INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

Optimization of an Autonomous Car Controller using a Self-Adaptive Evolutionary Strategy

Regular Paper

Tae Seong Kim, Joong Chae Na* and Kyung Joong Kim

Dept. of Computer Science & Engineering, Sejong University, Korea
* Corresponding author E-mail: jcna@sejong.ac.kr

Received 02 May 2012; Accepted 20 Jun 2012

DOI: 10.5772/50848

Abstract Autonomous cars control the steering wheel, acceleration and the brake pedal, the gears and the clutch using sensory information from multiple sources. Like a human driver, it understands the current situation on the roads from the live streaming of sensory values. The decision-making module often suffers from the limited range of sensors and complexity due to the large number of sensors and actuators. Because it is tedious and difficult to design the controller manually from trial-and-error, it is desirable to use intelligent optimization algorithms. In this work, we propose optimizing the parameters of an autonomous car controller using self-adaptive evolutionary strategies (SAESs) which co-evolve solutions and mutation steps for each parameter. We also describe how the most generalized parameter set can be retrieved from the process of optimization. Open-source car racing simulation software (TORCS) is used to test the goodness of the proposed methods on 6 different tracks. Experimental results show that the SAES is competitive with the manual design of authors and a simple ES.

Keywords Autonomous Car, Evolutionary Strategy, Self-Adaptation

1. Introduction

TORCS (The Open Racing Car Simulator) [1] is an open-source 3D car racing game which has a GNU license and has been used as a test bed for many AI (Artificial Intelligence) algorithms. Because of its realistic 3D simulation, it has been used for research on intelligent vehicles, robotics and games. For example, the simulator has been used in laboratory-level experiments with human drivers monitoring their brain/body signals [2][3], behavioural patterns [4] and 3D postures [5]. The simulator has been used for intelligent transportation research: the coordination of intelligent vehicles [6], simulated traffic scenarios [7] and digital human driver modelling [8].

Since 2008, international game AI competitions have changed their basic platform from 2D car racing to 3D car simulation, i.e., TORCS [9]. In competitions, a TORCS server provides each controller with 79 sensory values of a simulated car (*position* of the car on a track, *angle* between the car and the track axis, 19 *track* sensors, *speed* sensors, and so on). In particular, the 19 *track* sensors are

used to measure the distances between the car and the 19 edges of the track. By default, the sensors sample the space in front of the car for every 10° angle, spanning clockwise from -π/2 up to +π/2 with respect to the car axis. Each controller receives this sensory information from the server and sends the values of five actuators (steering wheel, accelerator, brake, gear and clutch). When the TORCS server receives the decision of each controller, the next time step is simulated for the car racing environments. This interactive process is called a 'game tick' in which the server (TORCS) and the client (controller) repeatedly exchange a UDP packet at every game tick until the race is completed. In this way, TORCS is able to offer realistic 3D car racing simulation for multiple players.

However, developing a sophisticated controller is not a trivial work. The *track* sensor values in TORCS a have limited distance (200m) and, thus, it is impossible to get the whole track data structure at once. Moreover, a client program is not supposed to do highly complex computation because the controller should respond to the server within the current game tick. On the basis of these difficulties, developing an excellent racing controller has been considered a challenging task and has received much attention lately.

In recent years, many kinds of computational intelligence methods have been used for developing a sophisticated controller. Onieva et al., the winners of the championships in both 2009 and 2010, proposed a parameterized modular architecture with a fuzzy system and a genetic algorithm [10][11]. Butz et al. developed their own controller named COBOSTAR and won the 2009 competition held by CEC and CIG [12]. Hoorn et al. proposed an imitative learning controller, adapting a multi-objective evolutionary algorithm [13]. Munoz et al. obtained data (trajectory and speed) from humans and then trained neural networks [14]. Cardamone et al. suggested a high-level action prediction model with high-level information about the track [15]. Seo et al. developed their own heuristic controller, which drives as safely as possible [16][17]. They used an ANN (Artificial Neural Network) and a linear regression to predict the desired speed in different situations. We call their controller the "BFB3", the name of the controller used at the CIG 2011 competition (5th rank) [16]. Quadflieg et al. developed a human-readable track recognition system which enables 'looking into the next curve', using a combination of advanced pre-processing steps and a simple track-type classifier [18].

In this paper, we propose optimizing the parameters of an autonomous car controller using ESs (Evolutionary Strategies) and describe how the most generalized parameter set can be retrieved from the process of optimization. For this, we propose a new controller of TORCS whose detailed behaviour is governed by a number of parameters. Our contributions in detail are as follows.

