Ahura: A heuristic-based racer for the open racing car simulator

Mohammad Reza Bonyadi, Zbigniew Michalewicz, Samadhi Nallaperuma, Frank Neumann

Abstract—**Designing automatic drivers for car racing is an active field of research in the area of robotics and artificial intelligence. A controller called Ahura (A HeUristic-based RAcer) for The Open Racing Car Simulator (TORCS) is proposed in this paper. Ahura includes five modules, namely steer controller, speed controller, opponent manager, dynamic adjuster, and stuck handler. These modules have 23 parameters all together that are tuned using an evolutionary strategy for a particular car to ensure fast and safe drive on different tracks. These tuned parameters are further modified by the dynamic adjuster module during the run according to the width, friction, and dangerous zones of the track. The dynamic adjustment enables Ahura to decide on-the-fly based on the current situation, hence, it eliminates the need for prior knowledge about the characteristics of the track. The driving performance of Ahura is compared with other state-of-the-art controllers on 40 tracks when they drive identical cars. Our experiments indicate that Ahura performs significantly better than other controllers in terms of damage and completion time especially on complex tracks (road tracks). Also, experiments show that the overtaking strategy of Ahura is safer and more effective comparing to other controllers.**

I. Introduction

DESIGNING a controller to drive a car is one of the most active research areas in the field of robotics and artificial intelligence. Many well-known companies such as Audi and Google have been investing in this topic and have gained some successes [1]. One of the main issues in this field of research is related to the difficulty of accessing facilities such as sensors, actuators, and the vehicle itself mainly due to the expenses. Hence, a realistic simulator is a good candidate to replace these resources and enable scientists to conduct research in this area. One of the most well-known car racing simulators is The Open Racing Car Simulator (TORCS) [2]. TORCS provides many different cars (bots) that can be driven through a controller. Each bot provides *all* information about the environment (e.g., the whole track and exact position of other cars) for the controller and the controller decides on the actuators (e.g., acceleration, brake, and clutch) accordingly. A more realistic extension of TORCS has been designed [3][1] that includes ten more bots, each of these bots have been equipped by some limited sensors to provide information about the car's environment at a fixed time rate (22ms in version 1.3.4 of TORCS). The bot "listens" for actions such as acceleration, clutch, braking, gear, and steer from the controller. The provided actions by the controller are applied to the vehicle and cause the vehicle to act on the track.

In this paper, a new controller called *Ahura* (A heuristic-based racer[2]) is proposed for TORCS. There are five modules in Ahura: steer controller, speed controller, opponent manager, dynamic adjuster, and stuck handler. Each of these modules contain some parameters that need to be determined. The parameters for the steer and speed controllers are adjusted for a particular vehicle using the covariance matrix adaptation evolutionary strategy (CMA-ES) [5] in a way that the controller drives fast on the track while avoids actions that apply damage to the car. After setting these parameters, the parameters for the opponent manager are set for that vehicle using another CMA-ES. This time, Ahura races against another controller (called Blocker) that has been designed to block the vehicle behind. The objective is to overtake Blocker with the smallest amount of damage in a limited time. Ahura also uses a dynamic parameter adjustment module that modifies the parameters of the controller during the race to perform according to mechanical specifications of the track.

There are four main features that make Ahura more successful comparing to the existing controllers (e.g., [6], [7], [8]):

1) Ahura evaluates the direction of the turn in front and prepares the vehicle's position (tentative position) at the track so that the centrifugal force is minimized while turning. This in fact increases the maximum safe speed (the speed that the vehicle can move with while it still does not leave the track) for taking the turn,
2) Ahura evaluates the angle of the turn in front and adjusts its acceleration policies to avoid unnecessary brakes and dangerous acceleration. This enables the controller to make a better decision on the safe

All authors are with the Optimisation and Logistics Group, The University of Adelaide, Australia. M. R. Bonyadi (mrbonyadi@cs.adelaide.edu.au, rezabny@gmail.com) and Z. Michalewicz are also with the data science group, Complexica, Adelaide, Australia. M. R. Bonyadi is also with the Centre for Advanced Imaging (CAI), the University of Queensland, Brisbane, QLD 4067, Australia. Z. Michalewicz is also with the Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland, Polish-Japanese Institute of Information Technology, Warsaw, Poland.

[1]From here on, whenever we refer to TORCS, we refer to this extension of the software rather than its original version. We used the version 1.3.4 of this extension of TORCS [4].

[2]Ahura-Mazda is the name of a higher divine spirit of an old religion called Zoroastrianism. We picked the name Ahura because of the relation to the well-known vehicles brand "Mazda".

speed to take a turn,

3) Ahura examines mechanical specifications (e.g., friction, boundaries, bumps) of the track during the run to adjust its driving decisions. This enables the controller to drive on a wide range of tracks,
4) Ahura evaluates all nearby opponents and generates a spatial map of the opponents so that it can find appropriate slots to perform overtaking by changing its tentative position on the track. This enables the controller to avoid being blocked by the opponents and overtake effectively,

The rest of this paper has been organized as follows. In section II we provide background information about TORCS and existing controllers. section III explains the proposed controller and its modules. Section IV compares Ahura with other controllers through some experiments, and section V concludes the paper.

II. Background

In this section a brief background on The Open Racing Car Simulator (TORCS) and existing controllers are given.

A. TORCS

TORCS is a well-known car racing simulator. There are ten special bots in TORCS [3], [4] that are controllable through network ports by a client (controller). There are some (virtual) sensors connected to these bots that observe the environment and send the information to the controller (see [3] for the full list and description of the sensors). There are 19 *proximity sensors* (the value of the i^{th} sensor is shown by $dist_i$) in front of the bot with predefined angles that provide the distance between the vehicle and the edges of the track towards their angle. The range of these sensors (shown by $maxD$ in this paper) is 200 meters in the current version of TORCS. The angle of the proximity sensor i is shown by $angle_i$ and it is a value in $[-90, 90]$ degrees. These angles can be set at the beginning of the simulation. The index of the proximity sensor in front of the vehicle is 0 (called the *zero sensor*) and the index of the other sensors is set from -9 to 9. In the rest of this paper we assume $angle_x = angle_{-x}$. There are 36 *opponent sensors* (the value of the i^{th} sensor is shown by opp_i) that are only sensitive to opponents. These sensors are all evenly distributed around the bot with every 10 degrees (the index of the sensor in front is 18) and their range is 200 meters. There is a track position sensor (the value is shown by $trackPos$) that provides a real value in the interval $[-\infty, \infty]$ (∞ refers to the maximum value of the type double in computer), where -1 represents the right side and 1 represents the left side of the track and other values translate to out of the track. There are 4 wheel spin sensors (one for each wheel) that calculate the speed of the wheels spin. The value for the wheel i is shown by v_i[3] and it is in m/s. There is a rpm sensor that provides the rotation per minutes of the engine and provides a real number in $[0, 10000]$. The current gear is an integer in $\{-1, 0, ..., 6\}$ (-1 is the rear gear and 0 is the neutral) that is also provided. There are 3 sensors for the current speed of the vehicle along its front (the value is shown by $xSpeed$), sides (the value is shown by $ySpeed$), and above (the value is shown by $zSpeed$). Finally, there is a sensor that calculates the current damage of the vehicle that is a real number in $[0, 10000]$.

A controller should provide appropriate values for the following actuators: acceleration pedal (shown by $accelPedal$ in this paper), braking pedal (shown by $brakePedal$ in this paper), and clutch pedal (shown by $clutchValue$ in this paper) that are real values in $[0, 1]$, gear (shown by $gearValue$ in this paper) that is an integer in $\{-1, 0, ..., 6\}$, and steer (shown by $steerValue$ in this paper) that is a real value in $[-1, 1]$, corresponding to full right and full left. The decision by the controller should be made within the simulation time interval (set to 22ms in the version 1.3.4 of TORCS). Hence, if the designed controller is slow then it might act by some delays which may cause inappropriate movements.

B. Existing drivers for TORCS

There have been some drivers based on evolutionary strategies, e.g., [8] and [7], for TORCS.

Cobostar [8] is a driver that maps sensory information to actual motor behavior. It maps this information by a function that converts sensory information to desired steering angle and speed. The parameters of this function are optimized by the covariance matrix adaptation evolutionary strategy (CMA-ES). Cobostar incorporates some other heuristically designed features such as Antilock Braking System (ABS) and Anti-Slip Regulation (ASR), gear shifting, and recovery when stuck. Cobostar won the simulated car racing competition at the Genetic and Evolutionary Computation Conference, GECCO, 2008 (see http://cig.dei.polimi.it/ for the results of simulated car racing competitions since 2008 at various conferences).

MrRacer [7] uses an expert system approach for driving. The basic design concept behind this driver is to use curvature estimation of a corner for setting a target speed, and to use two modifiers that reduce the effect of acceleration and brake in dependence of the current steering angle. CMA-ES was used to optimize the parameters of these functions. Additionally, MrRacer builds a track model during the warm up phase and uses this information during the race. The later versions of MrRacer follow a rather offensive opponent handling strategy that tries to overtake the opponents in front and blocks the opponents behind. Furthermore, an online

[3]The radius of the wheels is available in the car dynamics files of the simulator and, for the mentioned special cars in TORCS, it is 0.3179 meters for the front wheels and 0.3276 meters for the rear wheels.

adaptation is used by MrRacer to further adjust the parameters (previously tuned in off-line mode) during the warm up phase. MrRacer won the simulated car racing competition at the IEEE Computational intelligence and games (CIG) conference, 2011.

Similar to MrRacer, EVOR [9] is a driver that makes use of a track model built during the warm up phase. However, in contrast to the off-line parameter optimization strategy (e.g., in Cobostar and MrRacer), EVOR uses a dynamic optimization strategy to decide the actuator values. A dynamic optimization process optimizes the actuator outputs based on the static track model and the dynamic sensory information received during the race. The fitness function is based on how well the car trajectory resultant from the current status information suits the track model. The opponent handing is also considered implicitly within this strategy that if opponents intersect with the estimated trajectory they are considered as obstacles and the speed is adjusted to avoid any collision.

