A human-like TORCS controller for the Simulated Car Racing Championship

Jorge Muñoz, German Gutierrez, Araceli Sanchis

Abstract— **This paper presents a controller for the *2010 Simulated Car Racing Championship*. The idea is not to create the fastest controller but a human-like controller. In order to achieve this, first we have created a process to build a model of the tracks while the car is running and then we used several neural networks which predict the trajectory the car should follow and the target speed. A scripted policy is used for the gear change and to follow the predicted trajectory with the predicted speed. The neural networks are trained with data retrieved from a human player, and are evaluated in a new track. The results shows an acceptable performance of the controller in unknown tracks, more than 20% slower than the human in the same tracks because of the mistakes made when the controller tries to follow the trajectory.**

I. INTRODUCTION

The *Simulated Car Racing Championship* is a car racing competition where each participant has to submit a controller to drive a car. This competition is carried out in *The Open Racing Car Simulator* (TORCS) [1], a very realistic simulator with a sophisticated physic engine and several different cars and tracks, with their own features. TORCS takes into account aspects as the fuel consumption, the aerodynamics of the car, the collisions or the grip of the wheels, among other things.

As other competitions, the *Simulated Car Racing Championship* is growing each year increasing its difficulty and the skills of their participants. Although every year the rules are modified to increase the difficulty, the participants send better controllers. These controllers are really good, even better than the controllers included in TORCS, which have more information about the state of the game than the controllers of the competition due to its specifications. But, despite of the improvement in the submitted controllers each year, a skilled human player outperforms the results of any non-human controller, since reach the level skills of a good human players is incredibly complex. This is the reason why the games usually cheat on their own bots to simulate the same level skills as the human players.

Our goal is to create opponents with the same level skills of the human players. This can be done by means of collecting data of the human player first and then training an algorithm that can learn the human behaviour. We decided to use neural networks [1] to learn the trajectories followed by a human in different tracks and the speeds the human reaches in each position in the trajectory. We call this an *indirect controller* because the sensors information is not matched directly with the effectors, but these information is processed first, included in the state of the simulation and then a scripted policy decides the values of the effectors to achieve the objectives. This is, to follow the trajectory with the speed given by the neural networks. It is not a reactive controller but a deliberative controller. We will discuss about our election of an indirect controller in Section IV.

This paper structure is as follows: Section II references some related work on car racing games that tries to imitate some behavior. Section III explains the main features and rules of the *Simulated Car Racing Championship*, which information is provided by the application programming interface (API) of the competition and how is the car controlled. In Section IV we talk about the differences of the direct controller and the indirect controller and why we choose the latter. The detailed explanation of the controller is done in Section V where we talk about the trajectory and speed prediction, the policy used to follow the trajectory with the given speed and other issues of the controller. Section VI shows the experiments and their results to check the validity of the controller and in Section VII this is discussed. Finally, we talk about future works and improvements in Section VIII.

II. RELATED WORK

Imitate the human behavior is a very interesting researching topic, but very few times it has been applied to comercial games. One of the most popular commercial video game where the player is able to create non-player characters (NPC), that drives like him and can replace him in the game, is the famous *Forza Motosport* [2] for the *Microsoft XBox*. The drivers created by learing the player drive-style are called *Drivatars* (a complete description of how they works can be found in [2]). The behavior model that Drivatars build to imitate the human player is based on a statistical model of how the human drives in the different segments of the track. With this statistics model, when a similar segment in another track is found, a trajectory is created based on the model and followed by the car. The way the drivatar can follow the trajectory is done by increasing the grip of the car, this is a cheat but as the drivatar follow a human-like trajectory this fact is not noticed by other human players.

Other games where a human data has been used to train an algorithm to drive are the *Colin McRae Rally 2.0* [3], [4] (which uses neural networks to drive) and *Motocross The*

J. Muñoz, G. Gutierrez, A. Sanchis are with the Computer Science Department, Universidad Carlos III de Madrid, Avda. de la Universidad 30, 28911 Leganés, Spain (emails: { jmfuente, ggutierr, masm }@inf.uc3m.es).

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

[2]http://research.microsoft.com/en-us/projects/drivatar/forza.aspx

Force [5]. But in the last game the tracks are designed in a way that a bad controller does not crash but it makes worse lap time.