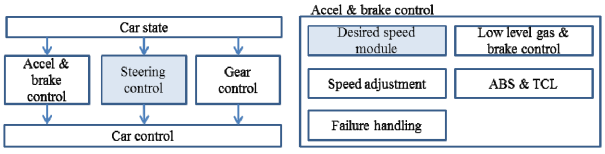
- We propose a new controller for TORCS by combining BFB3 [17] and Quadflieg et al.'s approach [18] to provide fast and accurate judgments of circumstantial situations by pre-processing track sensor values from a TORCS server. To accelerate in an accurate and aggressive manner, we modify the acceleration and brake control part and the steering control part of the BFB3 by introducing 10 new parameters and using the curvature information of a track segment, as in [18].
- We propose optimizing the parameters of our controller using a simple ES and a self-adaptive ES. To find the optimized values of the parameters and use 10+10 ESs. We obtained the track-dependent optimal parameter sets separately for 6 tracks (Alpine1, ASpeedway, Cgtrack2, Eroad, Wheel1 and Wheel2), i.e., 12 optimal parameter sets in total. Experimental results show that our controller significantly improves the performance of the BFB3 for the 6 tracks. Moreover, the self-adaptive ES is better than the simple ES on most tracks.
- We derive the most generalized parameter set from the process of optimization to develop a practical track-independent controller. We use intermediate generations as well as the latest generations produced during the process of finding the track-dependent optimal parameter sets. Among the parameter sets of these generations, we choose the one which minimizes the sum of the differences of lap times from track-dependent controllers for the 6 tracks. We experimentally compare the generalized controller and the BFB3 on 12 tracks.

This paper is organized as follows. Section 2 introduces the BFB3 that our controller is based on and describes the weaknesses of the BFB3. In Section 3, we present our modifications to improve performance of it. Specifically, we introduce new additional parameters and rules to enhance the controller and describe our learning approaches (ES) as applied to optimize our new parameters. In addition, we present how to derive the most general parameter set from the solutions found. Section 4 shows our experimental results, where we give comparative studies analysing two different learning approaches and compare the performances of the BFB3, the track-dependent controller and the track-independent one. In Section 5, we offer our conclusions.

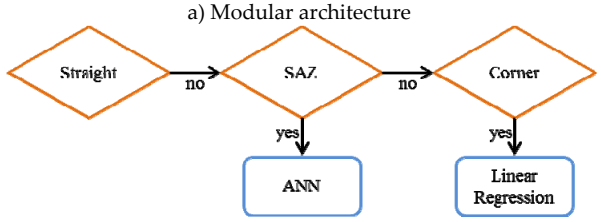
2. Overview of the BFB3

In this section, we give an overview of the BFB3, which our proposed controller is based on. The BFB3 follows the philosophy of "modular architecture" proposed by

Onieva et al. [10]. Figure 1 a) shows the modules making up the controller. Each module controls the acceleration and the brake, the steering, gears, etc., of the simulated car. The most important and complex module is the acceleration and brake control module composed of several sub-modules, such as the desired speed module, the ABS and the TCL module. We present a brief summary of the desired speed module and the steering control module, which are modified in our new controller. We keep the other components intact in our new controller.



a) Modular architecture



b) Desired speed module

Figure 1. Basic modular components of BFB3 (SAZ=Speed Adaptation Zone).

2.1 The desired speed module

The desired speed module determines the *Desired_Speed*, which the simulated car tries to reach. This module mainly uses the values of 19 *track* sensors, denoted by *track*[0], *track*[1], …, *track*[18]. Unlike the default setting of the TORCS controller, the BFB3 has the customized angles of track sensors in order to obtain more information of the front side of the car. Thus, there are a lot of track sensors on front side of the car, such that the customized angles are -90°, -1°, -45°, -30°, -25°, -20°, -15°, -10°, -5°, 0°, 5°, 10°, 15°, 20°, 25°, 30°, 45°, 1° and 90°.

The BFB3 uses different strategies according to the current situation of the car, which is categorized into three groups by the value of the centre *track* sensor (*CT*, the 9th *track* sensor). Figure 1 b) shows a rough flow chart of the desired speed module. The *Desired_Speed* determined in the desired speed module is used to adjust the acceleration and brake signal in the speed adjustment module. The acceleration and brake signal, within the range of -1 (full brake) ~ 1 (full acceleration), is computed by the difference between the current speed (*speedX*) and the desired speed (*Desired_Speed*), as follows:

$$Accel\ \&\ Brake\ signal = \frac{2}{(1+\exp(speedX - Desired\_Speed))} - 1 .$$

If CT ≥ 150m, the car is considered to be on a "straight" track segment (i.e., there is no obstacle in front of the car and it is less at risk of being stuck) and thus the *Desired_Speed* is set to the pre-defined *MAX_SPEED* value, as follows:

$$Desired\_Speed = MAX\_SPEED.$$

If 70m < CT < 150m, the car is considered to be in a *speed adaptation zone* (SAZ), which is a zone for reducing the speed of the car before a corner, whereby the desired speed is computed as follows:

$$Desired\_Speed = \frac{(\alpha - prob) \times (MAX\_SPEED - speedX) \times |T_{\max} - CT|}{|T_{\min} - CT|},$$

where α and *prob* are parameters, $T_{\max}$ = max(*track*[1], *track*[17]), $T_{\min}$ = min(*track*[1], *track*[17]) and *speedX* is the current speed.