Autopia [6] is also an expert system quite similar to MrRacer and Cobostar. The significant difference between this driver and the others mentioned before is that it uses fuzzy inferencing to determine the target speed [10]. The parameters related to the driving functions are tuned using CMA-ES. Furthermore, Autopia performs online parameter adaptation during the warm up phase where the target speed is adapted based on the damage. For example if the car is off the track at a position then the target speed at this position is reduced. Autopia has rather a defensive opponent handling strategy that is designed heuristically. Autopia has won the simulated car racing competition at various conferences, including CIG and GECCO 2010.

These four controllers are considered for the experiments in this study because they have shown good performance in the literature [8], [7], [6], [9] and they have been successful in several car racing competitions (see http://cig.dei.polimi.it/ for the results of simulated car racing competitions since 2008 at various conferences). There are also other controllers for TORCS based on expert systems [11], [12], artificial neural networks [13], [14], fuzzy expert systems [15], [16], and genetic programming [17], [18], however, their performances are not as competitive as those four controllers.

Despite their competitive performance, these drivers exhibit various drawbacks. For example, Cobostar and Autopia use the index of the proximity sensor that has the maximum value for the steer calculation, ignoring the forces that the car experiences during the turn. However, these forces enforce the vehicle to move slower during the turn to avoid leaving the track. Also, none of these controllers consider mechanical specifications of the track, such as friction, to adjust their driving style. However, such mechanical specifications of the track significantly affect the safe speed as well as acceleration and braking performances of the vehicle. Although some of these information (e.g., friction) are not available by sensors in TORCS, many of them can be estimated readily (see subsection III-E). Further, the fact that the proximity sensors have 200 meters of range has not been used by any of these drivers while such information can be useful to make better decisions on the speed and steer of the vehicle. As an example, this range can be effectively used to evaluate the turn in front and make better decisions. Finally, none of these controllers implement effective overtaking strategy, e.g., Autpia tries to avoid the opponent and Cobostar just simply ignores the opponents. However, opponent handling is obviously essential in racing.

III. Ahura: A heuristic-based racer for TORCS

In this section we propose a new controller called Ahura for TORCS. The controller uses five modules:

- **Steer controller**: this module uses the estimated angle of the turn in front and the vacant distance in front to determine the steer angle. The module can control how smooth or sharp the vehicle is going to turn.
- **Speed controller**: this module uses the estimated turn angle together with the vacant distance in front to decide the safe speed. In order to accelerate the vehicle to achieve that speed, the controller uses ASR/ABS technologies to avoid loosing the controller of the car.
- **Opponent manager**: this module creates a map of opponents around and finds the vacant slot to overtake. This action may entail modification of the speed and steer calculated by the speed and steer controller modules.
- **Dynamic adjuster**: this module uses the mechanical specifications of the track (friction, bumps) as well as recorded difficulties the controller has experienced during the earlier laps and adjusts the current driving style.
- **Stuck manager**: this module uses the idea proposed in [8] to control the vehicle when it is out of the track or it has stuck somewhere.

These modules are described in detail in the following subsections.

A. Steer controller

Steer controller determines the value for the steer of the vehicle without considering opponents. The main idea behind calculation of the steer angle is to find the proximity sensor that has the maximum empty space in front (called the *base sensor*), i.e., $base\ sensor = \{j | dist_j < dist_i,\ \ for\ any\ proximity\ sensor\ i\}$. The angle of the base sensor (the value of $angle_{base\ sensor}$) is then used to set the angle of the steer. However, moving exactly towards the base sensor might cause some issues (see subsection III-A2). Thus, the steer angle is calculated according to the base sensor and some sensors around it, called *auxiliary sensors*. Although the steer angle should

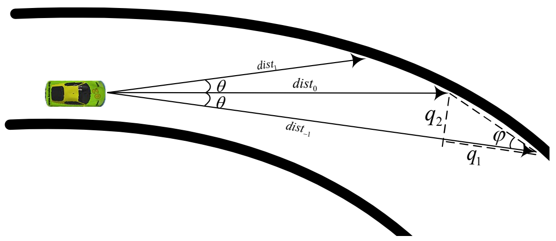

Fig. 1: The angle of the turn in front is estimated based on $dist_{-1,0,1}$ sensors.

Algorithm 1 Estimate the turn in front
Input: $dist$
Output: $estimatedTurn$ (estimated turn in front)

```
1: if $dist_1 > dist_0$ then
2:     $k = \frac{sin(\theta)dist_1}{dist_0 - cos(\theta)dist_1}$
3: else
4:     $k = \frac{sin(\theta)dist_0}{dist_{-1} - cos(\theta)dist_0}$
5: end if
6: $estimatedTurn = tan^{-1}(k)$
```

be determined when the vehicle arrives to a turn, the action before arriving to the turn is also important. Thus, the vehicle is prepared as soon as a turn was detected in front to take the turn with maximum speed and safety (see subsection III-A3). The preparation for turn involves estimation of the angle of the turn in front. Hence, we describe the algorithm to estimate the angle of a turn in subsection III-A1.

1) Estimation of the angle of a turn: To estimate the angle of the turn in front we use three proximity sensors -1, 0, and 1. Let us assume that the angle between these sensors is θ ($angle_{-1} = angle_1 = \theta$). The angle of the turn (φ in Fig. 1) in front is calculated by $tan^{-1}(\frac{q_2}{q_1})$. The value of q_1 and q_2 are calculated by $dist_{-1} - cos(\theta)dist_0$ and $sin(\theta)dist_0$. Thus, Algorithm 1 is used to estimate the turn in front.

Smaller values for θ lead to better estimations for the value of φ. The reason is that the proposed estimation procedure uses two sample points on the edges of the track (provided by $dist_{-1}$ and $dist_0$ or $dist_1$ and $dist_0$) to estimate the angle of the turn. Thus, closer samples lead to more accurate calculation of the turn angle. Hence, we set $\theta = 1$ in our experiments.

2) Taking a turn: The angle of the steer is set according to the angle of the base sensor (the value of $angle_{base\ sensor}$). The trajectory towards the base sensor, however, might be very close to the edges of the track that might cause the vehicle to move out of the track. Also, if the angle between the base sensor and the direction of movement is very large (a sharp turn) then setting the steer angle to this large value might cause losing the control of the vehicle and slipping to the sides of the track. To avoid these situations, the steer angle is calculated according to a weighted average of the base sensor and some other sensors around it (see Fig. 2). We

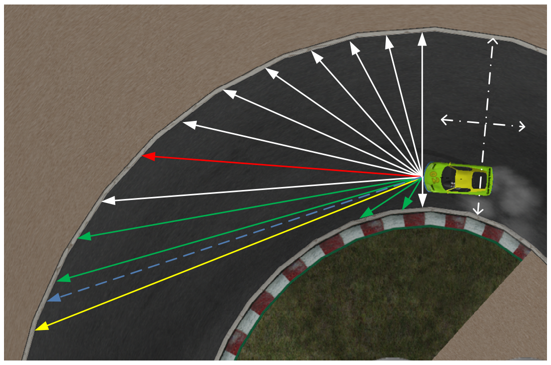

Fig. 2: The red vector, the yellow vector, green vectors, and the blue vector represent the zero sensor, the base sensor, auxiliary sensors, and the final angle to take the turn, respectively.

call these extra sensors as auxiliary sensors.

Because a weighted average over all auxiliary sensors and the base sensor is used for calculation of the steer angle, the smoothness of the changes of the steer angle is a function of the number of auxiliary sensors. If the distance in front of the vehicle is too small ($dist_0$ is small) then the vehicle needs to turn immediately and sharply. Thus, smaller number of sensors are preferred to enable the vehicle to change the direction more rapidly. However, if this distance is large then there is no need for large and sudden changes of the steer angle. Hence, the number of auxiliary sensors is a function of $dist_0$. We used a logarithm sigmoid function to map the value of $dist_0$ to the number of auxiliary sensors.

$$y = logSig(x_1, x_2, y_1, y_2, x) = \frac{a}{1 + e^{b(x+c)}} + d \qquad (1)$$

where $x = dist_0$ and a, b, c, and d are defined as

$$\begin{aligned} a &= y_2 - y_1 \\ d &= y_1 \\ b &= \frac{ln(\frac{a}{1.01y_1 - d} - 1) - ln(\frac{a}{0.99y_2 - d} - 1)}{x_1 - x_2} \\ c &= \frac{ln(\frac{a}{1.01y_1 - d} - 1)}{b} - x_1 \end{aligned} \qquad (2)$$

With these settings[4], the logarithm sigmoid function generates $0.99y_1$ for $dist_0 = x_1$ and $0.99y_2$ for $dist_0 =$

[4]There might be different ways to design a function to map $dist_0$ to the number of sensors, we designed this function manually. We need a function f that maps the distance in front (x) to the number of sensors (y), i.e., $y = f(x)$. The number of sensors should remain constant if the distance in front is larger than a value y_2 or smaller than another y_1 (assuming $y_1 < y_2$). The reason is that the maximum number of sensors is constant and the minimum number of sensors can not be smaller than 1. Hence, the function f has two asymptotes at $y = y_1$ and $y = y_2$. If we assume that the minimum and maximum number of auxiliary sensors are x_1 and x_2, respectively, then $f(x_1)$ should become very close to y_1 and $f(x_2)$ should become very close to y_2. Of course one option for such function is a line that connects the points (x_1, y_1) and (x_2, y_2) when $x \in [x_1, x_2]$ while has a constant value outside of the interval $[x_1, x_2]$. However, we experimentally found that the $f(x)$ is not linear in the interval $[x_1, x_2]$, hence, a logarithm sigmoid function was used. We assumed that $f(x_1) = y_1 + 0.01y1$ and $f(x_2) = y_2 - 0.01y_2$ (closer than 99% to y_1 and y_2 for $x = x_1$ or x_2, respectively). We then derived the values for a, b, c, and d accordingly.

