in the figure 6.

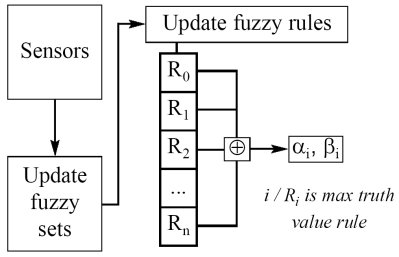

Fig. 6. Fuzzy rule example

B. Evolutionary algorithm

1) Base individual: one of the biggest issues that must be faced on applying evolutionary algorithms in autonomous driving is that a good starting point is needed. Automatic driving is so complex that it is really hard to obtain a driver configuration by chance, as is usually done in initial populations of evolutionary algorithms, where they are initialised randomly. For this reason we need to define a base individual to start evolving from, that must be able to drive on different tracks at least in a very simple way, as a novice would do when he is learning how to drive. The objective then is to evolve the configuration of this driver to obtain a competitive racer. In this case, we need to initialise two distinct parts of the controller: fuzzy rules and fuzzy set parameters.

Firstly, the set of fuzzy rules must be defined, establishing the relations between the available fuzzy sets as shown in previous sections. The algorithm designed does not evolve this set of rules, so the final individual will keep the same set as the base individual. Because of the amount of combinations among the available fuzzy entries and sets, nearly a hundred of rules were defined in this stage. They can be divided into different groups for a better understanding:

- General driving rules: a set of rules that is focused on keeping the car on the track while there is no predicted turn ahead. Its aim is to maintain the car as centered and as fast as possible.
- Turning prediction rules: these rules are focused on the track sensors in order to identify when a new bend is coming and, depending on its sense, proceed to drive through it in the best possible way.
- Emergency rules: the rules of this set are designed to drive the car in emergency cases, such as facing the track in the wrong direction or being outside it.
- Overhead rules: the last set of rules are intended to manage the situations where other cars are involved, so overhead actions can be taken.

Secondly, the parameters for every fuzzy set must be set up. These values are the ones that will be optimised by the evolutionary computation. As we stated before, no random initialisation will take place, therefore an initialisation is needed to generate the base individual.

Notice that there is an alternative design for the algorithm used keeping a very similar approach to this one: instead of maintaining the rules fixed and evolve the fuzzy set parameters, it could be done in other way, where the evolutionary algorithm would pick some rules up from the whole set of rules that can be composed, keeping the original parameters for the fuzzy sets unvariable. Nevertheless, that approach makes the initialisation of the fuzzy sets parameters very relevant. In other words, it is more difficult to determine the exact parameters for each fuzzy set rather than good rules.

2) Evolution: The evolutionary algorithm implemented to evolve the fuzzy sets is a genetic algorithm, using a steady state for each evolution step. The main characteristics of the implemented algorithm are:

- The individual: as said before, the individual is composed by the parameters of the fuzzy sets that form the controller.
- Initial population: the initial population is obtained by taking a base individual and creating mutated copies of it to fill the whole population. It is important to keep on mind that one non-mutated copy of the base individual is kept in the initial population, in order to assure at least one stable configuration.
- Selection: the selection process for choosing individuals to create new ones is a tournament of size 3.
- Crossover: the crossover between two individuals is implemented as an uniform crossover, the units exchanged between the parents are not the parameters, but the whole fuzzy set. For instance, when the new individual is being created, it will inherit the set "speed fast" from one of its parents, but not each parameter separately.
- Mutation: each parameter of every fuzzy set is mutated, obeying a mutation probability, adding an small amount for its value. Furthermore, the limits of the fuzzy entry must be checked in order to avoid overflow of the values.
- Fitness and evaluation: to evolve an individual, different races are performed in four distinct tracks. In each race, the car drives alone during a fixed amount of game tics and the fitness of the individual is retrieved as the number of meters raced during this time, minus a measure of game tics when the car was stucked against a wall, facing backwards or being outside the track. Finally, the mean of this calculation in the four circuits is assigned as the individual fitness.

The graph depicted in figure 7 represents the fitness evolution of the controller submitted to the competition on the training circuits.