Seo et al. tried to retrieve the generalized parameter α because the behaviour of the BFB3 relies upon the parameter α [16][17]. They chose α from {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0} and exhaustively searched in order to find the value of α with the minimum sum of the lap time as the generalized parameter. In addition, for more precise speed, the BFB3 predicted the chances of getting stuck (*prob*) using an ANN trained by standard back-propagation algorithms. This neural network computes a constant real number between 0 and 1 using four sensors (*track*[1], *track*[9], *track*[17] and *speedX*). If the car is less likely to be stuck, the value of *prob* will be near to 0 and the car will accelerate in proportion to α. Otherwise, it will be near to 1 and the car will accelerate less because *prob* is subtracted from α. In this way, the BFB3 can orient the desired speed with the current situation.

If CT ≤ 70m, the car is considered to be at a "corner" track segment and a linear regression analysis is used to determine the *Desired_Speed* given the environmental sensors. In this analysis, 19 *track* sensors (*track*[0]~*track*[18]) and 3 *speed* sensors (*speedX*, *speedY* and *speedZ*) are used and their weighted sum is the output of the prediction. They use a least median squared linear regression and feature selection algorithms to determine the value of each feature and the number of features involved in it. The best linear equation found in this linear regression analysis uses only 6 sensors (*track*[1], *track*[8], *track*[9], *track*[10], *track*[17], *speedX*), as follows:

$$\begin{aligned} Desired\_Speed = {} & 0.0639 \times track[1] + 1.2619 \times track[8] - 0.2008 \times track[9] \\ & + 1.1424 \times track[10] + 0.1808 \times track[17] + 0.3701 \times speedX \\ & - 7.9175 \end{aligned}.$$

2.2 The steering control module

The basic strategy of the BFB3 for the steering control is to have the car travel the longest free distance. The steering control module divides the current situation of the car into two cases.

If the car is in a straight track segment or a SAZ (i.e., CT > 70m), the controller steers the car towards the direction of the track axis. That is, if the car is leaning toward the right or left side of the track, the controller forces the car to move towards the centre of the track. In this case, the target steering value is as follows:

$$T_{steering} = \frac{0.5 \times (angle - trackPos)}{\pi / 4},$$

where "*angle*" is a degree between the car direction and the track axis, and "*trackPos*" is the distance between the car and the track axis.

If the car is in a corner track segment (i.e., CT ≤ 70m), the BFB3 steers the car towards the direction of the *track* sensor with the greatest value, and the target steering value is as follows:

$$T_{steering} = w[index] - 0.3 \times trackPos,$$

where *index* is the index of *track* sensor with the greatest value except for the three sensors (the centre, the rightmost and the leftmost *track* sensors) and w is a vector of weight values for the *track* sensors tuned manually.

2.3 The weaknesses of the BFB3

Despite the modules mentioned above, we found through a series of observations that the BFB3 is inefficient for two reasons. First, the probability *prob* predicted by the neural network for a SAZ is excessively large and it makes the BFB3 run the track in an unnecessarily defensive manner. Figure 2 shows the variation of the *prob* value which is estimated at the track "Wheel1." The values were measured at every game tick during a single lap. As shown in the figure, the probability is almost always 0 or 1; values within the middle range rarely appear. It means that the neural network is not able to handle the current situation on a minute level. Furthermore, the BFB3 always decelerates too much for upcoming corners.

Secondly, the BFB3 forces the car to move towards the centre of the track if the value of CT is greater than 70m, i.e., the car is in a straight track segment and in a SAZ. However, this strategy does not coincide with a common cornering strategy witnessed in real world racing. Figure 3 shows a variation of trackPos estimated at the part of Wheel1 (573m ~ 688m). The y-axis represents the position of the car, 1 and -1 mean that the car is on either of the edges of the track, and 0 means that the car is in the centre of the track. From 670m, BFB3 considers the track segment as a corner and starts to steer the car towards the direction of the track sensor with the greatest value. As can be seen, the BFB3 maintains the value of trackPos near to 0 up to 670m and changes its trackPos suddenly when it enters into the corner. This strategy causes great danger because a sudden change of steering causes the car to skid. In the real-world racing, it is common knowledge that car drivers should prepare for an upcoming corner by moving the car in the same direction as the corner in advance (see Figure 3).

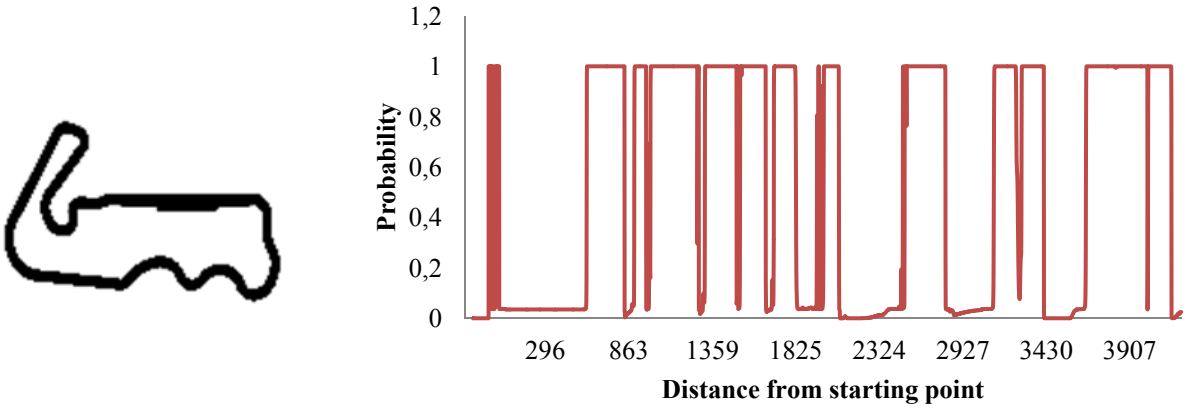


Figure 2. Variation of probability predicted by ANN during a single lap at Wheel1.

