Searching for the Optimal Racing Line Using Genetic Algorithms

Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, and Alessandro Pietro Bardelli

Abstract— **Finding the racing line to follow on the track is at the root of the development of any controller in racing games. In commercial games this issue is usually addressed by using human-designed racing lines provided by domain experts and represents a rather time consuming process. In this paper we introduce a novel approach to compute the racing line without any human intervention. In the proposed approach, the track is decomposed into several segments where a genetic algorithm is applied to search for the best trade-off between the minimization of two conflicting objectives: the length and the curvature of the racing line. The fitness of the candidate solutions is computed through a simulation performed with The Open Racing Car Simulator (TORCS), an open source simulator used as testbed in this work. Finally, to test our approach we carried out an experimental analysis that involved 11 tracks provided with the TORCS distribution. In addition, we compared the performance of our approach to the one achieved by a related approach, previously introduced in the literature, and to the performance of the fastest controller available for TORCS. Our results are very promising and show that the presented approach is able to reach the best performance in almost all the tracks considered.**

I. Introduction

The best racing line identifies the trajectory that a driver should follow to achieve the best lap-time on a given track with a given car. The best racing line depends on several factors [11], [4] including the track shape, the car aerodynamics, the grip, etc.

In racing games, non-player characters are typically designed to follow the best racing line [20]. In commercial games, racing lines are usually drawn by domain experts [20] and then tested and tuned by game developers through actual game-play. This hand-crafted process is time-consuming for game designers especially with the large number of tracks and car models available in more recent games. Noticeably, in the game research community, there are only few works that address the automatic design of racing lines using simple heuristics [10], [13], [3].

In this paper, we present an initial step toward the development of a tool for the automatic design, tune and test of the best racing line for a given track and car model. Our work is based on recent findings in real car racing research [4]. In this area, the search of the best racing line is modeled as a quadratic optimization problem to compute the optimal trade-off between two conflicting objectives: (i) driving fast, following the path with minimum curvature (MCP) and (ii) following the shortest path (SP).

Luigi Cardamone (cardamone@elet.polimi.it), Daniele Loiacono (loiacono@elet.polimi.it) Pier Luca Lanzi (lanzi@elet.polimi.it), and Alessandro Pietro Bardelli (alessandro.bardelli@mail.polimi.it) are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy. Pier Luca Lanzi (lanzi@illigal.ge.uiuc.edu) is also member of the Illinois Genetic Algorithm Laboratory (IlliGAL), University of Illinois at Urbana Champaign, Urbana, IL 61801, USA.

In this work, we extend the approach of [4] and apply it to the domain of racing games. In [4], the best racing line is computed as a linear combination of MCP and SP. In contrast, our approach applies a simple heuristics to decompose the track into meaningful sections and exploits a genetic algorithm to find the best trade-offs for all the sections at once. Moreover, while [4] applies a formal model to estimate the lap-time, we follow a simulation-based approach and compute the actual lap-time using a non-player character. Our approach is general in that it can be applied to any racing game. In this work, as a testbed, we used The Open Racing Car Simulator (TORCS) [1], a state-of-the-art open source racing simulator. We compared our approach against [4] and the heuristic approach used by Simplix [3], the best controller available for TORCS. Our results on 11 tracks available in the distribution of TORCS show that our approach always outperforms the approach of Braghin et al. [4] and it outperforms Simplix in 8 out of 11 tracks.

II. Related Work

Finding the best racing line possible for a given car on a given track is one of the basic problems that developers of racing games have to solve. Commercial racing games typically rely on human-designed racing lines designed by domain experts [20]. The few notable exceptions to such a hand-crafted approach include Colin McRae Rally[1] (Codemasters), where the racing line is computed with a neural network trained by domain experts [15], and the Forza Motorsport[2] series (Microsoft), where the supervised learning techniques are exploited to train the game AIs and evolutionary computation is applied to optimize their racing lines [29]. Open-source racing games usually rely on heuristics approaches that typically combine good practices with heuristics to generate a racing line for any given track automatically. The most successful examples of such approaches include the K1999 algorithm [10], which was developed by Remi Coulom and exploits gradient descent; Simplix [3], developed by Wolf-Dieter Beelitz for The Open Car Racing Simulator (TORCS)[1], based on a simple heuristic; the bot by Jussi Pajala for the Robot Auto Racing Simulator (RARS) [2] applies A*; the `DougE1` bot for RARS by Doug Elenveld applies a genetic algorithm. Although based on different techniques, almost all the previous approaches exploit a very similar heuristic that aims at reducing the curvature of the racing line as much as possible. In fact, the smaller the curvature of the racing line is, the higher the speed that can be achieved without losing grip. On the other hand, Casanova

