ISSN 1870-4069

Simultaneous Evolution of Neuro-Controllers for Multiple Car-like Robots

Antonio López Jaimes, Jorge Cervantes-Ojeda, Maria C. Gómez-Fuentes, A. Montserrat Alvarado-González

Universidad Autónoma Metropolitana, Department of Applied Mathematics and Systems, Mexico City, Mexico
`tonio.jaimes@gmail.com, jcervantes@correo.cua.uam.mx, mgomez@correo.cua.uam.mx, amontserrat@gmail.com`

Abstract. In this paper, we presented a methodology that allows the simultaneous evolution of neuro-controllers for multiple car-like robots. The methodology consists of three modules: the Evolutionary Algorithm module based on the Rank Genetic Algorithm to optimize the weights of the robots' neuro-controllers, the Robots Controllers Module based on a Feedforward Neural Network, and the Robots Module, where the TORCS car-like robots simulator was used for the experiments. Since TORCS' characteristics are realistic enough, the obtained results could provide insights applicable to real autonomous cars. We set up a challenging driving scenario in order to test the RankGA capabilities to find solutions within similar conditions as those found in real life. This algorithm should be able to maintain the current best individual and, at the same time, being able to escape local optima. The methodology successfully creates neuro-controllers able to keep the car in the track, to slow down to take curves, and to adjust the speed and certain gears to finish the race. However, we found that using evolution alone it is not enough to efficiently deal with situations presented in real-life driving (e.g., changing gears as needed, taking curves at high speed or avoiding collisions with other cars). Finally, we present some directives on possible future developments that could enlighten us on how to approach problems like this one. For instance, we suggest implementing a methodology that allows the neuro-controller to learn and optimize its parameters in real time.

Keywords: neural controller, rankGA multiple car-like robots.

1 Introduction

Autonomous navigation of car-like robots that travel through a given environment while avoiding fixed and mobile obstacles (such as each other) has been of great interest among researchers. Ideally, it would include both the creation of a plan of action based on the knowledge about the environment and a reactive system that allows a fast response to the changes in the environment [6]. Basically, the reactive system avoids collisions with obstacles by recognizing the environment based on the information obtained by the robot's sensors and reacts by modifying the robot's actuators (i.e. the robot controller). The reactive system can be built based on two different approaches: the classical and the cognitive approach. In the former approach, the robot controllers are manually generated based on human knowledge of the specific problem conditions. In the latter, the robots extract information from the data in order to automatically design the controller [10]. Designing a specific solution, as in classical robotics, has the disadvantage of having to think about all the factors that must be considered, and if one of them is missing, the solution would not be as efficient as it could. In contrast, in the cognitive approach, the provided solutions take into account all the information sensed by the robots. Neural Networks (NN) are widely employed in the cognitive approach to make robots navigate in different scenarios. It is expected that an NN learns by adjusting its parameters (weights) based on the acquired knowledge.

Although there are deterministic techniques (e.g., those based on gradient information) to obtain the weight values, the problem of autonomous navigation usually poses a scenario with several local optima which makes it very hard to solve by techniques based on local information. As a consequence, in the field of Evolutionary Robotics (ER) the use of Evolutionary Computation methods has been proposed [12] (e.g., Genetic Algorithms or Evolutionary Strategies) [8, 9] for obtaining a better approximation of the values of the weights that lead to a global optimal robot controller. Controllers designed this way are known as neuro-controllers. As far as we know, most of the proposals in Evolutionary Robotics are for a single car-like robot, i.e. [2, 3, 9, 13]. A review of studies which combine Neural Networks with Evolutionary Computation techniques in the ER area can be consulted in [5, 8, 12].

Controllers have been made for multiple robots with other paradigms, for example, in [1] they build a neuro-fuzzy controller for collaborative tasks, in [11] they build a neuro-fuzzy controller used for multiple robots that must avoid crashing each other. The training method is backpropagation with hand-made training data.

Simultaneous evolution means that the individuals in the population are evaluated in an environment where all of them can interact with each other. This has the advantage that individuals evolve to consider this complexity. Another advantage is that one simulation can be used to evaluate many individuals at once saving computing effort. Thus, in this work, we propose the simultaneous evolution of neuro-controllers based on an adapted version of a Rank Genetic Algorithm (RankGA) [4] to tune the Feedforward Neural Network (FFNN) weights to control multiple simulated car-like robots that compete between them.