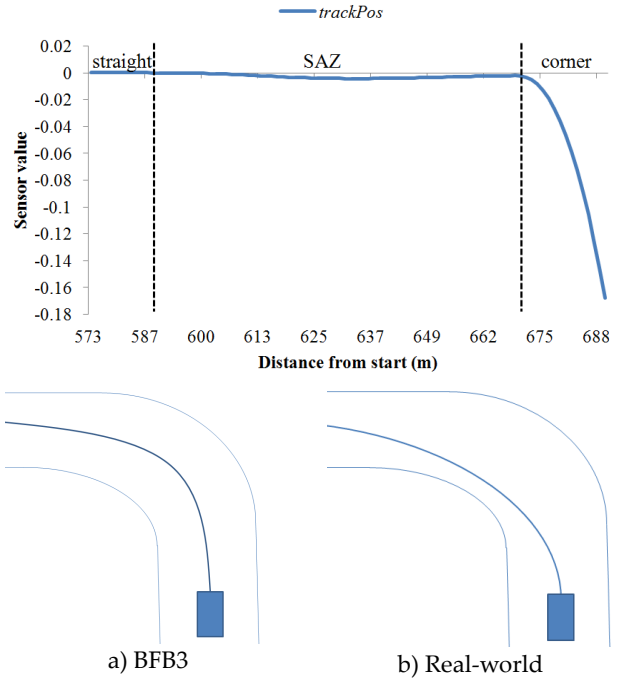


Figure 3. Variation of *trackPos* with the steering control module of the BFB3 and two different cornering strategies.

3. Proposed Methods for a Generalized Controller

In this section, we give a description of our controller based on curvature and learning. First, we describe the modules modified from the BFB3 and then present the learning approach to determine the parameters used in our modified controller.

3.1 The modified modules

To overcome the weaknesses of the BFB3, we modify the two modules - the desired speed module and the steering control module. We improve the desired speed module using the curvature information of a track segment. We first describe how to compute the curvature of the track segment and then present the desired speed module and the steering control module of our modified controller.

3.1.1 The curvature computation

Basically, our computation of the curvature is the same as the one used in [18]. The curvature is computed using the *track* sensors. Because the BFB3 customizes the angles of *track* sensors to obtain more information about the track in front of the car, we choose only 17 *track* sensors (excluding *track*[1] and *track*[17], whose angles are -1° and 1°, respectively) to obtain the curvature information.

First, we vectorize the 17 track sensors on the 2D polar coordinate. As they are already known, the angle and scalar (distance) values of each sensor, the i^{th} *track vector* $\vec{s}_i$, can be defined simply as:

$$\vec{s}_i = \begin{pmatrix} -\cos(\theta_i \times \frac{\pi}{180}) \times track[i] \\ \sin(\theta_i \times \frac{\pi}{180}) \times track[i] \end{pmatrix},$$

where θ_i is the angle of the i^{th} track sensor (e.g., $\theta_9 = 0°$).

After calculating these 17 *track* vectors, we derive the *outline vectors* from subtraction between two adjacent track vectors (Figure 4 a)) and compute the angle between two adjacent outline vectors $\vec{v}_1$ and $\vec{v}_2$ using the inner product of two vectors and the arccosine function, as follows:

$$\vec{v}_1 \angle \vec{v}_2 = \arccos\left(\frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1| \cdot |\vec{v}_2|}\right).$$

The curvature of the upcoming corner is represented by the sum of the angles between two adjacent outline vectors.

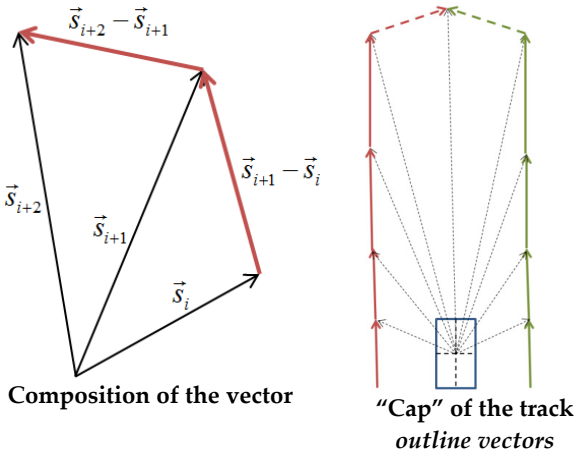


Figure 4. Descriptions of the computation for curvature information.