[1] `http://en.wikipedia.org/wiki/Colin_McRae_Rally`
[2] `http://en.wikipedia.org/wiki/Forza_Motorsport`

978-1-4244-6297-1/10/$26.00 ©2010 IEEE

[11] showed that to find the best racing line several aspects of the car dynamics must be taken into account, e.g., the braking points over the track, the acceleration capabilities of the car, the changes of direction, etc. Although very powerful, the approach in [11] requires a complete and very detailed formal model of the vehicles and of its interactions with the racing environment, which is typically unavailable in racing games. More recently, Braghin et al. [4] suggested that racing lines can be computed by solving the trade-off between minimizing the distance raced on the track and minimizing the curvature of the racing line followed. In particular, Braghin et al. [4] defined the problem of finding the best optimal racing line as a quadratic programming problem.

Finally, the problem of computing a racing line is also somehow related to the path and trajectory planning problem that has been widely studied in the fields of mobile robotics and autonomous vehicles [16], [27], [17], [18], [19], [12], [26], [23], [9]. However, although the mathematical formulation of the problem introduced in these works can be generally applied also to the problem studied in this paper, they are mainly concerned with satisfying the geometrical constraints on the generated paths rather than their optimality in terms of time.

III. Background

In this section we provide the definitions and the notation used in the remainder of the paper. First, we describe how the racing line is represented. Then, we introduce the shortest path (SP) along with the definition of the quadratic programming problem used to compute it. Similarly, the minimum curvature path (MCP) is introduced in the final part of the section.

A. Racing Line Representation

A racing line is defined as a sequence $\mathbf{P}_1, \ldots, \mathbf{P}_n$ of points on the track, where the last point $\mathbf{P}_n$ is connected to the first one $\mathbf{P}_1$. A convenient way to represent each point $\mathbf{P}_i$ of the racing line is by using the pair $\langle \delta_i, \alpha_i \rangle$, where α_i is the lateral distance of the point from one of the track borders (e.g., from the right border), normalized by the track width W; δ is the distance of the point from the track starting line, computed along the track axis. In this work, following the approach of [4], we also assume that the points of a racing line are equally spaced, that is, for every $i \in \{1, \ldots, n-1\}$, $\delta_{i+1} - \delta_i$ is constant. Accordingly, given the number of points n, the racing line is completely identified by the vector $\boldsymbol{\alpha}$ of lateral distances ($\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle^T$). An example of this representation is depicted in Figure 1.

B. Shortest Path

The *shortest path* (SP) is defined as the sequence of points that allows to complete one lap by racing the least distance possible. As suggested in [4], a good approximation of the SP can be computed by minimizing the following objective:

$$\tilde{S}^2 = \sum_{i=1}^{n} d(\mathbf{P}_{i+1} - \mathbf{P}_i), \tag{1}$$

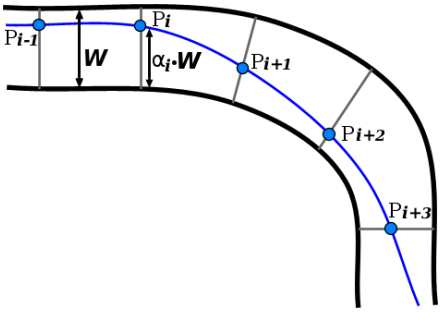


Fig. 1. Representation of a Racing Line.

