Evolving Competitive Car Controllers for Racing Games with Neuroevolution

Luigi Cardamone[†], Daniele Loiacono[†], Pier Luca Lanzi[†*]
[†]Artificial Intelligence and Robotics Laboratory (AIRLab)
Politecnico di Milano. P.za L. da Vinci 32, I-20133, Milano, Italy
[*]Illinois Genetic Algorithm Laboratory (IlliGAL)
University of Illinois at Urbana Champaign, Urbana, IL 61801, USA
cardamone@elet.polimi.it, loiacono@elet.polimi.it, lanzi@elet.polimi.it

ABSTRACT

Modern computer games are at the same time an attractive application domain and an interesting testbed for the evolutionary computation techniques. In this paper we apply NeuroEvolution of Augmenting Topologies (NEAT), a well known neuroevolution approach, to evolve competitive non-player characters for a racing game. In particular, we focused on The Open Car Racing Simulator (TORCS), an open source car racing simulator, already used as a platform for several scientific competitions dedicated to games. We suggest that a competitive controller should have two basic skills: it should be able to drive fast and reliably on a wide range of tracks and it should be able to effectively overtake the opponents avoiding the collisions. In this paper we apply NEAT to evolve separately these skills and then we combined them together in a single controller. Our results show that the resulting controller outperforms the best available controllers on a challenging racing task. In addition, the experimental analysis also confirms that both the

skills are necessary to develop a competitive controller.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Games*

General Terms

Algorithms, Experimentation, Performance

Keywords

NEAT, Games, TORCS, Simulated Car Racing

1. INTRODUCTION

Modern computer games today are very complex, realistic and team-oriented environments. As a result, programming

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

the artificial intelligence of games is an increasingly difficult and expensive task. In this scenario, computational intelligence is a promising technology to support the development of the artificial intelligence and to improve the game experience. In particular, the recent availability of cheap computing resources and several recent works in the literature [23, 11, 15] suggest that evolutionary computation techniques can be effectively applied to develop more interesting and attractive computer games. At the same time, computer games are an ideal testbed for evolutionary computation techniques as they provide very complex and realistic environments without the need of expensive simulators or real-world experiments [18]. Accordingly, in the recent years several scientific competitions dedicated to computer games have been organized at major international conferences in the evolutionary computation field [22, 9].

In this work we focused on The Open Racing Car Simulator (TORCS) [1] a state of the art car racing simulator, that features a sophisticated physics engine, and takes into account many aspects of a real racing car (e.g., car damage, fuel consumption, friction, aerodynamics, etc.). In particular, we applied NeuroEvolution with Augmenting Topology (NEAT) [17], a well known neuroevolution approach, to evolve a competitive controller for TORCS. To perform the experiments reported in this paper and to evolve a controller for TORCS, we used the framework of the Simulated Car Racing Competition, a competition based on TORCS organized in the last year in conjunction with two major conferences in the computational intelligence field (WCCI-2008 and CIG-2008). In this work we focus on two basic skills that we believe a competitive controller should have: it should be able to drive fast and reliably on a wide range of tracks and it should be able to effectively overtake the opponents avoiding the collisions. However, a reliable evaluation of both these skills at the same time is not an easy problem: it would require to design an evaluation process that covers a broad range of game conditions. Thus, we applied NEAT to evolve these skills separately and then combined them in a single controller.

To test our approach we compared our controllers to several controllers submitted to the Simulated Car Racing Competition. In the first set of experiments we evaluated the performance of the evolved skills, considering only one skill at once. Finally, we combined the evolved skills on a single controller and then we tested it on a challenging racing task.

2. RELATED WORK

In the recent years, a growing body of researches is focusing on the application of computational intelligence techniques to modern computer games, which are seen either as convenient environments to test new techniques or as an attractive application domain for the existing computational intelligence techniques. In particular, a lot of recent works [21, 19, 20, 18, 2, 23, 16] in the literature focused on racing games, a type of computer games where the goal is to control effectively a vehicle to accomplish a given task. Beside the problem of controlling the dynamics of a vehicle, the racing games involve additional challenges, like avoiding the collisions, finding the best trajectory to overtake an opponent, choosing the pit-stop strategy, etc. Such a wide range of issues makes it possible to easily define learning tasks of increasing complexity, ranging from a basic control problem to very rich and complex behaviors. In an early work, Pyeatt and Howe [12] applied reinforcement learning to learn racing behaviors in RARS, an open source car racing simulator. They show that the decomposition of the racing behavior could result in a speed-up of the learning process, although it might require some specific domain knowledge to combine successfully the learned behaviors. More recently, evolutionary computation techniques have been applied to improve the performance of a motocross game AI [2], to optimize the parameters in a sophisticated F1 racing simulator [23] and to evolve a neural network to predict crash in a car racing simulator [16]. Then, in the recent years, several works on learning controllers for racing games have been done by Togelius and Lucas [21, 19, 20, 18]. In particular, they evolved neural controllers both for radio-controlled car models and for simple racing simulator. They investigated several schema of sensory information (e.g., first person based, third person based, etc.) and studied the generalization capabilities of the evolved controllers. Finally, the computational intelligence techniques have been also applied to some commercial racing games. In *Colin McRae Rally 2.0 (Codemasters)* a neural network is used to drive a rally car, thus avoiding the need to handcraft a large and complex set of rules [5]: a feedforward multilayer neural network has been trained to follow the ideal trajectory, while the other behaviors ranging from the opponent overtaking to the crash recovery are programmed. In *Forza Motorsport (Microsoft)* the player, can train his own *drivatars*, i.e., a controller that learns the player's driving style and that can take his place in the races.