For more precise computation of the curvature, however, we have to carefully choose the *track* vectors used for the computation, as in [18]. Figure 4 b) illustrates the negative effect of some track vectors for the computation of the curvature. In this figure, it is clear that the dotted *outline vector* should be excluded for the computation of the curvature. To exclude vectors forming the shape of a cap - like in the figure - we first select the two *track* sensors *track*[*l*] and *track*[*r*] with the greatest value among the left *track* sensors (*track*[0], …, *track*[9]) and the right *track* sensors (*track*[9], …, *track*[18]) respectively. Next, we use only the *track* sensors *track*[0], …, *track*[*l*], and *track*[*r*], …, *track*[18] for the computation.

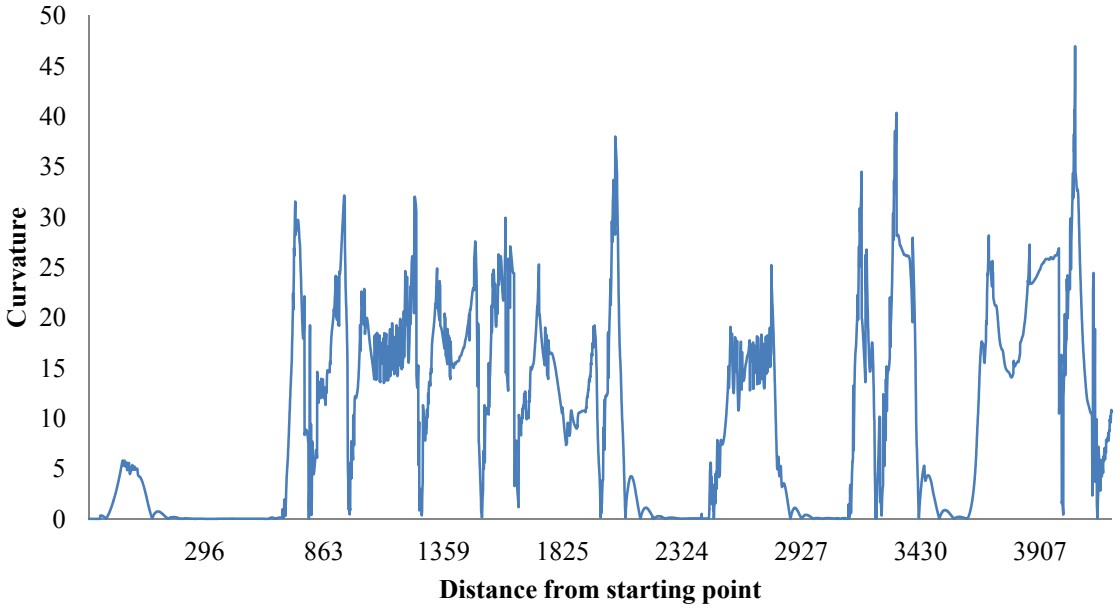


Figure 5. Variation of curvature information during a single lap at Wheel1.

Figure 5 shows the variation of the curvature values computed by the method above during a single lap at Wheel1. As can be seen, unlike the probability variation (**Figure 2**), the curvature information varies more frequently. For example, when the car is located at between 2324m and 2927m of the track, the curvature values do not exceed 30 but *prob* is always set to 1. This comparison means that the BFB3 decelerates greatly, even in a smooth corner or a SAZ, and cannot accelerate enough along the type of track which would be considered straight by a human.

3.1.2 The modified desired speed module

We utilized the curvature C of the track segment to determine the *Desired_Speed* more minutely. The curvature information is used even in the "straight" track segment. We divided the "straight" track segment of the desired speed module into three cases. The reason for this is that the controller needs to accelerate more radically in this part. In addition, we also defined three new rules for an SAZ to decrease an overrated probability. When the car is considered to be in a corner track segment, we do the same thing as with the BFB3.

In the "straight" track segment, two parameters S_1 and S_2 play the role of determining whether or not the car will be able to accelerate more rapidly (see Figure 6). If C is smaller than S_1, the desired speed is set to $MAX\_SPEED \times D_1$. If C is smaller than S_2, it is set to $MAX\_SPEED \times D_2$. Otherwise, it is set to *MAX_SPEED*. These additional rules and parameters make our controller more aggressive than the BFB3, especially where the track is almost straight. For example, if S_1 is 3 and D_1 is 5, our controller will accelerate 5 times faster than that of the BFB3 when the curvature value C is smaller than 3.

```
If ( CT ≥ 150m ) {   // the car is considered to be in a "straight" track segment
    If (C ≤ S1)
      Desired_Speed = MAX_SPEED×D1
    Else if (C ≤ S2)
      Desired_Speed = MAX_SPEED×D2
    Else
      Desired_Speed = MAX_SPEED
}
```

Figure 6. The desired speed computation for a straight track segment.

In an SAZ, the curvature value C is not used only for categorizing the cases but also for determining the *Desired_Speed* directly by adjusting *prob* with C. First, we divide the current situations into four cases using the values of *prob* and C and parameters S_3, S_4 and S_5. Then, we adjust the value of *prob* using the curvature C and the parameters D_3 and D_4 (see Figure 7). The three parameters S_3, S_4 and S_5 estimate how much *prob* is overrated, and D_3 and D_4 are involved in decreasing the overrated *prob*. In this way, we can reduce the overestimated probability value and prevent the car from decelerating fully.

