Genet Program Evolvable Mach (2014) 15:433–476
DOI 10.1007/s10710-014-9227-z

Driving as a human: a track learning based adaptable architecture for a car racing controller

Jan Quadflieg · Mike Preuss · Günter Rudolph

Received: 28 October 2013 / Revised: 16 May 2014 / Published online: 19 June 2014

Abstract We present the evolution and current state of the Mr. Racer car racing controller that excelled at the corresponding TORCS competitions of the last years. Although several heuristics and black-box optimization methods are employed, the basic idea of the controller architecture has been to take over many of the mechanisms human racing drivers apply. They learn the track geometry, plan ahead, and wherever necessary, adapt their plans to the current circumstances quickly. Mr. Racer consists of several modules that have partly been adapted and optimized separately, where the final tuning is usually done with respect to a certain racing track during the warmup phase of the TORCS competitions. We also undertake an experimental evaluation that investigates how the controller profits from adding some of the modules to a basic configuration and which modules are most important for reaching the best possible performance.

Keywords Car racing · Evolutionary computation · TORCS · Planning controller

J. Quadflieg (✉) · G. Rudolph
Chair of Algorithm Engineering, Computational Intelligence Group, Department of Computer Science, Technische Universität Dortmund, Dortmund, Germany
e-mail: jan.quadflieg@cs.tu-dortmund.de

G. Rudolph
e-mail: guenter.rudolph@tu-dortmund.de

M. Preuss
European Research Center for Information Systems (ERCIS), WWU Münster, Münster, Germany
e-mail: mike.preuss@uni-muenster.de

1 Introduction

There may be many different approaches to realizing a complex car racing controller that is able to cope well with different track and surface types and can also deal with opponent cars. Even with perfect information, this would not be trivial. However, within the TORCS competition environment, only partial information is available, which furthermore complicates the task.

When envisioning a controller architecture, one may think along two different lines, namely the degree of modularization and the amount of expert knowledge that shall be integrated, in contrast to utilizing black-box learning. From the start, it appeared most straightforward to us to shape the modules of the Mr. Racer controller similar to the different actions performed by a human driver and thereby make the individual parts relatively independent. However, the basis of all the high-level modules is formed by a track model. While there are other controllers out that drive completely without track model or with a very rough one, it has always been a fundamental part of the design philosophy of Mr. Racer to establish a most accurate track model and use it for deciding about specific target speeds, positions, and the possibility of special actions as overtaking. We do not claim that our approach is per se better than more machine-learning oriented ones; many ways may lead to very similar outcomes. However, it is well suited to think about racing as human drivers do, and learn from them where possible. For the remaining uncertainties in the design, we mostly applied heuristics that were tuned by means of black-box optimization methods.

Despite the fact that Mr. Racer won the TORCS Simulated Car Racing Championship (SCRC) from 2011 to 2013 (the Autopia controller [25] is still comparable in performance, but was not submitted to the competition in 2011, 2012 and 2013, thus it could not win), no complete description of its structure and mechanisms exists. From our first related publication [30] on, many details have changed, although the general structure has remained the same. The first task of this paper is thus to provide a consistent up-to-date description of the controller, and report on the changes done since 2010. This description is intended to be thoroughly enough to enable rebuilding the controller more or less from scratch.

Secondly, we go one step further and investigate how the newest, independently inserted or modified modules interact with each other and detect which combination of modules performs best. Most of these modules are internally optimized by a black-box evolutionary optimization method, the CMA-ES [2].[1] However, they have been designed largely independently, so that we cannot guarantee that enabling all modules at the same time leads to the best possible performance. Therefore, we experimentally compare the different module combinations. We emphasize that in general, our modules benefit from track knowledge provided by the track model. As examples, track knowledge helps us drive faster because we know if we are approaching a corner or exiting a corner (this cannot be distinguished from the current sensor data alone), allows to accelerate earlier than most other controllers, and enables positioning the car on the outside of a corner.

[1] See the tutorial [15] for a current description, as details have been changed meanwhile.

Thirdly, in order to find out about strengths and weaknesses of our controller in comparison to other recent competitors, we experimentally compare Mr. Racer with Autopia (Winner 2010), Mariscal & Fernández (2nd place 2012) and Ready2Win (3rd place 2012). This investigation is much more comprehensive than a single competition result as we take several tracks into account, not only three as done in one competition leg.

The remainder of this work is structured as follows: We give a short introduction to TORCS, the Simulated Car Racing Championship (Sect. 2) and related work (Sect. 3). We will present the architecture of the controller and details of the basic submodules in Sect. 4. The major submodule which creates the plan is described in great detail in its own Sect. 5. We will demonstrate how the performance of the base controller can be further improved (Sect. 6) and compare the advanced controller with the best controllers of the past championship (Sect. 7). We close with summary and conclusion.

2 TORCS

The open racing car simulator (TORCS) is an open 3D simulation environment that has been built in order to allow different AI controllers to drive against each other. We call these *native* bots because they use the programming interface provided by TORCS and are executed as part of the TORCS process. Figure 1 gives an impression of how the graphical display looks like. The Simulated Car Racing Championship for which the Mr. Racer bot was built has been started in 2008 and was held every year with only slight changes up to 2013. The official manual [18] provides details about the interface and winning conditions. As indicated in the right half of Fig. 1, the software interface of the Simulated Car Racing Championship provides a native TORCS bot which just acts as a server. The actual bot controlling the car connects to this server and has no direct access to the memory space of TORCS. Instead the controller perceives the current game state through a sensor model defined by the organizers of the competition. The communication via network happens in a fixed interval of 20 ms, the controller has 10 ms to send its response.

The biggest restriction imposed by the sensor model is the fact that a controller using the SCRC software interface cannot access the whole track. Instead, the controller only perceives the part of the track directly in front of the car by the information provided by the track sensors. These sensors provide the distance from the current position of the car to the track borders in steps of 10 degrees into the relative front direction (19 values). From 2011 on, the track sensor values contain artificially added relative noise, such that for an accurate track recognition, a noise filtering procedure has to be applied. Note that most other controllers simply use the longest sensor to detect a main direction and therefore do not need such filtering.

In addition to the track sensors, there are also opponent sensors that provide distances (in the same fashion as the track sensors) to the nearest cars. The other important values provided by the sensor model include the speed of the car v (where negative speeds indicate that the car is driving backwards), the normalized position

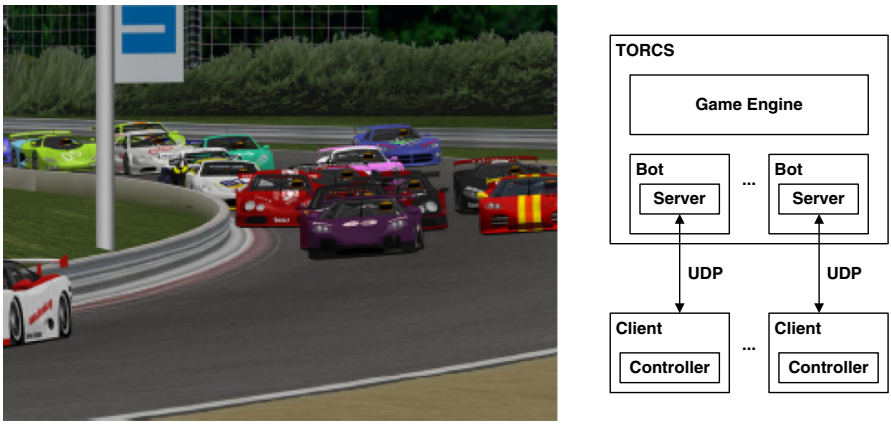


Fig. 1 TORCS screenshot with network-based game/controller interface as used in the context of the simulated car racing championship. The interface hides the internal state of the TORCS engine from controllers. Up to ten bots can connect at the same time

on the track τ (1 left edge, 0 center of the track, -1 right edge), the angle to the track axis ϕ and the distance from the start/finish line δ. As a response, the controller has to select which gear to use $(-1, 0, \ldots 6)$, the values for the virtual gas $(0 \ldots 1)$ and brake pedal $(0 \ldots 1)$ and the desired steering direction $(-1 \ldots 1)$. For more details, we refer to the official manual [18] which contains an in depth description of the sensors and actuators which are available to a TORCS bot in the context of the Simulated Car Racing Championship.

For each track of every leg in the Championship (three tracks were used for every leg, and usually, three legs were driven and points added, except for 2013, when only one leg was driven), a controller undergoes three stages, namely warmup, qualifying, and race. During warmup (10^5 gameticks $\approx$33 min), the controller drives alone on the track and has the chance to learn the track or try out different behaviors. The results of this stage are not used for the competition standings. In the qualifying stage (10^4 game tics or 200 s), the controller also drives alone and has to cover the greatest possible distance. The eight controllers with the largest distances qualify for the race, where all controllers drive together with some repeats. Further information can be found on the competition web page.[2]

In contrast to the convention in the competition manual, we always provide angles (e.g. the angle to track axis) in degrees, not radians. Furthermore, we use this notation: target values have a hat, e.g. target speed $\hat{v}$. A minimal value or lower bound is underlined, e.g. the minimal acceleration $\underline{a}$. A maximal value or upper bound has a bar, e.g. the maximal acceleration $\bar{a}$. Mirroring an input value means multiplying it by -1.

[2] http://cig.dei.polimi.it/?page_id=175.

3 Related work

The Simulated Car Racing Championship in its current form using TORCS has been established in 2008. Loiacono et al. [21] describe the general idea of the championship, the software interface developed for the competition, the methods used by the five entries and the results of the WCCI 2008 competition. This first championship already showed a wide variety of different controllers, from simple hand coded ones like the approach by Lucas to evolved neural networks (using NEAT [33]), submitted by Cardamone et al. [7, 9, 11]. Saez et al. [32] optimized the actions for a specific track using an evolutionary algorithm with astonishing results. The evolved controller drives faster than human drivers and faster than the bots provided by TORCS. But of course this approach does not generalize at all and the controller is only able to drive alone on the track used for the optimization process. Another early contribution to the TORCS problem has been published by Perez et al. [28] who discretize the possible sensor values and map the discrete values to the proper actions. The action values of this rule base have been optimized by an evolutionary algorithm. A major weakness of this first generation of controllers was the complete absence of a collision avoidance.

In 2009, the championship consisted of three legs run at three different conferences, with three races each (nine races overall), the results have been published by Loiacono et al. [19]. The 2009 edition was dominated by two controllers, Cobostar and Autopia. Cobostar by Butz and Lönneker uses direct mappings modeled by parameterized formulas. The 14 numerical parameters of the formulas have been successfully optimized with the CMA-ES [5]. A revised version of Cobostar entered to the last round of the 2009 competition by Butz, Linhardt and Lönneker has been upgraded with a simple collision avoidance [4]. Autopia was first submitted by Onieva et al. [26] and uses a TSK controller to derive a target speed based on the current sensor readings. Our controller Mr. Racer was submitted to the competition for the first time in 2009 as well [19]. Although only reaching a 5th place overall, Mr. Racer showed the potential of the track model when dominating the qualifying on the track Forza. Other controllers submitted to the 2009 competition have been described by Perez et al. [27] who use a rule base modeled with fuzzy logic (fourth place), Muñoz, Gutierrez and Sanchis [22] who use an artificial neural network trained with data gathered from a human player (7th place) and Ebner and Tiede [14] who used genetic programming with a tree representation (12th place). Similar to the approach by Muñoz et al., Cardamone, Loiacono and Lanzi discuss the usage of imitation learning based on data gathered form the Inferno bot [8].

The last major changes were introduced in the 2010 championship when artificial noise was added to the track sensors and the opponent sensors and the warmup stage was introduced. Since 2010 the organizers used tracks specifically created for the championship to make sure that none of the competitors could know the tracks in advance. In the following years, the Simulated Car Racing Championship has attracted a lot of attention and a wide variety of controllers has entered the competition, with performance differences between the best participators becoming

smaller and smaller every year. Although a lot of different groups participated in the championship not all of them published a description of the used techniques.

Uusitalo and Johansson [34] describe their entry Poowah (fourth place overall in the 2011 competition) which uses artificial potential fields. Agapitos et al. [1] employ genetic programming to coevolve a track representation and a controller which uses this representation to drive on a specific circuit.

Other later work often concentrated on specific aspects of a controller. Loiacono et al. applied reinforcement learning to improve the overtaking performance of a native TORCS bot [20]. Onieva et al. also tackled this specific problem in the context of the Autopia controller using fuzzy logic [24]. Cardamone et al. evolved racing lines for the native TORCS bot Simplix [13]. This work was later expanded by Botta et al. [3].

All the controllers can be divided into different groups according to different criteria, although we will see that the top performing ones fall into the same class nowadays. One criteria presented by Muñoz et al. [23] is whether or not the controller directly acts on the actuators. If it does, it is called a *direct controller*. The opposite is an *indirect controllers*, which first derives a target speed and a target position from the available sensor data and other information, eg. data collected during the warmup stage or the last lap. It then tries to reach the targets by working on the actuators. Another distinction is the exploitation of the warmup stage. In the 2012 competition, only the controller submitted by Ho Duc Thang ignored this, while all others used it to gather information about the track and fine tune their behavior.

Being the winner of the 2009 and 2010 championship, Autopia is probably the most prominent controller of the past years and can be regarded as the state of the art. Originally described by Onieva et al. [26], the most recent version is an indirect controller using fuzzy logic which also exploits the warmup stage [25].