Other researches have done imitation learning, but in these both cases the controller to be imitated was not a human. In [6] the author adds a new sensor — which is not present in the *Simulated Car Racing Championship* — with information about how are next segments in the track. With this new sensor the authors uses different learning methods as *neural networks*, *case base reasoning* (CBS) and *k-nearest neighbour* to learn the speed and position of the car in the segment. Then a scripted policy is used to drive based on the the target speed and target position, it is a indirect controller. The results shows good results but slower than the original controllers.

In [7] the author collects data from a bot controller and then train a neural network to imitate it. It is a direct controller where the outputs of the neural network are the values for the effectors of the car and the inputs are the sensors information. When the authors try the same learning method for a human driver the results shows that the NPC created can drive as the human in the easy parts of the track, but when it tries to steer in a complicated turn the car crash.

The authors of [8] use a *multi-objective evolutionary algorithm* to evolve neural networks with a human-like driving style at the same time the controller is competitive (reach the longest distance in the evaluations). The results show that drive like a human is a very hard problem.

Other work where the authors also try to model the human behavior with the aim of create more enjoyable tracks is [9].

For the *Simulated Car Racing Championship* different controllers have been submitted [10], [11], [12], [13], [14]. The controller proposed in [10] is a control architecture developed for the competition where the parameters has been optimized with an optimization algorithm. In [11] the authors also propose another architecture but in this case they use fuzzy sets to control the car. Fuzzy sets are also proposed by [13] and [12] although in the latter the fuzzy sets and rules are evolved with a genetic algorithm. In [14] the *neuroevolution of augmenting topologies* (NEAT) [15] is used to create the controller.

III. Simulated Car Racing Championship

A complete description of the *Simulated Car Racing Championship* is located in [16] and further information can be found in the competition web page [3], group [4] and past competitions [17]. We describe in this section only the most relevant aspects of the competition for our controller.

The *Simulated Car Racing Championship* is divided in 3 legs with different kind of tracks. In each leg the car runs in 3 unknown tracks, they can be tracks of the game or new tracks. For each track the competition is divided in three stages: warm-up, qualifying and the race.

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

[4]http://groups.google.com/group/racingcompetition

During the warm-up, the cars are allowed to run during 100000 game ticks (approximately 30 minutes of actual game time). The idea of the warm-up is to set up the parameters of the car and adapt it to the track features. We calcule that in an average track each lap is performed in 1 or 2 minutes for a good controller, this gives us between 15 and 30 laps to set up the car.

The first important stage is the qualifying. The controllers have 10000 game ticks (around 3 minutes and 20 seconds) to run alone in the track and reach the longest distance. The 8 best controllers in the qualifying run together in the race.

In the race the controllers are scored as in the Formula 1. For our work the race is no relevant due to our current work only focus in a controller that can drive as a human does, without other cars in the track at the same time. But we will talk about how the opponents could be included in the controller in Section VIII-A.

In the nexts Section, we describe the sensor information given by the API and the effectors to control the car, but only those ones that are relevant for our controller.

A. Sensors information and effectors

The information provided by the API is related with the lap, the status of the car, the track and the opponents. For us, the most important information is:

- Current lap time
- Distance from the start line
- Damage and fuel of the car
- Current gear and revolutions per minute (R.P.M.)
- Relative position of the car with the center of the track
- Spin of the wheels
- Current speed, lateral speed and vertical speed.
- 19 sensors to know the distance between the car and the limits of the road (range finders)
- 36 sensors of proximity for the opponents (useful for future versions of our controller)

Two differences of the *2010 Simulated Car Racing Championship* with previous years is that white noise has been included in the range finders and proximity sensors, and the angles of the range finders can be fixed by the controller at the initialization.

The effectors to control the car are:

- the accelerator
- the brake
- the steering
- the gear
- the clutch
- the value for the focus sensor (not used by our controller)

IV. Direct controller VS Indirect controller

We can divide the type of the controllers in two classes: the *direct controllers* and the *indirect controllers*. In our case that we are using neural networks to control the car, the indirect controller would be a direct matching between the input sensor information and the output effectors value, this is, the algorithm would learnt which are the right values in the effectors given a concrete sensor values. In the indirect controller the algorithm would process the information of the sensors first, then creates some kind of model of the world and objectives, and finally sets the proper values in the effectors to achieve its objectives

The problem with a direct controller is that all the noise in the sensors values as the range finders is directly applied into the effectors, and this could lead to some kind of instability in the control of the car. Other problem is related with the training of a neural network of this kind with human data. As the human player does not perform two laps in the same way, even does not perform the same actions with the same sensor information, there is a lot of noise in the data that avoids the network to learn how to drive [7]. So, our solution is use a indirect controller. First we create a model of the track, them we calculate with the model and the current sensor information where is the car in the track, and finally, we use an scripted policy to follow a trajectory. This trajectory is predicted by a neural network trained before with human data. Our controller does not train a neural network to control but trains it to predict the trajectory the human will follow given some information about the track.