If (70m < CT < 150m){
  prob = ANN(*Sensors*); // Artificial neural network
  If (*prob*×100 – *C* ≥ *S*3)
    prob = 0;
  Else if (*prob*×100 – C ≥ *S*4)
    prob = *prob* – *C*×D3;
  Else if (*prob*×100 – *C* ≥ *S*5)
    prob = *prob* – *C*×*D*4;
  If (*prob* < 0)
    prob = 0;

$$Desired\_Speed = \frac{\alpha - prob \times (MAX\_SPEED - speedX) \times |T_{\max} - track[9]|}{|T_{\min} - track[9]|};$$

// $T_{\max}$ = max(*track*[1], *track*[17]), $T_{\min}$ = min(*track*[1], *track*[17])

Figure 7. The desired speed computation for a SAZ.

3.1.3 The modified steering control module

To improve the steering control module of the BFB3, we deal with the "straight" track segment and the SAZ separately, which are processed identically in the BFB3. If the car is considered in the straight track segment, the controller keeps the car at the centre of the track axis, which is the same as the original steering control of the BFB3. Also, if the car is considered in the corner track segment, we do the same control as with the BFB3.

A different case from the BFB3 is when the car is considered in a SAZ. We introduce a new parameter O_1 in the steering module to prepare for an upcoming corner in advance. Due to the parameter O_1, the controller may not force the car to move towards the track axis when the corner is coming and so the car should not skid too much. If O_1 has a negative value, the controller will move the car in the opposite direction to the upcoming corner. For example, if the upcoming corner is right-handed, our steering control will move the car to the left-edge of the track. Otherwise, if O_1 has a positive value, the car will move in the same direction as the corner.

// *steerLock* = π/4
Procedure Get_Target_Steer( ){
  index = argmax(*track*[2], *track*[3], …, *track*[16]);
  If (CT ≥ 150) // in a straight track segment
    $T_{Steering}$ = 0.5×(*angle* – *trackPos*)/*steerLock*;
  Else If (70 < *CT* < 150) // in a SAZ
    $T_{Steering}$ = O_1×(*angle* – *trackPos*)/*steerLock*;
  Else // in a corner track segment
    $T_{Steering}$ = *w*[*index*] – 0.3×*trackPos*;
}

Figure 8. The modified steering control module.

3.2 Learning approach

3.2.1 Evolutionary strategy

Recently, evolutionary computation has been widely used to optimize robotic systems: for example, parallel robotic manipulators [19], motion planning for swarm robots [20] and humanoid motion planning [21]. We tuned the 10 new parameters D_1~D_4, S_1~S_5 and O_1 to find the optimal parameter set using an ES. Our first approach is to find the track-dependent parameter sets for 6 tracks, namely Alpine1, AspeedWay, Cgtrack2, Eroad, Wheel1 and Wheel2, as shown in **Figure 9**. For this work, we apply two kinds of ES methods.

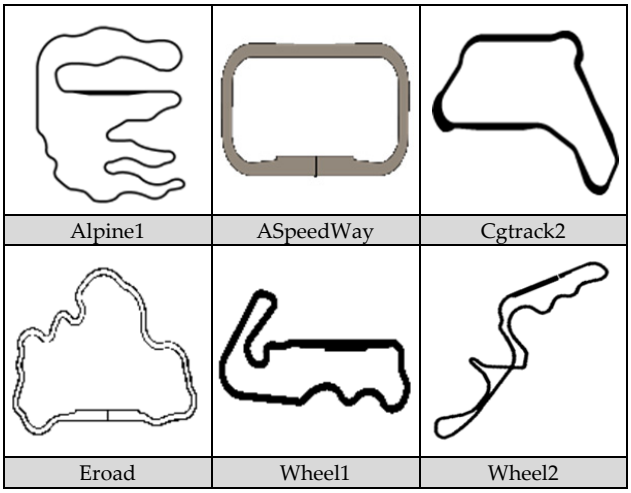


Figure 9. Six tracks used to optimize parameter sets.

The first method is a "simple" 10+10 ES. Initially, we make 10 individuals for the 1st generation. Each individual consists of the ten parameters and the initial value of each parameter is randomly chosen within the range 0~1. Next, we mutate every individual for the next generation. For each parameter p_i of an individual, its offspring parameter p'_i is defined as follows:

$$p'_i = p_i + N_i(0, 1), \qquad i = 1,...,n_p,$$

where N_i (0,1) is a standard Gaussian random variable for the i^{th} parameter and n_p is the number of parameters (i.e., 10).

At each generation, 10 new individuals generated by the mutation are evaluated with a single lap completion time. Among 20 individuals (the prior 10 individuals and the new 10 individuals), only the top 10 individuals survive for the next generation (Figure 10). However, the main drawback of this simple ES is that some parameters may increase repeatedly during whole generations.