where $\mathbf{P}_i$ is the i-th point of the path; $d(\mathbf{P}_{i+1} - \mathbf{P}_i)$ is the distance between $\mathbf{P}_{i+1}$ and $\mathbf{P}_i$ in the cartesian space. Since $\tilde{S}^2$ depends solely on the path points and on the track edges, Equation 1 can be rewritten as [4],

$$\tilde{S}^2 = \boldsymbol{\alpha}^T \mathbf{H_S} \boldsymbol{\alpha} + \mathbf{B_S} \boldsymbol{\alpha} + \text{cost}, \tag{2}$$

where $\mathbf{H_S}$ is a $n \times n$ matrix and $\mathbf{B_S}$ is a $1 \times n$ vector that depend only on the track edges coordinates. Accordingly, the shortest path SP can be computed by solving the following quadratic programming problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \boldsymbol{\alpha}^T \mathbf{H_S} \boldsymbol{\alpha} + \mathbf{B_S} \boldsymbol{\alpha} \\ & 0 \leq \alpha_i \leq 1 \quad \forall i \in \{1, \cdots, n\}. \end{aligned} \tag{3}$$

C. Minimum Curvature Path

The *minimum curvature path* (MCP) is defined as the sequence of points that allows to complete one lap following the racing line with the least curvature possible. The curvature of a track stretch is computed as the inverse of curvature radius of the same stretch. Curvature is a major factor in the determination of the optimal racing line. In fact, the smaller the curvature (i.e., the bigger the radius), the higher the speed that the car can maintain along the racing line according to the following relation:

$$v_{\max} = \sqrt{\mu \rho \left(g + \frac{F_a}{m} \right)}, \tag{4}$$

where $v_{\max}$ is the maximum speed sustainable (a higher speed would result in loosing the grip), m is the mass of the car, μ is the tire-road friction coefficient, ρ is the curvature radius, and F_a the aerodynamics downforce. Following [4], we can compute MCP, as the sequence of points that minimizes $\tilde{\Gamma}^2$,

$$\tilde{\Gamma}^2 = \sum_{i=1}^{n} \tilde{\Gamma}_i^{\,2}, \tag{5}$$

where $\tilde{\Gamma}_i$ represents the curvature in point $\mathbf{P}_i$ of the cubic spline that interpolates the racing line between $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$. The racing line that minimizes $\tilde{\Gamma}^2$ can be found by solving the following quadratic programming problem [4]:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \boldsymbol{\alpha}^T \mathbf{H_\Gamma} \boldsymbol{\alpha} + \mathbf{B_\Gamma} \boldsymbol{\alpha} \\ & 0 \leq \alpha_i \leq 1 \quad \forall i \in \{1, \cdots, n\}, \end{aligned} \tag{6}$$

where $\boldsymbol{\alpha}$ is the vector of lateral distances α_i that completely defines the racing line; $\mathbf{H_C}$ is a $n \times n$ matrix and $\mathbf{B_C}$ is a $1 \times n$ vector that can be computed from the track edges coordinates.

IV. Racing Line Optimization With GA

Neither the shortest path nor the minimum curvature path are usually the best possible racing line. In fact, as pointed out in [4], the optimal racing line is a trade-off between the shortest path SP and the minimum curvature path MCP. In particular, Braghin et al. assumed that the best racing line can be computed as [4],

$$\boldsymbol{\alpha} = (1 - \varepsilon) \cdot \boldsymbol{\alpha_{MCP}} + \varepsilon \cdot \boldsymbol{\alpha_{SP}}, \quad (7)$$

where α_{MCP} defines the minimum curvature path, α_{SP} defines the shortest path, and $0 \leq \varepsilon \leq 1$ is used to weight the two solutions. To find the best value of the weight ε, that depends on the car used as well as on the track considered, Braghin et al., applied a simple grid search: for a number of values of ε uniformly sampled in the interval $[0, 1]$, the corresponding racing line is evaluated computing a lap-time estimate. At the end, the best racing line found along with the corresponding weight ε^* are returned. The main limitation of this approach is that a single value of ε does not allow to find a different trade-off in different parts of the track (e.g., it is not possible to weight more the SP in the slowest parts of the tracks while weighting more the MCP in the fastest one). Accordingly, we extended the approach introduced in [4] in several respects: (i) we introduced a weight ε_{t_j} for each section t_j of the track; (ii) a genetic algorithm is applied to optimize the values of the ε_{t_j}; (iii) evolved solutions are evaluated through a simulation process rather than on the basis of a lap-time estimate.