V. Controller

The main goal is to create a controller able to drive as a human player, this is to imitate the behavior of the player. As the human player has a knowledge about all the track we need some kind of information about the track that is not given by the sensors information. For instance, with the current sensor information we do not know if the next turn is to the left or to the right or if car is in a right turn now or in a straight. We need this kind of information to imitate the human behavior. So the first step of our controller is to create a model of the track. This is explained later in Section V-B.

Once we have a model of the track we can use it to predict the actions the human will made in a concrete point of the track. We know that for car racing the humans try to follows trajectory to optimize the turns. Also remember the maximum speed they can reach in each turn and where need to start braking to keep the car on the trajectory. So our controller need some kind of predictions to calculate the trajectories and the speed of each points. We used for this predictions neural networks. Specifically we use 4 neural networks to do this, a further description is done latter in Section V-C for the trajectory prediction and Section V-D for the speeds prediction.

In order to follow the predicted trajectory with the predicted speed, we have implemented a scripted policy that takes into account the current error and sets the proper values in the steering, accelerator and brake to correct it. Section V-E explains how the steering is set and Section V-F describes how are the accelerator and brake set.

Notice that our work seems like the work done with the Drivatars in *ForzaMoto Sport* [2] and like in [6]. The differences are that we do not have so many information about the track to drive the car as in both works, we do not cheat to set the grip value as in former and we use human driver in order to learn not like in the latter.

A. Scripted aid policies

We have programed some scripted policies to aid in the gear change and to avoid the skids. In concrete, we have done an automatic gear change, an automatic control for the clutch, a traction control system (TCS) and a anti-lock braking system (ABS). All these scripted aid policies are used for the controller and for the human when we collect the data.

The gear change is an easy problem that could be solve by using a table with values that points out when it has to increase or decrease a gear. Table I shows the R.P.M. and the velocities we have use to increase or decrease a gear. For instance, if the car is in gear 2 and the R.P.M. are higher than 8900 or the speed is higher than $80km/h$ the gear is set to 3. We also include a restriction that it has to elapse 40 game ticks before change again the gear. This is done in order to avoid instability in the gear change.

TABLE I
Gear change policy

Current	R.P.M.		Speed km/h	
Gear	Gear Up	Gear Down	Gear Up	Gear Down
Reverse	-	3000	-	30
Neutral	5000	-	0	-
1	8500	3000	30	-
2	8900	5000	80	30
3	9000	6000	130	80
4	9000	7000	180	130
5	9000	7500	230	180
6	-	8000	-	230

In order to control the clutch, each time the gear changes the $clutch$ value is set to 0.5 and $clutchDec$ to 0.01. When the $clutch$ value is greater than 0 the value is update as in Equation 1, also the $clutchDec$ value is update as in Equation 2.

$$clutch = clutch - clutchDec \quad (1)$$
$$clutchDec = clutchDec \cdot 1.3 \quad (2)$$

The ABS is activated when the brake is activated and the current speed is bigger than $3m/s$. The $brake$ value us updated as in the Equation 3, where $speed$ is the current speed in m/s, $slip$ is the velocity of the wheels in m/s, and $absSlip$ and $absRange$ are two adjust values equals to 2.8 and 6.0.

$$brake = brake - \frac{(speed - slip) - absSlip}{absRange} \quad (3)$$

The TCS is only activated when the current speed is lower than $28m/s$, the car is accelerating and the difference of the speed of the rear wheels ($rearSp$) with the front wheels ($frontSp$) is greater than 3.0. The $accelerate$ value is update as in Eqcuation 4, where $tcsSlip$ and $tcsRange$ are two adjust values equals to 3.0 and 6.0.

$$accelerate = accelerate - \frac{(rearSp - frontSp) - tcsSlip}{tcsRange} \quad (4)$$

A recovery policy has been created to return the car to the road when it goes out. This process is not described here since is not relevant for the paper.