Algorithm 2 Weighted average of the sensors
Inputs: $angle$, $dist$, $base\ sensor$, α (a tentative position)
Output: $SteeringAnle$ (steer angle)

```
1: given $x = dist_0$, calculate $y$ by Eq. 1
2: $L = \{base\ sensor - y, ...base\ sensor, ..., base\ sensor + y\}$
3: $h = g = 0$
4: for each $i$ in $L$ do
5:     if $i$ is the base sensor then
6:         $d = 2dist_i$
7:     else
8:         if $angle_i > angle_{base\ sensor}$ then
9:             $d = \frac{dist_i}{\alpha}$
10:        else
11:            $d = \alpha \times dist_i$
12:        end if
13:    end if
14:    $h = h + d \times cos(angle_i)$
15:    $g = g + d \times sin(angle_i)$
16: end for
17: $steeringAngle = tan^{-1}(\frac{g}{h})$
```

x_2. The total number of auxiliary sensors is $2y$ because the value y refers to the number of auxiliary sensors at one side of the base sensor (in our implementation, the number of auxiliary sensors is set to $2round(y)$ where $round(y)$ is the closest natural number to y). The values of y_1, y_2, x_1, and x_2 are the parameters that are tuned by an evolutionary strategy in section III-D.

During the run, x is substituted by $dist_0$ in Eq. 1 to calculate y. The value of y is then used to generate a list L defined by $\{base\ sensor - y, ...base\ sensor, ..., base\ sensor + y\}$ (see section III-A for the calculation of the base sensor). The list L is used to determine the steer angle by the algorithm 2.

The value of α in Algorithm 2 is the *tentative position* of the vehicle at the track while moving. If $\alpha = 1.0$ then the vehicle tends to stay in the middle of the track. If $0.0 \le \alpha \le 1.0$ the distances that are at the left hand side of the base sensor become larger (as they are divided by $0.0 < \alpha < 1.0$) and the distances at the right hand side of the base sensor become smaller (as they are multiplied by $0.0 < \alpha < 1.0$). This multiplication/division results in more attraction towards the left hand side of the track. Also, if $\alpha > 1.0$ the distances that are at the right hand side of the base sensor become larger and the distances on the left hand side of the base sensor become smaller, resulting in more attraction towards the right hand side of the track. The value of $dist_{base\ sensor}$ has been multiplied by 2 in the algorithm to increase the effect of this sensor on the calculation of the steer angle.

3) Preparation for a turn: Although the steer angle should be determined to take a turn, the action before taking the turn is also important because of the forces that the vehicle experiences during the turn. The vehicle experiences a force during the turn outwards of the track perpendicular to its body, called *centrifugal force*. The centrifugal force is proportional to the velocity, mass, and the angle of the steer (a small angle refers to a smooth turn and a large angle refers to a sharp turn), i.e., for a constant mass, the smaller the steer angle is, the smaller the force will be. Hence, for a constant mass and a constant centrifugal force, a smaller angle of steer allows larger velocity. Therefore, during a turn, if a smaller angle for the steer is selected then the vehicle is able to move faster while the centrifugal force remains constant. In order to minimize the angle of the steer (and consequently maximize the velocity) to take a turn the vehicle needs to take the largest curve that in fact requires starting the turn from the further edge of the track (the red trajectory in Fig. 3). Thus, a strategy is needed to enable the vehicle to move towards the opposite side of the track before arriving to the turn and then takes the turn. We call this strategy *opposite sliding* (see also [19]). Although a longer distance should the vehicle travel if this strategy is used (comparing to the other strategy in Fig. 3, shown by a white trajectory), the velocity that the vehicle can move by using this strategy is considerably larger than the other's that not only compensates for the larger distance, but also causes saving more time [5].

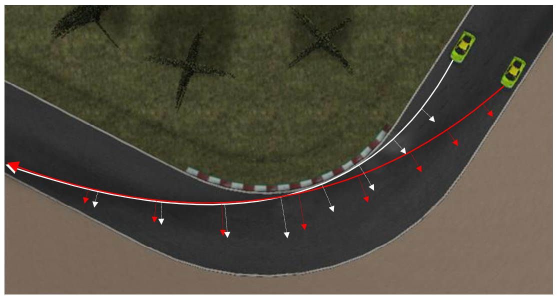
Fig. 3: Opposite sliding before the turn (red vector) in comparison to no sliding before the turn (white vector)

The opposite sliding in Ahura is implemented by setting the value of α in Algorithm 2. Let us assume that $\alpha = h^{s\beta}$ $(h > 0)$[6], where $s \in \{-1, 1\}$ determines the tentative position towards right/left side of the track and $\beta >= 0$ determines the amplitude of the position towards the sides of the track. The value of s can be determined according to the estimated turn angle in front: if the turn is to the left ($estimatedTurn < 0$ in Algorithm 1) then $s = -1$ to slide the vehicle to the right hand side of the track and vice versa.

The value of β is a function of the distance in front ($dist_0$). The reason is that if $dist_0$ is too large or too small then it is better to set $\alpha = 1.0$ that entails setting $\beta = 0$. If $dist_0$ is in the mid range then it is better to conduct the opposite sliding that needs $\alpha > 1.0$ or $\alpha < 1.0$ (depending on the turn direction in front) that entails

[5]This strategy is also used by professional human drivers in racing.
[6]Any positive value for h can be considered, however, we set this value to 4 in our experiments.

$\beta = 1.0$ (note that the direction of the slide is determined by s while β is only used to determine the amplitude of the slide). Hence, we used a trapezoidal function (Eq. 3) to calculate appropriate value for β as a function of $dist_0$.

$$y = trap(a,b,c,d,x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x \leq b \\ 1.0 & b < x \leq c \\ \frac{c-x}{d-c} + 1.0 & c < x \leq d \\ 0 & x > d \end{cases} \quad (3)$$

where $x = dist_0$, $y = \beta$, a is the distance where the vehicle should focus on taking the turn with no opposite sliding ($\beta = 0$ for $x \leq a$), b is the distance where the vehicle should begin taking the turn and ending the opposite sliding (β drops from 1 to 0 for $a < x \leq b$), c is the distance where the vehicle should be in the opposite sliding stage ($\beta = 1$ for any $b < x \leq c$), and d is the distance where the opposite sliding stage begins (β grows from 0 to 1 for $c < x \leq d$). The parameters a, b, c, and d are determined by an evolutionary strategy in section III-D.

As the opposite sliding procedure is not useful for wide angle turns, the value of β is set to zero if the estimated turn (determined by the Algorithm 1) is small. The threshold for the estimated angle of the turn to use the opposite sliding is set to 0.1 in our experiments.

B. Speed controller

The aim of the speed controller is to determine the speed that the vehicle can move by according to the current situation without considering opponents. This decision then is translated to the acceleration/breaking pedals.

1) Target speed calculation: A human driver decides the best speed according to the vacant distance and the angle of the turn in front of the vehicle. The target speed of the vehicle is a monotonically increasing function of $dist_0$. Also, for a constant $dist_0$, a more cautious policy is taken if the turn in front is sharper. Thus, we designed a function (Eq. 4) to calculate the speed that the vehicle can move by according to $dist_0$ and the angle of the turn in front.

$$targetSpeed = \left(\frac{dist_0}{maxD}\right)^{\lambda}(maxS - minS) + minS \quad (4)$$

where $targetSpeed$ is the speed that the vehicle should move by, $maxS$ is the maximum speed of the vehicle (set to 360 as this value is the maximum speed the available bot in TORCS can move by), and $minS$ is the minimum speed (set to 25). The value of λ is a function of the angle of the turn in front that determines how cautious the speed should be changed, i.e., for smaller values of λ the $targetSpeed$ grows rapidly (as a function of $dist_0$) that results in less cautious changes of the speed (see Fig. 4). Also, for larger values of λ the target speed only grows when $dist_0$ is large that results in more cautious changes of the speed.

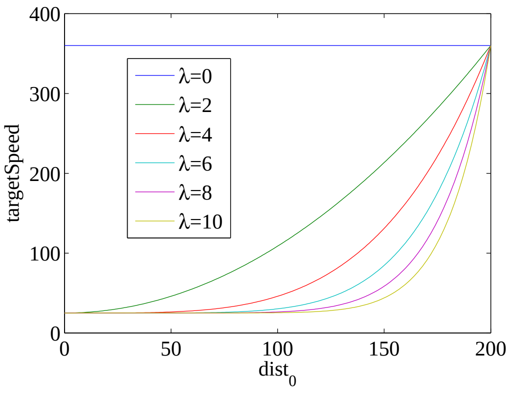

Fig. 4: The effects of λ on the target speed calculations.

We considered that the value of λ is a function of the estimated turn in front (estimated turn is calculated by Algorithm 1). For a sharp turn, the value of λ should be large so that the vehicle is more cautious and vice versa. We relate λ to the estimated turn through a logarithm sigmoid (Eq. 1) function. Let us assume that for e_1, an estimated angle of a turn, the value of λ is λ_1 and for e_2, an estimated angle for another turn, the value of λ is λ_2. We replaced the values of x_1, x_2, y_1, and y_2 in Eq. 1 by e_1, e_2, λ_1, and λ_2, respectively. The values of λ_1, λ_2, e_1, and e_2 are the parameters that are determined by an evolutionary strategy described in subsection III-D[7].

The target speed generated by Eq. 4 is translated to a value in the interval $[-1, 1]$, where negative values represent brake action and positive values represent acceleration action. We used the function proposed in [20] for this translation (see Eq. 5).

$$p = \frac{2.0}{1.0 + e^{b(xSpeed - targetSpeed)}} - 1.0 \quad (5)$$

If $p < 0$ then $brakePedal = |p|$ and $accePedal = 0.0$ and if $p \geq 0$ then $brakePedal = 0$ and $accePedall = p$. The value of b is set to 1.0 unless the vehicle is stuck (see section III-E).