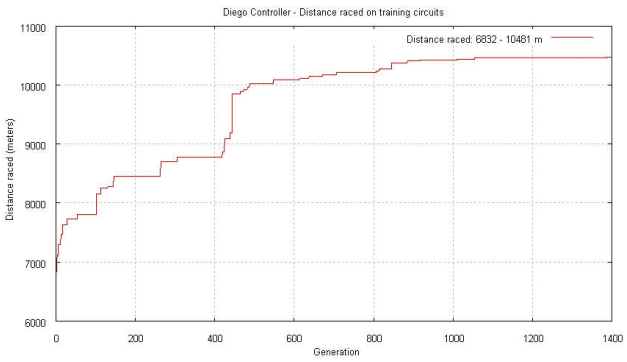

Fig. 7. Evolution of fitness on training circuits

Although this is only one of the experiments performed (indeed, the best one), the fact that the distance raced by the vehicle is increased in few generations suggests that this could be a good technique for evolving autonomous driving from a base individual. However, a lot of experiments and a statistical analysis must be done in order to demonstrate this empirically.

As it has been said before, the evolution steps of the algorithm modify the shape of the controller's fuzzy sets. As an example, figure 8 shows the evolution of one of these fuzzy sets: the position of the car when it is centered on the track. It can be seen that, through evolution, the shape of the function becomes narrower, what means that less values of the sensor will make the vehicle to be considered as centered on the track.

IV. Results

In this competition, the controllers were scored in three different tracks: *C-Speedway*, *E-Track 6* and *Wheel-2*. On the first stage, controllers raced alone in these tracks, and the distance raced within $10,000$ game tics were used as score. There were five participants in the competition: in the figures, labelled as Redjava (Chung Cheng Chiu, Academia

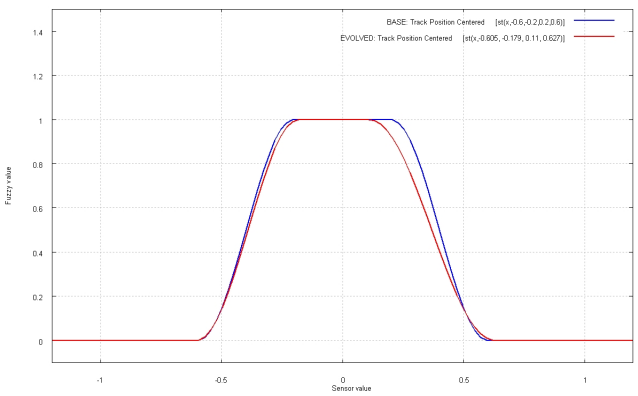

Fig. 8. Evolution of Track Position set

Sinica), Luigi (Luigi Cardamone, Politecnico de Milano), Diego (Diego Perez and Yago Saez, Universidad Carlos III), Matt (Matt Simmerson, New Zealand) and Aravind (Aravind Gowrisankar, UT Austin). Furthermore, three more non-participant controllers were used to compare with the competitors: Daniele (Daniele Loiacono, Politecnico de Milano), Julian (Julian Togelius, IDSIA) and the winner of the last contest (WCCI'08 champ).

Figures 9, 10 and 11 show the results obtained in the stage explained above:

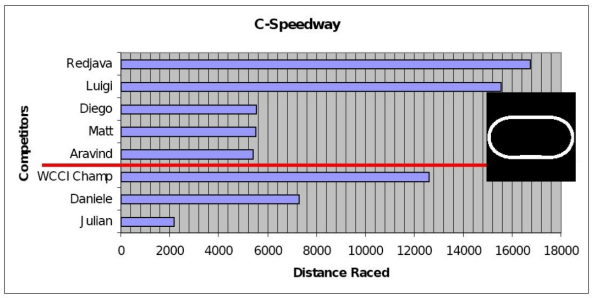

Fig. 9. Results on C-Speedway

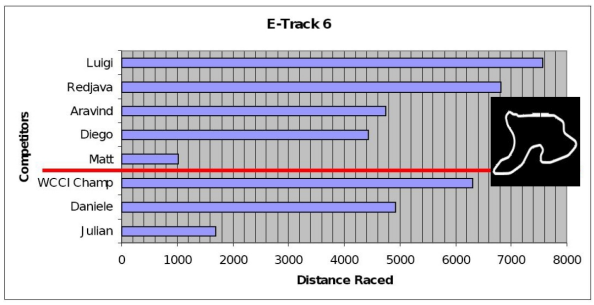

Fig. 10. Results on E-Track 6

As we can see, the participant controllers are divided clearly into two groups: Redjava and Luigi are much more faster than Diego, Matt and Aravind controllers. Organizers use Formula One scoring system to give points to the participants (10 points for the first, 8 for the second, 6 for the third, 5 for the fifth...), and results are shown in table III.