In the following of this section we provide additional details about the solution encoding and about the solution evaluation process used in our approach.

A. Solution Encoding

To apply a genetic algorithm to search for the best racing line, we decomposed the track into several sections $t_1, \cdots, t_m$. The start and the end points of each section of the track are identified by the intersections between the shortest path and the minimum curvature path. Thus, the section t_j of the track is defined as the part of the track between the $j-th$ and the $(j+1)-th$ intersections between the shortest path and the minimum curvature path. Figure 2 shows an example of such a track decomposition process.

On the basis of the track decomposition introduced above, each candidate solution is simply encoded as a vector of weights $\varepsilon_{t_1}, \cdots, \varepsilon_{t_m}$, where ε_{t_j} regulates the convex combination between the shortest path and minimum curvature path in the section t_j of the track. Accordingly, the genetic algorithm is applied to search for the best trade-off between the shortest path and the minimum curvature path in each section of the track. At the same time, with respect to using a weight ε_i for each point P_i of the racing line, our approach allows to reduce dramatically the size of the search space.

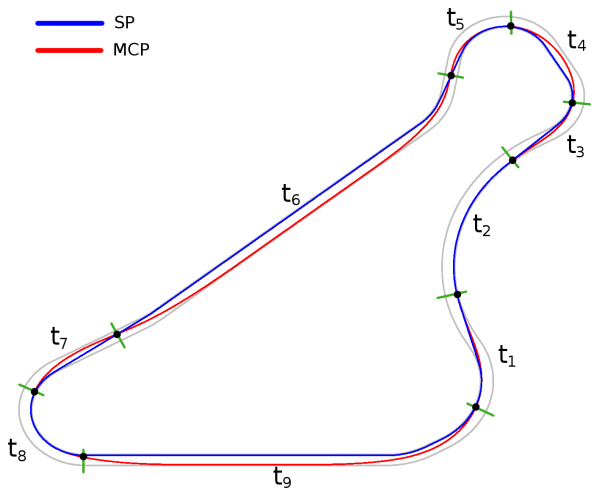


Fig. 2. An example of track decomposition process based on the intersection between the shortest path SP (reported as a blue line) and the minimum curvature path MCP (reported as a red line).

In addition, it is worthwhile to stress that the encoding introduced also guarantees that the racing line are always continuous. In fact, all the changes to the weight values happen only in correspondence of the intersections between the SP and MCP, i.e., where the shortest path and minimum curvature path are the same racing line.

B. Solution Evaluation

To evaluate a candidate solution a two steps process is performed. First, the genome of the solution is *decoded* to compute the racing line as follows. For each section t_j of the track, the shortest path and the minimum curvature path are combined according to the weight ε_{t_j} as,

$$\alpha_i = \alpha_{SP,i} \cdot \varepsilon_{t_j} + \alpha_{MCP,i} \cdot (1 - \varepsilon_{t_j}) \quad \forall \mathbf{P_i} \in t_j, \quad (8)$$

where α_i is the lateral deviation of the $i-th$ point $\mathbf{P}_i$ of the computed racing line (see Section III); $\alpha_{MCP,i}$ and $\alpha_{SP,i}$ are the lateral deviation of the $i-th$ point of the shortest path and of the minimum curvature path. Then, to evaluate the resulting racing line a simulation process is performed; that is, the fitness is computed on the basis of the lap-time achieved by following the racing line in the target racing simulator [3].

V. Experimental Setup

In this paper, we used as testbed The Open Racing Car Simulator (TORCS) [1], a state-of-the-art open source car racing simulator. TORCS provides several tracks, several cars, and a rather sophisticated physics engine, which takes into consideration many aspects of the racing car dynamics. In addition, several works recently focused on TORCS

[3] Although in this work we focused on The Open Racing Car Simulator (TORCS), an open source racing simulator, our approach can be applied to any racing game.

(e.g.,[5], [6], [7], [8], [14], [30], [22], [24], [25] and it has been also used in several scientific competitions [21].