2) Anti-lock Braking system and Anti-slip regulation: After calculation of the $accelPedal$ and $brakPedal$, it is necessary to make sure that these values are applied in an appropriate way. As an example, in a low friction road a sudden lock of the brakes is not as efficient as pulsing the brake pedal [21]. This is also the same for the acceleration pedal. The ABS and ASR technologies solve these issues through keeping the speed of the spinning wheels and the speed of vehicle as close as possible. During the brake, the spin of the wheels might become smaller than the speed of the vehicle that means the wheels are slipping. Also, during the acceleration the

[7]The value of λ_2 significantly impacts the changes of the speed of the vehicle. This value is further adjusted according to mechanical specifications of the track in section III-E.

spin of the wheels might become larger than the speed of the vehicle that means the vehicle is in traction.

To implement ABS, we calculated the difference between the speed of the vehicle and the speed that the wheels by $d = |xSpeed - \sum_{i=1}^{4} r_i v_i|$, r_i is the radius of the wheel i. If d is larger than $ABSSlip$ then the value for the brake pedal is revised to $brakePedal - \frac{d - ABSSlip}{ABSrange}$. Also, if $\sum_{i=1}^{4} v_i/4$ is smaller than $ABSMinSpeed$ then there is no need to apply ABS as the vehicle is moving slowly. The parameters $ABSSlip$, $ABSrange$, and $ABSMinSpeed$ are determined to maximize the efficiency of the braking system (see subsection III-D).

The implementation of ASR is very similar to that of ABS. If d is larger than the maximum ASR speed ($ASRSlip$) then the value for the acceleration pedal is revised to $accelPedal + \frac{ASRSlip - d}{ASRrange}$ (note that $ASRSlip - d$ is negative, thus p becomes smaller). If the value of $\sum_{i=1}^{4} v_i/4$ is larger than $ASRMaxSpeed$ then p is not changed. The parameters $maxASR$, $ASRrange$, and $ASRMaxSpeed$ are determined to maximize the efficiency of the acceleration system (see subsection III-D).

3) Gear control and clutch: We use a simple gear changing procedure that is based on the minimum and maximum values of the rpm for each gear. We set two thresholds for the motor's rpm for each gear i, one for the gear up (u_i) and one for the gear down (d_i). The values for $u_{1,...,6}$ and $d_{1,...,6}$ were set to $\{9500, 9500, 9500, 9500, 9500, 0\}$ (e.g., if gear=2 then change the gear to 3 if rpm is larger than $u_1 = 9500$) and $\{0, 3300, 6200, 7000, 7300, 7700\}$ (e.g., if gear=2 then change the gear to 1 if rpm is smaller than $l_2 = 3300$) through some experiments.

The clutch is used to change the gears. However, clutch for the first gear is used in a special way by professional human drivers. When the gear is 1, the clutch is pushed half way through and it is released slowly in a way that the spin of the wheels remains as close to the speed of the vehicle as possible. This prevents the wheels from slipping too much (maximizes the friction between the wheels and the track surface) so that the vehicle accelerates faster with less traction.

C. Opponent manager

Ahura's opponent manager contains two modules, steer reviser and speed reviser, that are responsible to revise steer and speed for overtaking. Ahura builds a spacial map of the position of opponents based on the information provided by the *opp* sensors. This map is used to revise steer and speed of the vehicle for overtaking purposes.

1) Opponents spacial map: The information about opponents provided by *opp* sensors contain their distance and angle from the current position of the vehicle. This means that the position of opponents is provided in a polar coordinates system with the center of the measuring vehicle. Ahura builds a spacial map of the opponents positions according to the given *opp* sensors. This map is

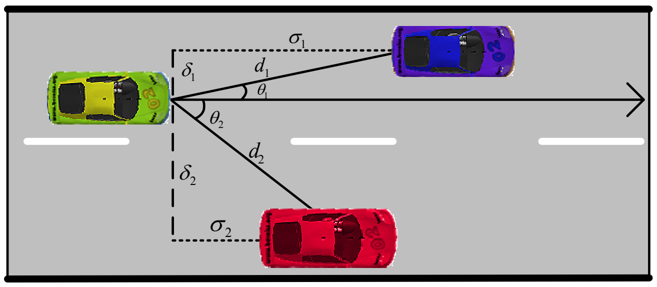

Fig. 5: Building a spacial map of other opponents using the opp sensors information.

built using basic conversions from the Polar coordinate (distance-angle) to the Cartesian coordinate (x-y).

As an example, in Fig. 5 the values for θ_1, θ_2, d_1, and d_2 are provided by opp sensors. Clearly, $\sigma_1 = d_1 cos(\theta_1)$, $\sigma_2 = d_2 cos(\theta_2)$, $\delta_1 = d_1 sin(\theta_1)$, and $\delta_1 = d_2 cos(\theta_2)$. Ahura uses this calculation to determine the position of opponents (δ_i and σ_i for each opponent i). The result is a spacial map of the position of opponents in the Cartesian coordinate. According to this map, Ahura is able to find vacant positions at the track and determine the closest achievable vacant position that can fit in (this position is called $bestVacant$). Also, the generated spacial map is used to determine the distance between Ahura and the opponent in front, if there is any.

2) Steer reviser: In order to overtake, the calculations in section III-A for the steer angle should be revised to prevent crashes with opponents and overtake successfully. The $bestVacant$ position calculated in section III-C1 is the target position that Ahura tries to go to. To do so, the value of α in Algorithm 2 is set in a way that the tentative position of the vehicle is set to $bestVacant$. However, it is not safe to slide to the found vacant position very fast because it might need intense changes of the steer angle that might result in losing the control of the vehicle. Hence, the intensity of the changes in the value of α is set according to the distance between the current position of the vehicle and the vacant position, i.e., $\alpha = q|trackPos - bestVacant|$ where q is the *maneuver factor*. The value of α (and consequently the steer angle) changes more intense for larger values of the maneuver factor. In reality, the faster the vehicle approaches a vehicle in front, the larger the changes in the value of steer angle should be. This means that for a large relative speed (approaching the vehicle in front very fast) the steer value need to change faster, that entails a larger value for q. Also, if the relative speed is small then a small value for q is sufficient. We assumed that $q = \omega_1 v' + \omega_2$, where v' is the relative speed and ω_1 and ω_2 are parameters.

Calculation of v', the relative speed between Ahura and the car in front, entails estimation of the distance between the two vehicles in the next time slice. The distance between Ahura and the opponent in front can be calculated from the generated spacial map. Ahura records the changes of the distance of the opponent in front and uses the Lagrange interpolation technique with 4 points [22] (set experimentally) to estimate the distance in the next simulation time slice (next 22ms). This estimation of the distance is then used to estimate the speed of the vehicle in front using the basic velocity calculation formula: $v = \frac{x_2 - x_1}{t_2 - t_1}$ where x_2 is the extrapolated distance, x_1 is the current distance, and $t_2 - t_1$ is the time slice duration. If Ahura recognized that there is an opponent on one of its sides and it is vacant in front then the tentative position is kept constant until the opponent is passed by a far enough distance.

Ahura determines the distance to start overtaking (shown by $minDistToOvertake$) according to the relative speed with the vehicle in front. The faster Ahura is approaching the vehicle in front, the earlier it needs to overtake. We assumed that there is a linear relationship between $minDistToOvertake$ and v' , i.e., $minDistToOvertake = \omega_3 v' + \omega_4$. If the distance from the opponent in front is smaller than $minDistToOvertake$ then Ahura starts changing the steer to overtake, otherwise, no action is taken. The parameters ω_1, ω_2, ω_3, and ω_4 are set in section III-D.

If an opponent was detected and Ahura started to overtake, the number of auxiliary sensors (Algorithm 2) is set to maximum possible (10 in our implementations) to make the movement as accurate and smooth as possible. In addition, if Ahura detected an opponent at either of its sides then the $targetSpeed$ is multiplied by 1.1 to make the overtaking process faster.

3) Speed reviser: The aim of the speed reviser is to revise $targetSpeed$ to avoid crashes with other opponents. It is obvious that a vehicle A that is chasing a vehicle B in front does not crash with B if it is moved at the same speed or slower than B. Thus, Ahura estimates the speed of the vehicle in front in the next time slice and sets its own speed accordingly. To estimate the speed of the vehicle in front, the value of the current speed of Ahura ($xSpeed$) is added to the relative speed v' (calculated in section III-C3). The $targetSpeed$ is then set to the speed of the vehicle in front. The speed is revised only if the vehicle in front is closer than $minDistToBrake = \omega_5$, otherwise, it remains unchanged. The parameter ω_5 is set in section III-D.

Fig. 6 summarizes the speed calculation, steer calculation, and opponent management and their relation.