3. NEUROEVOLUTION WITH AUGMENTING TOPOLOGY

In this study, we focused on Neuroevolution with Augmenting Topology or NEAT [17], one of the most successful and widely applied neuroevolution approach. NEAT is specifically designed to evolve neural networks without assuming any a priori knowledge on the optimal topology nor on the type of connections (e.g., simple or recurrent connections). NEAT is based on three main ideas. First, in NEAT the evolutionary search starts from a network topology as simple as possible, i.e. a fully connected network with only the input and the output layers. Complex structures emerge during the evolutionary process and survive only when useful. Second, NEAT deals with the problem of recombining networks with different structures through an

Figure 1: A screenshot from TORCS.

historical marking mechanism. Whenever a structural mutation occurs, a unique *innovation number* is assigned to the gene representing this innovation. This mechanism is then used to perform the recombination and to identify similarities between the networks without the need of a complex and expensive topological analysis. Third, NEAT protects the structural innovations through the mechanism of *speciation*. The competition to survive does not happen in the whole population but it is restricted to niches where all the networks have a similar topology. Therefore, when a new topology emerges, NEAT has enough time to optimize it. Again, the historical marking is used to identify the niches by using the innovation numbers of the genes for measuring the similarities between different topologies. To prevent a single species from taking over the population, the networks in the same niche share their fitness [4]. Then, in the next generation more resources will be allocated to the species with greater fitness.

4. SIMULATED CAR RACING

In recent years, several scientific competitions dedicated to computer games have been organized at major international conferences [22, 9]. Among the various games used in competitions such as Pac-Man [9] or Othello [10], simulated car racing has recently received more and more attention. The first event organized during CEC-2007 was based on a simple simulator written in Java and developed by Julian Togelius. The subsequent competitions moved to a more realistic platform, the Open Racing Car Simulator (TORCS) [1], a state-of-the-art open source car racing simulator which has been used for the competitions organized at WCCI-2008 and CIG-2008. In addition, three similar competitions based on the same platform are planned for CEC-2009, GECCO-2009 and CIG-2009. Accordingly, in this study we focused on the same platform and we applied NEAT to evolve a fast and challenging controller for TORCS. The rest of this section is organized as follows. First of all we introduce TORCS. Then, we provide an overview of the API provided with the Simulated Car Racing Competition to control the car in the game. Finally we briefly describe the controllers submitted to the Simulated Car Racing Competition, that we compared to our approach in the experimental analysis reported in this paper.

4.1 TORCS

The Open Racing Car Simulator (TORCS) [1] is a state-of-the-art open source car racing simulator which provides a full 3D visualization (Figure 1)), several tracks, many types of cars, and different game modes (e.g., practice, quick race, championship). The car dynamics is accurately simulated with a sophisticated physics engine, that takes into account many aspects of the racing car (e.g. traction, aerodynamics, fuel consumption, etc.). Each vehicle is controlled by its own automated driver or *bot*. The game is provided with a lot of human programmed bots that users can easily extended and customize to develop their own bots. The game provides a lot of information on the current state of the car and of the race, including the position on the track, the distance between the bot and the other cars, the current speed, etc. However such information can be also easily preprocessed to provide the desired representation of the surrounding game environment. At each control step, each bot controls the gas/brake pedals, the gear stick, and steering wheel on the basis of sensory information it perceives. Although TORCS was not specifically designed to perform machine learning research, it is well suited to our purposes. In fact, it combines many features typical of commercial racing simulators with a modular and customizable software architecture. The experimental analysis performed in this paper has been carried out using the same version of TORCS (1.3.0) used for the Simulated Car Racing Competition held in conjunction with WCCI-2008 [8].

4.2 Competition API

In the Simulated Car Racing Competition, the competitors have been provided with a specific software interface developed on a client/server basis. The controllers run as external programs and communicate with a customized version of TORCS through UDP connections. Each controller perceives the racing environment through a number of sensor readings which would reflect both the surrounding environment (the track and the opponents) and the current game state and they could invoke basic driving commands to control the car. The complete list of sensors is reported in Table 1 and includes rangefinders to perceive the distance of nearby opponents, the current speed, the current gear, the fuel level, etc (we refer the interested reader to the software manual of the competition [7] for additional details). Table 2 reports all the driving commands: besides the rather typical driving commands (i.e., the steering wheel, the gas pedal, the brake pedal, and the gear change) a meta-command is available to reset the state of the race from the client-side.