The remainder of this paper is organized as follows. In Section 2, we present research related to the evolution of neuro-controllers of multiple car-like robots. Then, in Section 3, we explain the details of the proposed methodology. In Section 4, we present the experimental setup. In Section 5, we present key results and discussion about the experiments designed to evaluate the neuro-controllers' behavior. Finally, in Section 6, we provide some conclusions and possible future research paths.

2 Related Work

In what follows, we review the works related to neuro-controllers combining Neural Networks with Evolutionary Computation techniques to drive a single robot.

In [9], Hui and Pratihar address the problem of a car-like robot that has to find a collision-free and time-optimal path into an environment which has a few moving obstacles. They compare three approaches, In the first approach, they use a Feedforward NN trained with Back-propagation to build the NN-based controller. In the second approach, they use a Genetic Algorithm (GA) to evolve an NN-based controller, and the third one is a motion planner (Potential Field Method), which has the task of determining the acceleration and the steering angle of the robot in order to reach the target and avoid collisions. The aim is to find the controller that makes the robot able to reach the target in the lowest possible traveling time and avoiding collisions with fixed obstacles. They found that the second approach outperforms the others. Note that the problem they deal with has various simplifications. For instance, the robot entries are only two: the robot's wheels are considered to move due to pure rolling action, and the back hitting of the robot by the obstacles is neglected.

In [13], Togelius and Lucas use a genetic algorithm to train the neuro-controller of a car-like robot in a 2D simulation environment. They made neuro-controllers for different car racing tracks and reported that general proficient controllers can be obtained if neuro-controllers are evolved from scratch for one track and then, when it reaches certain proficiency, additional tracks are added to the tracks set. They did experiments with 6 tracks and obtained good results when training first with the easier tracks and then with the harder ones. The obtained controllers do not perform well in unknown tracks.

In [2], Capi and Toda also use a FFNN trained with a GA to control a real robot that has to move through a very simple path in an office. They make a preprocessing of the image captured by the robot's camera, with this preprocessing, the number of entries in the NN is reduced and so, learning is faster. However, the amount of information that enters in the NN is less, but in this case, the problem is simple, then the obtained information was enough to solve it.

In [3], Cervantes and Flores propose a GA was used to evolve a fixed topology of a FFNN for controlling a real robot which has to follow a simple path to reach its target. They propose GA operators suitable for noisy fitness functions. As in [2], the problem to solve is very simple.

3 Evolutionary Algorithms and Neural Networks - Base Scheme for Controlling Multiple Mobile Robots

In this work, we propose the simultaneous evolution of neuro-controllers based on an adapted version of a Rank Genetic Algorithm (RankGA) [4] to tone the Feedforward Neural Network weights to control multiple simulated car-like robots that compete between them.

As previously mentioned, the goal of the methodology we present is to find the most adequate neuro-controllers to control multiple simulated car-like robots that compete between them, while avoiding collisions and reach their goal position in the shortest possible time.

The methodology consists of three modules: the Evolutionary Algorithm (i.e., the RankGA) module, the Robots Controllers Module (i.e., the Feedforward Neural Network), and the Robots Module. Figure 1 illustrates the flow of information that interconnects them.

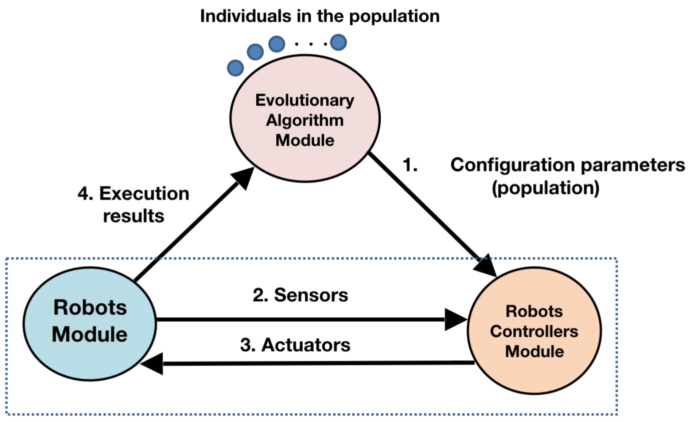
Individuals in the population
Evolutionary Algorithm Module
1. Configuration parameters (population)
4. Execution results
Robots Module
2. Sensors
3. Actuators
Robots Controllers Module

Fig. 1. Methodology: The Evolutionary Algorithm Module evolves a population of weights vectors that are sent to the Robot Controllers Module for evaluation (1) and orders the Robots Module to start. The Robots Module sends, every few milliseconds, the robots' sensors values to the Robots Controllers Module (2) which, in turn, sends the robot's actuators values back (3). Finally, when the end criterion is met, the Robots Module sends to the Evolutionary Algorithm Module the performance results of each robot in order to evaluate the population (4).