In this section we provide additional details on the design of the experimental analysis presented in this paper.

Tracks. Table I shows the 11 tracks used in our experimental analysis. All the tracks are available in the standard distribution of TORCS and include representatives of very different types of track: two speedways (A-Speedway and Michigan Speedway), several technical and complex tracks (e.g., Aalborg, Alpine 1 and 2), very fast tracks (e.g., Forza, Wheel 1, CG Speedway) and narrow tracks with hills and bumps (i.e., Olethros Road and Ruudskogen).

Computing SP and MCP. For each track used in our experimental analysis (see Table I) we computed the SP and MCP as follows. In TORCS the track is represented as a sequence of segments, which can be either straights, left turns or a right turns. Each straight is completely defined by two parameters: the *width* of the track along the segment and the *length* of the segment. Turns have a similar representation: the *width* of the track, the *arc* covered by the turn and its *radius*. In addition, for each straight as well as for each turn, the coordinates of the four corners of the segment are stored in a dedicated data structure. Accordingly, we computed the coordinates of the track borders following a straightforward iterative process: for each segment of the track we computed the equation of the track borders exploiting the parameters about the current segment (i.e., *length*, *width*, *arc*, and *radius*) and, starting from the bottom corners of the segment, we uniformly sampled the track borders. Finally, with the coordinates of the track borders we computed H_Γ, H_S, B_Γ, and B_S (see Section III) and solved the related quadratic programming problem using the MATLAB Optimization Toolbox [4].

Evaluation Process. In this work we followed a simulation approach to evaluate the performance of the racing lines evolved with the genetic algorithm. We developed a *follower bot* based on a slightly modified version of the Simplix bot [3] that is able to follow any given racing line. Accordingly, the evaluation process is a two-laps simulation in TORCS of a *follower bot* that follows the racing line to evaluate. The performance is measured as the lap-time achieved by the *follower bot* in the second lap.

VI. Experimental Results

In this section we present three sets of experiments, where we applied the grid search, our approach and the Simplix bot to find the best racing line in 11 tracks (see Table I) available in TORCS.

A. Grid Search

In the first set of experiments, we applied a grid search to find the best racing line in the 11 tracks reported in Table I. The grid search has been performed sampling 100 values of the parameter ε between 0 and 1 using an uniform step of size 0.01. For each value of ε, the resulting racing line has been computed according to the Equation 7 and evaluated in TORCS (as described in the previous section). Table II shows the results of grid search: the column ε^* reports the best value of ε discovered, the column `Lap-Time` reports the performance of the corresponding racing line (measured as the lap-time). It can be noticed that, except for the CG Speedway and the Olethros tracks, the best weights ε^* found are either equal or very close to zero, i.e., the best convex combination of the shortest path and of the minimum curvature path is much more close to the latter. This is not surprising and support the common experience that suggests that the MCP is a very effective racing line. These results also confirm the findings reported in [4].

TABLE II
Results of the grid search applied to find the best racing line in 11 tracks.

Track	ε^*	Lap-Time (s)
Aalborg	0	70.693996
Alpine 1	0	122.544002
Alpine 2	0	93.075997
A-Speedway	0	25.137999
Forza	0	85.686000
CG Speedway	0.33	39.793999
Michigan Speedway	0.04	33.890000
Olethros Road	0.25	112.925999
Ruudskogen	0	63.208003
Street 1	0	76.124002
Wheel 1	0	75.406003

B. Genetic Algorithm

In the second set of experiments, we applied our approach (see Section IV) to the same 11 tracks of the previous experiment. To this purpose we used the C++ Genetic Algorithm Toolbox [28] with the following parameters setting: the population size was set to 30; tournament selection without replacement was used and the tournament size was set to 2; one point crossover is applied with probability $p_\chi = 0.9$ and each gene is mutated using a Gaussian mutation with probability $p_m = 0.1$. For each track we performed 10 runs and each run was stopped after 100 generations.