4.3 Submitted Controllers

In this section we briefly introduce the controllers submitted to the past editions of the Simulated Car Racing Competition, that have been used in the experimental analysis reported in this paper (we refer the interested reader to [8] for more details about the competition and the entries submitted). The sources of all the controllers described below are available on the homepage of the Simulated Car Racing Competition [6], where are also available two sample controllers programmed in C++ and Java. In the remainder of the paper we refer to each controller with the name of the first author, except for the sample controllers, dubbed as *C++ example* and *Java example*.

Name	Description
angle	Angle between the car direction and the direction of the track axis.
curLapTime	Time elapsed during current lap.
damage	Current damage of the car (the higher is the value the higher is the damage).
distFromStartLine	Distance of the car from the start line along the track line.
distRaced	Distance covered by the car from the beginning of the race
fuel	Current fuel level.
gear	Current gear: -1 is reverse, 0 is neutral and the gear from 1 to 6.
lastLapTime	Time to complete the last lap
opponents	Vector of 18 sensors that detects the opponent distance in meters (range is [0,100]) within a specific 10 degrees sector: each sensor covers 10 degrees, from $-\pi/2$ to $+\pi/2$ in front of the car.
racePos	Position in the race with respect to other cars.
rpm	Number of rotation per minute of the car engine.
speedX	Speed of the car along the longitudinal axis of the car.
speedY	Speed of the car along the transverse axis of the car.
track	Vector of 19 range finder sensors: each sensors represents the distance between the track edge and the car. Sensors are oriented every 10 degrees from $-\pi/2$ and $+\pi/2$ in front of the car. Distances are in meters and sensors are limited

	to 100 meters. When the car is outside of the track (i.e., pos is less than -1 or greater than 1), these values are not reliable!
trackPos	Distance between the car and the track axis. The value is normalized w.r.t the track width: it is 0 when car is on the axis, -1 when the car is on the left edge of the track and +1 when it is on the right edge of the car. Values greater than 1 or smaller than -1 means that the car is outside of the track.
wheelSpinVel	Vector of 4 sensors representing the rotation speed of the wheels.

Table 1: Description of available sensors in the competition API.

Name	Description
accel	Virtual gas pedal (0 is no gas, 1 is full gas).
brake	Virtual brake pedal (0 is no brake, 1 is full brake).
gear	Gear value defined in {-1,0,1,2,3,4,5,6} where -1 is reverse and 0 is neutral.
steering	Steering value: -1 and +1 means respectively full left and right, that corresponds to an angle of 0.785398 rad.
meta	This is meta-control command: 0 do nothing, 1 ask competition server to restart the race.

Table 2: Description of available effectors.

Chiu. This controller was submitted to the Simulated Car Racing Competition held in conjunction with CIG-2008. It is a human programmed controller built upon the C++ example controller provided with the competition API. The gas and the brake pedals are controlled with a policy similar to the one used in the C++ sample controller, but without any speed limit. The steering behavior basically consists of driving along the direction corresponding to the furthest distance to the track edges (according to *track* sensor).

Kinnaird-Heether et al. This controller resulted to be the second best entries to the Simulated Car Racing competition held in conjunction with WCCI-2008 [8] with an overall performance very close to the winner one. It exploits a Cultural Algorithm [13] to optimize the parameters of a programmed controller. First a programmed controller is developed decomposing it into four behaviors: (i) *acceleration*, that deals with the gas and the brake pedals to achieve the target speed; (ii) *steering*, that controls the steering wheel of the car; (iii) *shifting*, that implements the gear shifting policy; (iv) *error correction*, that is basically used to recover from critical situations. Then, a Cultural Algorithm is applied to optimize the target speed during turns used in the *acceleration* behavior. The fitness of individuals is computed simply as the distance raced in a fixed amount of time and the best solution found was submitted as a controller for the competition.

Lucas. This controller was submitted to the Simulated

Car Racing Competition held in conjunction with WCCI-2008 [8]. It is a human programmed controller that improves the Java example controller provided with the competition API. It basically increased the speed limit of the Java example controller and extended the steering and the braking policies to deal with the higher speed.