B. Track model. Exploration

The track model is created during the competition in the warm-up stage and before collecting the data for the experiments. In the competition we have around 30 minutes in the warm-up, in this time we need to perform a complete lap without going out of the track to get enough information to create the model of the track.

The way in which the model is built by means of storing information about the distance from the start line of the car, the angle with the track, the distance of the car to the center of the track, the distance of the car with the track edges and the distance the rear wheels cover between two game ticks (for this the spin value of the wheels have been used). We store all this information for each game tick. It is very important that the car never skids or jumps because it would mean that wrong information has been stored and a wrong model of the track will be created. In order to avoid this problems the controller that makes the exploration to collect this data drives in a low speed, without accelerating or braking abruptly, and trying to stay in the middle of the track. Notice that the distance of the car with the track edges is measure by the range finders which have noise. To reduce this noise we use 7 range finders in each lateral of the car to measure the distance with the track edges, the average value is the one used.

In order to create the model we also need the distance between the wheels, we do not know this value but we set it as 1.94 meters as is the common value for the most of the cars in TORCS. Now we are going to explain how the track model is created. We start setting initial points for the wheels, the middle of the track and the track edges. We know the distance covered by the wheels (L_1 and L_2) between two game ticks and we also know that the trajectory of the both wheels must follow two differents circles with the same center in the line that join both wheels and radius which difference is the separation between the wheels, then we can calculate the new points of the wheels. See Equation 8 to know how is the radius calculated and Figure 1 which shows the update of the wheels: points *A* and *B* are the initial points of the wheels, points *D*, *F* are the final points of the wheels and point *C* is the center of the circles, L_1 is the distance between *A* and *D*, L_2 the distance between *B* and *E*, $angle$ is the angle among *A*, *C* and *D*, r_1 is the distance between *A* and *C* and r_2 is the distance between *B* and *C*.

$$L_1 = angle \cdot r_1 \quad (5)$$

$$L_2 = angle \cdot r_2 \quad (6)$$

$$r_1 = r_2 - 1.94 \quad (7)$$

$$r_1 = \frac{1.94}{1 - \frac{L_2}{L_1}} \quad (8)$$

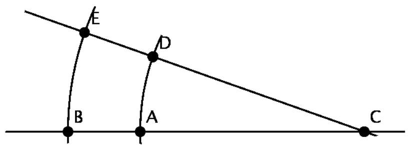

Fig. 1. Calculus of the new wheel positions

Once we have the next points of the wheels it is very easy calculate with some trigonometry the new points of edges of the tracks with the rest of the stored information: the angle with the track axis, the distance to the edges of the tracks and the distance of the car with the track center. We continue with the next stored information about the next point until we do not have more points. And finally we have a complete description of the track. Next step in simplify the track model we have. We do not need all the track points we get. We decided to use points separated 4 meters to create the model of the track, this is a tracks divided in segments of 4 meters. We try more distances than 4 meters but with this value we get good models of the tracks. Figure 2 shows the track model divided into segments, black lines are the edges of the track and dotted gray lines divide the segments.

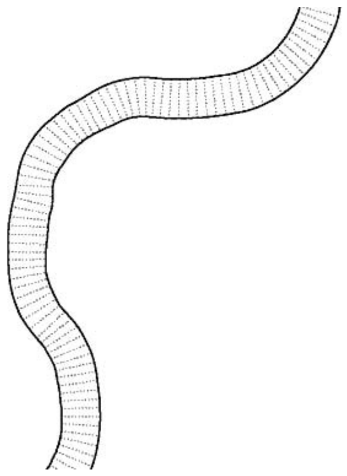

Fig. 2. Detail of a track segmentation. Dotted line divide the segments of the track.

The final information that is stored in the model of the track is the angle between the segments and their width.

The tracks used in the experiments and their corresponding models can be shown in Figure 3. We can see in the figure that the models of the tracks seem similar to the original tracks but they are not closed. This is due to the process of segmentation and the mistakes in the stored data when the car complete the lap.

C. Trajectory model

To create a trajectory model based on the human data we trained two neural networks. Both networks have the same inputs but the output of the first network is the predicted position of the car in the current segment (from here when we refer to a segment we mean a segment of the track model described in the previous section), and the output of the second network is the difference in the position of the car between the current segment and the last segment. The reason to use two networks is to reduce the noise and soft changes in the trajectory.