Table III shows the performance achieved from our approach on each track. In particular, the performance of our approach is reported by the column *GA*, the column *MPC* reports the performance of the minimum curvature path, and the column *Gap* reports the difference between the performance achieved with our approach and the performance of the minimum curvature path. The results show that the evolved racing line is always better than the MCP in all the tracks considered, whereas the linear combination of the SP and the MCP with a single weight ε^* found with a simple grid search, resulted in a better racing line only in 3 tracks out of 11 (Table II). In addition , the improvements achieved with the proposed approach is not negligible with respect to the MCP: as the Table III shows, following the racing lines evolved with our approach leads to improving the lap-times

[4] `http://www.mathworks.it/products/optimization/`

TABLE I

TRACKS USED IN THE EXPERIMENTAL ANALYSIS.

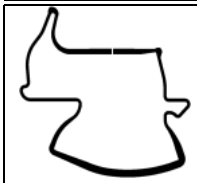
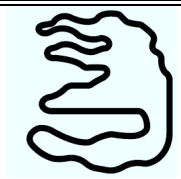
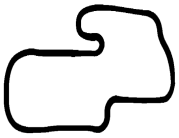
Aalborg	Alpine 1	Alpine 2	A-Speedway	Forza	Michigan Speedway
					
CG Speedway	**Olethros Road**	**Ruudskogen**	**Street 1**	**Wheel 1**	
					

TABLE III

PERFORMANCE OF OUR APPROACH APPLIED TO FIND THE BEST RACING LINE IN 11 TRACKS. STATISTICS ARE AVERAGED OVER 10 RUNS.

Track	GA (s)	MCP (s)	Gap (s)
Aalborg	69.928 ± 0.023	70.693996	0.766
Alpine 1	121.481 ± 0.012	122.544002	1.063
Alpine 2	92.527 ± 0.017	93.075997	0.549
A-Speedway	24.701 ± 0.004	25.137999	0.437
Forza	85.210 ± 0.002	85.686000	0.476
CG Speedway	39.372 ± 0.003	40.132001	0.760
Michigan Speedway	33.866 ± 0.001	33.918002	0.052
Olethros Road	111.656 ± 0.068	112.984002	1.328
Ruudskogen	62.732 ± 0.007	63.208003	0.476
Street 1	75.613 ± 0.080	76.124002	0.510
Wheel 1	74.887 ± 0.109	75.406003	0.519

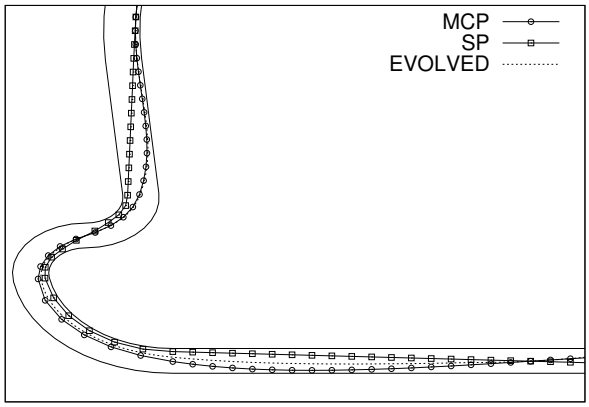


Fig. 3. Comparison of the racing line evolved with our approach on a section of the Aalborg track (reported as `EVOLVED`), the SP (reported as `SP`) and the MCP (reported as `MCP`).

of the MCP from an half of a second up to more than one second (the only exception being the Michigan Speedway). Figure 3 shows as an example a short excerpt of the racing line evolved with our approach on the Aalborg track. From this simple example it is possible to understand why our approach is more effective than using a linear combination between the SP and the MCP as proposed in [4]: while in the first segment reported (i.e, the straight at bottom of Figure 3) the evolved racing line (the dashed line) is in the middle between the SP and the MCP, in the second segment (starting from the middle of the tight chicane) the evolved racing line coincides almost perfectly with the MCP. Accordingly, the evolved racing line allows at the same time to race a smaller distance in the straight (as it is not possible to exploit a very small curvature due to the subsequent tight turn) and to exploit the MCP in the second part of the chicane to maximize the exit speed.