Besides the Simulated Car Racing Championship, TORCS has been the subject of other competitions as well. The TORCS Endurance World Championship [36] is a competition for native TORCS bots. Cardamone et al. optimized the car setup for the Simplix bot and successfully tested the improved controller in the Endurance World Championship [10]. The problem to optimize the setup of a racing car has also been the subject of a competition in the scientific community [16].

Cardamone et al. have dealt with the question how the behavior learned in the context of TORCS can be used to ease the learning process in a different car racing game (VDrift) [6]. Finally, Loiacono et al. [17] and Cardamone et al. [12] developed methods to procedurally generate racetracks using evolutionary computation. The resulting circuits were used for the Simulated Car Racing Competition.

4 Controller architecture

4.1 Overview

The controller consists of nine high level modules. The separation in various modules with well defined functionality and clear interfaces makes this architecture

very flexible. It is easily possible to replace modules with more advanced versions, which we will demonstrate in the comparison between the base controller and the advanced controller in subsequent sections. Expert knowledge, if available, can be incorporated into various components while unknown relations between input and output of a component can be learnt. Due to this kind of flexibility it is easy to compare several models of the relations. Moreover, finding good parametrizations of the modules is facilitated because parameters of submodules can be learned or optimized independently. Figure 2 illustrates the controller's architecture and the flow of information between its modules.

In the presence of noise, the sensor data is first filtered according to our approach presented in [29]. That is, a moving average is performed on the sensor data and two regression polynomials are fitted to the average sensor values, one for each side of the track. The two polynomials are then resampled to gain the final, filtered sensor values. The filtering only takes place in the presence of noise, which can easily be detected by observing the sensor values during the few seconds before the start of the race. During this time, the car is standing perfectly still and without noise, the sensors deliver exactly the same values in each consecutive gametick. In contrast, differing sensor values are a clear sign of noisy sensors.

The (filtered) sensor values are then classified using the vector based approach from [30] and the resulting information is incorporated into the track model. Depending on the classification, we can decide whether the car is in a regular state or not. If the state is irregular, eg. because the car has left the track or got stuck at an obstacle, the recovery behavior is used. Otherwise, the planning module creates a plan based on the sensor information and the data stored in the track model. This plan contains a target speed and in the case of the advanced controller also a target position. The remaining modules work on the actuators to reach these targets. The

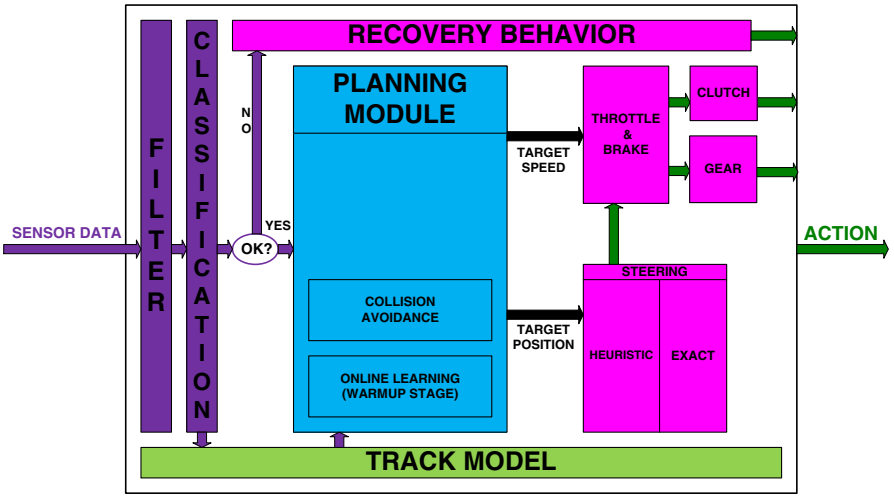


Fig. 2 Overview of the controller architecture. The controller receives a data packet containing the current sensor values from the competition server

interplay of throttle, brake, gear and clutch lead to the target speed, whereas the steering module moves the car to the target position. Details about these modules are given in the following sections.

4.2 Track segment classification

The track segment classification has always been one of the key features of our controller. It is based on the information provided by the 19 track sensors indicating the distances to the track borders ahead of the car for given angles. From the directions and lengths of these vectors it is possible to reconstruct an imminent segment of the upcoming race track by a linear interpolation [30]. Depending on the estimated degree ρ of the track segment's curvature the track segment is classified as a straight lane, full speed curve, medium speed curve, slow speed curve or hairpin curve. The approximated track segments and their classification are then used for building a model of the entire race track. If the car is not on the track and the track sensors deliver no data, the situation is classified as *OffTrack* triggering the activation of the recovery module.

4.3 Track model

Algorithmic details of our approach to learn a global model of the race track from the available sensor information using the classification from the previous section can be found in [30]. But because of the importance of the track model for the planning process described subsequently, we recap the most important aspects here.

We do not strive for reconstructing the exact outline of the track in form of a polygonal line, although this might be possible. Instead we aim for a more abstract representation which only contains information about the direction (left, right or straight) of the track and, in the case of a corner, its degree of curvature. Based on the classification described in the previous section, we divide the track into *segments* of the same direction. For each *segment*, we obviously store its direction, its beginning (as the distance from the starting line in meters) and its length (in meters). A segment can also have the state *unknown* if no information about this part of the track is available.

In the case of a corner, a segment consists of one or more *subsegments*. A *subsegment* is a part of the track which has been classified as the same type (*full*, *medium*, *slow* or *hairpin*). Subsegments surrounded by *faster* subsegments contain an apex. For those subsegments (with apex) we store the maximum absolute value of ρ seen during the creation of the track model. This value is then mapped to a target speed during the planning process.

The process of learning the track model exploits the *warmup* stage introduced with the 2010 competition. When the car crosses the finish line for the first time, the length of the track can be determined and the track model is initialized: One segment of the state *unknown* which covers the whole track is created and the width and length of the track are stored. During the first lap the car drives in the middle of the track with a slow speed of about 50 km/h. New information from the track segment classifier is incorporated into the track model, constantly reducing the

length of the unknown part until the model covers the whole track. Theoretically the learning could take more than one lap if the car leaves the track but in practice we never saw this happen. We can therefore assume that the track model is available after the first lap of the warmup in most cases and that it is certainly available during the qualifying and the actual race.

Figure 3 illustrates the structure of the track model by showing the first three segments of the track WHEEL2 on the left and a graphical representation of the corresponding segments from the track model on the right. The first segment is a straight part, starting at the finish line (0m) with a length of 307m. The second segment is a right hand corner starting at 307m with a length of 390m and seven subsegments: It begins with a subsegment of type full, followed by a subsegment of type medium which represents the first apex of this corner. Not shown in the illustration is the maximum absolute value of ρ recorded for this subsegment which is 24.65°. The fifth subsegment of type slow represents the second apex of this corner with recorded maximum value for $\rho = 55.86$. The third segment represents a left hand corner consisting of three subsegments and only one apex. The complete track model generated for WHEEL2 consists of 29 segments.

Since the estimated track width *tw* per segment is stored in the model the track model also offers the opportunity to calculate the absolute track position τ_{abs} measured in meters with the left edge at zero meters and the right edge at *tw* meters. Since the normalized position τ given by the sensor model indicates the left edge of the track segment with $\tau = 1$ and the right edge with $\tau = -1$ the absolute track position is determined via

$$\tau_{abs} = tw - \frac{\tau + 1}{2}\, tw = \frac{1 - \tau}{2}\, tw\,.$$

Taking into account the width of the car (ca. 2.2 m) it is now possible to decide if the car is actually on or off the track. Since the normalized track position τ provided by the sensor model references the *center* of the car and is normalized to the width of the race track, $|\tau| > 1$ actually means that at least half the car already left the track, which might be problematic. The absolute track position provides more precise information and both steering modules and the online learning module take advantage of this. The other modules do not for historical reasons: They have been developed before the track model became available. These modules might benefit from the more precise information as well but due to time constraints, they have never been adapted to use the absolute track position τ_{abs}.

4.4 Recovery behavior

The recovery behavior is used in situations not covered by the normal driving behavior. It is activated if

1. the car is not on the track ($|\tau| \geq 1$) or
2. the absolute angle between the car and the track axis is larger than 45° or
3. the speed is smaller than −0.15 km/h or
4. the car is stuck.

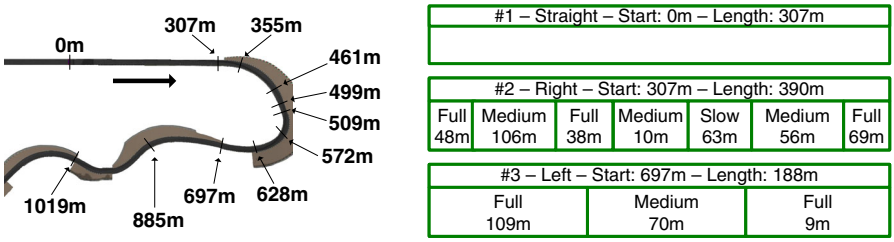


Fig. 3 *Left* birds eye view of the first three segments of the track WHEEL2. *Right* graphical representation of the corresponding segments from the track model

While on the track, the global stuck detection constantly checks if the car is not accelerating properly ($v_t < 5\text{km/h} \wedge v_t - v_{t-1} \leq 0.25\text{km/h}$) and activates the recovery behavior if necessary. As soon as the car is back on the track and the speed is higher than -0.15 km/h and the absolute track angle between the car and the track axis is smaller than 9° the recovery behavior is deactivated and the controller returns to using the normal driving behavior.

Figure 4 shows the architecture of the recovery behavior, which has been mainly designed, implemented, and parametrized using expert knowledge. Two subcomponents are used to handle special cases: The *OffTrack* behavior gets the car back on the track and the *OnTrackTurning* component turns the car around if it is on the track but facing the wrong direction.

The recovery behavior uses the well known heuristic to steer in the direction of the track sensor with the highest value, which leads to the steering outputs shown in Table 1. The unfiltered sensor values are used to determine which index to use. As can be seen in the table, we multiply the output by two for slow corners and by three for hairpins. This proved to be necessary after initial tests, which showed that the basic heuristic does not handle tight corners very well. Simple rules are used for the acceleration and the brake. The car fully accelerates up to 68 km/h. If it is faster, it starts to brake ($b = 0.5$). It fully brakes when driving faster than 150 km/h. The standard gear change behavior described in Sect. 4.7 is used to determine the output for the gear. The clutch is always set to 0.

The recovery behavior delegates its actions to the *OffTrack* behavior when the car is not on the track ($|\tau| > 1$). The *OnTrackTurning* component is used when the car is stuck or heading in the wrong direction ($|\phi| > 80°$).

4.4.1 *OffTrack behavior*

The *OffTrack* behavior takes advantage of the fact that the behavior has to be modeled for only one side of the road. Should the car leave the track on the other side, the same behavior can be used by mirroring the input value for the angle to track axis and mirroring the output value for the steering. We here describe the behavior for the left side of the track. If the car is facing away from the track ($\phi < 0$) or is stuck, it drives backwards otherwise forwards. The behavior has its own rulebase to recognize if the car is stuck: It constantly checks if the speed hasn't

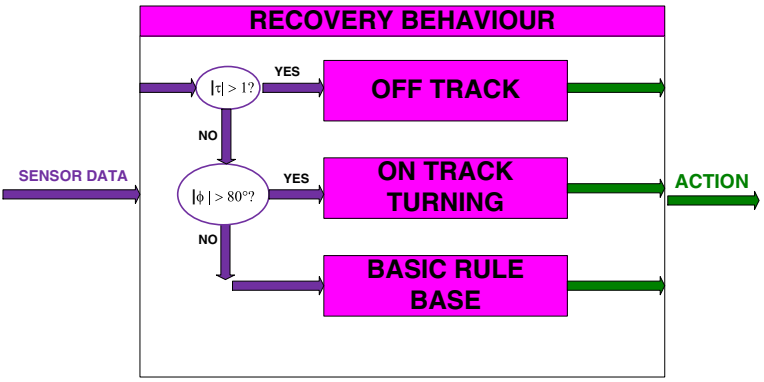


Fig. 4 The recovery module, which consists of three submodules

Table 1 Steering output s of the recovery behavior, depending on the track classification and the index of the track sensor with the highest value

Sensor	0–4	5	6	7	8	9	10	11	12	13	14–18
$s_{hairpin}$	1	1	1	1	0.66	0	−0.66	−1	−1	−1	−1
s_{slow}	1	1	1	0.89	0.44	0	−0.44	−0.89	−1	−1	−1
s_{else}	1	0.89	0.67	0.44	0.22	0	−0.22	−0.44	−0.67	−0.89	−1

increased between gameticks while driving slower than 10 km/h or if the difference. If the former condition has been true for 50 consecutive gameticks, the car is considered stuck and the *OffTrack* behavior switches to driving backwards. While driving backwards, the stuck detection is inactive. The outputs for the steering s, the acceleration a, the brake b and the gear g are as follows.

Forward We can assume that $\phi \geq 0$ because otherwise the car would be driving backwards. The angle $\hat{\phi}_f$ is the optimized heading the car tries to reach. At first, the steering s is determined via

$$s = \begin{cases} 1 & \phi > \hat{\phi}_f + 90 \\ \dfrac{\phi - \hat{\phi}_f}{90} & otherwise \end{cases} \tag{1}$$

ensuring that the steering is zero if the car is going into the right direction. The acceleration depends on the steering and is set to

$$a = \underline{a}_f + (\bar{a}_f - \underline{a}_f) * (1 - |s|)$$

if $-1 \leq v \leq 50$, and $a = 0$ otherwise.