The target position in the track is calculated as shows the Equation 9, where t_p is the target position, p_p is the predicted position by the neural network, p_{dp} is the predicted difference position by the other network, t_{p-1} is the target position of the previous segment and α is parameters equals to 0.5.

$$t_p = \alpha \cdot p_p + (1 - \alpha) \cdot (t_{p-1} + p_{dp}) \tag{9}$$

The inputs of both networks are 68:

- the angles of the next 50 segments
- the angles of the last 15 segments
- the track width
- the next turn (0 for left, 1 for right and 0.5 for a straight)
- the current turn (as next turn)

Figure 4 shows an example of predicted trajectory. The gray line is the predicted trajectory.

D. Velocities model

For the velocities model we use a similar process than for the trajectory model. We also use two neural networks, the first one predicts the speed and the second one the difference in the speed in the current segment with the last segment.

Target speed is calculated as shows the Equation 10, where t_s is the target speed, p_s is the predicted speed by the neural network, p_{ds} is the predicted difference speed by the other network, c_s is the current speed of the car and β is parameters equals to 0.2 when p_d is positive and 0.6 when p_d is negative.

$$t_s = \beta \cdot p_s + (1 - \beta) \cdot (c_s + p_{ds}) \tag{10}$$

The inputs of both networks are 115:

- the angles of the next 50 segments
- the angles of the last 15 segments
- the trajectory angles of the next 40 segments
- the trajectory angles of the last 10 segments

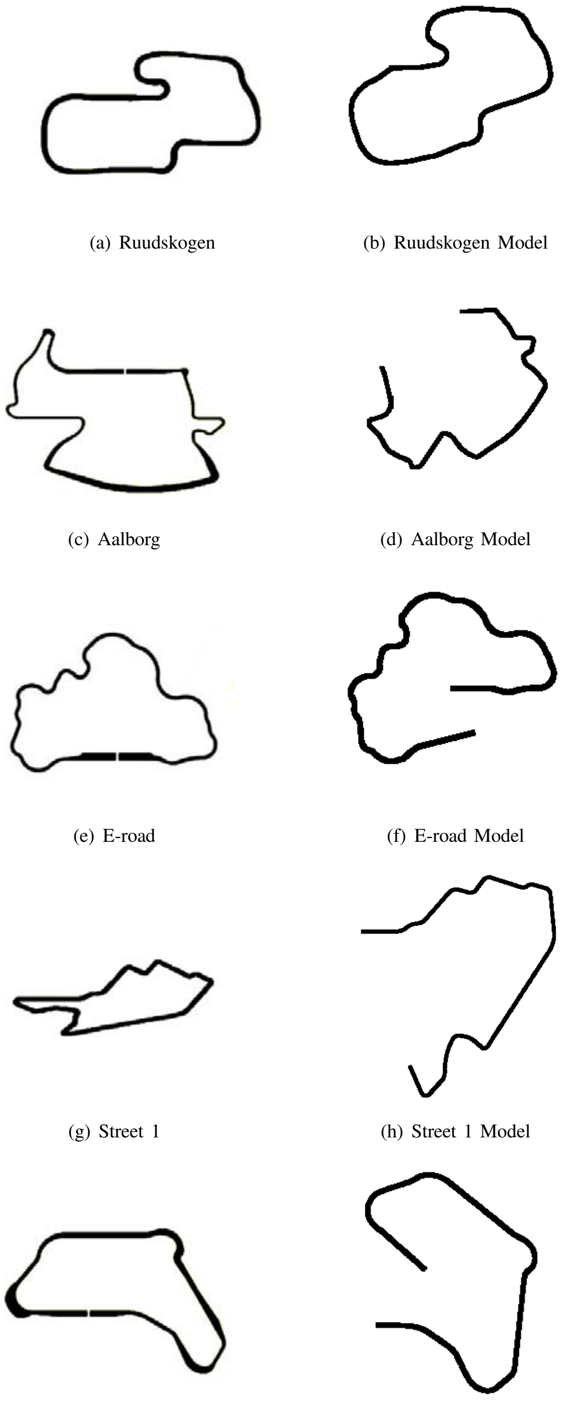
(a) Ruudskogen (b) Ruudskogen Model
(c) Aalborg (d) Aalborg Model
(e) E-road (f) E-road Model
(g) Street 1 (h) Street 1 Model
(i) CG track 2 (j) CG track 2 Model

Fig. 3. Tracks used in the experiments and their models

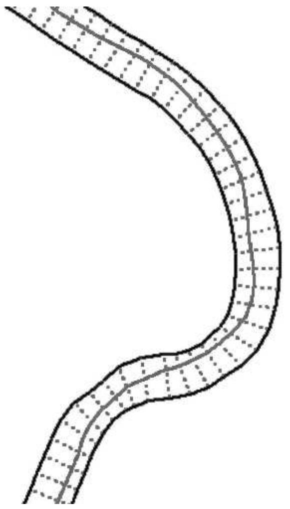

Fig. 4. Example of predicted trajectory in a turn.