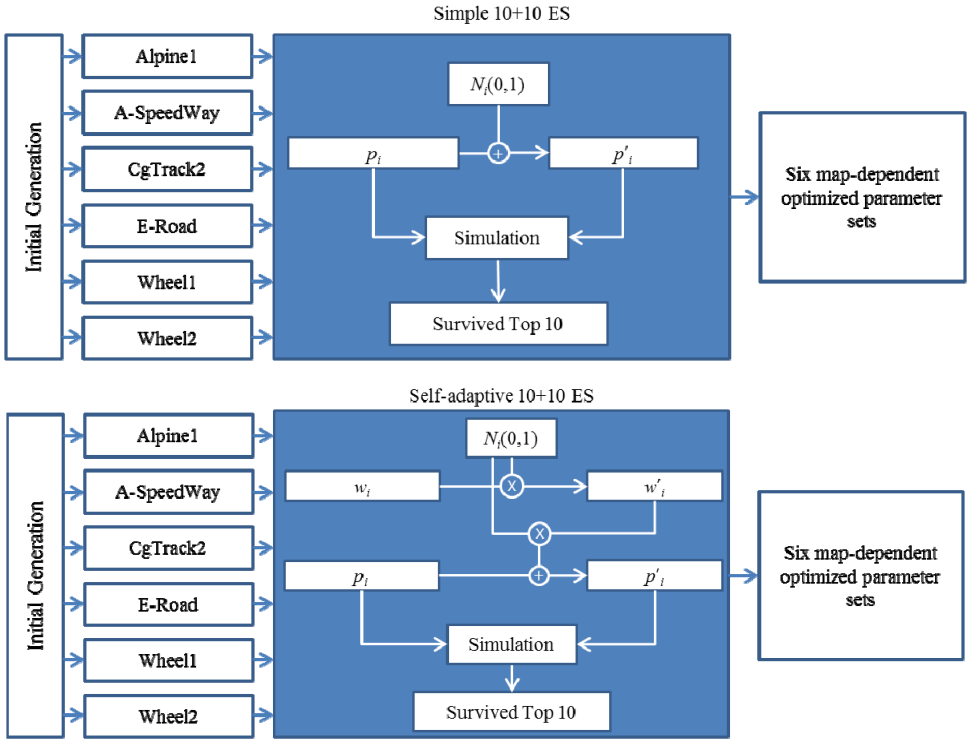


Figure 10. Overview of the simple and self-adaptive evolutionary strategy.

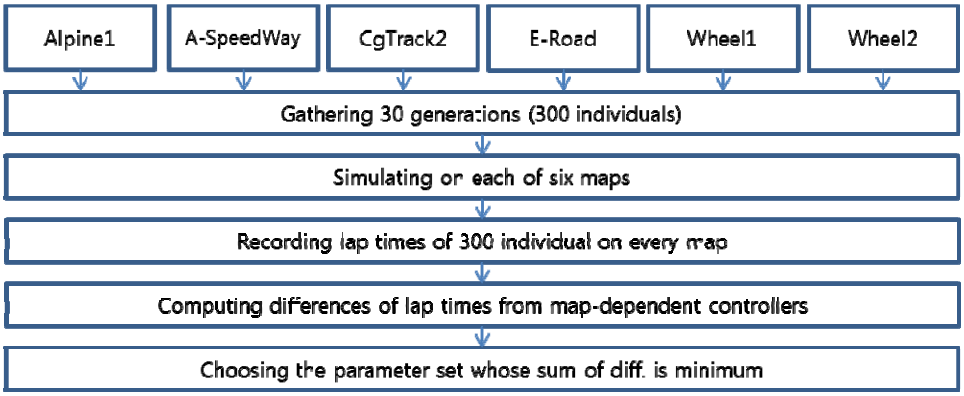


Figure 11. Flowchart for choosing the most general parameter set.

The second method is a "self-adaptive" 10+10 ES. This method is the same as the simple ES, except for the mutation operation used. For each individual, we define a weighted value (w_i) for each parameter (p_i) and this weighted value (the step size of the mutation) is also mutated during the generations. For each parent p_i, its offspring parameter p'_i is defined as follows:

$$w'_i = w_i + \exp(\tau \times N_i(0,1)), \qquad i = 1,\ldots,n_p$$
$$p'_i = p_i + w'_i \times N_i(0,1), \qquad i = 1,\ldots,n_p$$
$$\tau = \frac{1}{\sqrt{2\sqrt{N_p}}} = \frac{1}{\sqrt{2\sqrt{10}}} = 0.397635.$$

Note that, unlike the simple ES, the self-adaptive ES uses the two-stage mutation process (Figure 10).

3.2.2 The generalization of the parameter set

We generalize the parameter set by observing the behaviour of the optimized controller for each track. The parameter sets obtained by the learning strategy are dependent on the tracks and the sets are not practical for the competition. In order to find a general parameter set, we gather 30 generations by extracting 5 generations (the 20th, the 40th, the 60th, the 80th and the 100th) from 100 generations for the 6 tracks. In other words, we have 300 parameter sets in total. For each selected individual, we estimate the lap times for the 6 tracks and compute the sum of the lap time differences from the track-dependent optimized controllers. Among the 300 parameter sets, we choose the parameter set with the minimum standard deviation as the most general parameter set (Figure 11).

4. Experimental results

In this section, we experimentally analyse our controller. First, we compute parameter sets using the two strategies and analyse the performance of the controller with the parameter sets. Next, we generalize the parameter set and analyse the performance of the controller with the generalized parameter set.