Perez et al. This controller was submitted to the Simulated Car Racing Competition held in conjunction with WCCI-2008 [8] and an improved version of the same controller was submitted to Simulated Car Racing Competition held in conjunction with CIG-2008. In this controller the knowledge is represented as a set of rules that are evolved with an evolutionary algorithm. Each rule consists of a *condition* on the car sensors (i.e., when to apply the rule) and an *action* on the car effectors (i.e., how to apply the rule). Among the available sensors, only the following one have been considered: (i) the angle w.r.t. the track axis (*angle*); (ii) the distance between the car and the track axis (*trackPos*); (iii) the current speed (*speedX*); (iv) the leftmost, the rightmost and the frontal rangefinders to detect the track edges (a subset of *track* sensors). The effectors are all the ones available in the API: the gas and brake pedals, the steering wheel and the gear shifting. The evolutionary process basically consists of selecting two rules in the knowledge base, recombining them, applying a mutation and then adding it to the knowledge base replacing the rule most similar to it. New rules are kept in the knowledge base only if they lead to a better controller, i.e., a controller with an highest fitness, computed on the basis of the lap-time and of the damage suffered by the car.

Simmerson. This controller resulted the winner of the Simulated Car Racing competition held in conjunction with WCCI-2008 [8]. It basically consists of a neural controller evolved using NEAT4j [14], a Java implementation of NEAT. The network has three outputs that control respectively the gas and brake pedal, the steering wheel and the gear shifting. The inputs of the network are chosen in a subset of the ones available in the competition API (see Table 1): (i) the angle w.r.t. track axis (*angle*); (ii) the current speed (*speedX*); (iii) the 19 track rangefinders (*track*); (iv) the current gear (*gear*); (v) the spin speed of wheels (*wheelSpinVel*); (vi) the distance between the car and the track axis (*trackPos*); (vii) the current RPM of the engine (*rpm*). The fitness used in the evolutionary process is computed as the distance raced within a fixed amount of game tics, penalizing it for the amount of damage received and the time spent out of the track.

Tan et al. This controller was submitted to the Simulated Car Racing Competition held in conjunction with WCCI-2008 [8]. It was developed with a three-step process. First, the sensory information was aggregated and preprocessed; second, a parametrized controller based on simple rules was designed; finally, the parameters of controller were optimized using evolution strategies. The resulting controller drives in the direction where the rangefinder sensors indicate the largest free distance, with a speed dependent on that distance.

5. EVOLVING THE DRIVING SKILL

First we focused on evolving a controller that drives as fast as possible when racing alone on the track. In the following of this section we describe the design of the evolved controller, i.e., we define the inputs and the outputs of the neural network evolved. Then, we describe the experimental setup used to evolve the neural controller, including the fitness function used. Finally we report the experimental results.

5.1 Controller Design

In order to apply successfully NEAT to evolve a controller for TORCS, the choice of the proper inputs and outputs of the network plays a key role. As inputs of the neural network we focused on a subset of the sensory information provided with the competition API: (i) six rangefinder sensors to perceives the track edges (provided by the *track* sensor) along the directions $\{-90°, -60°, -30°, +30°, +60°, +90°\}$; (ii) an improved frontal sensor computed as the biggest value among the ones returned from the three rangefinders along the directions $\{-10°, 0°, +10°\}$; (iii) the current speed of the car (*speedX*). Our results show that such a small subset of the sensory inputs available is enough to evolve a fast controller with NEAT. In addition, our empirical analysis suggests that replacing the frontal rangefinder, i.e., the one parallel to the car axis, with the improved frontal sensor leads to a better controller. In fact, the frontal sensor is exploited from the controllers to detect either when a turn is approaching or when it is over: the improved frontal sensor detect more reliably the begin and the end of turns, especially when the car is not perfectly aligned to the axis of the track (see the example in Figure 2). Concerning the outputs of the neural controller, we used two continuous outputs in the range [-1,1]. The first one is used to control the steering wheel, according to the *steering* effector described in Table 2. The second one is used to control the gas and brake pedals as follows. If the output is less than zero it is assigned, as a positive value, to the *brake* effector, resulting

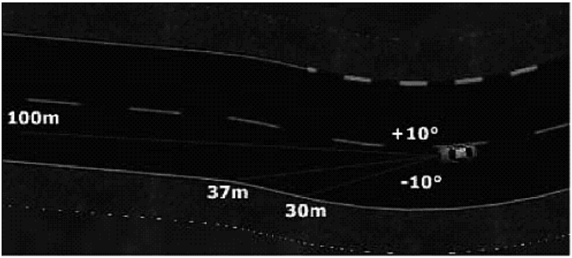
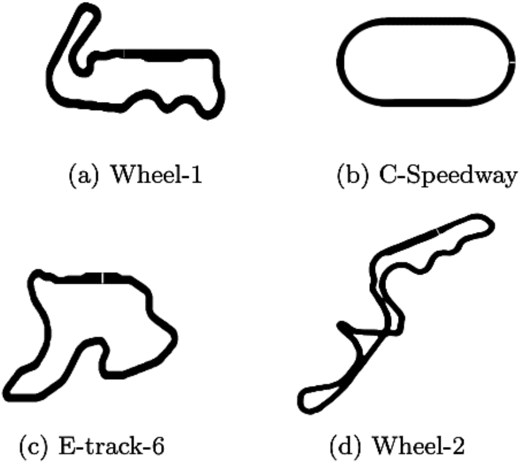


Figure 2: An example of how the improved frontal sensor works. In this example, the rangefinders parallel to the car axis returns 37m even if the turn is almost over, preventing the controller from applying a full acceleration. The improved frontal instead returns the biggest among the three rangefinders reading, i.e., 100m.