In the Evolutionary Algorithm Module (EAM) the RankGA algorithm applies mutation, crossover, and selection operations to a set $\mathbf{P}$ (the population) of individuals p_i, where $i = 1, \ldots, N_I$, and N_I is the number of individuals. Each individual p_i is a vector of real values $p_{i,j}$, where $j = 1, \ldots, N_G$, and N_G is the number of genes in the individual. The EAM sends the population $\mathbf{P}$ to the Robot Controller Module (RCM) and orders the Robots Module (RM) to start sensing and to acquire data.

The RCM uses the values $p_{i,j}$ of each individual p_i as the parameters of one of the robot controllers R_i in the Robots Module. Specifically, it uses these values as the weights of the synaptic connections of the neurons in the Feedforward Neural Network.

The objective of the EAM is to find the optimal set of weight values to achieve the best performance of a group of car-like robots. For each robot's step t, the RM sends the robots sensor's status to the RCM. After this, based on its current weights, the RCM sends the updated values to the actuators namely: acceleration, brake, clutch, steering angle, and gear. Once the termination criteria are met (e.g., the race is finished or the time is reached), the results of every robot are sent from the RM to the EAM. The above process is repeated for each generation.

In what follows we will explain all modules in further detail.

3.1 The Robots Module: TORCS

The Robots Module of the proposed methodology is designed in such a way that can be applied to various scenarios, namely, using either real or simulated car-like robots. For our study, we adopted a simulated environment for carrying out the experiments.

The Open Racing Car Simulator (TORCS) [14] is an open source simulator for car racing in 3D for multiple players. Figure 2 shows the simulation environment set for the experiments. Each car has up to 79 sensors including speed, the position of the car on a track, the angle between the car and the track axis, and 19 track sensors, to name a few. At each game tick, each car controller receives its sensory information from the TORCS server and sends back, as an answer, the computed values of five actuators: steering wheel, accelerator, brake, gear, and clutch. The description of these values is presented in Table 1.

Subsequently, the TORCS server simulates the next simulation step. This process is repeated until the race is completed. In this simulation environment the controller must be fast enough to respond to the server within the current game tick since if the server does not get an answer in time, it repeats the last seen actuators values. The most important sensors obtain information about the possible obstacles within a radius of 200 meters. There two kinds of obstacles in a race, namely: track edges and other cars. For detecting track edges, the sensors sample the space in front of the car for every $10°$ angle, spanning clockwise from $-\pi/2$ up to $+\pi/2$ with respect to the car axis (see Fig. 3. Similar to track sensors, for detecting other cars, the controller receives information from 36 sensors that detect cars within the same distance. The only difference is the covered range that goes from $-\pi$ to π. These collision sensors and others are briefly described in Table 3.

It is worth to mention that TORCS is implemented as a client/server architecture. The server is the process that carries out the simulation and obtains the current values of the sensors of each car in the race. On the other hand, each robot plays the role of the client, so that it only receives the values of its own sensor values and sends to the server the actuator values computed somehow.

Fig. 2. TORCS: The simulation environment.

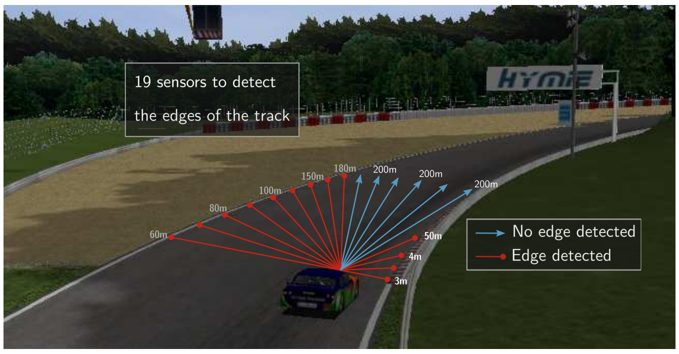

Fig. 3. Vectors for detecting track edges within a radius of 200 meters.

Table 1. Actuators adopted as outputs for the neural network.