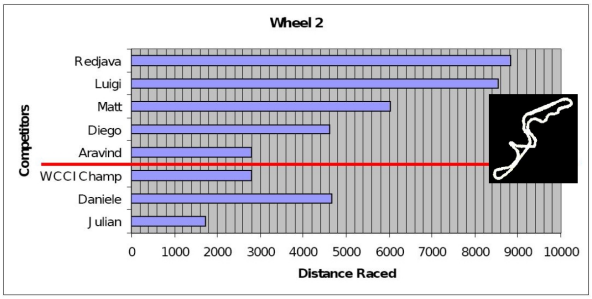

Fig. 11. Results on Wheel-2

TABLE III
CLASSIFICATION OF FIRST STAGE

Round 1	C-Speedway	E-Track 6	Wheel-2	TOTAL
REDJAVA	10	8	10	28
LUIGI	8	10	8	26
DIEGO	6	5	5	16
MATT	5	4	6	15
ARAVIND	4	6	4	14

Our controller has ended this first stage in third position, on top of the second group of controllers. These three controllers, as well as the first two ones between them, are very close to each other, but there is a big difference between these two groups.

The classified controllers for the second stage were the ones of the first group (Redjava and Luigi) and the winner of the previous competition. The winner of CEC 2009 Simulated Car Racing Competition was Luigi, and the results are shown in table IV.

V. CONCLUSIONS

One of the problems found on this controller is that it is not able to accelerate and brake completely. Although there are respective fuzzy sets for that actions (full acceleration, full brake), the values used by the car are the results of applying the center of gravity function. As the shape of the fuzzy set membership function is defined by some parameters that can not take values outside of the sensor limits, the result of the deffuzzifier function can not be the maximum (1: full acceleration value) and minimum (0: full brake) values. This restriction makes the car unable to get very high speeds or perform sudden brakings, so the usage of fuzzy sets on actuators must be improved to allow that.

However, the use of fuzzy sets on sensors has been quite useful, as can be seen on the fitness curve on training circuits.

TABLE IV
FINAL STAGE RESULTS

Round 2	C-Speedway	E-Track 6	Wheel-2	TOTAL
LUIGI	9	11	10	30
REDJAVA	10	8	9	27
WCCI	6	6	6	18

Additionally, the results obtained on this competition are quite better than the ones obtained by the authors controller prepared for the last competition, WCCI 2008.

These facts can guide us to design controllers for the next competition that will take place on CEC 2009. Furthermore, it reinforces the idea that the usage of fuzzy sets is a good choice to produce evolution on autonomous driving, even if it is not focused on races but centered on autonomous driving on traffic environments.

ACKNOWLEDGMENTS

This work was supported in part by the Spanish MCyT project MSTAR, Ref: TIN2008-06491-C04-03.

REFERENCES

[1] J. Bernard, J. Gruening, and K. Hoffmeister. Evaluation of vehicle/driver performance using genetic algorithms. *SAE International Congress and Exposition*, (980227), 1998.

[2] A. Niehaus and R. F. Stengel. Probability-based decision making for automated highway driving. *Vehicle Navigation and Information Systems Conference, 1991*, 2:1125–1136, Oct. 1991.

[3] G. Siegle, J. Geisler, F. Laubenstein, H. Nagel, and G. Struck. Autonomous driving on a road network. *Intelligent Vehicles '92 Symposium., Proceedings of the*, pages 403–408, Jun-1 Jul 1992.

[4] R. Sukthankar, D. Pomerleau, and C. Thorpe. Shiva: Simulated highways for intelligent vehicle algorithms. In *In Proceedings of IEEE Intelligent Vehicles*, pages 332–337, 1995.