C. Simplix

Finally, in the last set of experiments we applied the Simplix bot [3], a state of the art controller in the TORCS development community, to 11 tracks used also in the previous experiments. Table IV reports the performance of Simplix (`Simplix`) for each track and compares it with the ones achieved from our approach (`GA`), the minimum curvature path (`MCP`), and the best linear combination of the SP and the MCP (`Grid Search`). The results shows that Simplix is outperformed by our approach in 8 tracks out of 11, achieving a lap-time up to almost 3 seconds slower (e.g., in Olethros Road and A-Speedway) than the one registered with our approach. However, Simplix clearly outperforms our approach in Wheel 1 and in Street 1 tracks, where also the performance achieved by the minimum curvature path is clearly worse than the one of Simplix. Accordingly, even the GA optimization step exploited in our approach is not able to fill such a quite large gap. Such a difference can be explained as follows. Since the minimum curvature path aims

TABLE IV

COMPARISON OF THE DIFFERENT APPROACHES APPLIED TO COMPUTE THE RACING LINE. PERFORMANCE IS MEASURED AS THE LAP-TIME ACHIEVED (IN SECONDS).

Track	GA	MCP	Grid Search	Simplix
Aalborg	$\mathbf{69.928 \pm 0.023}$	70.694	70.694	70.762
Alpine 1	$\mathbf{121.481 \pm 0.012}$	122.544	122.544	122.876
Alpine 2	$\mathbf{92.527 \pm 0.017}$	93.076	93.076	94.334
A-Speedway	$\mathbf{24.701 \pm 0.004}$	25.138	25.138	27.558
Forza	$\mathbf{85.210 \pm 0.002}$	85.686	85.686	86.608
CG Speedway	$\mathbf{39.372 \pm 0.003}$	40.132	39.794	40.120
Michigan Speedway	33.866 ± 0.001	33.918	33.890	**33.742**
Olethros Road	$\mathbf{111.656 \pm 0.068}$	112.984	112.926	114.630
Ruudskogen	$\mathbf{62.732 \pm 0.007}$	63.208	63.208	63.206
Street 1	75.613 ± 0.080	76.124	76.124	**74.680**
Wheel 1	74.887 ± 0.109	75.406	75.406	**73.924**

to a *global* minimization of the racing line curvature, it does not guarantee that the curvature of the racing line computed by Simplix is bigger in any point of the track. Accordingly, in some particular tracks the racing line followed by Simplix might have a lower curvature in some critical points of the track and thus resulting in a better lap-time.

VII. CONCLUSIONS

In this paper we introduced a novel approach to find the best racing line for a given track and car. In particular, we extended the approach introduced by Braghin et al. in [4] for real cars and we applied it to the domain of car racing games. To find the best racing line, we decomposed the track into several segments, where a genetic algorithm is applied to search for the best trade-off between the shortest path and the minimum curvature path. Each solution candidate is then evaluated through a simulation process preformed directly in TORCS, an open-source racing simulator used as a testbed in this work. To test our approach we applied it to 11 tracks that are provided with the TORCS distribution. The results obtained are very promising and showed that our approach always outperforms the approach of Braghin et al. and the Simplix bot [3] , one of the most successful controller developed for TORCS, in 8 tracks out of 11.

Future works include co-evolving the best car parameters along with the best racing line and exploits multi-objective evolutionary computation instead of searching for the best trade-off between the two objectives.

REFERENCES

[1] The open racing car simulator. `http://torcs.sourceforge.net/`.

[2] Robot auto racing simulator. `http://rars.sourceforge.net/`.

[3] Wolf-Dieter Beelitz. The SIMPLy mIXed best practice TORCS robot. `http://www.wdbee.gotdns.org:8086/SIMPLIX/SimplixDefault.aspx`.

[4] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni. Race driver model. *Comput. Struct.*, 86(13-14):1503–1516, 2008.

[5] Martin V. Butz and Thies D. Lonneker. Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 317–324, Sept. 2009.

[6] L. Cardamone, D. Loiacono, and P.L. Lanzi. On-line neuroevolution applied to the open racing car simulator. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2622–2629, May 2009.

[7] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179–1186, New York, NY, USA, 2009. ACM.