(a) Wheel-1 (b) C-Speedway

(c) E-track-6 (d) Wheel-2

Figure 3: Tracks used in all the experiments reported in this paper.

in a braking command. Otherwise it is assigned to the *accel* effector, resulting in an acceleration command. In addition, when the car is in a straight segment of the track, i.e., when the frontal sensor does not perceive the track edge within 100 meters, the gas pedal is set to 1 and the brake pedal is set to 0. Such a design choice forces the controller to drive fast since the early generations and prevents the evolutionary search from wasting time with safe but slow controllers.

Finally, to deal with the gear shifting, we used a programmed policy: while it is quite complex to develop a good policy that controls the speed and the trajectory of the car, an effective gear shifting policy can be quite easily programmed. The controller is also provided with a scripted recovery policy to be used when the car goes outside of the track.

5.2 Controller Training

To train the driving behavior we evolved a population of 100 networks for 150 generations with the standard C++ implementation of NEAT [17]. The evolved networks are evaluated on the basis of their performance when racing alone on the `Wheel 1` track depicted in Figure 3(a). This track was chosen because it is a fast track but, at the same time, it involves several complex turns, being in our opinion

Controller	`C-Speedway`	`E-Track 6`	`Wheel 2`
NEAT driver	15528.90	**7566.40**	8534.31
Kinnaird-Heether et al.	14573.80	5375.44	6804.25
Simmerson	12629.50	6386.44	2792.97
Tan et al.	12590.00	5136.41	6823.36
Perez et al.	5540.09	4428.98	4612.84
Chiu	**16721.50**	6820.00	**8823.05**
Lucas	12240.30	5022.28	3943.58
C++ example	7265.36	4930.86	4664.39
Java example	5711.37	2932.35	2355.26

Table 3: Distance raced by each controller within 10000 game tics on the `C-Speedway`, `E-Track 6`, and `Wheel 2` tracks. Statistics reported are the medians computed over 10 runs.

a good mix of the tracks available in TORCS. The evaluation process consists of two laps on the `Wheel 1` track. First, in the *warm-up* lap, the network to be evaluated is loaded into the bot that starts to race. Then, in the *evaluation* lap, the performance achieved is recorded and used to compute the fitness of the network as follows:

$$F = C_1 - T_{out} + C_2 \cdot \bar{s} + d,$$

where T_{out} is the number of game tics the car is outside the track; $\bar{s}$ is the average speed (meters for game tic) during the evaluation; d is the distance (meters) raced by the car during the evaluation; C_1 and C_1 are two constants introduced respectively to make sure that the fitness is positive and to scale the average speed term (both C_1 and C_2 have been empirically set to 1000 in all the experiment reported here).

5.3 Experimental Result

To test our approach we compared the controller evolved to the ones submitted as entries to the past editions of the Simulated Car Racing Competition. The comparison has been carried out following the same approach used for the competition: each controller is scored when racing alone on three different tracks. The performance of the controller is measured as the distance raced by each controller in 10000 game tics, corresponding to 200 seconds of simulated game. In the following experimental analysis we used the same three tracks used for the last edition of the Simulated Car Racing Competition: `C-Speedway`, `E-Track 6`, and `Wheel 2` (depicted respectively in Figure 3(b), in Figure 3(c), and in Figure 3(d)). Table 3 compares the performance of the controllers described in Section 4 to our controller, dubbed as *NEAT driver*. The first five controllers reported in Table 3 have been developed applying an evolutionary algorithm, while the last four controllers are entirely programmed. The results show that the controller evolved with our approach has the highest performance among the evolved controllers, while the programmed controller developed by Chiu appears to be slightly faster in two tracks out of three. Therefore, NEAT is able to evolve a neural controller almost as fast as the best programmed controllers also on tracks different from the one on which it was trained. In addition, the analysis of behavior of the evolved controller reveals that although it does not always follows the optimal trajectory, it controls the car very reliably avoiding to race outside the edges of the track.

To make a fair comparison of the controllers reported in

Table 3, we should underline that they are evolved using two different approaches. The first approach, followed by Simmerson and by Perez et al. consists of evolving the controller from scratch without using any prior knowledge on the structure of the controller. The second approach, followed by Kinnaird-Heether et al. and by Tan et al. exploits an evolutionary algorithm to optimize the parameters of a designed driving behavior. Instead, our approach falls somewhere between the former and the latter: the driving skill is evolved from scratch but then is combined with a programmed gear shifting policy and a programmed recovery policy to develop the final controller. Accordingly, the proposed approach does not require any strong assumption on the structure of the searched controller, but at the same time does not involve to deal with a too complex and expensive optimization problem. The results suggest that the proposed approach is effective in practice and leads to a better performance than the one achieved following different approaches.