Actuator	**Description**	**Range**
Steering Wheel	Angle of the steering wheel. -1 and 1 represent 20° to the right and to the left, respectively.	$[-1, 1]$
Accelerator	The gas pedal. A value of 0 means no gas, while 1 full gas.	$[0, 1]$
Brake	The break pedal. A value of 0 means no brake, while 1 full brake.	$[0, 1]$
Gear	The selected gear. -1 means reverse, 0 neutral, while remainder values engage the other gears.	$[-1, 6]$
Clutch	The pedal clutch. A value of 0 means do not press the clutch, while 1 full clutch.	$[0, 1]$

Table 2. Sensors adopted as entries for the neural network.

Sensor	**Description**	**Range**
Angle	Angle between car direction and the tangent line of current segment of the track.	$[-\pi, \pi]$
TrackPos	Position between the car and the track axis. 0 means that the car is in the center, -1 means the car is the right edge, +1 means the car is in the left edge, and values beyond -1 and 1 means the car is outside the track.	[-1,1]
Track	Values of 19 sensors which measure the distance between the edge track and the car in a range of 200 meters.	$[-\pi/2, \pi/2]$
Gear	Current gear of the car.	[-1,. . . ,6]
SpeedX	Current speed along axis of abscissas.	[0,1]
SpeedY	Current speed along axis of ordinates.	[0,1]
Opponents	Values of 36 radial sensors at intervals of $10°$ which measure the distance to the closest opponent in a range of 200 meters.	[0,200]

3.2 The Evolutionary Algorithm Module

We use an adapted version of the RankGA [4] to find the optimal configuration of the robot's neuro-controller. The reason to use this algorithm is that it has been proven [4] to outperform a simple GA in difficult fitness landscapes where it is necessary to have a good balance between exploration and exploitation. This balance is needed when the fitness landscape presents many local optima and where a modular solution can be constructed by evolution. Also, each module's solution is hard to find in such difficult problems. The problem in this study is very hard and complex, so we expect it to have many local optima. It is also expected to have some sort of modular structure because of the various tasks that need to be performed simultaneously such as stay in the track and avoid collisions with objects while being fast. Thus, the Rank GA seems to be a plausible algorithm for the problem.

Here we use an adapted version of the algorithm in [4] to floating point genes. In this algorithm, each individual's genotype is a set of all the Neural Network weights as floating point numbers. Individuals are initialized randomly. The individuals of the population are ranked from best to worst in terms of their fitness before each of the genetic operators are applied.

In this study, the main goal is the time to finish the race. However, when the time limit runs out or the car gets stuck, the simulator returns zero for the race time. Therefore, several solutions might have zero time, although some of the cars get closer to the finish line than others. Thus, in order to rank solutions, we used a lexicographic order where the first criterion is the distance to finish the race and the second one is the lap time.

Then, the application of these genetic operators depends on the rank of each individual in the population. The top-ranked individuals tend to stay unchanged while others tend to vary increasingly with their rank trying to escape from local optima.

Algorithm 1 RankGA.

```
procedure MAIN
    initialization
    Evaluation and Sort
    while not end Criteria met do
        RankSelection
        RankRecombination
        Evaluation and Sort
        RankMutation
        Evaluation and Sort

procedure RANK SELECTION
    clones ← null
    for $i$ in $[0, \ldots, N_I - 1]$ do
        $r_i \leftarrow i/N_I$
        $N_C \leftarrow \lfloor K(1 - r_i)^{(K-1)} \rfloor$                ▷ Note the $\lfloor x \rfloor$
        for $j$ in $[0, \ldots, N_C - 1]$ do
            $clones.add(\mathbf{p}_i)$                        ▷ $\mathbf{p}_i$ is cloned $N_C$ times
    $i \leftarrow 0$
    while $clones.size() < N_I$ do
        $r_i \leftarrow i/N_I$
        $N_C \leftarrow K(1 - r_i)^{(K-1)}$                   ▷ without $\lfloor x \rfloor$
        $f \leftarrow N_C - \lfloor N_C \rfloor$              ▷ $f$ is the fractional part of $N_C$
        if $random(0, 1) < f$ then
            $clones.add(\mathbf{p}_i)$                        ▷ extra clone of $\mathbf{p}_i$
        $i \leftarrow (i + 1) \bmod N_I$
    $\mathbf{p} \leftarrow clones$                            ▷ replace population
    Sort

procedure RANK RECOMBINATION
    for $i$ in $[0, \ldots, N_I - 2]$ step 2 do               ▷ for each even individual
        for $g$ in $[0, \ldots, N_G - 1]$ do                  ▷ for each gene
            if $random(0, 1) < 0.5$ then                      ▷ uniform crossover
                Swap($\mathbf{p}_{i,g}$, $\mathbf{p}_{i+1,g}$)        ▷ mating $\mathbf{p}_i$ with $\mathbf{p}_{i+1}$

procedure RANK MUTATION
    $c \leftarrow ln(N_G)/ln(N_I)$
    for $i$ in $[0, \ldots, N_I - 1]$ do
        $r_i \leftarrow (i/N_I)^c$
        for $g$ in $[0, \ldots, N_G - 1]$ do
            $\mathbf{p}_{i,g} \leftarrow (1 - r_i)\mathbf{p}_{i,g} + (r_i)gaussian(0, R)$
```