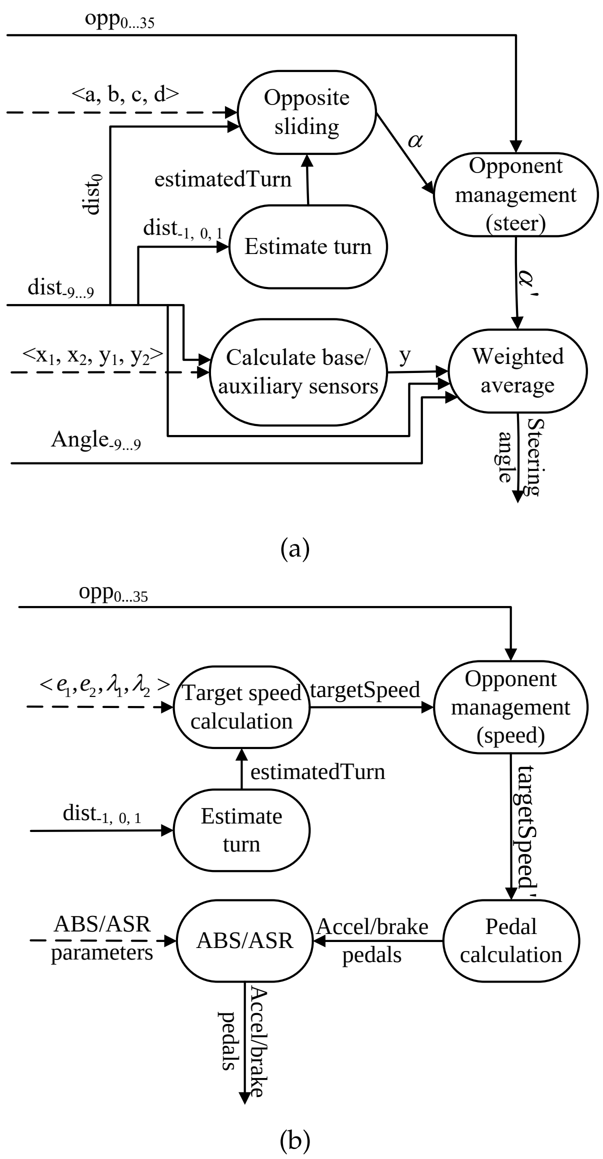

Fig. 6: How (a) steer and (b) speed are calculated in Ahura. Dashed lines represent values that are set by an evolutionary algorithm and solid lines are the values that are provided by the sensors. The parameter values indicated by dash (′) are the values revised by the opponent manager module.

D. Optimization of parameters

The parameters for steer and speed controllers as well as the opponent manager give Ahura the flexibility to be able to drive a large range of vehicles. In this section, we use an optimization-based approach to find the best values for these parameters so that Ahura can drive a specific bot in TORCS.

There are 23 parameters in Ahura that need to be determined:

- 8 parameters for the steer controller (x_1, x_2, y_1, y_2, a, b, c, and d),
- 10 parameters for the speed controller (e_1, e_2, λ_1, λ_2, $minABSSpeed$, $minASRSpeed$, $ABSrange$, $ASRrange$, $minABS$, and $minASR$),
- 5 parameters for the opponent manager (ω_1, ω_2, ω_3, ω_4, and ω_5).

Eighteen of these parameters are *independent* from opponents (interaction between the vehicle and the track) while 5 others are related to handling opponents (*dependent* parameters). There are four important characteristics in optimizing these parameters that leads us to select an appropriate optimization algorithm.

1) All parameters are in the continuous space,

TABLE I: Average and standard deviation of the objective value over 10 runs (1 lap each) when the optimized paramters for one track were tested on other tracks. Each column represents the average objective values (total time + damage/2) where Ahura used the parameters optimized for the track specified in the heading of that column.

	$P_{\text{Wheel 2}}$	P_{Eroad}	$P_{\text{Street 1}}$	$P_{\text{Alpine 2}}$
Wheel 2	**118.2 (0.29)**	127.2 (1.22)	154.1 (1.41)	144.4 (2.09)
Eroad	69.7 (1.01)	**68.2 (0.25)**	73.8 (1.29)	79.2 (1.91)
Street 1	130.4 (0.86)	135.3 (1.03)	**87.7 (0.19)**	109.6 (0.91)
Alpine 2	113.1 (0.92)	151.5 (1.04)	143.0 (1.11)	**102.6 (0.24)**
Average	**107.8**	120.55	110.9	112.7

2) The effect of the changes of the value of parameters on the behavior of the vehicle is non-linear (changing the value of λ affects the speed in a non-linear way, see Fig. 4),
3) There is no constraint,
4) Some parameters are non–separable (changing the value of one parameter changes the best possible choice for another), e.g., if the parameters are set in a way that the vehicle moves faster then the best choice for steering parameters are changed.

We use Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [5] for the optimization purposes because it has a good performance in continuous space, works with non-linear systems, no constraint handling technique is required, and it is appropriate for non-separable search spaces [23].

CMA-ES was used to set the independent parameters of Ahura where each individual represented all 18 independent parameters. We selected four different tracks, namely Wheel 2, Eraod, Street 1, and Alpine 2 (see these tracks in Fig. 7) to optimize these parameters for. These tracks were used because, all together, they could cover a wide range of possible combination of turns that might exist in a track. For each track, we run CMA-ES for 20 times for 5000 function evaluations and selected the parameters combination that had the best objective value *total time* + *damage/2* for that track over all 20 trials. This procedure led us to 4 parameters combinations, each combination represented the best found parameters for one of the tracks. To select the final parameters, we run Ahura with each of these parameters on all 4 tracks and selected the parameters combination that had the minimum value for *total time* + *damage/2* over all tracks. Table I shows the average and standard deviation of the results. Note that the standard deviation of the objective value are small in these tests because the environment was almost static, hence, the decisions made by the controller were almost the same at each run. According to this table, the minimum average of objective function for these tracks achieved when the best found parameters for the track Wheel 2 are selected.

One should note that, as these four tracks can represent a wide variety of situations, we expect that the parameters which lead to a good performance of the controller at all of these tracks also show a good performance on most other tracks. Of course there might be some situations that is not experienced by the controller in these tracks that might take place at other tracks. However, we experimentally found that such situations are rare and, if they occur, they can be handled by the dynamic adjustment procedure described in section III-E.

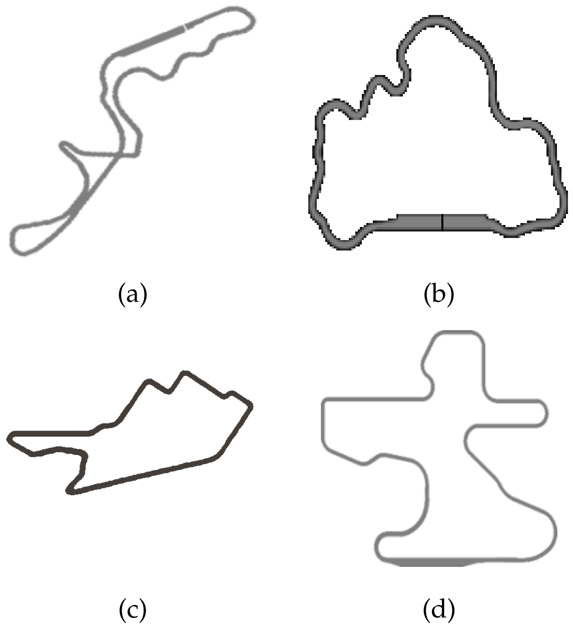

Fig. 7: four tracks used for optimization (a) wheel-2, (b) eroad, (c) street-1, and (d) Alpine 2.

The final independent parameters were set as follows:

- For the number of auxiliary sensors (section III-A2): $x_1 = -0.230$, $x_2 = 95.303$, $y_1 = 2.012$, $y_2 = 6.231$,
- For the opposite sliding (section III-A3): $a = 50.951$, $b = 90.480$, $c = 164.116$, $d = 187.908$,
- For the target speed (section III-B1): $e_1 = -5.079$, $e_2 = 5.382$, $\lambda_1 = 0.020$, $\lambda_2 = 6.900$,
- For the ABS and ASR parameters (section III-B2): $ABSRange = 3.198$, $ABSMinSpeed = 3.401$, $ABSSlip = 1.018$, $ASRRange = 1.183$, $ASRMinSpeed = 149.190$, $ASRSlip = 1.190$

Fig. 8 shows the convergence curve of CMA-ES for four selected tracks for parameter setting purposes.

We also used CMA-ES to set 5 dependent parameters of Ahura. In order to evaluate individuals, we run Ahura against another bot called *Blocker* using the values of parameters in that individual. Blocker is an instance of Ahura, except for its $maxS$ that is set to 300 to make it slower and it does not use the opponent sensors at its front. Blocker uses the opponent sensors at its back and tries to block the behind vehicle. Using the values in the CMA-ES individuals for the dependent parameters, we run Ahura against Blocker to minimize the following objective: $500\times$*standing*+*damage*, where standing is the position of Ahura after the race. This was done on the same tracks and same procedure as for setting the

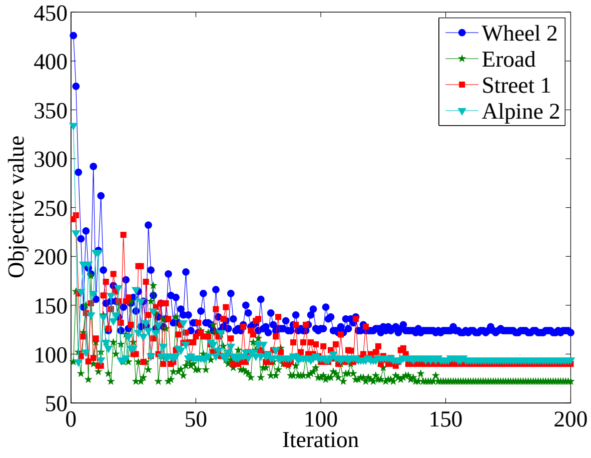

Fig. 8: CMA-ES was run for four tracks to set the independent parameters of Ahura.

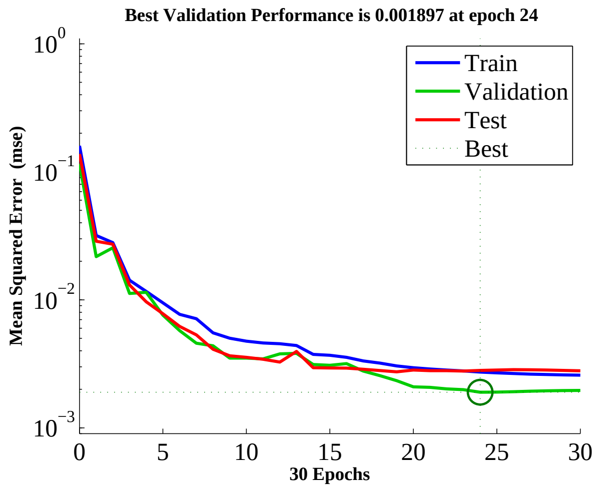

Fig. 9: the convergence curve of the neural network to estimate the friction of the track.

independent parameters. The final values for the dependent parameters were set to: $\omega_1 = 0.583$, $\omega_2 = 15.007$, $\omega_3 = 0.034$, $\omega_4 = 1.001$, $\omega_5 = 10.075$.