6. EVOLVING THE OVERTAKING SKILL

In the previous section we showed that NEAT can be applied to evolve a controller with good driving skills. Unfortunately this is not enough to develop a competitive car controller for a racing game, where the capabilities of overtaking the opponents and avoiding the collisions are very important. Nevertheless, most of the controllers submitted to the Simulated Car Racing Competition fails to deal with this issue. Accordingly, in the past editions of the Simulated Car Racing Competition the winner was not the fastest controller submitted but the one with the best tradeoff between driving and overtaking capabilities. Therefore, in this section we apply NEAT to evolve a controller with overtaking skills. The section is organized as the previous one. First, we define the controller architecture, then we describe the experimental setup used to train the controller, and finally we discuss the experimental results.

6.1 Controller Design

To define the inputs of the neural controller, we focus again on a subset of the available sensors: we use the same inputs described in the previous section and some additional inputs to perceive the presence of nearby opponents on the track. In particular, we introduced eight additional inputs provided by the *opponents* sensor of the competition API: (i) four beams that cover the frontal area of the car between -20° and + 20° with respect to the car axis; (ii) two diagonal beams that cover respectively the area between - 40° and - 50° and the area between + 40° and + 50° with respect to the car axis; (iii) finally two lateral beams that cover the area between - 70° and - 80° and + 70° and + 80° with respect to the car axis. Similarly to what found in the previous experiment, few inputs are enough to evolve an effective overtaking behavior: in the most critical area of the car, i.e. the frontal area, we use four inputs to represent the presence of opponents while in the lateral and diagonal we use only two inputs. The outputs of the neural controller are the same of the controller previously evolved: one output is used to control the steering wheel and one output is used to control the gas and brake pedals.

6.2 Controller Training

In order to evolve the overtaking skill, we designed a specific evaluation process that involves an *overtaker* bot that races against an *opponent* bot. At the beginning of each evaluation, a programmed controller is used to align the *overtaker* and the *opponent* in a random segment (i.e., it can be a straight strong turn, a chicane, etc.) of the track as shown in Figure 4: the bots are placed with an horizontal *offset* with respect to axis of the track that is randomly selected in $\{-3m, 0m, +3m\}$; the *opponent* is placed ahead the *overtaker* at a *distance* randomly chosen between $10m$ and $20m$. As this initial setup is completed, the neural controller

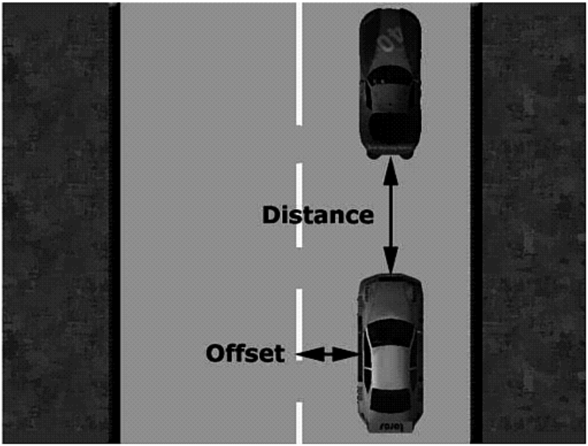


Figure 4: Initial setup used to evaluate the evolved overtaking skill. The blue car (at the top) is the *opponent*, while the yellow car (at the bottom) is the *overtaker*.

is loaded in the *overtaker* bot and the neural controller tries to overtake the *opponent*. After 2000 game tics, corresponding to 40s of simulated time, the evaluation is over and the performance of the controller is computed as:

$$P = C - \alpha \cdot T_{out} - \beta \cdot T_{collision} - \gamma \cdot \Delta,$$

where T_{out} is the number of game tics the *overtaker* is outside the track; $T_{collision}$ is the number of game tics a collision with the *opponent* is detected, Δ is the difference between the position of the *opponent* and the position of the *overtaker* (i.e., a negative value means that the overtake succeeded while a positive means that it failed); C is a constant used to make sure that the fitness is positive, while α, β, γ are used to weights the contribution of each term to the fitness (in the experiments reported in this paper, we empirically set $C = 8000$, $\alpha = 5$, $\beta = 10$, and $\gamma = 3$). Finally, the fitness of each controller in the population is computed as the average performance achieved over 20 evaluations, in order to assess the quality of the solution in several conditions, i.e., in different track segments.