The Rank of an Individual The individuals in the population are sorted from best to worst. Then, for the i-th individual, where $i = 0, \ldots, N_I - 1$, its rank is given by

$$r_i = i/N_I. \tag{1}$$

Rank Selection Individuals are given a number of instances (clones) for the next generation. The number of clones N_C for an individual i with rank r_i is given by

$$N_C(r_i) = K(1 - r_i)^{K-1}, \tag{2}$$

where K is the population's selective pressure. This N_C value is separated in its integer and fractional parts. The integer part determines directly a minimum number of clones of an individual. The fractional part determines the probability to produce an extra clone of that individual. Random numbers between 0 and 1 are drawn for each current population individual to check if an extra clone will be created cyclically until the total number of clones equals N_I in the original population.

Rank Recombination Mating is made between contiguous individuals in the fitness sorted list of individuals, i.e., between similarly ranked individuals.

Rank Mutation Each gene of an individual with rank r_i, being a real number without bounds, is modified as follows

$$p_{i,j}^{(t+1)} = (1 - r_i^c)\, p_{i,j}^{(t)} + r_i^c\, Gaussian(0, R), \tag{3}$$

where

$$c = \frac{\ln(N_G)}{\ln(N_I)}, \tag{4}$$

R is the reach (as a standard deviation) of random mutation, and t is the current generation.

3.3 The Robot Controller Module: Feedforward Neural Network

As mentioned before, we employed an NN as the robot's controller. For this study, we used a three-layered fully-connected Feedforward NN with a single hidden layer, as shown in Figure 4.

In this FFNN we adopted a sigmoid activation function for each neuron. Besides, the neurons of the input layer are connected to each of the robot sensors (see Table 2), and the five neurons of the output layer are connected to the robot actuators (Table 1) to be applied in the next simulation step. In the output layer, neurons y_j, where $j = 1, \ldots, 5$, have the following output:

$$y_j = \text{sigmoid}\left(\sum_{i=1}^{N_H} w_{ij}^{y} h_i\right), \tag{5}$$

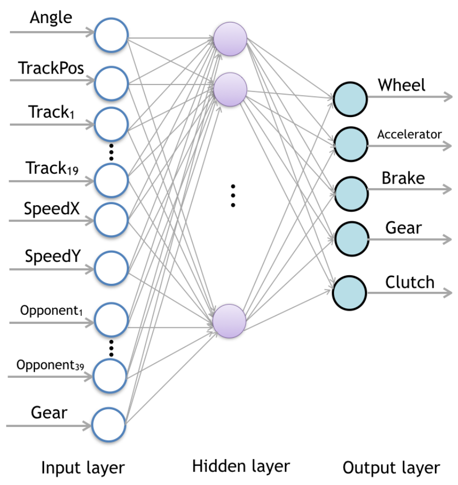

Fig. 4. Feedforward Neural Network. The input layer has 60 nodes representing the sensors: speed, the position of the car on a track, the angle between the car and the track axis, and 19 track sensors, gear, and 36 opponents values (see Table 2 for further details about the sensors). The hidden layer has 10 nodes. The output layer has five nodes representing the: steering wheel, accelerator, brake, gear, and clutch (see Table 2 for further detail about the actuators).

where N_H is the number of neurons in the hidden layer (including one neuron that produces constant output 1, w^y_{ij} is the weight of the output h_i of the hidden layer for the j-th neuron in the output layer. Also, the outputs h_j in the hidden layer are given by

$$h_j = \text{sigmoid}\left(\sum_{i=1}^{N_S} w^h_{ij} x_i\right), \tag{6}$$

where N_S is the number of input sensors, $w^h_{i,j}$ is the weight of the sensor value x_i for the j-th neuron in the hidden layer.