The car brakes with a value of 0.5 if the speed is lower than -1 km/h. If it is faster, the value for the brake output is set to zero. The output for the clutch is always 0, the gear is always 1.

Backward When driving backwards, the car tries to reach the heading $\hat{\phi}_b$ using the steering

$$s = \begin{cases} 1 & \phi \leq \hat{\phi}_b - 90 \\ -1 & \phi > \hat{\phi}_b + 90 \\ -\dfrac{\phi - \hat{\phi}_b}{90} & else \end{cases} \tag{2}$$

Again, the acceleration depends on the calculated steering angle and the optimized values for the minimal ($\underline{a}_b$) and maximal ($\bar{a}_b$) acceleration and is set to

$$a = \underline{a}_b + (\bar{a}_b - \underline{a}_b) * (1 - |s|)$$

if $-50 \leq v \leq 1$, and $a = 0$ otherwise.

The car brakes with a value of 0.5 if the speed is higher than 1 km/h. If it is lower, the value for the brake output is set to zero. The output for the clutch is always 0, the gear is always set to -1.

The six parameters of the *OffTrack* behavior have been optimized for the GECCO 2011 submission using the CMA-ES in a scenario specifically designed to evaluate the efficiency of the recovery behavior. Table 2 shows the optimized values.

4.4.2 OnTrackTurning

The component *OnTrackTurning* also distinguishes between driving backwards or forwards and takes advantage of the fact that the behavior has to be modeled for only one orientation of the car. We here describe the behavior used when the car faces the left side of the road ($\phi \leq 0$). In the other case, the same behavior can be used by mirroring the input values for the angle to track axis and the track position and mirroring the output value for the steering. When the component *OnTrackTurning* is called for the first time during a recovery maneuver it checks if the car is already driving backwards and if that is the case, it continues to do so. It also drives backwards if the special stuck detection of the component *OnTrackTurning* has detected that the car is stuck while trying to drive forward. The stuck detection checks if the car is slower than 10 km/h and the difference in speed between two gameticks is slower than 2 km/h. If this condition is true for 50 consecutive gameticks the car is considered stuck and starts to drive backwards. While driving backwards, no stuck detection is performed. The outputs for the steering s, the acceleration a, the brake b and the gear g are as follows. The clutch is always set to 0.

Table 2 Optimized parameters used by the off-track recovery behavior

Parameter	Value
Target angle backward $\hat{\phi}_b$	$-44.51°$
Min. acceleration backward $\underline{a}_b$	0.80
Max. acceleration backward $\bar{a}_b$	0.87
Target angle forward $\hat{\phi}_f$	31.13°
Min. acceleration forward $\underline{a}_f$	0.23
Max. acceleration forward $\bar{a}_f$	0.70

Forward When driving forwards and $\phi \leq -9$ the steering is set to

$$s = \begin{cases} -1 & \phi \leq -90 \\ \dfrac{\phi}{90} & -90 \leq \phi \leq -9. \end{cases}$$

If $\phi > -9$ we use the heuristic to steer in the direction of the track sensor which has the highest value. The acceleration and the brake only depend on the current speed of the car. If the speed is lower than -1 km/h the brake is set to 1 and the acceleration is set to 0. Otherwise the brake is set to 0 and the acceleration set to 0.8. The gear is set to one or, if the car is faster than 30 km/h, the standard gear change behavior (see Sect. 4.7) is used.

Backward The main goal when driving backwards is to get the car away from the curb. Therefore, the controller keeps driving backwards until either

- the center track sensor with index 9 has a value greater than 1m and the angle to track axis ϕ is bigger than $-20°$
- or the position on the track τ is smaller than -0.5.

If the first condition is true, the car is still close to the curb but facing the right direction to give driving forward a try. If the second condition is true, the car is far away from the left curb and can turn in the right direction by driving forwards. Otherwise it drives backwards using the steering:

$$s = \begin{cases} 1 & \phi \leq -90 \\ -\dfrac{\phi}{90} & \phi > -90 \end{cases} \tag{3}$$

The backward part of the *OnTrackTurning* behavior is the only submodule of the recovery behavior that uses an indirect approach for the acceleration. That means, a target speed is calculated before the outputs for the acceleration and brake are determined. The target speed depends on the normalized distance d_ϕ to the desired orientation (-20) with

$$d_\phi = \frac{|\phi + 20|}{160}$$

and the normalized distance d_τ to the desired track position (-0.5) with

$$d_\tau = \frac{|\tau + 0.5|}{1.5}.$$

The target speed $\hat{v}$ is then

$$\hat{v} = -(15 + 20 \cdot \min(d_\phi, d_\tau))$$

The acceleration simply depends on the current speed and the calculated target speed: If the current speed is higher than the target speed, the acceleration a is set to 0.8 and the brake b is set to 0. If the speed is equal or lower, the acceleration is set to 0 and the brake to 0.8. An optimization of the desired orientation and position, similar to the optimization of the parameters of the *OffTrack* behavior, was considered but never done.

Despite the optimization of some numerical values, the recovery behavior described here is probably not optimal. But it is reliably able to get the car back on the track in a reasonable amount of time. In the context of the competition it is rarely used at all because the tracks used there do not have run-off areas but barriers very close to the road. As a side note, the recovery behavior could act as a very simple controller on its own, although it obviously does not achieve competitive lap times.

4.5 Steering

The heuristic steering behavior is an extension of the simple heuristic used by the recovery behavior. It is based on the same idea to steer in the direction of the track sensor which has the highest value. In contrast to the recovery behavior, the normal steering module uses a smoother increase of the steering output for tight corners depending on the measure ρ calculated by the track segment classifier:

$$s = f(\rho) \cdot h(k^*) \tag{4}$$

where

$$f(\rho) = \begin{cases} 1 & |\rho| < 20 \\ 1 + \dfrac{|\rho| - 20}{15} & |\rho| \geq 20 \wedge |\rho| < 35 \\ 2 + \dfrac{|\rho| - 35}{25} & |\rho| \geq 35 \wedge |\rho| < 60 \\ 3 & |\rho| \geq 60 \end{cases} \tag{5}$$

is used to increase the steering output and $h(k^*)$ is the k^*th entry of the last row of Table 1 with k^* denoting the index of the sensor with highest value.

Further enhancements are two heuristic rules activated when the car gets to close to the curbs in a corner. As usual, these rules have to be modeled for only one direction of a corner, we here describe the rules used in a left hand corner. In a right hand corner, the same rules can be used by mirroring the input values for the track position τ, the track angle ϕ, the steering s calculated with Eq. 4 and the resulting

modified steering value after the application of the rules. The rules are not used on a straight.

The first rule covers the inside of a corner. We assume that the car is 2.2 m wide, the distance d between the left side of the car and the curb is then $d = \tau_{abs} - 1.1$. The rule is triggered when $d < 1\text{m}$ and the car is oriented towards the left curb ($\phi < 0$). To get the car away from the curb, the steering s is modified as follows:

$$s = \begin{cases} s \cdot (2d - 1) & d > 0.5\text{m} \\ \phi \cdot (-2d + 1) & d \leq 0.5\text{m} \end{cases} \tag{6}$$

This equation causes the steering to open until the car drives exactly forward at a distance of 0.5m and then gradually turns into the opposite direction depending on the angle to the track axis.

The second rule is triggered when the right side of the car gets to close to the right curb on the outside of the corner. In that case we want to further increase the steering to stay on the track. This is done using the following equation:

$$s = s \cdot \left(1 + \frac{|\rho|}{100}\right). \tag{7}$$

4.6 Throttle and brake

This module determines the values for the throttle a and the brake b, based on the desired target speed $\hat{v}$ and the current speed of the car v. Depending on whether the car is faster or slower than the desired target speed $\hat{v}$, the throttle a and the brake b are initialized in an almost binary reaction according to Eqs. 8 and 9.

$$a = \begin{cases} 1 & v < \hat{v} - 5\text{km/h} \\ \dfrac{\hat{v} - v}{5} & \hat{v} - 5\text{km/h} \leq v < \hat{v} \\ 0 & v \geq \hat{v} \end{cases} \tag{8}$$

$$b = \begin{cases} 0 & v \leq \hat{v} \\ 1 & v > \hat{v} \end{cases} \tag{9}$$

Finally, ABS filtering is performed if the car is braking ($b > 0$). This has been taken unchanged from the SimpleDriver provided with the client package, which is actually the same approach as described by Wymann[3] using a different value of 3 for the variable ABS_SLIP. In contrast to other competitors, our controller does not have a traction control. The advanced version achieves a similar effect using the clutch (see Sect. 6.3).

[3] http://www.berniw.org/tutorials/robot/tutorial.html.

4.7 Gear

A simple hand coded rulebase is used to select the value of the gear output. The only inputs used are the currently selected gear and the revolutions per minute of the engine. If the latter reaches a predefined threshold, the gear is incremented or decremented. Table 3 shows the threshold values. To avoid oscillating between two gears, we allow only one gear change per second.

5 A flexible planning module

Driving fast on a race track is actually a fairly simple process: Fully accelerate until it is necessary to reduce speed for mastering the next corner. The task of the planning module is to create a plan which contains the target speed $\hat{v}$ the car should reach. The target speed should obviously be as high as possible to cause the acceleration module to accelerate. At a certain point, the target speed then has to be low enough to cause the acceleration module to brake to safely pass the next corner. Unfortunately this simple and intuitive approach is hindered by four obstacles which will be eliminated by our solutions presented next.

The first obstacle is the missing global knowledge about the race track in the context of the Simulated Car Racing Championship. This problem has been solved by the trackmodel described in the previous Sect. 4.3.

The second obstacle is the missing knowledge of how fast a certain corner can be passed in a safe manner. To forecast that speed limit, we stick to the approach described in our previous work [31], i.e., we map the maximum absolute value of ρ stored in the track model for each corner to the desired target speed $\hat{v}(\rho)$ using the generalized logistic function

$$\hat{v}(\rho) = 330 - \frac{280}{\left[1 + Q \cdot e^{-B(|\rho| - M)}\right]^{\frac{1}{v}}}. \tag{10}$$

The upper and lower asymptotes have been fixed to reasonable values. The upper asymptote of 330 km/h cannot be reached by the fictitious car used by SCRC interface. This enables the function to generate high target speeds, which cause the acceleration module to keep accelerating. The lower asymptote of 50 km/h is slow enough to safely pass every corner. The remaining four unknown parameters of the function (Q, B, M and v) are included in the learning process.

The third obstacle is the missing knowledge of the car's brake behavior, which is necessary to calculate at which point the car has to start braking. This can be easily

Table 3 Gear change thresholds

Current gear	1	2	3	4	5	6
Shift up	> 9200	> 9200	> 9200	...	> 9200	
Shift down		< 3500	< 6000	...	< 6000	< 6000

solved by using the simple formula learnt in driving school, as we have described in our previous work [31]. Here, we use a more sophisticated approach described by Wymann [35], who was able to find an analytical solution which takes drag into account. In the context of braking, drag is actually an advantage because it improves the brake performance by the opportunity to plan a fail-safe breaking at a later moment. For the reduction of the current speed v to the target speed $\hat{v}$, $\hat{v} < v$, both converted to m/s, the required breaking distance s (in meter) is calculated via

$$s = -\ln\left(\frac{c + v^2 \cdot d}{c + \hat{v}^2 \cdot d}\right) \cdot \frac{1}{2\,d} \cdot \frac{1}{bcc} \tag{11}$$

where c and d are place holders for values which can be calculated from known constants: $c = \mu \cdot G$ and $d = \frac{ca \cdot \mu + cw}{mass}$ with constant of gravitation G, downforce coefficient ca, drag coefficient cw, mass of the car $mass$ and friction coefficient μ of the track we are racing on. The car specific constants can be found in the TORCS configuration files and are set to $ca = 3.65$ and $cw = 0.08$. The friction μ will become a part of the controllers parameter set. The parameter bcc is fixed to 1 when we use Eq. 11 for a straight part of the track. For corners, $bcc \in [0...1]$ is included in the learning process. A value smaller than 1 leads to longer brake distances in corners. This should reflect the fact that one might not be able to brake as well in a corner as on a straight. One might argue that it is undesirable to use the values found in the xml configuration files. Without doing this, one could still easily solve the problem of predicting the brake distance by using a lookup table, just as we did for the speed prediction described below. But we here chose the pragmatic way of simply using the available information.

The fourth and last obstacle results from the solution of the third obstacle: To calculate the brake distance, we need a forward model of the car's acceleration to predict the speed of the car. This is needed for the simple brake calculation formerly used as well as for the advanced version used here. The trivial solution is the creation of a lookup table: A simple controller fully accelerates on a long straight using the gear change behavior described in Sect. 4.7. One can then take measurements of which speed the car has reached after a certain distance.

As can be seen in Table 4, the granularity of the lookup table is not equidistant, because we aimed at keeping the table at a reasonable size: The entire table has 114 entries. To predict the speed of the car, one first searches the entry in the table which is closest to the current speed of the car, e.g. the entry *210 m / 182.81 km/h* if the car is currently traveling with of a speed of 181 km/h. We then add the distance for which we want to accelerate to the distance in the entry. E.g. a straight of 663 m length is ahead of the car, $210 + 663 = 873$. In the final step, we search the entry closest matching this distance, in our case *880 m / 266.789 km/h*. We then know that the car will be driving with a speed of 267 km/h at the end of the straight. This information can then be used to calculate the brake distance.