6.3 Experimental Results

In this second set of experiments, we compared the controller with the *overtaking skill* evolved, dubbed *NEAT overtaker*, (i) to the *NEAT driver*, (ii) the winner of the WCCI-2008 edition of the Simulated Car Racing Competition, i.e., the Simmerson's controller and (iii) to the the best programmed controller submitted so far to the Simulated Car Racing Competition, i.e. the Chiu's controller. To com-

Controller	Overtaking Time	$T_{collision}$	T_{out}	Success Rate
NEAT driver	351.77 ± 151.21	109.03 ± 107.99	1.11 ± 11.00	55.2%
NEAT overtaker	**271.49 ± 135.57**	**47.10 ± 97.85**	13.50 ± 39.38	**86.7%**
Chiu	296.65 ± 165.93	152.38 ± 183.14	7.36 ± 22.16	64.1%
Simmerson	397.76 ± 102.62	145.59 ± 150.53	**0.00 ± 0.00**	56.8%

Table 4: Performances of each controller on the overtaking of a slower opponent. Statistics reported are the averages computed over 1000 overtakes.

pare the four controllers we performed 1000 overtakes following almost the same experimental design used to evolve the *overtaking skill*, except for the initial *offset* and *distance* of the cars that are uniformly chosen respectively in the interval $[-3m, +3m]$ in the interval $[10m, 20m)$, to test the controllers in a broad range of conditions. Table 4 compares the performance of the four controllers. The first column, *Overtaking Time*, reports the average number of game tics necessary to overtake the opponent; the second column, $T_{collision}$, reports the average number of game tics in which a collision with the opponent is detected; the third column, T_{out}, reports the average number of game tics in which the controller is out of the track edges; finally, the last column, *Success Rate*, reports the percentage of successfull overtakes performed by the controller, where an overtake is defined as successfull if, after 500 game tics, corresponding to 10s, the distance between the controlled car and the opponent is at least $10m$. The results show that the evolved overtaking behavior, reported as *NEAT overtaker*, outperforms the other controllers except for T_{out}, the average number of game tics the controller is out of the track.

We applied a *one-way* analysis of variance or ANOVA [3] to test whether the differences in Table 4 are statistically significant. We also applied the typical post-hoc procedures (SNK, Tukey, Scheffé, and Bonferroni) to analyze the differences among the four controllers. The analysis of variance shows that there are differences statistically significant at the 99.99% confidence level. In particular, according to the post-hoc procedures: (i) in terms of average overtaking time, the *NEAT overtaker* is significantly better than the others; (ii) in terms of collision avoidance capabilities ($T_{collision}$) the *NEAT overtaker* is significantly better than the others and the *NEAT driver* is significantly better than the Simmerson's and Chiu's controllers; (iii) finally, in terms of capabilities of keeping the track (T_{out}) the *NEAT overtaker* is significantly worse than the other controllers. The results are not surprising as they suggest that the evolved *NEAT overtaker* is able to overtake the opponents faster and more often than the others controller, avoiding the collision as much as possible. However the *NEAT overtaker* also appears to have worse driving capabilities than the other controllers, resulting in a higher average number of game tics spent outside of the track.

7. COMBINING THE SKILLS

In the final experiment we compares the controllers on a complete race against several opponents, to test whether the evolved overtaking skill really gives a competitive advantage in a racing environment. The experiment consists of a complete race against six opponents chosen among the controllers used in the experiments reported in Section 5: (i) the

Kinnaird-Heether's controller, (ii) the Tan's controller, (iii)

Controller	C-Speedway	E-Track 6	Wheel 2	Total
NEAT mixed	9	8	**10**	**27.0**
NEAT overtaker	8	4	9	21.0
NEAT driver	**10**	3.5	5	18.5
Chiu	5	4	4.5	13.5
Simmerson	7	**10**	4	21.0

Table 5: Comparison of the scores achieved by each controller racing against 6 opponents and starting from the last position on the `C-Speedway`, `E-Track 6`, and `Wheel 2` tracks. Statistics reported are the medians computed over 10 races.

the Perez's controller, (iv) the Lucas' controller, (v) the Java example controller and (vi) the C++ example controller. In this experiment we compared the four controllers considered in the previous section to a new controller that combines the *driving skill* and the *overtaking skill* evolved. This controller, dubbed *NEAT mix*, was developed in a straightforward way: it behaves as the *NEAT overtaker* when it perceive an opponent at a distance equal or smaller than $40m$, otherwise it behaves as the *NEAT driver*. For each controller, we run 10 races on the three tracks of TORCS used also in the previous section: `C-Speedway`, `E-Track 6`, and `Wheel 2`. In each run, the controller to test starts from the last position of the starting grid, while the first six position are populated with a random permutation of the six opponents. At the end of each race, a score is assigned to the controller according to the F1 point system, following the scoring procedure used also in the Simulated Car Racing competition [8]: 10 points to the first, 8 points to the second, 6 points to third, 5 points to the fourth and so on.

Table 5 compares the median score of the five controllers for each track and, in the last column, it reports the total score computed as the sum of the median score collected in each track. The results show, that the *overtaking skill* is very important when racing against several opponents; e.g., fast controllers with poor overtaking capabilities, like the Chiu's one and the *NEAT driver*, have a worse score than a slower controller with better overtaking capabilities, like the *NEAT overtaker*. On the other hand, the *driving skill* is very important too; e.g., although the the *NEAT overtaker* has the best overtaking capabilities, it does not outperform the other controllers due to its inferior driving capabilities. Accordingly, *NEAT mix* outperforms all the other controllers, as it combines good driving and overtaking capabilities to win the races.