In TORCS each robot is implemented as a different client that needs to be connected to the Robots Module, which plays the role of the server. For instance, if we have four controlled cars, that means that there are four different clients connected to the Robots Module. Each of these clients only receives the values of its own sensor values and should send back the computed actuators values. On the other hand, the specific set of weights for each client are received from the Evolutionary Algorithm Module.

The values for the accelerator, brake and clutch (i.e., outputs y_1, y_2 and y_3) are already in proper range [0,1]. However, since the Robots Module expect values in other ranges for the other two actuators, we adjust the corresponding outputs in the following way:

– For the steering wheel angle, $y_4 = 2 \times \text{sigmoid}(\Sigma) - 1$, to obtain a value in the range $[-1, 1] \in \mathbb{R}$.
– For the gear, $y_5 = \text{round}(7 \times \text{sigmoid}(\Sigma) - 1)$, to obtain a value in $[-1, 6] \in \mathbb{Z}$.

4 Experimental Design

In this section, we report on the design of experiments to test the performance of the methodology introduced in the previous section.

As our first concern was to discover if the obtained neuro-controllers are capable of autonomously driving following the track and taking the curves without any collision, we used the A-Speedway race track. This race track is a free-obstacle oval with a length of 1.9km with no slopes (see Figure 5).

As previously explained, each car has up to 79 sensors, however, given the flat track, we are using for the experiments we only employ 60 of them since the information for this type of sensors is irrelevant in this scenario. For instance, in this first experiments, our main concern was to test the ability of neuro-controllers to follow the track and avoid opponents. In these conditions, sensors values like current lap time, fuel or car's elevation over the surface are not necessary to achieve our goals.

Regarding the actuator values, given that in preliminary experiments we noticed that it is hard for the RankGA to find an adequate value for the clutch actuator, we decided to set its value to zero.

On the other hand, we fixed some parameters for both the RankGA and the NN algorithms of the EAM which can be seen in Table 3. In contrast to the usual setting employed in benchmark problems, in the experiments we used only 40 generations for each configuration because of the simulation time; a typical run takes around two hours using a computer with 16 cores. For the RankGA and the FFNN, we used parameter values based on our experience in a similar work [7].

Notice that, as previously stated, in this study the objective function to be optimized by the RankGA is the time to finish the race.

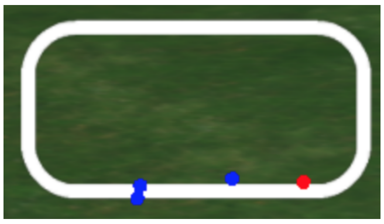

Fig. 5. The A-Speedway oval racetrack with length of 1.9Km.

Table 3. Parameters used for simulation.

Description	Value
Population size N_I	80
Number of Generations	40
Crossover Rate	1.0
Reach R of random mutation	50
Population's selective pressure K	3
No. of hidden neurons N_H	10
No. of input neurons(sensors) N_S	60
No. of simulations	6400
No. of car-like robots	1,2,4,8,16

5 Results and Discussion

In this section, we report and discuss the design and results of our experiments. The goal is to test the performance of our methodology: *i)* when varying the number of car-like robots in a single race, and *ii)* avoiding obstacles.

It is worth to note that each car-like robot is a neuro-controller (i.e., an individual of the population).

In order to simultaneously evolve the neuro-controllers for multiple car-like robots, we take groups of individuals from the population until all of them are evaluated. For instance, in a race with four cars, the first four best individuals are enrolled in a race. This way, while each car is being evaluated, it also plays the role of an obstacle for the opponents. Subsequently, the following four solutions are taken from the population, and so on, until the entire generation is evaluated.

The parameters to rank the individual's performance are the following: *i)* the distance remaining to finish the competition, and *ii)* the time to complete the race.

The results of the experiments are summarized in Table 4. The table shows the time and distance remaining to finish the competition and the damage of the car after completing the race. Although the latter value is not taken into account by the RankGA to evaluate individuals, it helps to explain the kind of controller obtained. Finally, the last column indicates the number of individuals that finished the race.

The damage is related to finishing the race because when the maximum damage is reached, the cars are removed from the race. On the other hand, for each simulation, there is a maximum time to finish the race in order to avoid waiting indefinitely for cars that get stuck. Therefore, even if a car is not damaged enough, it might not finish the race because the time limit ran out.

Judging by the number of cars finishing the race, it is notable that the complexity seems to increase as the number of simultaneous cars in a single simulation increases. In particular, for races with 8 and 16 car-like robots, the RankGA was not able to evolve a neuro-controller that finishes the race, while for 8 cars, only one neuro-controller could manage to complete the race, although with a great damage in the car. This behavior does not mean that

Table 4. Performance evaluation.