The forward models of the brake and acceleration behavior can be used to answer two more questions which will arise during the planning process: How fast can we approach a point ahead of the car under the condition that we want to reduce the speed by a certain amount beyond that point using a certain distance? And which brake distance do we *really* need? The first question can easily be answered by

Table 4 Excerpts with varying resolution from the lookup table used for the speed prediction

Distance (m)	Speed (km/h)	Distance (m)	Speed (km/h)
0.0	0.0	210.0	182.81
5.0	29.4447	220.0	185.734
…	…	…	…
195.0	178.564	390.0	220.729
200.0	179.873	400.0	222.107
420.0	225.029	840.0	266.263
440.0	227.717	880.0	266.789
…	…	…	…
780.0	262.465	1,960.0	294.795
800.0	263.748	2,000.0	295.228

solving Eq. 11 for the speed v with known target speed $\hat{v}$ and known brake distance s resulting in

$$v = \sqrt{\exp(-2\,d \cdot s \cdot bcc) \cdot \left(\frac{c}{d} + \hat{v}\right) - \frac{c}{d}} \tag{12}$$

At a first glance, the second questions seems to be a stupid one already answered by Eq. 11 combined with the forward model of the acceleration. But the simple approach to predict the speed and then calculate the brake distance is inaccurate: The brake distance reduces the distance available to accelerate, which decreases the predicted speed, which in turn decreases the needed braking zone, etc. More accurate results are achieved by an iterative search for the needed brake distance using Algorithm 1. Three iterations proved to be accurate enough in practice.

Algorithm 1 Iterative calculation of the brake distance

Input: $\nu > 0$ and $\hat{\nu}$ and $d_{acc} > 0$

```
// Predict speed using the lookup table
ν_predicted = predictSpeed(ν, d_acc)

// Calculate initial brake distance using equation 11
d_brake = calcBrakeDistance(ν_predicted, ν̂)

for i = 1 to 3 do
   // Reduce distance available to accelerate
   d'_acc = d_acc − d_brake

   // Predict speed again using the lookup table
   ν_predicted = predictSpeed(ν, d'_acc)

   // Calculate brake distance using equation 11 again
   d_brake = calcBrakeDistance(ν_predicted, ν̂)
end for

return d_brake
```

Using these components, it is the task of the planning module to create a plan which covers a certain distance ahead of the car. This distance should not be too large for several reasons: A voluminous plan takes longer to create, which is undesirable in a realtime scenario. The creation of the plan should happen quickly, to stay within the 10 ms time limit. A longer planning period has also a higher chance to become obsolete because of changes in the environment, like other cars coming into the coverage of the opponent sensor. Even if no outside factors cause a change in the plan, an update might become necessary due to inaccuracies in the plan. The forward models used are approximations of the real simulation and far from perfect. We therefore propose to create a plan which covers only the part of the track in front of the car up to and including the next corner. A new plan is created when

- the next expected race distance is not covered by the plan or
- the opponent observer or the online learning module requests a replanning or
- the deviation between the current position on the track and the racing line becomes too large.

The last point is only important in the context of the advanced controller, which is able to create and follow a racing line. A similar check could be performed on the predicted speed generated by the forward model of the acceleration and the current speed of the car, but we do not do that. The second point partly lies outside the scope of this paper since we do not cover the opponent handling here. The interaction between the online learning module and the planning module is described in Sect. 5.5. In the majority of cases the first point is the reason to create a new plan. The next expected race distance can be derived from the current race distance delivered by the sensor model, the current speed of the car and the fact that we get new sensor information every 20 ms. The plan is based on the race distance because it continuously increases during the race and is therefore easier to handle than the distance from the start line. The target speeds we want to plan are a mapping from the race distance to the desired speed. We will see that piecewise constant functions are good enough to model this mapping.

To illustrate the planning process, we will use the first part of the track *Wheel2* as shown in Fig. 3 on page 9 as an example, assuming that the car is standing at the start line.

Whenever a replanning becomes necessary, the first step is to find the next corner or unknown segment in the track model. Only during the first lap of the warmup, we will find unknown segments to plan for in the track model. Therefore, for the rest of this section, we will only talk about "planning towards the next *corner*" as this is the usual case. During the search for the next corner, a stack is built which collects all necessary data used for the planning process. For every track segment, a new data element is created and pushed on the stack containing the index of the track segment, the race distance at which the plan for this segment should start and end, and the predicted speed at the end of the segment. We start with the track segment in which the car is currently located and create a stack element for it. The beginning of the plan is the current race distance, whereas the end of the plan is the beginning of the plan plus the end of the track segment minus the current distance from the

starting line. The current speed is known so that the speed at the end of the segment can be predicted. For our example, this can be seen in Fig. 5. The car is currently in segment 29, with 4m remaining and a current speed of 0 km/h. This leads to a predicted speed of 29.4 km/h at the end of segment 29. We then iterate through the track model to find the next corner. For every straight segment found, we add a new stack element as described before, using the whole length of the track segment for the speed prediction. In our example, this is the case for segment 1, as can be seen in Fig. 5. A last element is added to the stack for the corner, omitting the speed prediction for the end of the segment as this is not needed.

In a second step, we check if we have to take care of slower segments beyond the corner to avoid unpleasant surprises: We start with the track segment containing the point 200m beyond the end of the corner to determine what we call the *approach speed* for that segment. A look ahead of 200m proved to be enough in practice. The *approach speed* for this segment is initialized as follows.

Straight: We initialize the approach speed with 330 km/h assuming that no special care is needed for straights.

Corner: We use the target speed for the first apex of the corner and calculate the distance between the start of the segment and the point at which we want to have reached that target speed. The details of this will be explained in Sect. 5.3, as we do the same calculation during the actual planning. The result is the distance available within the corner to reduce speed. We then use Eq. 12 to calculate the speed with which we can arrive at the start of the corner and set the approach speed to this value.

This initial approach speed is then *propagated back* through the segments of the track model until we reach the corner we want to plan for as follows:

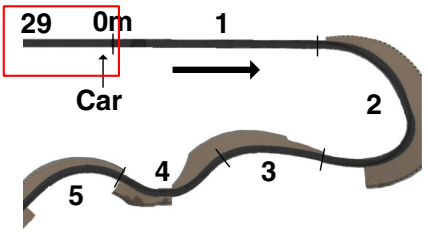
Straight: Using the current approach speed as the target speed $\hat{v}$ and the length of the segment as the brake distance s, we calculate the new approach speed using Eq. 12.

Corner: We simply reinitialize the approach speed in the same way as described above.

The result is the maximum speed we can allow at the end of the plan we are creating.

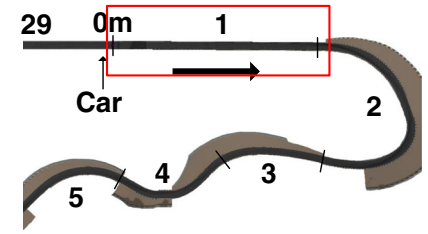
For our example, this process is illustrated in Fig. 6. The look ahead falls into segment 4, for which we determine an approach speed of 213.93 km/h. This is propagated backwards according to the rules above: Segment 3 is again a corner, so we simply replace the approach speed by the new one for segment 3. Since segment 3 is very fast corner its approach speed is unlimited (330 km/h).

The third step is the actual planning process, which we perform *back to front*. That means that we first plan for the top element of the stack which represents the corner. The result is a *plan element* containing the target speeds for this segment and the new approach speed for this segment. The top element is popped from the stack and the whole process is repeated for the next data element on the stack until the stack is empty.



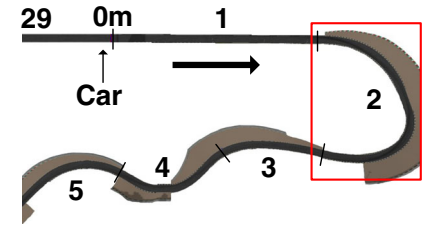
Stack

#	Start	End	v-Predicted
29	0m	4m	29.4km/h



Stack

#	Start	End	v-Predicted
1	4m	311m	208.7km/h
29	0m	4m	29.4km/h



Stack

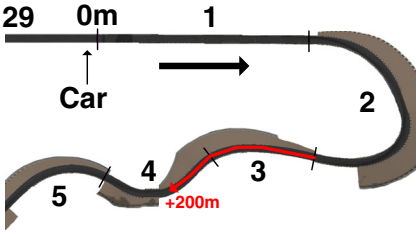
#	Start	End	v-Predicted
2	311m	701m	./.km/h
1	4m	311m	208.7km/h
29	0m	4m	29.4km/h

Fig. 5 Illustration of the first phase of the planning process. Starting with the segment in which the car is currently located, we store information about the segments ahead of the car on the stack, until we find the next corner

The overall result is a plan in form of a list of plan elements, each plan element covers a certain race distance ahead of the car. The planning for the various types of track segments is described in detail in the following subsections.

5.1 Planning for unknown segments

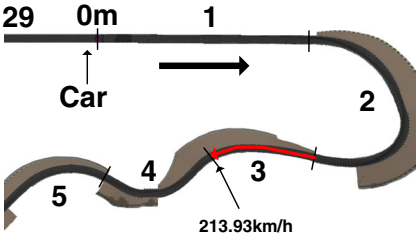
Under normal circumstances it is only necessary to plan for unknown segments during the first lap of the warmup when the track layout is unknown. The trivial solution is to plan a constant target speed $\hat{v}$ of 50 km/h. This is slow enough to safely pass every corner.



Approach Speed = ?

Stack

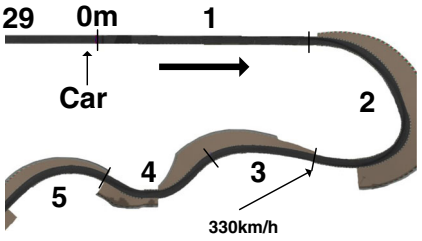
#	Start	End	v-Predicted
2	311m	701m	./.km/h
1	4m	311m	208.7km/h
29	0m	4m	29.4km/h



Approach Speed = ?

Stack

#	Start	End	v-Predicted
2	311m	701m	./.km/h
1	4m	311m	208.7km/h
29	0m	4m	29.4km/h



Approach Speed = 330km/h

Stack

#	Start	End	v-Predicted
2	311m	701m	./.km/h
1	4m	311m	208.7km/h
29	0m	4m	29.4km/h

Fig. 6 Illustration of the second phase of the planning process. Starting at the end of the corner for which we want to plan, we look 200 m ahead to determine the *approach speed*. This is the save speed, with which we want to approach the segment after the corner. In our example, the final approach speed is 330 km/h, which means that no special care has to be taken. That is because segment 3 is a very fast corner which can be driven almost flat out

5.2 Planning for straight segments

Planning for a straight segment is also rather simple. First we check if the next element on the stack also represents a straight segment. If that is the case, we handle both of them together.[4]

We know the speed of the car, either the real speed if the car is in this segment or the predicted speed at the beginning of the segment which is stored on the stack. We

[4] This can only happen at the start/finish line. For the rest of the track, a straight segment in the track model is always prepended and followed by a corner.

also know the approach speed, the speed with which we want to arrive at the beginning of the next segment. And we know the length of the segment, that is the distance for which we need to plan. If the car is in the segment, we reduce this distance to the remaining part of the segment.

Using all this information, one can plan the target speeds as follows:

1. Calculate the brake distance to reduce the current or predicted speed of the car to the desired approach speed, using Algorithm 1.
2.
   (a) If the resulting brake distance is zero, we can plan a constant target speed of 330 km/h for the whole segment. The new approach speed passed to the next planning iteration is also set to 330 km/h.
   (b) If the resulting brake distance is longer than the segment, we need the whole segment for braking. Since we want to brake down to the approach speed of the next segment, we plan this approach speed as the constant target speed for the whole segment. The new approach speed passed to the next planning iteration is calculated using Eq. 12.
   (c) If the brake distance is non zero but shorter than the (remaining) length of the segment, we can divide the segment in two parts: The braking zone at the end of the segment, for which we plan a constant target speed equal to the given approach speed. And a part at the beginning of the segment which we can use to accelerate. We therefore plan a constant target speed of 330 km/h for this part. The new approach speed passed to the next planning iteration is also set to 330 km/h.

In our example, the brake distance is zero: We will see that we can pass the first apex of segment 2 at 242 km/h but we will arrive with only 209 km/h at the beginning of segment 2 so no braking is needed. Therefore we can plan a constant target speed of 330 km/h for segments 29 and 1.

5.3 Planning for corners

Good planning for a corner is not trivial: A corner segment contains one or more subsegments, as described in Sect. 4.3. Some of these subsegments represent apexes at which we want to drive at most with the speed determined by the generalized logistic function from Eq. 10. But the track model is an abstract representation and we do not have accurate information about the exact position of an apex within the corresponding subsegment. Because of this, we will plan to hold that target speed for a certain fraction of the length of the subsegment containing the apex. The parameter cf (*c*orner *f*raction), $cf \in [0...1]$, which determines how much of the subsegment should be driven with constant speed, is included in the learning process. Subsegments at the beginning of a corner, before the first apex, can be used for braking. Subsegments at the end of a corner, behind the last apex, can be used for accelerating. Finally, if the corner has more than one apex, subsegments between two apexes can be used for *both*, *accelerating* and *braking*, depending on the layout of the corner. Figure 7 illustrates what we want to achieve in general: Brake down

to the desired target speed, hold that speed at the apex and accelerate at the exit of the corner.

When we plan for a corner segment, we first check if the segment contains the position of the car. If that is the case, we pretend that the car is positioned at the beginning of the segment. By doing this, we can always plan for the whole segment and avoid special cases. Planning for a corner is an iterative process, starting with the end of corner. During each iteration, we handle one apex until the last iteration handles the corner entry. During each iteration, we keep track of the planning process using a number of helper variables: v_n is the next speed for which we want to plan, v_p is the previous speed from which we are planning.