[8] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Learning drivers for torcs through imitation using supervised methods. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 148–155, Sept. 2009.

[9] T. A. de A Costa. Trajectory generation with curvature constraint based on energy minimization. In *ABCM Symposium Series in Mechatronics*, volume 3, pages 300–307, 2008.

[10] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[11] R.S. Sharp D. Casanova. *On minimum time vehicle manoeuvring: the theoretical optimal time.* PhD thesis, Cranfield University, 2000.

[12] H. Delingette, Martial Hebert, and Katsushi Ikeuchi. Trajectory generation with curvature constraint based on energy minimization. In *Proceedings 1991 IEEE/RSJ International Conference On Intelligent Robotic Systems (IROS '91)*, November 1991.

[13] Christos Dimitrakakis. Online statistical estimation for vehicle control. IDIAP-RR 13, IDIAP, 2006.

[14] Marc Ebner and Thorsten Tiede. Evolving driving controllers using genetic programming. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 279–286, Sept. 2009.

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

[16] Y. Kanayama and N. Miyake. Trajectory generation for mobile robots. In *In Proc. of the Int. Symp. on Robotics Research*, 1985.

[17] Y. Kanayama, A. Nilipour, and C. Lelm. A locomotion control method for autonomous vehicles. In *ICRA*, pages 1315–1317, 1988.

[18] Yutaka J. Kanayama and Bruce I. Hartman. Smooth local-path planning for autonomous vehicles. *Int. J. Rob. Res.*, 16(3):263–284, 1997.

[19] K. Komoriya and K. Tanie. Trajectory design and control of a wheel-type mobile robot using b-spline curve. In *Intelligent Robots and Systems '89. The Autonomous Mobile Robots and Its Applications. IROS '89. Proceedings., IEEE/RSJ International Workshop on*, pages 398 –405, sep 1989.

[20] Stefano Lecchi. Artificial intelligence in racing games. In *CIG'09: Proceedings of the 5th international conference on Computational*

Intelligence and Games, pages 1–1, Piscataway, NJ, USA, 2009. IEEE Press.

[21] D. Loiacono, J. Togelius, P.L. Lanzi, L. Kinnaird-Heether, S.M. Lucas, M. Simmerson, D. Perez, R.G. Reynolds, and Y. Saez. The wcci 2008 simulated car racing competition. In *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, pages 119–126, Dec. 2008.

[22] Jorge Munoz, German Gutierrez, and Araceli Sanchis. Controller for torcs created by imitation. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 271–278, Sept. 2009.

[23] B. Nagy and A. Kelly. Trajectory generation for car-like robots using cubic curvature polynomials. *Field and Service Robots 2001*, 2001.

[24] E. Onieva, D. A. Pelta, J. Alonso, V. Milanes, and J. Perez. A modular parametric architecture for the torcs racing engine. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 256–262, Sept. 2009.

[25] Diego Perez, Gustavo Recio, and Yago Saez. Evolving a fuzzy controller for a car racing competition. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 263–270, Sept. 2009.

[26] F.G. Pin and H.A. Vaaseur. Autonomous trajectory generation for mobile robots with non-holonomic and steering angle constraints. In *Intelligent Motion Control, 1990. Proceedings of the IEEE International Workshop on*, volume 1, pages 295 –299, aug 1990.

[27] Alireza Sahraei, Mohammad Taghi Manzuri, Mohammad Reza Razvan, Masoud Tajfard, and Saman Khoshbakht. Real-time trajectory generation for mobile robots. In *AI*IA '07: Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence on AI*IA 2007*, pages 459–470, Berlin, Heidelberg, 2007. Springer-Verlag.

[28] K. Sastry. Single and multiobjective genetic algorithm toolbox for matlab in c++. Technical Report 2007017, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 S. Mathews Avenue Urbana, 2007.

[29] David Stern and Joaquin Qui nonero Candela. Playing machines: Machine learning applications in computer games. In *CIG'09: Proceedings of the 5th international conference on Computational Intelligence and Games*, pages 1–1, Piscataway, NJ, USA, 2009. IEEE Press.

[30] N. van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber. Robust player imitation using multiobjective evolution. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 652–659, May 2009.