Also, the angles for the proximity sensors were set to $\{-90, -75, -50, -35, -20, -15, -10, -5, -1, 0, 1, 5, 10, 15, 20, 35, 50, 75, 90\}$ through trials.

E. Dynamic adjustments and stuck manager

The calculated parameters for the speed and steer controllers need to be revised based on the specifications of the track. The reason is that the parameters have been set for a limited number of tracks while new tracks might have different specifications. The main characteristics of tracks that may affect the best choice for parameters are friction, width, and bumps.

Also, Ahura uses the strategy described in [6] to handle stuck situation. In fact, if Ahura recognized that it is out of the track, it reduces the maximum value of the acceleration pedal to prevent too much traction. It finds the correct direction first and tries to get back to the track. If it detects that the car is not moving, the gear is changed to -1 (rear gear) and the steer is adjusted accordingly to repair the direction of movement.

1) Adjustment based on friction: We propose a simple strategy based on a supervised neural network to estimate friction of the track during the run. Clearly the difference between the speed of the spin of wheels and the speed of the vehicle ($d = |xSpeed - \sum_{i=1}^{4} r_i v_i|$) is larger if the friction of the surface is low when the power on the wheels is positive (rpm is non-zero). However, this difference is also affected by other parameters, e.g., lower gears usually cause larger d, larger rpm usually causes higher d, downhills usually cause a smaller d, small acceleration pedal usually causes smaller d, etc. One can assume that friction is a function of d, rpm, gear, and slope of the current segment of the track. As the slope is not given by the sensors, the value of $zSpeed$ was rather used which is directly related to the slope of the track. In order to increase the precision of the friction calculation based on these four parameters, we limited the friction estimation process by the following conditions:

- gear is either 2 or 3,
- rpm is in [7000, 8000],
- the vehicle is not taking a turn ($|steer\ angle|$ is small, set to 0.05 in our implementations).

Note that, the angle of the steer also affects the value of d. Thus, the value of friction is estimated only when the steer angle is small. These settings reduce the effects of other potential parameters on the calculation of the friction.

A multi layer perceptron [24] was used to find the mapping from $< d, gear, rpm, zSpeed >$ to the friction value. In order to train the network, we run Ahura on different tracks with different frictions (including dirt tracks and snowy tracks) and extracted some sample points (i.e., $< d, gear, rpm, zSpeed >$) under the mentioned conditions. For each track, the friction was also extracted (the friction is available in the specification files for each segment of the track) that was used as the output of the mapping for the extracted samples. From this sample pool, we selected 650 samples in a way that the they included different combinations of $< d, gear, rpm, zSpeed >$ for different values of frictions. We used 70% of these samples for training, 15% for test, and 15% for validation (selected randomly). The number of neurons in the hidden layer of the perceptron was set experimentally to 10 and the levenberg-marquardt [25] method was used for the training purposes (Fig. 9 shows the convergence curve).

During the run, Ahura samples the difference between the spin speed of the wheels and the speed of the vehicle (i.e., d) whenever the mentioned conditions are met (gear is 2 or 3, rpm is in [7000, 8000], and steer angle is smaller than 0.05). Also, the values for $gear$, rpm, $zSpeed$ are available directly from sensors. The sampled values are fed into the trained neural network and the estimated friction is calculated. The calculated friction then is used to revise the value of λ_2 in the following formula: $\lambda_2 = \tau/\mu^2$ where τ is a constant for a track that its friction (μ) is 1.0 and μ is the calculated friction. τ was set to 6.9 as this value for λ_2 is appropriate for a track with the friction 1.0 (Alpine-2 has the friction equal to 1.0 and the best value of λ_2 for that track is 6.9).

2) Adjustment according to width and bumps: The value of λ_2 is revised according to the width of the track in a way that λ_2 decreases when width is larger and vice versa. The reason is that at a wide track the vehicle has more space at both sides and, hence, it can move faster at the turns while still prevent moving out of the track.

In addition, if a vehicle is jumping then it needs to be controlled in a special way. As an example, when it gets back on the ground and the steer angle is large then it losses its control. Thus, we implemented a simple jump handling strategy for Ahura. As the distance of the vehicle from the ground is available by a sensor, called z, if the vehicle is jumping ($z > 0.65$ set experimentally) then the steer is set to zero until the vehicle gets back on the ground ($z < 0.35$ set experimentally). This prevents the vehicle from being uncontrollable after landing on the ground.

3) Adjustment for unknown situations: A human driver relies on experience in addition to in-line collected information. There might be many different reasons behind it such as the lack of experience on that situation, the lack of relevant information to handle that situation, or delay of gathering in-line information (because of the physical limitations of sensors) to handle that situation. Ahura uses the same strategy as human to memorize and handle unknown situations that have difficulties to handle. During the run, the location of unsuccessful actions (if the vehicle goes out of the track or seriously damaged) are stored in a list. In the next lap, the target speed of the vehicle is multiplied by $max(0.9^{severity}, 0.7)$ before (set experimentally to 250 meters) it arrives to that location. Severity of the unsuccessful action is calculated based on the number of times that the action has been unsuccessful. Every unsuccessful action results in multiplication of the target speed by 0.9, however, this coefficient does not reduce to values less than 0.7.

IV. Experimental results

In this section the performance of Ahura is compared with other controllers (Cobostar, EVOR, Autopia, and MrRacer) when they are run at different tracks. These controllers were used for comparison purposes because they have shown good performance in the literature [8], [7], [6], [9] and each of them has ranked first in several car racing competitions (Car racing competitions in IEEE CEC 2008-2010, ACM GECCO 2009-2015, and IEEE CIG 2009-2012).

All controllers were tested under Windows 7 environment, at 40 different tracks[8] with different specifications. All controllers were run on a warm-up session for 100,000 ticks of the game (almost 36 minutes) on each track before the actual races. During the warm-up, the controllers evaluate the track and tune their parameters if necessary (Ahura and Cobostar do not need the warm up session as they make decisions during the run). The results of testing the controllers under different runs (individual run, race, etc) are reported and compared in the next subsections. In order to discuss the results, we introduce five parameters for the tracks and investigate their impacts on the performance of the controllers using a decision tree [26]. This analysis helps us to determine advantages/disadvantages of different controllers under different specifications of the tracks.

A. Tracks specifications

Friction, rolling resistance, roughness, width, and complexity of tracks are used to study the performance of the controllers. The values for friction, roughness, rolling resistance, and width were extracted from the specification files of each track [9]. We use a simple method to calculate the complexity of the tracks as follows. We run an instance of Ahura with 10 auxiliary sensors (constant during the run), $\alpha = 1$ (remains in the middle of the track during the run), and $targetSpeed = 50$. This setting causes the vehicle to stay in the middle of the track and move slowly. The vehicle records the value of $dist_0$ every 1 second for each track under this setting. We expect that the numerical difference ($\frac{\Delta dist_0}{\Delta t} = x_i - x_{i-1}$) of this recorded data fluctuates more for a complex track with many turns comparing to a track with few turns. The reason is that the distance in front starts dropping as the vehicle approaches a turn while it starts growing when the turn is finished. As the first order difference of a signal represents changes of that signal, we expect that the numerical difference calculated by $\frac{\Delta dist_0}{\Delta t} = x_i - x_{i-1}$ also estimates such changes.

Fig. 11(a) shows the recorded data for two tracks called Aalborg (a complex road track, shown in Fig. 10(a)) and Michigan (a simple oval track, shown in Fig. 10(b)). It is obvious that the fluctuation of the recorded data ($dist_0$ vs time) for the Aalborg track is much higher than that of Michigan track. Hence, if we calculate the numerical difference ($\frac{\Delta dist_0}{\Delta t} = x_i - x_{i-1}$) then the resulted signal for more complex tracks should be fluctuating more (see Fig. 11(b)). Hence, the variance of this numeric difference can measure the complexity of the tracks. As an example, the complexity of the track Michigan is measured as 42.16 by using this method while the complexity of Aalborg is measured as 524.69.

[8]Tracks, driver codes, detailed results, and videos are available online at https://sites.google.com/site/mohammadrezabonyadi/standarddatabases/simulated-car-racing.

[9]Some tracks include different segments with different mechanical specifications such as friction and rolling resistance. For those tracks, the average of the parameter over all segments was used.

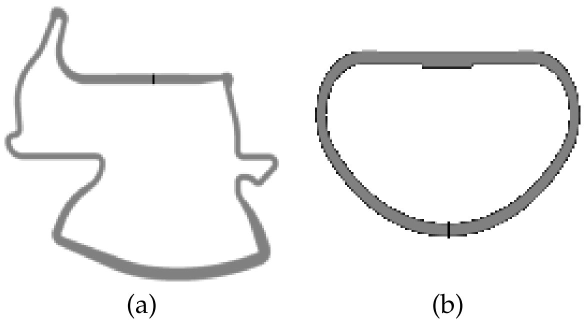

Fig. 10: Two sample tracks from TORCS, (a) Aalborg (a complex track), and (b) Michigan (a simple track).