1. We start the planning process with the last apex. We know the speed with which we want to approach the next segment and the desired speed at the apex. v_n is set to the approach speed and v_p is set to the apex speed. We can also calculate the distance d_1 we want to drive with a constant speed using the length of the subsegment containing the apex and the factor *cf*. The remaining distance within the subsegment plus the subsegments beyond the apex are the distance d_2. We then calculate the brake distance s to reduce v_p to v_n with distance d_2 using Algorithm 1.
   (a) Ideally, the resulting brake distance is zero. That means, that we can use the exit of the corner to accelerate. We therefore plan a constant target speed of v_n for the distance d_2.
   (b) If the brake distance is bigger than d_2, we cannot accelerate but obviously have to brake. This is the case when we are planning for a fast corner which is followed by a slow corner. We plan a constant target of v_n for d_2. And we calculate a new value for v_p using Eq. 12, with v_n as the target speed and d_2 as the available distance.
   (c) If the brake distance s is nonzero and smaller than d_2, we can plan a constant target speed of 330 km/h for the distance $d_2 - s$ to accelerate. And we plan a constant target speed of v_n for the distance s. Finally, we plan a constant target speed of v_p for the distance d_1.
2. If the corner has more than one apex, we handle the rest *back to front* in the same way. For the next iteration, v_n is set to v_p and v_p is set to the target speed of the apex handled next. d_1 can once again be calculated from the subsegments length and the factor cf, d_2 is set to the distance between the two apexes.
3. In a last step, we plan the corner entry. This can be done in the same way as before: We set the distance d_1 to zero, because this part is not needed for the corner entry and set the distance d_2 to the length of the corner entry. v_n is set to v_p and vp is set to the predicted speed with which we will arrive at the corner. This information can be found in the next stack element. If there is no other element on the stack, we use a fictitious speed of 0 km/h.

When the planning is finished, we pass the last value of v_p as the approach speed to the planning of the next track segment.

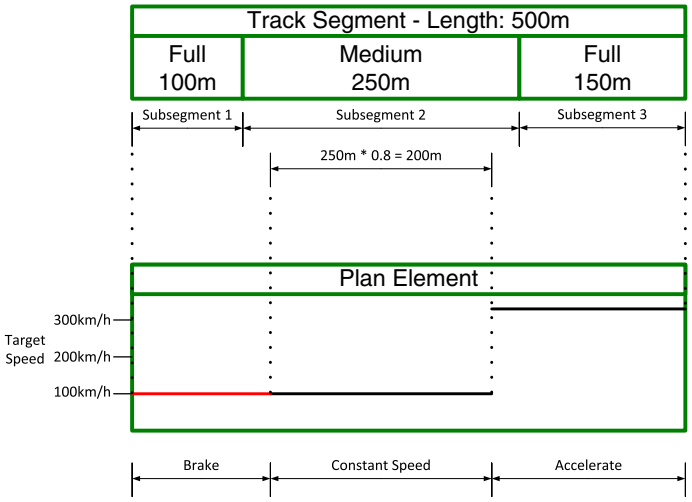


Fig. 7 Illustration of the general corner planning. *Top* a track segment representing a fictitious medium speed corner, *bottom* the plan element created during the planning process. We assume that the optimized value for the parameter *cf* is 0.8 and that the generalized logistic function delivers a target speed of 100 km/h for the apex in the *middle* subsegment. We want to hold that target speed for $250\text{m} * 0.8 = 200\text{m}$ and plan a constant target speed of 100 km/h for that distance. We do the same for the part in front of the apex (drawn in *red*), because we need to brake down to the desired target speed. Behind the apex, we plan a constant target speed of 330 km/h because we want to use this part of the corner to already accelerate again (Color figure online)

For our example, the planning for the corner segment is illustrated in Fig. 8. Starting with the last apex, we can plan a target speed of 330 km/h for the exit of the corner and a target speed of 96 km/h for the apex. In the second iteration, we plan for the first apex which we can pass with 242 km/h. Nearly the whole distance between the two apexes is needed to reduce this speed to the target speed of the second apex. Finally, we plan the corner entry. An illustration of the complete plan for our example can be seen in Fig. 9.

5.4 Using the plan

To use the plan, we first determine the next expected race distance and check if it is covered by the plan. If it is not, a new plan is created as described before. Otherwise, we search the list of plan elements for the one containing the next race distance to get the planned target speed and in the context of the advanced controller the next target position. These are then passed to the steering and acceleration modules which work on the actuators.

The result of our example is shown in Fig. 10 which contains a plot of the planned target speeds and the actual speeds reached by the car while driving the part of the track covered by the plan.

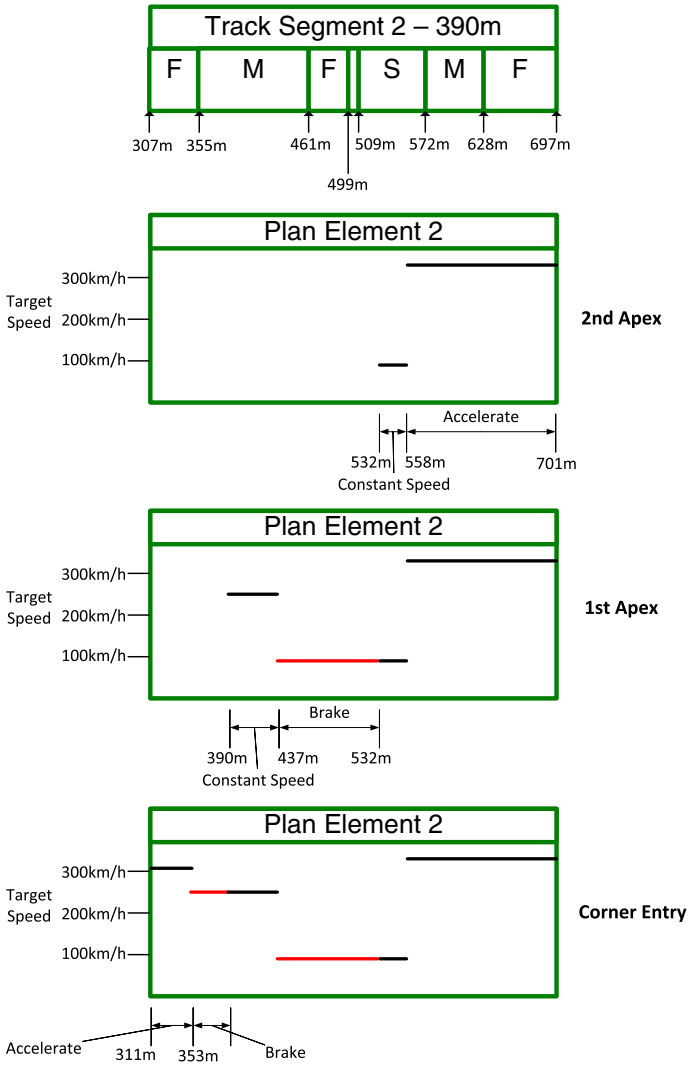


Fig. 8 Illustration of the planning for the corner in our example. *Top* track segment 2 for which we want to plan. *Below* graphical representations of the plan segment created for the corner during the various iterations of the planning. *First step* plan for the second apex and the exit of the corner. *Second step* plan for the first apex, *3rd step* plan the corner entry. Each stage of the plan element contains plots of the piecewise constant functions for the planned target speeds. In this example, the optimized corner fraction *cf* is 0.4 which means 40 % of a subsegment containing an apex should be driven with constant speed

5.5 Online learning

The online learning module implements a heuristic rule base to further adapt the learned behavior to unknown tracks during the *warmup* stage of the competition. It is only used during the warmup and deactivated when the stage is set to *qualifying*

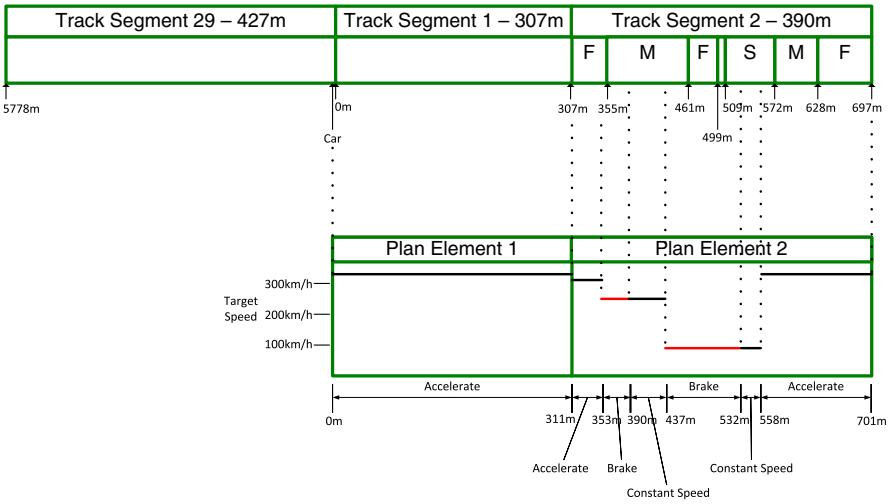


Fig. 9 Illustration of the resulting plan for our example. *Top* the three segments of the track model for which we plan. *Below* a graphical representation of the resulting plan elements containing plots of the planned target speeds

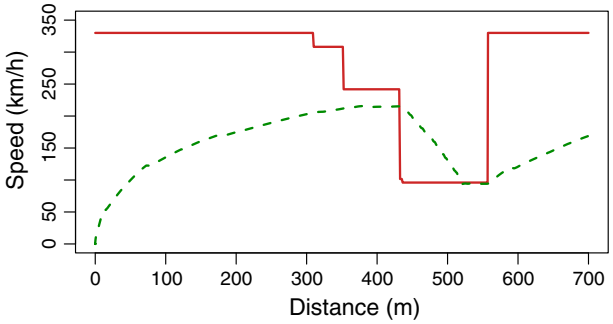


Fig. 10 Planned target speeds (*red*, *solid*) and actual speeds (*green*, *dashed*) reached by the car for our example. With standing start, the car does not reach the planned target speeds up to the first apex of the corner. It brakes as planned for the second part of the corner and drives with a constant speed through the second apex. It then already accelerates at the end of the corner (Color figure online)

or *race*. The limited time during the warmup has to be used as efficiently as possible. The basic idea is to collect data while the car is driving, interpret the data and then alter the behavior of the controller to improve the performance. One can regard this approach as a kind of reinforcement learning. We have seen that the planning module contains a lot of variables which could be changed to adapt the behavior. But to keep the online learning module as simple as possible, we chose to only adapt the target speeds. That means, that the learned generalized logistic function (Eq. 10) is only used to initialize the target speeds for a new track. These target speeds are then fine-tuned and saved with the track model at the end of the

warmup. When the stage is set to *qualifying* or *race*, the controller uses the target speeds stored in the track model and no longer uses Eq. 10.

The online learning consists of four distinct phases:

1. Creation of the track model during the first lap.
2. Test parameter sets and select the best one.
3. Correct target speeds down to drive without crashing.
4. Correct target speeds up to drive faster.

The first phase, learning the track layout, is obviously mandatory. The second phase is always performed, even if the controller has only one parameter set and there is no option to choose from. But the information gathered during phase two is needed for decisions in the subsequent phase(s) anyway. Phase three is optional and can be skipped, if the controller did not crash during the relevant laps in phase 2. The goal is to spend as much time as possible in phase four to drive as fast as possible. Whenever the module changes values during the online learning it triggers the creation of a new plan, which is then based on the tuned values. In the following subsections, we will describe what data is collected and how it is interpreted during the various phases of the online learning.

5.5.1 Collecting data

For every lap driven, we record the time needed and if the lap has been completed. We also remember if the lap has been a flying one, which means that it was the second lap driven while testing some tuned values.

Apart from phase one, we also collect data for every track segment during each lap. Such a data set contains the following values:

Problematic	Flag to indicate that we once had trouble (e.g. left the track) in this segment.
OnLimit	Flag to indicate that we are close to the limit in this segment.
OffTrack	Flag to indicate that we left the track in this segment.
Lateral speed	Integral of the lateral speed of the car over the length of the segment.
Dec	Decrement for target speeds.
Inc	Increment for target speeds.
Target speeds	Target speed(s) used in this lap for this segment if the data belongs to a corner.

We collect data for every track segment, although we currently only use the data collected for corners when we make the decisions on how to tune the target speeds.

The boolean flags are initialized as being unset when we collect data for the first time (usually the second lap). For each subsequent lap, the flags OnLimit and Problematic are copied from the data collected during the previous lap. By doing this, once a flag has been set for a segment, it stays set for the rest of the warmup. The decrement is initialized with 20 % and the increment is initialized with 0.

While driving, the data sets are updated each gametick as follows using the current sensordata and the sensordata from the last gametick.

Lateral speed integral If the car has driven forwards ($\epsilon_t > \epsilon_{t-1}$), we increase the integral by $|v_l| * (\epsilon_t - \epsilon_{t-1})$. Once the lap is completed, the integrals of all segments are normalized to the segments length.