No. of robots	Time (seg)	Distance to finish (km)	Damage	No. of solutions finishing the race
1	59.018	0	764	8
2	54.818	0	796	10
4	63.318	0	19	4
8	NA	1.465	5951	0
10	59.164	0	6924	1
16	NA	1.612	6	0

the RankGA is bad for the task. Since the task becomes more complex with more cars on track, it is expected to have worse results. In order to solve the more complex task, we need to provide the system with more power by either putting more intermediate neurons in the controllers or by giving the algorithm more evaluations or both. Also, the fact that only a few individuals are able to complete the race is an expected result. The design of the RankGA is such that it always works with some very bad individuals because there are always randomly generated individuals who are there to provide exploration while only a few provide exploitation of the good genes that have been found.

In order to evaluate the obstacle avoidance, we manually selected the best individual of a race and used it as the unique neuro-controller of eight cars. It is noticeable that such individual was the one that finished the race without crashing because it was always the leader (i.e., it never found any car in its way). As a result, the individual did not learn to avoid obstacles. Thus, when facing this new scenario, the cars did not move or they crashed between them. This problem is expected to be solved by evaluating individuals all together in a single race with the best ones starting the race last. Thus, if they can't overtake without crashing, they would not be preferred by selection.

In order to appreciate the behavior of the kind of neuro-controllers obtained by our methodology, we recorded the values of the four active actuators (remember that the value of the clutch is fixed to 0), namely, wheel angle, acceleration, brake, and gear. Figures 6–8 show the values of each of these actuators at each simulation tic for the experiment in only one race.

From Figure 6 we can observe the moments in which the car turns left in each of the four curves of the track (i.e., the four peaks in the plot). Notice that a positive value means to turn left, while a negative value to turn right.

On the other hand, as we can see in Figure 7, the actuator value is positive all the time, which means the car keeps accelerating.

One interesting thing to note is that the neuro-controller of the car never brakes during the race, as seen in Figure 8. However, as seen in Figure 9, in order to decelerate to take the curves, the neuro-controller decides to change from 3^{rd} gear to 2^{nd}. Then, the car returns to 3^{rd} gear. This fact can be seen in the figure where the vertical lines denote the start of each curve of the track. The change to 2^{nd} gear is done about the middle of the curves.

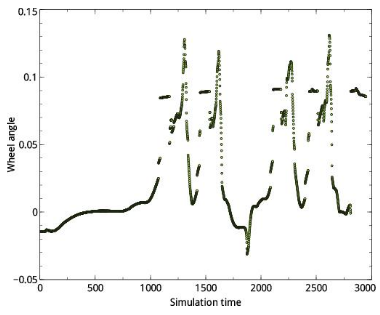

Fig. 6. Values of the wheel angle actuator during the race.

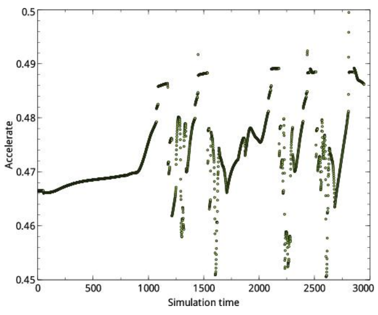

Fig. 7. Values of the accelerator pedal actuator during the race.

Finally, even we have achieved an important progress, there is still room for improvement. First, the methodology is based purely on evolution, i.e., it does not learn during the simulations. That is, once the neural networks are trained, their weights are not modified in real time when used. Thus, the information obtained by each neuro-controller during the race (i.e., the consequences of their actions) is not incorporated into the model.

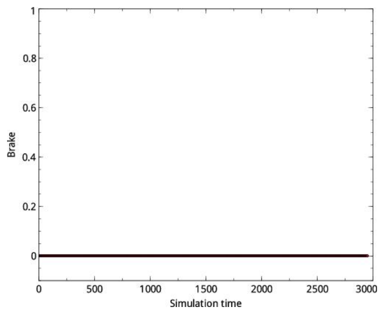

Fig. 8. Values of the brake pedal actuator during the race.

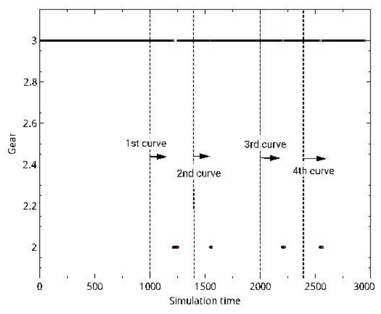

Fig. 9. Values of the gear selection actuator during the race.