[5] D. Pomerleau and T. Jochem. Rapidly adapting machine vision for automated vehicle steering. In *IEEE Expert*, 11(2):19–27, Apr 1996.

[6] R. Sukthankar, J. Hancock, S. Baluja, D. Pomerleau, and C. Thorpe. Abstract adaptive intelligent vehicle modules for tactical driving. In *In Proceedings of AAAI-1996 Workshop on Intelligent Adaptive Agents.*

[7] C. Thorpe, T. Jochem, and D. Pomerleau. Automated highway and the free agent demonstration. In *In Proceedings of 1997 IEEE Conf. on Intelligent Transportation Systems*, pages 496–501, 1997.

[8] M. Bertozzi, A. Broggi, G. Conte, and R. Fascioli. The experience of the argo autonomous vehicle. In *in Procs. SPIE'98 - Aerosense Conf*, volume 3364, pages 218–229, 1998.

[9] J. M. Collado, C. Hilario, A. de la Escalera, and J. M. Armingol. Self-calibration of an on-board stereo-vision system for driver assistance systems. In *Intelligent Vehicles Symposium, 2006 IEEE*, pages 156–162, June 2006.

[10] Wu, B.-F. Chen, C.-J. Chiang, H.-H. Peng, H.-Y. Perng, J.-W. Ma, L.-S. Lee, T.-T. The Design of an Intelligent Real-Time Autonomous Vehicle, Taiwan iTS-1 In *Journal - Chinesse institute of engineers*, volume 30; 5, pages 829–842, 2007.

[11] P. Lamon, S. Kolski, R. siegwart The Smarter vehicle for fully autonomous navigation In *Proceedings of CLAWAR 2006*, Brussels, Belgium, 2006.

[12] Q. Chen, U. Ozguner, and K. Redmill. Ohio state university at the 2004 darpa grand challenge: developing a completely autonomous vehicle. In *Intelligent Systems, IEEE*, 19(5):8–11, Sept.-Oct. 2004.

[13] S. B. and R. Caruana. Removing the genetics from the standard genetic algorithm. In *The Int. Conf. on Machine Learning 1995*, pages 38–46, San Mateo, CA, 1995. Morgan Kaufmann Publishers.

[14] S. Baluja, R. Sukthankar, and J. Hancock. Prototyping intelligent vehicle modules using evolutionary algorithms. pages 241 – 257, 1998.

[15] L. D. Pyeatt, A. E. Howe, and C. W. Anderson. Learning coordinated behaviors for control of a simulated robot. In *Technical report* Computer Science Dept, Colorado State Univ., Ft. Collins, CO 80523, 1996.

[16] D. Floreano, T. Kato, D. Marocco, and E. Sauser. Coevolution of active vision and feature selection. In *Biological Cybernetics*, 2004.

[17] Z. Sun, G. Bebis, and R. Miller. On-road vehicle detection using evolutionary gabor filter optimization. In *IEEE Transactions on Intelligent Transportation Systems*, (6):125–137, 2005.

[18] I. Tanev, M. Joachimczak, H. Hemmi, and K. Shimohara. Evolution of the driving styles of anticipatory agent remotely operating a scaled model of racing car. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, 2:1891–1898 Vol. 2, Sept. 2005.

[19] I. Tanev, M. Joachimczak, and K. Shimohara. Evolution of driving agent, remotely operating a scale model of a car with obstacle avoidance capabilities. In Mike Cattolico, editor, *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1785–1792, New York, NY, USA, 2006. ACM.

[20] J. Togelius and S. M. Lucas. Evolving controllers for simulated car racing. In *The 2005 IEEE Congress on Evolutionary Computation, 2005*, 2:1906–1913 Vol. 2, Sept. 2005.

[21] J. Togelius and S.M. Lucas. Evolving robust and specialized car racing skills. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1187–1194, 2006.

[22] J. Togelius and S. M. Lucas. Arms races and car races. In *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference, Reykjavik, Iceland, September 9-13, 2006, Procedings*, volume 4193 of *Lecture Notes in Computer Science*, pages 613–622, 2006.