OnLimit If the car is on the track according to the sensor model ($|\tau| < 1$), but off track according to the absolute track position ($\tau_{abs} < 1.1 \vee \tau_{abs} > tw - 1.1$, where 1.1m is half the width of the car), we say that the car is *on the limit*. The OnLimit flag is therefore set for the current segment if it represents a right hand corner and the car is on the left (outside) edge and vice versa if the current segment represents a left hand corner. Should the car leave the track on the *inside* and the previous segment is also a corner and we are at most $3 \times tw$ m away from the previous segment, *we blame the previous segment* by setting its OnLimit flag.

OffTrack When the car completely leaves the track in a corner ($|\tau| \geq 1$), we set the OffTrack flag for this segment. The flag is also set when the car got more when 20 points of damage between the last gameticks. This reflects the fact that the car cannot leave the track on the circuits used in the championship because these do not have run off areas. Instead of leaving the track, the car gets damage by crashing into the barriers when it drives too fast. Sometimes the car collects a few damage points simply by driving fast without crashing at all. We assume that this should reflect tire wear. That is why we chose to only activate the flag if the damage increased by 20 points or more.

Whenever the car crosses the finish line, the collected data is interpreted as described in the following sections.

5.5.2 Phase 1: Track learning

While learning the track model, the online learning module is idle. Once the track model is complete, usually after the first lap, we log the time of the first lap and switch to phase two.

5.5.3 Phase 2: Testing parameter sets

In the context of the Simulated Car Racing Championship, a controller faces a wide variety of possible race tracks. To cope with this challenge, our controller uses two parameter sets when submitted to the competition: One optimized for race tracks with a tarmac surface and one optimized for dirt tracks with a slippery surface. In the second phase of the online learning, the module tests the available parameter sets to choose the best one.

To test a parameter set, all variables of the controller are set to the values stored in the set and the target speeds for the apexes are initialized using Eq. 10. The controller drives two laps with the parameter set to make sure that it drives one

flying lap. When this has been done for all parameter sets, we compare the lap times of the flying laps and choose the parameter set with the best (smallest) lap time. The module then switches to phase three.

5.5.4 Phase 3: Adjusting target speeds down

The goal of phase three is to get the controller to a safe state, that is to drive without leaving the track or crashing in any other way. To check if this is necessary at all, we analyze the flying lap from phase two belonging to the parameter set chosen at the end of phase two: If this lap was driven without any problems, one can directly switch to phase four.

Otherwise, the target speeds for segments in which the controller left the track are lowered by the decrement (20 %) and the segments are marked as being problematic. The adjusted target speeds are then tested by driving one lap. If the controller crashes again, the target speeds are lowered again, otherwise we drive a second lap using the same target speeds. Once the car has driven two laps without problems we switch to phase four.

5.5.5 Phase 4: Driving faster

The goal of the last phase is to drive as fast as possible by incrementing the target speeds. When we switch to phase four, the first thing we have to do is to initialize the increments for the target speeds. To do this, we analyze the last flying lap, either the one from phase three which we drove without any problems or the one from phase two, if phase three was skipped.

For problematic segments, we initialize the increment to 10 %, in the hope that we can regain some of the speed lost earlier, when we lowered the target speeds. For the other corners, we initialize the increment based on the recorded lateral speed integral: A low value is an indicator that the car passed the corner very stable without sliding. This indicates a great potential for a higher target speed and leads us to the following rules: If the lateral speed integral is lower than 1, we initialize the increment with 10 %. If it is lower than 10, we initialize the increment with 5 % and 1 % if it is higher than 10.

With the increments we can calculate the new, higher target speeds, these are then tested. If we have problems with the higher target speeds, the target speeds are reset to the previous values in the affected segments. Once we drove two laps without any problems, the target speeds are incremented again. This process is repeated until we run out of time.

5.5.6 Selecting the optimal values

At the end of the warmup, we have to chose and save the fine tuned, optimal values. The optimal parameter set is the one chosen in phase two. It is saved to disk and later used by the controller for the qualifying and the race. The optimal target speeds are obviously the ones used during the fastest flying lap driven. This information is

contained in the collected data. The target speeds are stored in the track model and the track model is saved to disk to be later used during the qualifying and the race.

5.6 Parameter optimization for the basic controller

Although the various modules of the controller contain a significant amount of expert knowledge, there remain some unknown values which need to be learned. In this first experiment, we demonstrate that these unknown parameters can successfully be learned offline in a black box optimization scenario using the well known CMA-ES. To ease the understanding of the experiment, we will first summarize the parameters of the controller before we describe the experimental setup and the results.

Table 5 summarizes all parameters of the base controller. The first six values are unknown and define the parameter space of the optimization problem. The other parameters are either known or have already been optimized in an independent scenario.

Next to presenting examples for the finally obtained solutions, we also need to tackle the **research question** if the chosen optimization method (CMA-ES) and the employed representation reliably lead to satisfying results. In the following, we repeatedly utilize the same 7-part scheme to describe our experimental findings.

Pre-experimental planning Initial runs led to the insight that slightly increased population sizes [set to (10, 20) instead of the default values of (5, 10)] may provide more reliability, and that initial step sizes shall be chosen slightly smaller than usual (0.2 here). Run length is chosen to 1,500 evaluations as longer runs result in little additional progress.

Task A time in the range of an excellent human driver ($\approx$360 s) has to be achieved in all runs.

Setup We run the CMA-ES with the specific parametrization given above 10 times on the Wheel2 track (one run takes about two days on a modern 4-core machine which can handle four parallel evaluations). To evaluate the fitness of a parameter set, the car drives three laps on the track Wheel2 which contains a wide variety of different corners. An evaluation is aborted after 9 min to avoid spending too much time with uncompetitive solutions. The track model for Wheel2 is known (it has been created in advance) and the controller is set to a mode in which it always uses the generalized logistic function to determine the target speeds while planning. The optimization was done without noise and the lap time limit, the damage and the fuel consumption were all disabled using the command line flags for TORCS introduced with the SCRC patch.

Results/observations/discussion Figure 11 documents the optimization progress for all runs performed. One can see that the CMA-ES is able to reliably find good solutions. Figure 12 shows a plot of the generalized logistic function belonging to the overall best solution found. The best solutions from the other runs show a similar shape. Figure 13 shows boxplots of the optimized values for the corner fraction *cf* and the brake coefficient *bcc* from the best solution found during each of

Table 5 Parameters of the base controller

Parameter	Value	Range	Comment
cf	?	0…1	Corner fraction
bcc	?	0…1	Coefficient for the brake calculation in corners
$\hat{v}_Q$	?	0.01…1	Parameter Q of Eq. 10
$\hat{v}_B$	?	0…10	Parameter B of Eq. 10
$\hat{v}_M$	?	0…1	Parameter M of Eq. 10
$\hat{v}_v$	?	0.01…1	Parameter v of Eq. 10
μ	1.15	–	Parameter μ of Eq. 11
ca	3.65	–	Parameter ca of Eq. 11
cw	0.08	–	Parameter cw of Eq. 11
$mass$	1150	–	Parameter $mass$ of Eq. 11
$\hat{\phi}_b$	−44.51°	–	Recovery: target angle backward
$\underline{a}_b$	0.80	–	Recovery: min. acceleration backward
$\bar{a}_b$	0.87	–	Recovery: max. acceleration backward
$\hat{\phi}_f$	31.13°	–	Recovery: target angle forward
$\underline{a}_f$	0.23	–	Recovery: min. acceleration forward
$\bar{a}_f$	0.70	–	Recovery: max. acceleration forward

The first six values are unknown and have to be learned in an optimization scenario. Values for the next four parameters can be found in the configuration files of TORCS. The last six parameters are used by the recovery behavior. They have already been optimized in an independent scenario

the ten runs. The optimal corner fraction seems to in the range 0.2 to 0.3, which means that 20–30% of the subsegment containing an apex should be driven with constant speed. The optimal brake coefficient is always 1 or very close to 1. From this we conclude that the factor bcc is not needed at all, because the abs makes it possible to brake as well in a corner a on a straight.

6 Improved controller with advanced modules

Observations during the parameter optimization for the basic controller exposed a number of shortcomings. First of all, the output of throttle and brake seems to be too extreme. Watching the car driving, one gets the impression that it slides a lot due to rough throttle and brake usage.

Secondly, the performance may be improved by an exact steering module provided by Tim Delbrügger in his bachelor thesis "Modellierung des Lenkverhaltens eines Fahrzeugs in einer Aurorennsimulation".[5] This module is able to follow a given racing line. However, planning a good racing line proved to be a difficult task. The advanced version of the planning module is able to plan a trivial racing line for unknown parts of the track and a simple racing line for straight

[5] provided in German, translated into English: "Modelling the steering behavior of a vehicle in a car racing simulation".

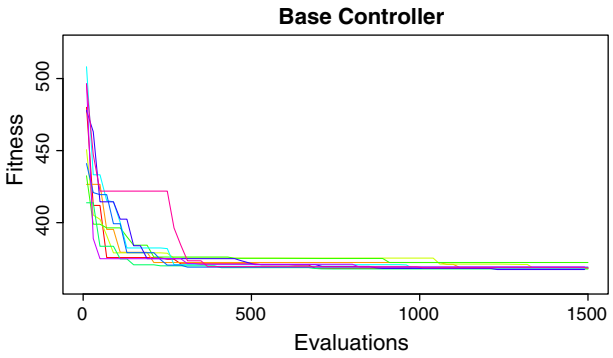


Fig. 11 Best ever fitness (time in seconds for three laps) for 10 runs of base controller optimization. The CMA-ES reliably finds competitive solutions. Big improvements generally happen early, but the last (very important) few seconds of laptime are obtained very late

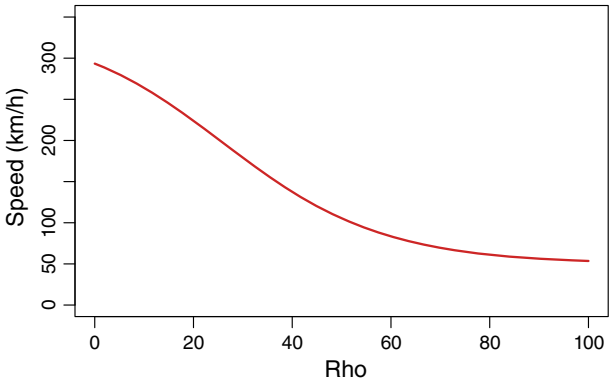


Fig. 12 Learned generalized logistic function of the overall best individual, which has a fitness of 367.5. The optimized parameters are $Q = 0.83, M = 0.22, B = 5.85$ and $v = 0.69$

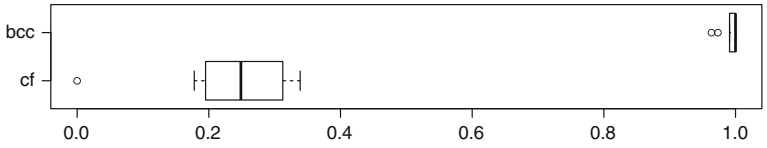


Fig. 13 Boxplot of the optimized values for the corner fraction *cf* and the brake coefficient *bcc* from the best solution found during each of the ten runs

segments. Whenever a racing line is available, the advanced controller uses the exact steering, otherwise it relies on the heuristic steering already described in Sect. 4.5.

6.1 Planning a racing line

The advanced planning module is able to plan a racing line for unknown and straight segments. As the plan is based on the distance raced, a racing line is a function which maps the distance raced covered by the plan to the desired track position $\hat{\tau}$.

For unknown segments, the trivial solution is to plan a constant target position $\hat{\tau} = 0$, which causes the car to drive in the middle of the track. For straight segments, the planned racing line should cause the car to approach the next corner at the outside of the track. To achieve this, we first define what we call *anchor points* for the possible combinations of a first and a second track segment. These can be seen in Table 6. We target a smooth racing line and therefore use a cubic spline to model the mapping of the race distance to the target position. Three points define the spline: The first point is the beginning of the plan element and the target position is either the current track position if the car is in this segment or the anchor point according to Table 6. The third point is the end of the plan element and the anchor point according to Table 6. The second point is the average of the first and third point. The derivation at the beginning and end of the spline is set to zero.

6.2 Damping of the brake and acceleration output

The damping of the brake and acceleration output is rather simple. We assume that the damping should depend on the steering angle, the idea being that the output should be damped the more the bigger the absolute steering angle is. As we are looking for monotone dampening functions, we chose a variation of the generalized logistic function as shown in Eq. 13.

$$d(s) = \left(1 - \frac{1}{(1 + Q * e^{-B*(|s|-M)})^{\frac{1}{v}}}\right) * f \tag{13}$$

The lower and upper asymptotes have been fixed to 0 and 1. Four parameters $B \in [1 \ldots 10]$, $Q \in [0.01 \ldots 1]$, $M \in [0 \ldots 1]$ and $V \in [0.01 \ldots 1]$ are included in the learning process. The parameter f is used to scale the function so that $d(0) = 1$, that is no damping is performed when the car drives straight forward. The advanced controller contains two of this functions, one to dampen the acceleration value $a, a = a * d_a(s)$ and one to dampen the brake value $b, b = b * d_b(s)$.

6.3 Clutch

This module determines the value for the clutch which we mainly use to improve the acceleration of the car, especially at the start of the race. The behavior of the clutch has been modeled as a generalized logistic function, with an extra parameter $\bar{v}$. $\bar{v}$ is the maximum speed up to which the clutch is used, above this speed the clutch is always set to zero.