E. Steering

The *steering* value is calculated based on the difference of the car position with the trajectory. Equation 11 shows how is the steering value calculated, where t_a is the angle between the car and the track, $maxTurn$ is the maximum turn of the wheels (0.785398), $e_{current}$ is the current error in the trajectory and $e_{advance}$ is the error in the trajectory tree segments ahead if the car would not steer.

$$steering = t_a \cdot maxTurn + e_{current} \cdot 0.8 + e_{advance} \tag{11}$$

Both $e_{current}$ and $e_{advance}$ are calculated as in Equation 13 where $error$ is the error in meters, and γ is an adjust parameter set to 4 for $e_{current}$ and 8 for $e_{advance}$. The results of Equation 13, $e_{relative}$, is a value between 0 and 1.

$$e_{absolute} = minimum(1, maximum(-1, \frac{error}{\gamma})) \tag{12}$$

$$e_{relative} = 1 - \frac{1}{exponential(e_{absolute} \cdot 8 + 5)} \tag{13}$$

When we have the steering value we multiply it by a value depending on the brake value in order to avid brake and turn at the same time (see Equation 14).

$$steering = steering \cdot (1 - brake \cdot 0.8) \tag{14}$$

F. Accelerate and brake

In order to calculate the $accelarate$ and $brake$ values we need the *target speed* (s_t) predicted by the neural networks. Then we calculate the *speed adjust* (s_a) as shows Equation 15 and the *difference speed* (s_d) between the *target speed* and the current speed (s_c) as in Equation 16.

$$s_a = 40 \cdot |e_{current}| \tag{15}$$

$$s_d = s_t - s_c \tag{16}$$

If the *difference speed* is positive the car accelerates and the $brake$ is set to 0, if it is negative the car brakes and the $accelerator$ is set to 0. When the car accelerates the $accelerator$ value is set as in Equation 17 when the *difference speed* (s_d) is bigger than the *speed adjust* (s_a) , otherwise is set to 1. When the car brakes the $brake$ value is set as in Equation 18 when the *difference speed* (s_d) is bigger than 10 , otherwise is set to 1.

$$accelerator = \frac{s_d}{s_a} \tag{17}$$

$$brake = \frac{s_d}{10} \tag{18}$$

(19)

VI. Experiments

In the experiments 5 tracks included in TORCS were used: *Ruudsokgen*, *Aalborg*, *E-Road*, *Street 1* and *CG track 2* (see Figure 3). For each of the tracks a human complete 6 laps in a competitive way without get out of the track in any moment. The human results are shown in Table II, first column is the track name (*Track*), the second the number of laps performed (*Laps*), next three columns shows the times in seconds of the 6 laps (*Total Time*), the average lap (*Average Lap*) and the best lap (*Best Lap*). Last two columns are the top speed (*Top Speed*) and the damages (*Damages*) suffered by the car.

The data of the human game play is stored, except the first lap of each track that is removed in order to avoid the noise of the start. Then for each one of the tracks we used the others as the training data for the neural networks, and the current track to evaluate the learning. This is, for *Ruudskogen* we use *Aalborg*, *E-Road*, *Street 1* and *CG track 2* as the data set for training the neural networks, and then we evaluate these networks in *Ruudskogen*. We do not use the Ruudskogen to train the networks, so we have a fair evaluation. We do the same for all the tracks. The networks have two hidden layers with 20 and 10 neurons each one, they were trained during 1000 cycles with a variable learning rate that starts in 0.8 and ends in 0.0001. We selected these parameters because we got good results in previous tests. Table III shows the number of patterns used in the training of each track.