6 Conclusions and Future Work

In this paper, we presented a methodology that allows the simultaneous evolution of neuro-controllers for multiple car-like robots. The methodology consists of three modules: the Evolutionary Algorithm module based on the RankGA, the Robots Controllers Module based on a Feedforward Neural Network, and the Robots Module, where the TORCS simulator was used for the experiments.

The advantage of simultaneously evolving the resulting neuro-controllers is that the simulations are faster. Regarding the achieved results, we observed that the neuro-controllers were able to properly handle the wheels to move forward and take curves. They also learned to change the gear and the accelerator pedal to both reduce or increment the speed.

However, further work is needed to produce satisfactory performing neuro-controllers. For instance, the neuro-controllers developed in our methodology did not learn to avoid obstacles efficiently. It seems to be very difficult to get the adequate weights for the Neural Network to learn to drive on the track as fast as possible while trying to avoid a crash with other cars. We think that incrementing intermediate neurons in the Neural Network could help. Also, running the evolutionary algorithm for more generations should produce better results.

As future work, we suggest implementing a methodology that allows the neuro-controller to learn and optimize its parameters in real time. Additionally, it would be interesting to train them for each type of tracks (i.e., to have specialists in each terrain).

Finally, it would be recommended to include the damage allowed as a constraint to improve the performance.

Acknowledgements. The authors would like to thank Rodolfo René Suárez Molnar for providing us with institutional support and Sergio Hernández for his feedback.

References

1. Awad, H.A., Koutb, M.A., Al-zorkany, M.A.: Multiple Mobile Robots Navigation in a Cluttered Environment using Neuro-Fuzzy Controller. In: Abraham, A., Dote, Y., Furuhashi, T., Köppen, M., Ohuchi, A., Ohsawa, Y. (eds.) Soft Computing as Transdisciplinary Science and Technology. pp. 893–903. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
2. Capi, G., Toda, H.: Evolution of neural controllers for robot navigation in human environments. Journal of Computer Science 6(8), 837 (2010)
3. Cervantes, J., Flores, D.: Rank based evolution of real parameters on noisy fitness functions. In: 10th Mexican International Conference on Artificial Intelligence, MICAI 2011. pp. 72–76. Special Session (2011)
4. Cervantes, J., Stephens, C.R.: Limitations of existing mutation rate heuristics and how a rank GA overcomes them. IEEE Transactions on Evolutionary Computation 13(2), 369–397 (2009)
5. Chin, K.O., Teo, J.T.W.: Evolution of RF-signal cognition for wheeled mobile robots using pareto multi-objective optimization. International Journal of Hybrid Information Technology 2(1), 31–44 (2009)
6. Dudek, G., Jenkin, M.: Computational Principles of Mobile Robotics. Cambridge University Press, New York, NY, USA, 2nd edn. (2010)
7. Flores, D., Cervantes, J.: Rank based evolution of real parameters on noisy fitness functions: Evolving a robot neurocontroller. In: 10th Mexican International Conference on Artificial Intelligence. pp. 72–76 (2011)

8. Gupta, S., Singla, E.: Evolutionary robotics in two decades: A review. Sadhana 40(4), 1169–1184 (Jun 2015)
9. Hui, N.B., Pratihar, D.K.: Neural network-based approaches vs. potential field approach for solving navigation problems of a car-like robot. Machine Intelligence and Robotic Control 6(I), 29–60 (2004)
10. Nguyen-Tuong, D., Peters, J.: Model learning for robot control: a survey. Cognitive Processing 12(4), 319–340 (2011)
11. Pradhan, S.K., Parhi, D.R., Panda, A.K.: Neuro-fuzzy technique for navigation of multiple mobile robots. Fuzzy Optimization and Decision Making 5(3), 255–288 (Jul 2006)
12. Pratihar, D.K.: Evolutionary robotics-A revew. Sadhana 28(6), 999–1009 (2003)
13. Togelius, J., Lucas, S.M.: Evolving robust and specialized car racing skills. In: IEEE Congress on Evolutionary Computation, CEC 2006. pp. 1187–1194 (2006)
14. Wymann, B., Dimitrakakisy, C., Sumnery, A., Guionneauz, C.: The Open Racing Car Simulator (jun 2018), http://torcs.sourceforge.net