Table 6 Desired target positions $\hat{\tau}$ at the connection of the two segments depending on each segments type

1st segment	2nd segment	Position $\hat{\tau}$
Straight	Left corner	$-1+\frac{4}{tw}$
Straight	Right corner	$1-\frac{4}{tw}$
Right corner	Straight	$1-\frac{4}{tw}$
Left corner	Straight	$-1+\frac{4}{tw}$

$$c(v) = \begin{cases} 1 - \dfrac{1}{(1 + Q * e^{-B*(0.003 \cdot v_{max} - M)})^{\frac{1}{v}}} & v \leq \bar{v} \\ 0 & v > \hat{v} \end{cases} \tag{14}$$

Five parameters of Eq. 14 are unknown. These are optimized for the competition in a separate scenario.

6.4 Experimental evaluation of the improved controller

With the second experiment, we want to answer the **research question** how the performance of the controller can be improved with the advanced modules described before. Due to time constraints, we ignore the clutch for now which leaves us with three modules to be tested: acceleration damping, brake damping and the planning of the racing line. For each module we have the option to use or not use it, which results in $2^3 = 8$ variations. One of these has obviously already been evaluated: The base controller is equivalent to the variant with all advanced modules disabled. That leaves us with seven alternatives yet to evaluate. Our assumption is that the advanced controller with all modules enabled performs best.

Pre-experimental planning We chose to optimize each alternative in the same scenario as used for the optimization of the base controller with the same settings used for the CMA-ES. The advanced controllers have more variables leading to a more complex optimization problem but initial test runs demonstrated that the CMA-ES can cope well with the higher dimensions of the decision space.

Task A time in the range of an excellent human driver ($\approx$360 s) has to be achieved in all runs for all variations.

Setup For each variation, we run the CMA-ES with the specific parametrization given above 10 times on the Wheel2 track (one run takes about two days on a modern 4-core machine which can handle four parallel evaluations). To evaluate the fitness of a parameter set, the car drives three laps on the track Wheel2 which contains a wide variety of different corners. An evaluation is aborted after 9 min to avoid spending too much time with uncompetitive solutions. The track model for Wheel2 is known (it has been created in advance) and the controller is set to a mode in which it always uses the generalized logistic function to determine the target speeds while planning. The optimization was done without noise and the lap time

limit, the damage and the fuel consumption were all disabled using the command line flags for TORCS introduced with the SCRC patch.

Results/visualization Figure 14 shows box plots of the best fitness for each run and for each variation. The leftmost box plot summarizes the optimization results for the base controller whose single results have been displayed in Fig. 11. Then follow the different variations with single or multiple modules added, and MrRacer is the variation with all three modules turned on.

Observations We find two unexpected behaviors in the results: (a) adding the steering module (only) leads to a decrease in performance, and (b) adding acceleration and brake damping performs better than adding all three modules.

Discussion It is obvious from the results that racing line planning (and using this with the exact steering module) alone does not improve performance, it needs adding brake damping to improve the controller. However, this leads to even better times than adding all three modules (the full advanced controller Mr. Racer). We presume that this the brake damping plus steering variation is subject to overfitting, so that these controller configurations cannot be adapted very well by the online learning to different tracks. We therefore test the two best variants (brake damping plus steering and all three modules turned on) together with the online learning on four different asphalt tracks and report the results in Table 7. Although both variants perform similarly in terms of the ranking, the added up times clearly favor the full advanced controller. This supports our assumption of overfitting of the brake damping plus steering variation. The full advanced controller with all modules turned on seems to be better balanced towards using it on different tracks.

7 Comparison to other controllers

In the previous sections we described in great detail the architecture of our controller and the functioning of the various modules. We have shown that the unknown numerical parameters can be successfully learned using an evolutionary algorithm and that the behavior can be further adapted to unknown tracks using the online learning component used in the warmup stage. The remaining question is, how the advanced controller performs against other top competitors from the Simulated Car Racing Championship. To answer this question, we test Mr. Racer (all modules enabled, see Sect. 6.4) as it was submitted to the competition in 2013 against three other top performing controllers: (1) Autopia, (2) the controller by Mariscal & Fernández and (3) Ready2Win.

7.1 Preparation of the controller

To compare Mr. Racer with the other top competitors, we prepared the controller in the same manner as we do when we submit it the Simulated Car Racing Championship: We first optimize the parameters of the clutch two times

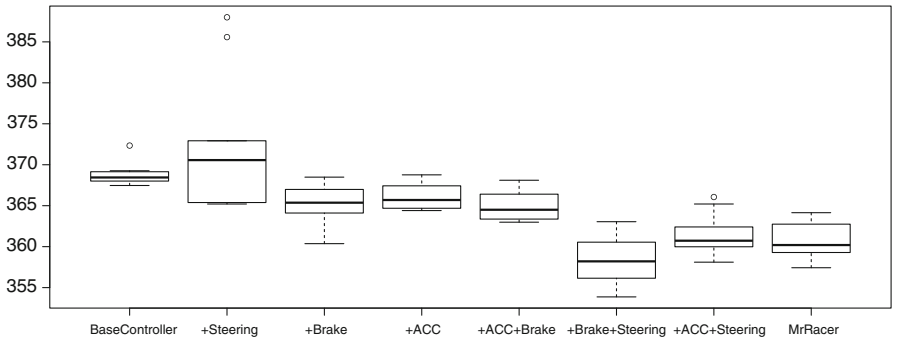


Fig. 14 Box plots of the driving times on the Wheel2 track for the best configuration found for each of 10 runs. Leftmost: the base controller, then the 7 alternatives with different modules and their combinations switched on (the 3 modules are the exact steering, brake damping, and acceleration damping). The tag MrRacer stands for all modules enabled

Table 7 Comparison of the two best variants: the advanced controller with all modules enabled (Mr. Racer) and the variant using brake damping and racing line planning

Track	Results	Brake damping and racing Line	Mr. Racer
Forza	Time (s)	**195.44**	198.76
	Points	**10**	8
Wheel2	Time (s)	244.89	**232.8**
	Points	8	**10**
CG-Speedway	Time (s)	**83.41**	87.88
	Points	**10**	8
Mueda	Time (s)	152.24	**133.47**
	Points	8	**10**
Overall	Time (s)	675.98	**652.91**
	Points	**36**	**36**

Bold entries highlight the best result for each track

independently from the other parameters: Once for a tarmac track with friction $\mu = 1.15$ and once for a dirt track with friction $\mu = 0.85$. This optimization is done using the CMS-ES with the same settings as described before. The evaluate the clutch, we use a controller which fully accelerates on a long straight for 1500 gameticks. The goal of the optimization is to maximise the distance raced.

With the clutch optimized for two different track types, we optimized the rest of the parameters for the two tracks in the same way as described in Sect. 6.4 with the clutch enabled. This results in two parameter sets, one optimized for tarmac tracks and one for dirt tracks. The parameter set best fitting the present track is selected during the warmup, as described in Sect. 5.5. The optimized values of the two parameter sets are shown in Table 8.

Table 8 Parameters of Mr. Racer used for the comparison with the other controllers

Parameter	Value set 1 (Tarmac)	Value set 2 (Dirt)
cf	0.36	0.10
bcc	1	1
$\hat{v}_Q$	0.62	0.44
$\hat{v}_B$	6.94	1
$\hat{v}_M$	0.31	0.92
$\hat{v}_v$	0.65	0.74
Clutch Q	0.57	0.89
Clutch B	4.96	3.42
Clutch M	0.03	0.27
Clutch v	0.34	0.87
Clutch $\bar{v}$	300.0	268.82
Brake damping Q	0.01	0.13
Brake damping B	4.77	2.46
Brake damping M	0.2	0.21
Brake damping v	0.42	0.68
Brake damping f	16.67	4.02
Acceleration damping Q	1	0.42
Acceleration damping B	1.56	10
Acceleration damping M	0.33	0.33
Acceleration damping v	0.29	0.29
Acceleration damping f	1.03	1
μ	1.15	0.85
ca	3.65	3.65
cw	0.08	0.08
$mass$	1150	1150
$\hat{\phi}_b$	−44.51°	−44.51°
$\underline{a}_b$	0.80	0.80
$\bar{a}_b$	0.87	0.87
$\hat{\phi}_f$	31.13°	31.13°
$\underline{a}_f$	0.23	0.23
$\bar{a}_f$	0.70	0.70

For a specific track, Mr. Racer selects one of the two parameter sets during the warmup phase as described in Sect. 5.5.3

7.2 Let the race begin

Pre-experimental planning We selected six tracks to test the controllers: Wheel2 and Forza, which are distributed with TORCS, because they resemble considerably different real racetracks (Suzuka and Monza, the first being extremely challenging, the second very fast). Furthermore, we employ CG-Speedway, a fictional track also

distributed with TORCS, and three tracks taken from previous competitions: Mueda, Kerang and Zvolenovice. The layout of the tracks can be seen in Fig. 15.

Task We want to find the best controller of the set of four given above. A clear dominance would be achieved by performing best on at least half of the tracks. In absence of a clear decision, we choose the controller with the best overall ranking (according to the SCRC point system with 10, 8, 6, and 5 points for the 1st, 2nd, 3rd, and 4th rank) as the best one. Noisy and non-noisy results should count separately, so that possibly two winners emerge.

Setup Each controller has to perform the warmup for each track twice, one time without noise and one time with noisy sensors, thereby providing it with the chance to adapt its behavior to the different tracks and conditions. The warmup phase length is set to last 100,000 gameticks, and lap time limit, damage and fuel consumption are all disabled. To evaluate the controllers, each one drove alone for two laps, without using any specific TORCS options (except setting the stage to 'qualifying'). Each controller is evaluated once without noise and once with noisy sensors, each time using the data learned in the corresponding warmup sessions.

Results/visualization Tables 9 and 10 summarize the results of the evaluation without and with noise, respectively. The last line shows the sum of points obtained by each controller on the full set of 6 tracks (for the point system please see the task description).

Observations For the noiseless scenario, the situation is very clear: Mr. Racer dominates the other controllers in the sense defined above: it wins on three tracks and comes in second for the other three. Including noise, it wins only two but still defends the second place whenever it does not win. Interestingly, there is one controller (Mariscal & Fernández) that is not able to achieve a first or second place at all. Of the three remaining controllers, Autopia performs very similar with and without noise and wins the same tracks again, whether Ready2Win seems to improve with noise enabled. Mr. Racer mostly looses some seconds on the noisy tracks but gets faster by a clear margin at least on Zvolenovice.

Discussion According to the task set above, Mr. Racer dominates the noiseless scenario, and wins the noisy scenario, with a slightly worse result. This is well in line with the expectations, as the whole controller layout has been done with a noiseless scenario in mind and then adapted to the noisy scenario. However, the loss is considerably small and even in the noisy scenario, Mr. Racer is the best guess to run for an unknown track. While the losses are small for Mr. Racer, they are rather inconsistent for the Mariscal & Fernández controller. However, the reason for this may be that the controller has been designed with a noisy scenario in mind and cannot cope too well with a noiseless scenario. Ready2Win behaves a bit more predictable and degrades only gradually without noise, but generally has the same tendency. One may state that Autopia and Mr. Racer cope with the noise, where Ready2Win and Mariscal & Fernández rely on it. In that sense, the comparison without noise is a bit unfair. However, it provides some insight into how the

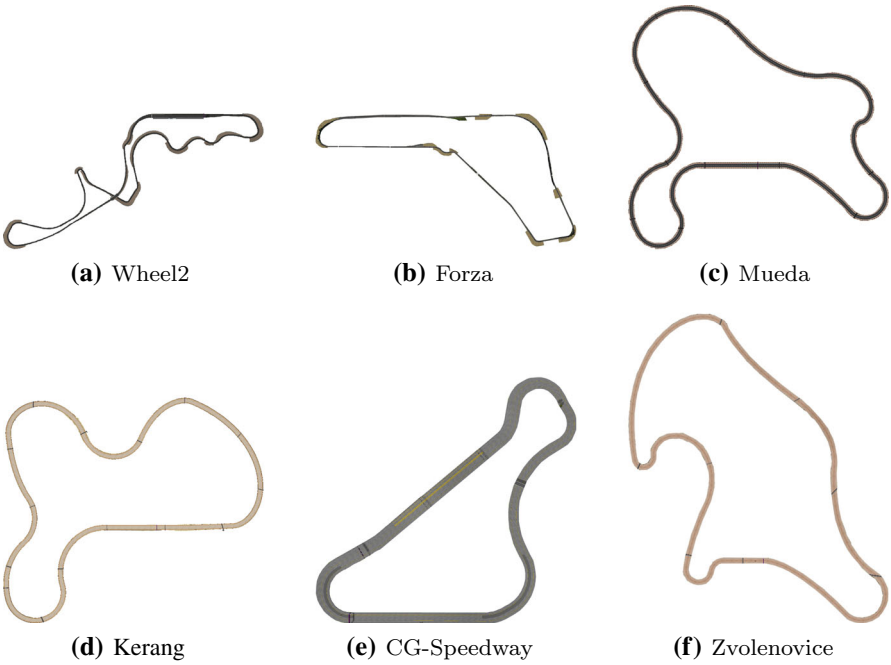


Fig. 15 Bird's eye views of the six tracks used to compare the controllers. Forza is 11 m wide, Wheel2 12 m and the other four 15 m. CG-Speedway is the shortest track with a length 2058 m, Zvolenovice is 3,174 m long, Mueda 3,611 m and Kerang 4,199 m. The tracks which resemble real race tracks are the longest with 5,784 m (Forza) and 6,205 m (Wheel2)

Table 9 Results of the evaluation without noise, 2 laps driven alone on the track after finishing the warmup

Track	Results	Ready2Win	Autopia	Mariscal&F.	Mr. Racer
Forza	Time (s)	207.26	206.27	209.28	**201.84**
	Points	6	8	5	**10**
Wheel2	Time (s)	251.38	254.15	255.7	**237.08**
	Points	8	6	5	**10**
CG-Speedw.	Time (s)	89.87	88.18	88.35	**83.36**
	Points	5	8	6	**10**
Kerang	Time (s)	162.93	**148.91**	222.57	153.27
	Points	5	**10**	6	8
Mueda	Time (s)	144.75	**136.5**	141.45	137
	Points	5	**10**	6	8
Zvolenovice	Time (s)	**166.04**	173.71	235.74	170.33
	Points	**10**	6	5	8
Overall	Points	39	48	33	**54**

Bold entries highlight the best result for each track

For the overall ranking (bottom line), we use the SCRC point scheme that assigns 10, 8, 6, and 5 points for the 1st, 2nd, 3rd, and 4th place, respectively

Table 10 Results of the evaluation with noise, under the same conditions as in Table 9, only with noise added

Track	Results	Ready2Win	Autopia	Mariscal&F.	Mr. Racer
Forza	Time (seconds)	**200.34**	209.35	210.44	201.59
	Points	**10**	6	5	8
Wheel2	Time (seconds)	**249.43**	256.23	276.22	250.41
	Points	**10**	6	5	8
CG-Speedw.	Time (seconds)	88.73	87.49	90.71	**84.65**
	Points	6	8	5	**10**
Kerang	Time (seconds)	159.31	**150.21**	200.08	154.25
	Points	6	**10**	5	8
Mueda	Time (seconds)	147.17	**136.92**	150.48	145.78
	Points	6	**10**	5	8
Zvolenovice	Time (seconds)	162.01	164.01	188.29	**155.33**
	Points	8	6	5	**10**
Overall	Points	46	46	30	**52**

Bold entries highlight the best result for each track

Ready2Win and Mariscal & Fernández have a tendency to perform better under noise, Autopia and Mr. Racer slightly worse

controllers work. Nevertheless, the comparison with noise is held under competition conditions and seeing Mr. Racer as the winner here as well is very satisfactory.