TABLE III

NUMBER OF PATTERNS FOR TRANNING

Track	Number of patterns
Ruudskogen	119009
Aalborg	114768
E-Road	119033
Street 1	112758
CG track 2	126180

Table IV shows the same information as Table II but with the results of the learnt controller. The last column (*Best Lap*

TABLE II
HUMAN PLAYER TIMES

Track	Laps	Total Time	Average Lap	Best Lap	Top Speed	Damages
Ruudskogen	6	445.38	74.23	71.23	246	17
Aalborg	6	506.41	84.4	81.89	227	17
E-Road	6	442.26	73.71	71.56	262	0
Street 1	6	539.55	89.92	87.25	276	9
CG track 2	6	341.71	56.95	54.47	273	36

Increment) shows the increment of the time for the best lap compared with the human time. Notice that in some tracks the controller does not perform the 6 laps, this is because it suffers so damage that the simulation is over.

VII. Discussion

The times of the controller we trained are good enough for us although slower than a human player. The biggest problem that we have seen when the controller drives is that the car can not follow correctly the trajectory and some times this leads to an instability that makes the car to skid and crash or get out of the road. This is also the reason why the times are slower than the human times. The normal behavior of a human when he can not follow the trajectory he wants to follow, or he has made a mistake, is not to try to turn more to recover the mistake but to create a new not-optimal trajectory to recovery. He also learns the mistake he did and tries to avoid they in the next laps. These are things that our controller can not done yet, but could be improved in the future.

We have seen that in *Ruudskogen* the controller goes out of the track in the hard turn, maybe because the networks were trained without a turn of same characteristics. In *Aalborg* the controller is unstable in the straights and crash in some turns, but not in ones that we think were the most difficult. In *E-road* the controller is also unstable in the high speed turns, making the car crash. In *Street 1* the car does not crash as often as in the other tracks, even perform a lap without crash. And finally, in *CG track 2*, the easiest track, the car never crash and performs good laps. A common problem of all the tracks is that the top speed reached by the controller is much lower than the top speed of the human. We think this is a problem with the training data because there are few patterns with the top speed, we think this could be solved including in the training a track with very high top speed during a lot of time.

The training should be done with similar tracks as the evaluation track. Of course, this will improve the results because their similarity, but we have seen that the human takes some laps to adapt his behavior to the features of the new track. So, to be fair enough the controllers should have the same possibilities, and the only way to make this is to use similar tracks in the training.

VIII. Future works

Some of the works that can be done in our controller to improve it are: the overtaking and avoid overtaking, the trajectory adaptation to correct the mistakes the car done, include a grip parameter to our networks to get better predictions and into the scripted policy to get a better control of the car, and avoid the jumps of the track. The current scripted policy for the gear change, and accelerator, brake and steering control, could be improved or modified to decrease the mistakes made by the controller when follows a trajectory. Another improvement is to adjust the inputs and outputs of the neural networks used in the predictions in order to remove useless inputs and include other useful sensor information.

Next sections explain some of the improvements that could be done.

A. Overtaking

As we said we do not take into account the opponents for our controller but this work could be done easily. We only have to include some modifications to overtake opponents and avoid the overtaking. One easy way to perform the overtaking in our controller is taking into account the opponents in the trajectory. The controller can modify the trajectory and the target speed in order to pass the opponent cars when these are close enough to our car. To avoid the overtaking the trajectory and target speed could be also modified, but in this case the controller should follow a trajectory that avoids the overtaking, this is trying to be located all the time in front of the opponents car.

B. Trajectory adaptation

When our controller is trying to follow a trajectory it makes some mistakes and never follow the trajectory perfectly. The main reason is that the scripted policy to follow the trajectory is based on the error of the current position of the car with the trajectory, so there is needed some kind of error to follow the trajectory. But this error is more or less the same in each lap, so we could be able to take into account this error to include it in the scripted policy or in the trajectory in order to follow the trajectory better. As the track is divided in segments we can update the error in each segment and take it into account in the next lap to modify the steering with the aim of advancing the future errors and correct them before they happen.

TABLE IV
CONTROLLER TIMES

Track	Laps	Total time	Average Lap	Best Lap	Top Speed	Damages	Best Lap Increment
Ruudskogen	6	605.35	100.89	90.38	212	1183	+26.88%
Aalborg	3	360.03	120.01	115.75	207	10090	+41.34%
E-Road	6	738.55	123.09	105.10	215	6344	+46.86%
Street 1	4	472.69	118.17	104.40	217	10121	+19.65%
CG track 2	6	417.28	69.54	67.57	217	898	+24.04%

C. Grip

The tracks are different and have different grips, also the grip is modified at the same time the fuel is consumed and the wheels are worn. The grip of the car affects directly to some systems we implemented as the ABS and the TCS, but also over the scripted policy to control the car. This is the main reason that the grip should take part, not only in the TCS and ABS, but in the scripted policy too. The problem is that the grip is not constant and depends on the track. So one future improvement is to create an algorithm able to estimate this parameter and updates it while the car state changes. The warm-up stage matches perfectly to set this value and make some test before the race.