[23] A. Agapitos, J. Togelius, and S. M. Lucas. Evolving controllers for simulated car racing using object oriented genetic programming. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1543–1550. ACM Press, 2007.

[24] Alessandro Saffiotti; Sr. Ruspini Enrique H.; Sr. Konolige Kurt G. A fuzzy controller for flakey, an autonomous mobile robot. In *Technical Note 529, Artificial Intelligence Center*, SRI International, 333 Ravenswood Ave. Menlo Park, CA 94025, USA. 1993.

[25] Alessandro Saffiotti. The uses of fuzzy logic for autonomous robot navigation. In *Journal of Intelligent and Robotic Systems*, Volume 40, Issue 1, Pages: 45–88. May 2004.

[26] Astudillo, L., Castillo, O., Melin, P., Alanis, A., Soria, J., Aguilar, L. Intelligent Control of Autonomous Mobile Robot Using type2 fuzzy logic. In *Journal of Engineering Letters*, 13(2):93–97, September 2006.

[27] Baltes, J.; Otte, R. A Fuzzy Logic Controller for Car-like mobile robots. In *Computational Intelligence in Robotics and Automation, 1999. Proceedings. 1999 IEEE International Symposium*, pages: 89–94. 1999.

[28] Lee, Michael A. and Takagi, Hideyuki. Integrating design stages of fuzzy systems using genetic algorithms In *Proc. 2nd IEEE Inter. Conf. on Fuzzy Systems*, San Francisco, CA, pp. 612–617. 1993.

[29] K.C. Ng, Y. Lee. Design of sophisticated fuzzy logic controllers using genetic algorithms. In *Proceedings of the Third IEEE International Conference on Fuzzy systems (FUZZ-IEEE'94)*, Orlando, USA, pp. 1708–1712, 1994.

[30] Andrea Bonarini. Evolutionary Learning of Fuzzy Rules - competition and cooperation In *W. Pedrycz(Ed.), Fuzzy Modelling: Paradigms and Practice, Kluwer Academic Press, Norwell, MA*, 1996.

[31] Tunstel and Jamshidi. On genetic programming of fuzzy rule-based systems for intelligent control. In *International Journal of Intelligent Automation and Soft Computing*, 2(3), pages 273-284, 1996.

[32] Karr, C. L., Freeman, L. M. Genetic algorithm based fuzzy control of spacecraft autonomous rendezvous. In *Engineering Applications of Artificial Intelligence*, 10(3), pages 293-300. 1997.

[33] Frank Hoffmann, Gerd Pfister. Automatic Design of Hierarchical Fuzzy Controllers Using Genetic Algorithms. In *Proceedings of the Second Conference on Intelligent Techniques and Soft Computing (EUFIT'94)*, Aachen, Germany, pp. 1516–1522, 1994.

[34] S. Huang and W. Ren. Use of Neural Fuzzy Networks with Mixed Genetic/Gradient Algorithm in Automated Vehicle Control. In *IEEE Transactions on Industrial Electronics 46*, No. 6, pages 1090-1102, December 1999.

[35] Julian Togelius, Simon Lucas, Ho Duc Thang, Jonathan M. Garibaldi, Tomoharu Nakashima, Chin Hiong Tan, Itamar Elhanany, Shay Berant, Philip Hingston, Robert M. MacCallum, Thomas Haferlach, Aravind Gowrisankar, Pete Burrow. The 2007 IEEE CEC simulated car racing competition In *Genetic Programming and Evolvable Machines*, Volume 9, Issue 4, Pages: 295–329 , December 2008.

[36] Daniele Loiacono, Julian Togelius, Pier Luca Lanzi, Leonard Kinnaird-Heether, Simon M. Lucas, Matt Simmerson, Diego Perez, Robert G. Reynolds, Yago Saez The 2008 IEEE WCCI simulated car racing competition In *Proceedings of IEEE Computational Intelligence and Games 2008*. Pages to appear.

[37] Y. Sáez, D. Perez, Gustavo Recio, and P. Isasi. Evolving a rule system controller for automatic driving in a car racing competition. In *IEEE Symposium on Computational Intelligence and Games (CIG'08)* Pages 336–342, 2008.