8 Conclusion

Mr. Racer has come a long way. Right from the start, it was our firm belief that the limited sensor range imposes a limit of how fast a controller based on the SCRC software interface can be and that one can only go beyond that limit by somehow gaining global knowledge about the track. We still think that this was the right approach as all the top performing controllers nowadays store global information about the track in one way or the other while other very good controllers which have not been extended by some kind of track model became more and more uncompetitive (e.g. Cobostar). Our controller was the first one to learn a model of the track using the limited information from the track sensor and although we only used a very simple heuristic to use that information, this payed off when Mr. Racer dominated the qualifying on the Forza track in the 2009 competition. The introduction of the warmup stage in 2010 played in our hands because it gives our controller the possibility to establish a track model ahead of the actual evaluation. Unfortunately, the artificial noise also introduced 2010 completely broke our track segment classifier and Mr. Racer was not able to show its true potential in the 2010 competition.

Apart from the 2009 version, every version of the Mr. Racer bot has been developed with the track model in mind. Since 2011, we sticked to the architecture

presented here and only modified submodules of the controller. As the comparison of the base controller with the advanced versions demonstrated, this modular design pays off and the incremental improvements finally led to a version of Mr. Racer which is at least on par with the state of the art, Autopia.

Another design principle of Mr. Racer is to use the available expert knowledge where possible and to learn only what is not known at all.

There are, obviously, downsides to our approach as well. More than once, we missed better results because of trivial bugs in the controller, because parts of the controller (especially the planning module) became more and more complex. This is of course not a fundamental problem of the design: it can easily be resolved by using state of the art principles of software development, e.g. a test driven approach. On the other hand, we experienced that small glitches sometimes do not matter. For example, we used wrong values to calculate the brake distance for a long time which was compensated by the EA during the optimization process.

The bigger problem is the proper selection of the best optimized values for the various parameters. In the experiments presented here, we simply chose the best individual from the evolutionary optimization which has a tendency to be overfitted to the track used for the optimization process (Wheel2). There is of course no guarantee that the same individual works well on other tracks as well. To us, being able to select a parameter set from a number of good solutions which generalizes well to other tracks is one of the last things missing to reliably outperform Autopia. A first step was taken in our work on multi-objective track selection [29] which we want to extend in the future.

Another plan for the future is the inclusion of the opponent handling into the planning process. This work has already been started but is not presented here due to space constraints.

Finally, planning an optimized racing line is probably the last possibility to further improve the performance of the current generation of controllers, including Mr. Racer. Unfortunately, it is not trivial to transfer the insights gained by e.g. Cardamone et al. [13] to our controller. Cardamone et al. use a native TORCS bot which has perfect information in contrast to Mr. Racer which has been built using the SCRC interface. It remains to be seen if the abstract representation in form of a track model allows for reliably planning a good race line, covering the whole track and not just the straight parts as we do now. We hope to find an answer to this problem in the near future.

References

1. A. Agapitos, M. O'Neill, A. Brabazon, T. Theodoridis, Learning environment models in car racing using stateful genetic programming, in *2011 IEEE Conference on Computational Intelligence and Games (CIG)* (2011) pp. 219–226. doi:10.1109/CIG.2011.6032010
2. A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in *The 2005 IEEE International Congress on Evolutionary Computation (CEC'05)*, ed. by B. McKay, et al., vol. 2 (2005) pp. 1769–1776

3. M. Botta, V. Gautieri, D. Loiacono, P.L. Lanzi, Evolving the optimal racing line in a high-end racing game, in *2012 IEEE Conference on Computational Intelligence and Games (CIG)* (2012) pp. 108–115. doi:10.1109/CIG.2012.6374145
4. M. Butz, M. Linhardt, T. Lönneker, Effective racing on partially observable tracks: indirectly coupling anticipatory egocentric sensors with motor commands. IEEE Trans. Comput. Intell. AI Games **3**(1), 31–42 (2011)
5. M. Butz, T. Lönneker, Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge, in *Proceedings of the IEEE Symposium on Computational Intelligence and Games* (2009), pp. 317–324
6. L. Cardamone, A. Caiazzo, D. Loiacono, P.L. Lanzi, Transfer of driving behaviors across different racing games, in *2011 IEEE Conference onComputational Intelligence and Games (CIG)* (2011), pp. 227–234. DOI:10.1109/CIG.2011.6032011
7. L. Cardamone, D. Loiacono, P.L. Lanzi, Evolving competitive car controllers for racing games with neuroevolution, in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09* (2009), pp. 1179–1186. doi:10.1145/1569901.1570060
8. L. Cardamone, D. Loiacono, P.L. Lanzi, Learning drivers for torcs through imitation using supervised methods, in *IEEE Symposium on Computational Intelligence and Games, 2009 (CIG 2009)* (2009), pp. 148–155. doi:10.1109/CIG.2009.5286480
9. L. Cardamone, D. Loiacono, P.L. Lanzi, On-line neuroevolution applied to the open racing car simulator, in *Proceedings of the IEEE Congress on Evolutionary Computation* (2009), pp. 2622–2629
10. L. Cardamone, D. Loiacono, P.L. Lanzi, Applying cooperative coevolution to compete in the 2009 torcs endurance world championship, in *2010 IEEE Congress on Evolutionary Computation (CEC)* (2010), pp. 1–8. doi:10.1109/CEC.2010.5586041
11. L. Cardamone, D. Loiacono, P.L. Lanzi, Learning to drive in the open racing car simulator using online neuroevolution. IEEE Trans. Comput. Intell. AI Games **2**(3), 176–190 (2010)
12. L. Cardamone, D. Loiacono, P.L. Lanzi, Interactive evolution for the procedural generation of tracks in a high-end racing game, in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11* (ACM, New York, 2011), pp. 395–402. doi:10.1145/2001576.2001631
13. L. Cardamone, D. Loiacono, P.L. Lanzi, A.P. Bardelli, Searching for the optimal racing line using genetic algorithms, in *2010 IEEE Symposium on Computational Intelligence and Games (CIG)* (2010), pp. 388–394. doi:10.1109/ITW.2010.5593330
14. M. Ebner, T. Tiede, Evolving driving controllers using genetic programming, in *Proceedings of the IEEE Symposium on Computational Intelligence and Games 2009* (2009), pp. 279–286
15. N. Hansen, The CMA Evolution Strategy: A Tutorial. https://www.lri.fr/hansen/cmatutorial110628.pdf (2011). 28 June 2011
16. M. Kemmerling, M. Preuss, Automatic adaptation to generated content via car setup optimization in torcs, in *2010 IEEE Symposium on Computational Intelligence and Games (CIG)* (2010), pp. 131–138. doi:10.1109/ITW.2010.5593361
17. D. Loiacono, L. Cardamone, P.L. Lanzi, Automatic track generation for high-end racing games using evolutionary computation. IEEE Trans. Comput. Intell. AI Games **3**(3), 245–259 (2011). doi:10.1109/TCIAIG.2011.2163692
18. D. Loiacono, L. Cardamone, P.L. Lanzi, *Simulated Car Racing Championship: Competition Software Manual* (Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Italy, 2013). http://arxiv.org/abs/1304.1672v2
19. D. Loiacono, P.L. Lanzi, J. Togelius, E. Onieva, D.A. Pelta, M.V. Butz, T.D. Lönneker, L. Cardamone, D. Perez, Y. Saez, M. Preuss, J. Quadflieg, The 2009 simulated car racing championship. IEEE Trans. Comput. Intell. AI in Games **2**(2), 131–147 (2010)
20. D. Loiacono, A. Prete, P.L. Lanzi, L. Cardamone, Learning to overtake in torcs using simple reinforcement learning, in *2010 IEEE Congress on Evolutionary Computation (CEC)* (2010), pp. 1–8. doi:10.1109/CEC.2010.5586191
21. D. Loiacono, J. Togelius, P.L. Lanzi, L. Kinnaird-Heether, S.M. Lucas, M. Simmerson, D. Perez, R.G. Reynolds, Y. Saez, The wcci 2008 simulated car racing competition, in *Proceedings of the IEEE Symposium on Computational Intelligence and Games* (2008), pp. 119–126
22. J. Muñoz, G. Gutierrez, A. Sanchis, Controller for torcs created by imitation, in *Proceedings of the IEEE Symposium on Computational Intelligence and Games* (2009), pp. 271–278

23. J. Muñoz, G. Gutierrez, A. Sanchis, A human-like torcs controller for the simulated car racing championship, in *Proceedings of the IEEE Conference on Computational Intelligence and Games* (2010), pp. 473–480
24. E. Onieva, L. Cardamone, D. Loiacono, P.L. Lanzi, Overtaking opponents with blocking strategies using fuzzy logic, in *2010 IEEE Symposium on Computational Intelligence and Games (CIG)* (2010), pp. 123–130. doi:10.1109/ITW.2010.5593364
25. E. Onieva, D. Pelta, V. Milanés, J. Pérez, A fuzzy-rule-based driving architecture for non-player characters in a car racing game. Soft Comput. Fusion Found. Methodol. Appl. **15**, 1617–1629 (2011). doi:10.1007/s00500-011-0691-6
26. E. Onieva, D.A. Pelta, J. Alonso, V. Milanés, J. Pérez, A modular parametric architecture for the torcs racing engine, in *Proceedings of the IEEE Symposium on Computational Intelligence and Games* (2009), pp. 256–262
27. D. Perez, G. Recio, Y. Saez, P. Isasi, Evolving a fuzzy controller for a car racing competition, in *Proceedings of the IEEE Symposium on Computational Intelligence and Games* (2009), pp. 263–270
28. D. Perez, Y. Saez, G. Recio, P. Isasi, Evolving a rule system controller for automatic driving in a car racing competition, in *Proceedings of the IEEE Symposium on Computational Intelligence and Games* (2008), pp. 336–342
29. M. Preuss, J. Quadflieg, G. Rudolph, Torcs sensor noise removal and multi-objective track selection for driving style adaptation, in *2011 IEEE Conference on Computational Intelligence and Games (CIG)* (2011), pp. 337–344. doi:10.1109/CIG.2011.6032025
30. J. Quadflieg, M. Preuss, O. Kramer, G. Rudolph, Learning the track and planning ahead in a car racing controller, in *2010 IEEE Symposium on Computational Intelligence and Games (CIG)* (2010), pp. 395–402. DOI:10.1109/ITW.2010.5593327
31. J. Quadflieg, M. Preuss, G. Rudolph, Driving faster than a human player, in *Applications of Evolutionary Computation, Lecture Notes in Computer Science* ed. by C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekrt, A. Esparcia-Alczar, J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, G. Yannakakis, vol 6624. (Springer, Berlin, 2011), pp. 143–152. doi:10.1007/978-3-642-20525-5_15
32. Y. Saez, D. Perez, O. Sanjuan, P. Isasi, Driving cars by means of genetic algorithms, in *Proceedings of tenth International Conference on Parallel Problem Solving from Nature* (2008), pp. 1101–1110
33. K. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002). doi:10.1162/106365602320169811
34. T. Uusitalo, S. Johansson, A reactive multi-agent approach to car driving using artificial potential fields, in 2011 IEEE Conference on Computational Intelligence and Games (CIG) (2011), pp. 203–210. doi:10.1109/CIG.2011.6032008
35. B. Wymann, *Torcs Robot Tutorial—Optimal Brake Distance*, (2013). http://www.berniw.org/tutorials/robot/torcs/robot/ch3/braking3.html. 25 Oct 2013
36. B. Wymann, The Torcs Racing Board, (2014). http://www.berniw.org/trb/index.php. 26 Mar 2014