D. Jumps

Finally, some of the tracks can include jumps or other points that the car must avoid, or minimize their influence in the control in order to get better results. The sensors provide some information about the vertical speed of the car that can be used to know when the car jumps. These points could be stored and in the next laps try to avoid it or do other things like braking to minimize their effects in the control.

ACKNOWLEDGMENT

This work was supported in part by the University Carlos III of Madrid under grant PIF UC3M01-0809 and by the Ministry of Science and Innovation under project TRA2007-67374-C02-02.

REFERENCES

[1] X. Yang and J. Zheng, “Artificial neural networks,” *Handbook of Research on Geoinformatics*, p. 122, 2009.

[2] M. Tipping and M. Hatton, *Drivatars^{TM} and Forza Motorsport*, 2006. [Online]. Available: http://www.vagamelabs.com/drivatars-trade-and-forza-motorsport.htm

[3] J. Matthews, *Colin McRae Rally 2.0. Interview with Jeff Hannan*, 2001. [Online]. Available: http://www.generation5.org/content/2001/hannan.asp

[4] M. Buckland, *Colin McRae Rally 2.0. Interview with Jeff Hannan*. [Online]. Available: http://ai-junkie.com/misc/hannan/hannan.html

[5] B. Chaperot and C. Fyfe, “Improving artificial intelligence in a motocross game,” in *2006 IEEE Symposium on Computational Intelligence and Games*, 2006, pp. 181–186.

[6] L. Cardamone, D. Loiacono, and P. Lanzi, “Learning drivers for TORCS through imitation using supervised methods,” in *Proceedings of the 5th international conference on Computational Intelligence and Games*. IEEE Press, 2009, pp. 148–155.

[7] J. Muñoz, G. Gutierrez, and A. Sanchis, “Controller for torcs created by imitation,” in *IEEE Symposium on Computational Intelligence and Games*, September 2009, pp. 271–278.

[8] N. Van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber, “Robust player imitation using multiobjective evolution,” in *Proceedings of the IEEE Congress on Evolutionary Computation (in press)*. Citeseer, 2009.

[9] J. Togelius, R. De Nardi, and S. Lucas, “Making racing fun through player modeling and track evolution,” *Optimizing Player Satisfaction in Computer and Physical Games*, p. 61, 2006.

[10] M. Butz and T. Lonneker, “Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge,” in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, Milano, Italy*, 2009.

[11] E. Onieva, D. Pelta, J. Alonso, V. Milanés, and J. Pérez, “A modular parametric architecture for the TORCS racing engine,” in *Proceedings of the 5th international conference on Computational Intelligence and Games*. IEEE Press, 2009, pp. 256–262.

[12] D. Perez, G. Recio, Y. Saez, and P. Isasi, “Evolving a fuzzy controller for a car racing competition,” in *Proceedings of the 5th international conference on Computational Intelligence and Games*. IEEE Press, 2009, pp. 263–270.

[13] D. Ho and J. Garibaldi, “A Fuzzy Approach For The 2007 CIG Simulated Car Racing Competition,” in *IEEE Symposium On Computational Intelligence and Games, 2008. CIG’09*, 2008, pp. 127–134.

[14] L. Cardamone, D. Loiacono, and P. Lanzi, “On-line neuroevolution applied to the open racing car simulator,” in *Evolutionary Computation, 2009. CEC ’09. IEEE Congress on*, May 2009, pp. 2622–2629.

[15] K. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[16] D. Loiacono, L. Cardamone, and P. L. Lanzi, “Simulated Car Racing Championship 2010: Competition Software Manual,” Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, Tech. Rep., 2010.8.

[17] D. Loiacono, J. Togelius, P. Lanzi, L. Kinnaird-Heether, S. Lucas, M. Simmerson, D. Perez, R. Reynolds, and Y. Saez, “The wcci 2008 simulated car racing competition,” in *IEEE Symposium On Computational Intelligence and Games*, Dec. 2008, pp. 119–126.