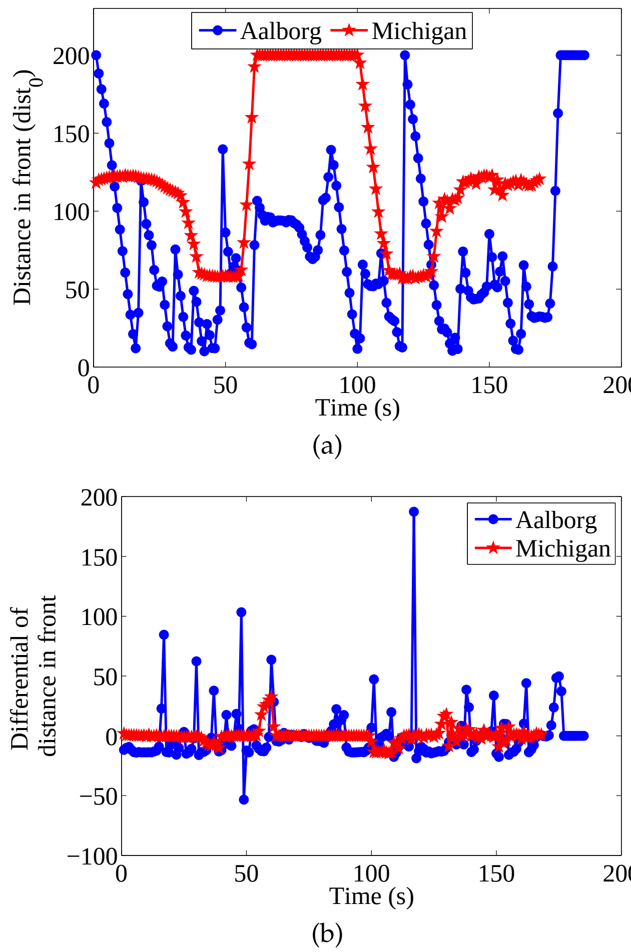

Fig. 11: Complexity of two example tracks measured by Ahura, Michigan: 42.16 and Aalborg: 524.69.

The friction of tracks was in [0.85, 1.4], the rolling resistance was in [0.001, 0.03], the roughness was in [0, 0.5], the width was in [10, 30], and complexity in [12, 735].

B. Qualification results

After a warm-up session, each controller was run on each track for 25 runs where each run consisted of 5 laps. Table II shows the results. In the qualification phase (row: Qualification), we averaged the total time each controller needed to finish 5 laps over all 25 runs on each track (in total, 40 numbers for each controller, each number represents the average of time the controller needs to finish 5 laps on a track). These average times were used to determine the rank (Rank (First) column in the table) and assign scores (F1 score)[10] to the controllers.

In order to calculate the average rank (Rank (Avr) in the table), we considered all runs on all tracks for each controller (40x25 values for each controller, each value represents the rank of the controller in 5 laps on one of the tracks) and averaged those values. The Wilcoxon test was used to determine if the difference between the Ahura's results (difference of the median of these values) is significant ($p < 0.05$) comparing to other controllers' results. Asterisked values in the table in front of a controller (row) for a specific performance measure (column) indicate that the result of Ahura was significantly different from the result of that controller in that specified performance measure. The Rank (Avr) performance measure in the qualification run shows that Ahura could achieve significantly better rank in comparison to the tested controllers.

The time per lap column in table II for qualification has been calculated by dividing the total time to finish all runs on all tracks by 40x25x5 laps (average time per lap). The table indicates that Ahura's time per lap on all tracks is significantly (Wilcoxon test) better than other controllers. The reported damage in the table is the average over all runs. Again, based on the Wilcoxon test, the damage that Ahura has applied to the car is significantly lower than the damage other controllers applied to the car.

The classification of the results in the qualification stage using a decision tree (Fig. 12) shows that friction (F) and complexity (C) of the tracks play a role in the achieved results. According to the decision tree, Evor performs better than other bots in simple oval tracks where $C < 31.48$ (this category included 5 tracks). In more complex tracks with small friction, where $F < 1.3$ and $C \geq 31.48$ (this branch included 34 tracks) Ahura shows a better performance. This category in fact refers to most usual road tracks (high complexity with moderate friction). For very high friction, Autopia has performed the best over all other bots. However, the

[10]The Formula 1, F1, scoring system assigns scores to the drivers in order with following numbers: 25, 18, 15, 12, 10, 8, 6, 4, 2, 1, 0.

TABLE II: Comparison results for different algorithms for qualification, race, and race* stages. The best results have been shown by bold face. All results are averages over all 5 trials.

	Alg. name	F1 score	Rank (First)	Rank (Avr)	$\frac{Time}{lap}$	Damage
Qualification	Ahura	**820**	**23**	**1.69**	**80.41**	**407.12**
	Cobostar	675	6	2.73*	92.38*	893.12*
	Autopia	578	3	3.37*	89.6*	1620.34*
	EVOR	644	8	3.05*	95.13*	1121.57*
	MrRacer	478	0	4.42*	103.01*	1892.03*
Race	Ahura	**877**	**26**	**1.53**	**82.52**	**1733.3**
	Cobostar	670	7	2.51*	91.23*	3332.8*
	Autopia	513	1	3.825*	94.99*	3114.2*
	EVOR	660	6	2.65*	88.97*	4147.2*
	MrRacer	485	0	4.1*	100.77*	5913.21*
Race*	Ahura	**782**	**16**	**2.025**	**84.18**	3010.2
	Cobostar	673	8	2.495	92.11*	4572.55*
	Autopia	520	0	3.75*	98.93*	**2409.1**
	EVOR	703	13	2.52*	92.19*	5926.32*
	MrRacer	525	3	3.85*	110.33*	6574.3*

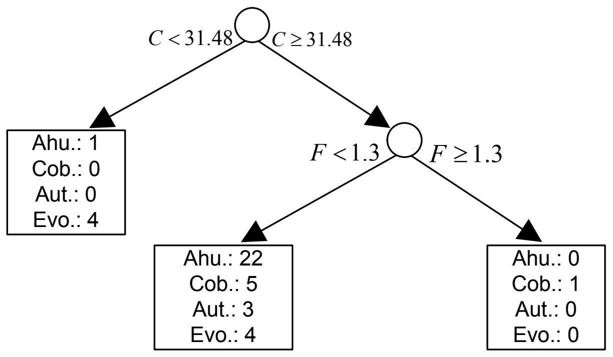

Fig. 12: Decision tree for classification of controllers on different tracks in the qualification stage. C represents Complexity and F represents Friction. The values in the graph have been calculated by the decision tree. The values are the number of times each controller won within that group of tracks.

latter results are not reliable as the number of tracks with these specifications was only 1.

The better performance of Ahura on low to mid friction tracks is the result of adjusting the driving style according to the estimated friction. Also, the rather complex steer controller of Ahura has led the controller to drive effectively at complex tracks. However, Evor has shown a better performance at oval tracks. Our results showed that the average (standard deviation) of standing for Evor for this group of tracks (included 5 tracks) was 1.2 (0.45) while it was 2.2 (1.1) for Ahura (Ahura was in the second place after Evor in this group of tracks). This difference comes from the steer controlling style for Evor. Evor usually prefers to take the smallest curve to take a turn that is more effective in oval tracks (small complexity and wide) than Ahura's strategy (opposite sliding).

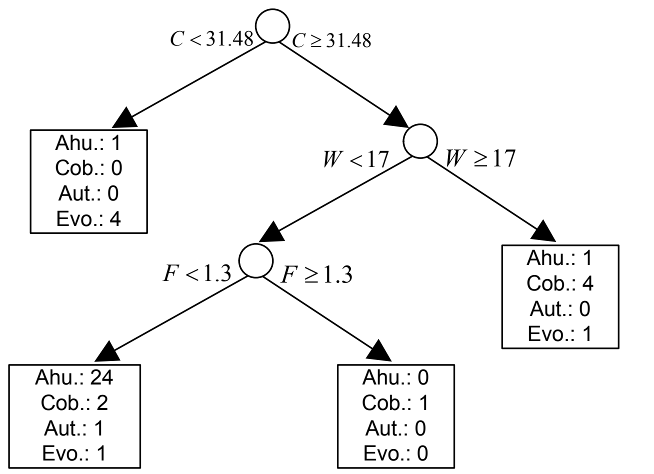

Fig. 13: Decision tree for classification of controllers on different tracks in the race stage. W is the width of the track. The results are the average number of standing first over 5 runs.

C. Race results

The bots were placed on the tracks to compete with each other based on the average time to complete those tracks recorded in the qualification run (25 runs for each track, 5 laps each). Results in table II (row: Race) show that Ahura is the best controller in terms of the F1 score, the average number of times that won the races, the average of rank over all runs (Wilcoxon test), the average of the time per lap (Wilcoxon test), and average of the damage (Wilcoxon test) among all controllers.

The decision tree based on the tracks specifications in the race stage (Fig. 13) shows that EVOR is the best controller when the complexity of the track is smaller than 31.48 (simple oval tracks). At higher complexity tracks, if the track width is $W \geq 17$ (very wide tracks) then Cobostar is the best choice among other controllers. However, if the track width is tighter than 17 meters then Ahura is usually the winner. Although Cobostar shows a better performance for very high friction tracks, the data points in that category is very small that makes the results unreliable. This result is aligned with what was discussed for qualification test. Ahura was in the second place (after Evor) when $C < 31.48$ and in the third place (after Cobostar and Evor) when $C >= 31.48$ and $W >= 17$.

D. Overtaking test

This section compares overtaking strategies of controllers. The results of two tests are reported here. The first test focuses on the overtaking ability of Ahura against other tested bots in a competition where Ahura was placed the last at all tracks. The second test is the result of competing different controllers against Blocker (see section III-D).

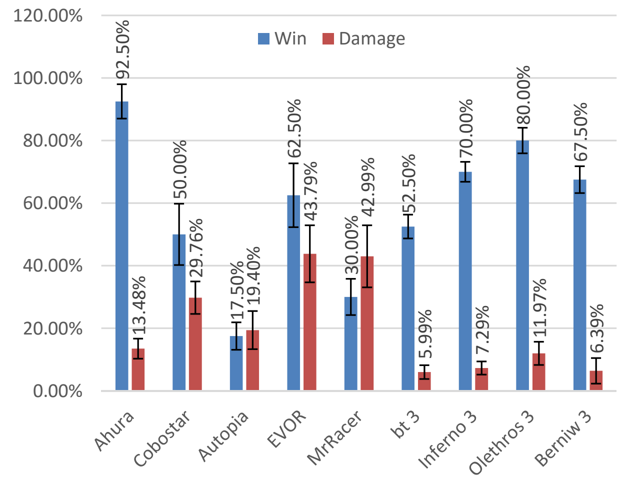

Fig. 14: Results for overtaking the Blocker.