8. CONCLUSIONS

In this work we applied NEAT to evolve a neural controller for TORCS, that is able to drive fast when racing alone

as well to behave reliably in presence of opponents. First, we applied NEAT to evolve a *driving skill*, that is a neural controller specifically devised to race alone as fast as possible on different types of tracks. Then, we extended the same approach to evolve an *overtaking skill*, that is we evolved a neural controller that is able to overtake an opponent and to avoid collisions in a broad range of situations. Finally, we combined the evolved skills in a single controller that is able to drive fast as well to challenge several opponents in a race.

To test our approach, an empirical analysis was performed following the same guidelines used to evaluate the entries submitted to the Simulated Car Racing Competition. In the first experiment we tested the performance of the evolved behaviors alone. Our results show that NEAT is able to evolve a *driving skill* that is competitive with the best human programmed controller available. Similarly, the *overtaking skill* evolved by NEAT outperformed both the best programmed and the best evolved controllers. In the final experiments we compared the best controllers of the Simulated Car Racing Competition to the ones evolved with our approach: a controller with only the *driving skill*, a controller with only the *overtaking skill*, and a controller that combines both the skills. Such a comparison was performed on a challenging task: racing against six opponents starting from the last position of the starting grid. The results suggest that either the *driving skill* or the *overtaking skill* alone does not lead to a very competitive and reliable car controller. Instead, the controllers that exploits both the skills is able to outperform all the other controllers, including the best submitted to the past editions of the Simulated Car Racing Competition.

9. REFERENCES

[1] The open racing car simulator website. Online.

http://torcs.sourceforge.net/.

[2] Benoit Chaperot and Colin Fyfe. Improving artificial intelligence in a motocross game. In *IEEE Symposium on Computational Intelligence and Games*, 2006.

[3] S. A. Glantz and B. K. Slinker. *Primer of Applied Regression & Analysis of Variance*. McGraw Hill, 2001. second edition.

[4] David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[5] Jeff Hannan. Interview to jeff hannan, 2001. http://www.generation5.org/content/2001/hannan.asp.

[6] Daniele Loiacono, Julian Togelius, and Pier Luca Lanzi. The car racing competition homepage. Online. http://cig.dei.polimi.it/.

[7] Daniele Loiacono, Julian Togelius, and Pier Luca Lanzi. Software manual of the car racing competition. Online. http://mesh.dl.sourceforge.net/sourceforge/cig/CIG2008-Manual-V1.pdf.

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

[9] Simon M. Lucas. Ms pac-man competition. *SIGEVOlution*, 2(4):37–38, 2007.

[10] Simon M. Lucas and Thomas P. Runarsson. Temporal difference learning versus co-evolution for acquiring othello position evaluation. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, 2006.

[11] Steffen Priesterjahn, Kramer, Alexander Weimer, and Andreas Goebels. Evolution of human-competitive agents in modern computer games. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, 2007.

[12] Larry D. Pyeatt and Adele E. Howe. Learning to race: Experiments with a simulated race car. In *Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference*, pages 357–361. AAAI Press, 1998.

[13] R.G. Reynolds and M. Ali. Computing with the social fabric: The evolution of social intelligence within a cultural framework. *Computational Intelligence Magazine, IEEE*, 3(1):18–30, February 2008.

[14] Matt Simmerson. Neat4j homepage. Online, 2006. 2006. [Online]. Available: http://neat4j.sourceforge.net.

[15] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, 2005.

[16] Kenneth O. Stanley, Nate Kohl, Rini Sherony, and Risto Miikkulainen. Neuroevolution of an automobile

crash warning system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)*, 2005.

[17] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[18] Julian Togelius. *Optimization, Imitation and Innovation: Computational Intelligence and Games.* PhD thesis, Department of Computing and Electronic Systems, University of Essex, Colchester, UK, 2007.

[19] Julian Togelius and Simon M. Lucas. Evolving controllers for simulated car racing. In *Proceedings of the Congress on Evolutionary Computation*, 2005.

[20] Julian Togelius and Simon M. Lucas. Arms races and car races. In *Proceedings of Parallel Problem Solving from Nature.* Springer, 2006.

[21] Julian Togelius and Simon M. Lucas. Evolving robust and specialized car racing skills. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2006.

[22] Julian Togelius, Simon M. 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, and Pete Burrow. The 2007 ieee cec simulated car racing competition. *Genetic Programming and Evolvable Machines*, 2008.

[23] Krzysztof Wloch and Peter J. Bentley. Optimising the performance of a formula one car using a genetic algorithm. In *Proceedings of Eighth International*

Conference on Parallel Problem Solving From Nature, pages 702–711, 2004.