1) Against other bots (Race):* In the Race* stage, all bots were placed on the track based on their achieved results in the qualification stage except for Ahura that was always placed at the last position. The bots compete with each other on 40 tracks, 25 repetition of each track, 5 laps each. Results (table II, row: Race*) indicate that Ahura is still the best controller in terms of the F1 score, Rank (First), and time per lap (Wilcoxon test). However, at this stage Ahura has received the second place in terms of damage (Autopia outperforms Ahura in terms of damage). This was expected as the bot that is placed in the last position at the beginning of the race needs to overtake 4 other bots that usually involves more damage. Ahura received significantly less damage (Wilcoxon test) than Cobostar, EVOR, and MrRacer. Also, in terms of the Rank (Avr), Ahura was significantly better (Wilcoxon test) than Autopia, EVOR, and MrRacer while was statistically similar to Cobostar (although better in average).

2) Against Blocker: All controllers (Ahura, Cobostar, Autopia, EVOR, and MrRacer) plus four other controllers that are available with TORCS (called bt 3, Inferno 3, Olethros 3, and Berniw 3) were run against Blocker (see section III-D) on all tracks, 5 runs for each track and each run consisted of 5 laps. As Blocker is significantly slower than its opponents, if a vehicle is unsuccessful in overtaking Blocker then one can conclude that its overtaking strategy is not effective.

Results (Fig. 14) show that Ahura's overtaking strategy is more effective than other controllers against Blocker (successful in 92% of tracks with 13.48 percent of damage). Although Ahura damaged slightly more than bt 3, Inferno 3, Olethros 3, and Berniw 3, the percentage of successful overtakes for Ahura is higher than that of those controllers. Comparing to Cobostar, EVOR, and MrRacer, despite the aggressive overtaking policy taken by these controllers (evident by gaining more damage during the overtaking process), none of these controllers could overtake Blocker as effective as (lower damage and higher success percentage) Ahura.

V. Conclusions and future work

In this paper, a high performing controller called Ahura for TORCS was presented. The parameters for Ahura were set using a CMA-ES to drive a particular car and it was run against other state-of-the-art controllers. Results showed that Ahura performs better than other controllers in many instances in terms of driving and overtaking. Also, results from the decision tree on the qualification stage showed that Ahura is effective on tracks with wide range of frictions (from 0.85 to 1.3). The reason is that Ahura adjusts its driving style according to the friction of the road. The test on very large friction tracks included only on track that does not really indicate that Ahura is not effective on high friction tracks. However, it seems that Ahura can be improved to drive on tracks with low complexity. This can be done through a procedure that recognizes the low complexity tracks and adjusts the driving style accordingly. Further, results from decision tree on the race stage indicated that Ahura outperforms other controllers at most tracks with low/mid range friction and high complexity that refer to usual road tracks. This again suggests that Ahura is effective on road tracks but needs improvement on oval tracks. Other tests showed that Ahura is very effective at overtaking against other bots and Blocker (a bot that blocks the vehicle behind). One should note that Ahura is able to drive any other bots in TORCS, i.e., Ahura is vehicle independent. However, the parameters of the controller have been optimized only for one particular type of bot in this paper.

There are several different areas that can be further enhanced in Ahura. One potential enhancement is to enable the controller to generate a map of the track in the warm-up session and use this information to make better decisions. As an example, this map can be useful to decide the best tentative position before arriving to turns and make better decisions for the preparation phase for taking a turn. Another potential enhancement is to enable the controller to handle noisy sensors. One can use the strategies described in [27] to enable Ahura to handle noisy environments. Also, measuring specifications of the tracks and adjusting the driving style (e.g., friction estimation in Ahura) can significantly improve the controller performance. Hence, recognizing key features that distinguish tracks and designing methods to estimate them during the run is valuable as these information can be useful by controllers to adjust their driving style. Finally, our experiments in section III-D showed that optimizing Ahuras parameters for a specific track might result in significant performance impairment on other tracks. One way to address this issue would be to perform a single optimization process that evaluates the performance of Ahura on a set of tracks (rather than one track) at the same time to find a parameters set that perform well on that set of tracks. Another way to address this issue would be to find optimum parameters sets for Ahura for a set of tracks (one parameters set for each track) and design a mechanism that identifies which of these parameters set is a better fit when the controller drives on a new track. This can be done by analyzing the performance of the controller during the warm-up stage when it uses different parameters set or by identifying how similar is the current track to other previously evaluated tracks and use the best parameters set accordingly.

VI. Acknowledgment

This work was partially funded by the ARC Discovery Grant DP130104395 and by grant N N519 5788038 from the Polish Ministry of Science and Higher Education (MNiSW).

References

[1] J. Markoff, "Google cars drive themselves, in traffic," *The New York Times*, vol. 10, p. A1, 2010.

[2] B. Wymann, E. Espie, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "TORCS, the open racing car simulator," *Software available at http://torcs.sourceforge.net*, 2000.

[3] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lonneker, L. Cardamone, D. Perez, and Y. Sez, "The 2009 simulated car racing championship," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 131–147, 2010.

[4] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship: Competition software manual," *arXiv preprint arXiv:1304.1672*, 2013.

[5] N. Hansen, S. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with co-variance matrix adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[6] E. Onieva, D. A. Pelta, J. Godoy, V. Milans, and J. Prez, "An evolutionary tuned driving system for virtual car racing games: The AUTOPIA driver," *International Journal of Intelligent Systems*, vol. 27, no. 3, pp. 217–241, 2012.

[7] J. Quadflieg, M. Preuss, and G. Rudolph, *Driving faster than a human player*. Springer, 2011, pp. 143–152.

[8] M. V. Butz and T. Lonneker, "Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge," in *Computational Intelligence and Games*. IEEE, 2009, pp. 317–324.

[9] S. Nallaperuma, F. Neumann, M. R. Bonyadi, and Z. Michalewicz, "EVOR: an online evolutionary algorithm for car racing games," in *Genetic and evolutionary computation*. ACM, pp. 317–324.

[10] E. Onieva, J. Godoy, J. Villagra, V. Milanés, and J. Pérez, "On-line learning of a fuzzy controller for a precise vehicle cruise control system," *Expert Syst. Appl.*, vol. 40, no. 4, pp. 1046–1053, 2013.

[11] T. S. Kim, J. C. Na, and K. J. Kim, "Optimization of an autonomous car controller using a self-adaptive evolutionary strategy," *Int J Adv Robot Syst*, vol. 73, no. 9, 2012.

[12] J. Munoz, G. Gutierrez, and A. Sanchis, "A human-like TORCS controller for the simulated car racing championship," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, 2010, pp. 473–480.

[13] K.-J. Kim, J.-H. Seo, J.-G. Park, and J. C. Na, "Generalization of TORCS car racing controllers with artificial neural networks and linear regression analysis," *Neurocomp*, vol. 88, pp. 87–99, 2012.

[14] D. Galanopoulos, C. Athanasiadis, and A. Tefas, "Evolutionary optimization of a neural network controller for car racing simulation." in *SETN*, ser. Lecture Notes in Computer Science, I. Maglogiannis, V. P. Plagianakos, and I. P. Vlahavas, Eds., vol. 7297. Springer, 2012, pp. 149–156.

[15] L. Cardamone, P. L. Lanzi, D. Loiacono, and E. Onieva, "Advanced overtaking behaviors for blocking opponents in racing games using a fuzzy architecture," *Expert Systems with Applications*, vol. 40, no. 16, pp. 6447–6458, 2013.

[16] D. Perez, G. Recio, Y. Saez, and P. Isasi, "Evolving a fuzzy controller for a car racing competition," in *Proceedings of the 5th International Conference on Computational Intelligence and Games*, ser. CIG'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 263–270.

[17] A. Agapitos, J. Togelius, and S. M. Lucas, "Evolving controllers for simulated car racing using object oriented genetic programming," in *Genetic and evolutionary computation*. ACM, 2007, pp. 1543–1550.

[18] J. Togelius, S. M. Lucas, H. D. Thang, J. M. Garibaldi, T. Nakashima, C. H. Tan, I. Elhanany, S. Berant, P. Hingston, R. M. MacCallum, T. Haferlach, A. Gowrisankar, and P. Burrow, "The 2007 IEEE CEC simulated car racing competition," *Genetic Prog. and Evolvable Machines*, vol. 9, no. 4, pp. 295–329, 2008.

[19] B. Beckman and N. B. R. Club, "The physics of racing, part 5: Introduction to the racing line," *online] http://www. esbconsult. com. au/ogden/locust/phors/phors05. htm*, 1991.

[20] D. Galanopoulos, C. Athanasiadis, and A. Tefas, *Evolutionary optimization of a neural network controller for car racing simulation*. Springer, 2012, pp. 149–156.

[21] H. Pacejka, *Tire and vehicle dynamics*. Elsevier, 2005.

[22] D. Sheen, "Introduction to numerical analysis," 1980.

[23] N. Hansen, R. Ros, N. Mauny, M. Schoenauer, and A. Auger, "Impacts of invariance in search: When CMA-ES and pso face ill-conditioned and non-separable problems," *Applied Soft Computing*, vol. 11, no. 8, pp. 5755–5769, 2011.

[24] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," DTIC Document, Tech. Rep., 1961.

[25] C. T. Kelley, *Iterative methods for optimization*. Siam, 1999, vol. 18.

[26] J. R. Quinlan, "Learning decision tree classifiers," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 71–72, 1996.

[27] M. Preuss, J. Quadflieg, and G. Rudolph, "Torcs sensor noise removal and multi-objective track selection for driving style adaptation," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2011, pp. 337–344.