4.1 Evolutionary strategy

We obtained 12 track-dependent parameter sets optimized by the two different learning strategies (the simple ES and the self-adaptive ES). We produced 100 generations to search the space. Table 1 shows the values of 12 optimal parameter sets optimized by two learning strategies for the 6 tracks and the lap times of the BFB3. As shown in the table, our controller significantly improved the performance of the BFB3. In addition, the self-adaptive ES found better parameters than the simple ES except for the tracks ERoad and Wheel2.

	Alpine1	ASpeedWay	CgTrack2	ERoad	Wheel1	Wheel2
S_1	-3.58	3.95	0.86	-1.35	-1.17	0.31
S_2	-4.50	-0.32	0.09	-1.17	-0.42	1.55
S_3	-0.69	3.39	0.15	1.17	2.66	-0.67
S_4	0.85	-2.09	0.08	-3.60	-0.16	-3.94
S_5	3.16	-1.43	0.47	0.14	2.19	-0.42
D_1	0.92	4.53	0.80	-1.52	0.58	2.29
D_2	-4.52	2.06	0.66	-2.05	2.57	2.57
D_3	0.83	3.59	0.05	-0.42	-0.57	2.46
D_4	0.36	-1.15	0.95	0.78	-1.01	0.95
O_1	3.14	0.01	0.43	1.49	-0.07	0.11

(a) Track-dependent parameter sets optimized by the simple ES.

	Alpine1	ASpeedWay	CgTrack2	ERoad	Wheel1	Wheel2
S_1	-0.03	6.43	0.34	3.01	3.24	0.92
S_2	2.11	1.41	0.12	-0.58	0.57	0.70
S_3	-0.90	-5.77	0.71	7.36	12.92	3.33
S_4	0.67	0.11	1.89	0.93	0.06	1.04
S_5	2.21	3.47	0.65	1.20	0.47	2.19
D_1	-0.33	19.29	1.37	2.72	9.07	3.03
D_2	1.45	2.31	0.28	3.17	-5.71	-1.57
D_3	0.47	0.28	1.86	0.91	0.31	2.39
D_4	1.13	5.59	0.21	-0.02	-0.24	-0.23
O_1	-0.01	15.26	-0.01	-0.90	-0.08	2.48

(b) Track-dependent parameter sets optimized by the self-adaptive ES.

	Alpine1	ASpeedWay	CgTrack2	ERoad	Wheel1	Wheel2
BFB3	162.63	36.67	65.92	88.25	93.34	139.97
Simple ES	159.42	36.33	64.12	**79.30**	90.63	**132.61**
Self-adaptive ES	**158.49**	**35.27**	**64.05**	79.54	**90.17**	133.02

(c) Lab times of three different controllers on the six tracks. (Bold means the best one)

Table 1. Results of evolutionary learning

Track-dependent controller	Tracks					
	Alpine1	Aspeed Way	Cgtrack2	Eroad	Wheel 1	Wheel 2
Alpine1	**158.49**	36.53	64.58	86.16	92.85	133.37
AspeedWay	161.71	**35.27**	65.44	81.43	91.95	135.51
Cgtrack2	161.75	36.35	**64.05**	86.40	118.99	134.07
Eroad	161.99	37.77	64.98	**79.54**	91.93	134.89
Wheel1	161.27	37.21	65.06	82.84	**90.17**	134.71
Wheel2	162.27	37.13	65.11	80.62	92.28	**133.03**

Table 2. Lap times of track-dependent controllers on the six tracks.

Figure 12 shows the comparison of the performance of the simple ES and the self-adaptive ES over 100 generations. The x-axis represents the generation number and the y-axis represents the lap time (in seconds). For CgTrack2 and ERoad, the performance of the two strategies is similar. For Wheel2, the self-adaptive ES is worse than the simple ES during the early generations but the two strategies are almost the same towards the end of the generations. For the other three tracks, the self-adaptive ES is better than the simple ES. Consequently, the self-adaptive ES generally escapes from a local minimum within a relatively short time and, thus, the self-adaptive ES is more suitable for learning the track-dependent optimal controllers than the simple ES.

Table 2 shows the lap times of track-dependent controllers trained by the self-adaptive ES on the 6 tracks. Obviously, the parameter set with the best record (indicated in bold face) for each track is the parameter set trained on the same track. However, the optimal parameter set of a particular track may witness shoddy performance on the other tracks. Specifically, the controller with the parameter set optimized on the track "Cgtrack2" ran the track "Wheel1" in 118.99 sec, which is 28 sec worse than the parameter set optimized on the track "Wheel1." Therefore, we need to find the general parameter set to make a practical track-independent controller.

4.2 Extracting the general parameter set

We generalized the parameter sets using the method described in Section 3.2. We used the self-adaptive ES for the generalization. Table 3 shows the top 5 individuals (parameter sets) from the search space of the self-adaptive ES whose sum of differences are the least. We denote the name of the parameter set by its generation number and individual number, such as <track name>_<generation number>_<individual number>. In our experiment, the best parameter set was Wheel1_100_8 and the second was Wheel2_20_1.

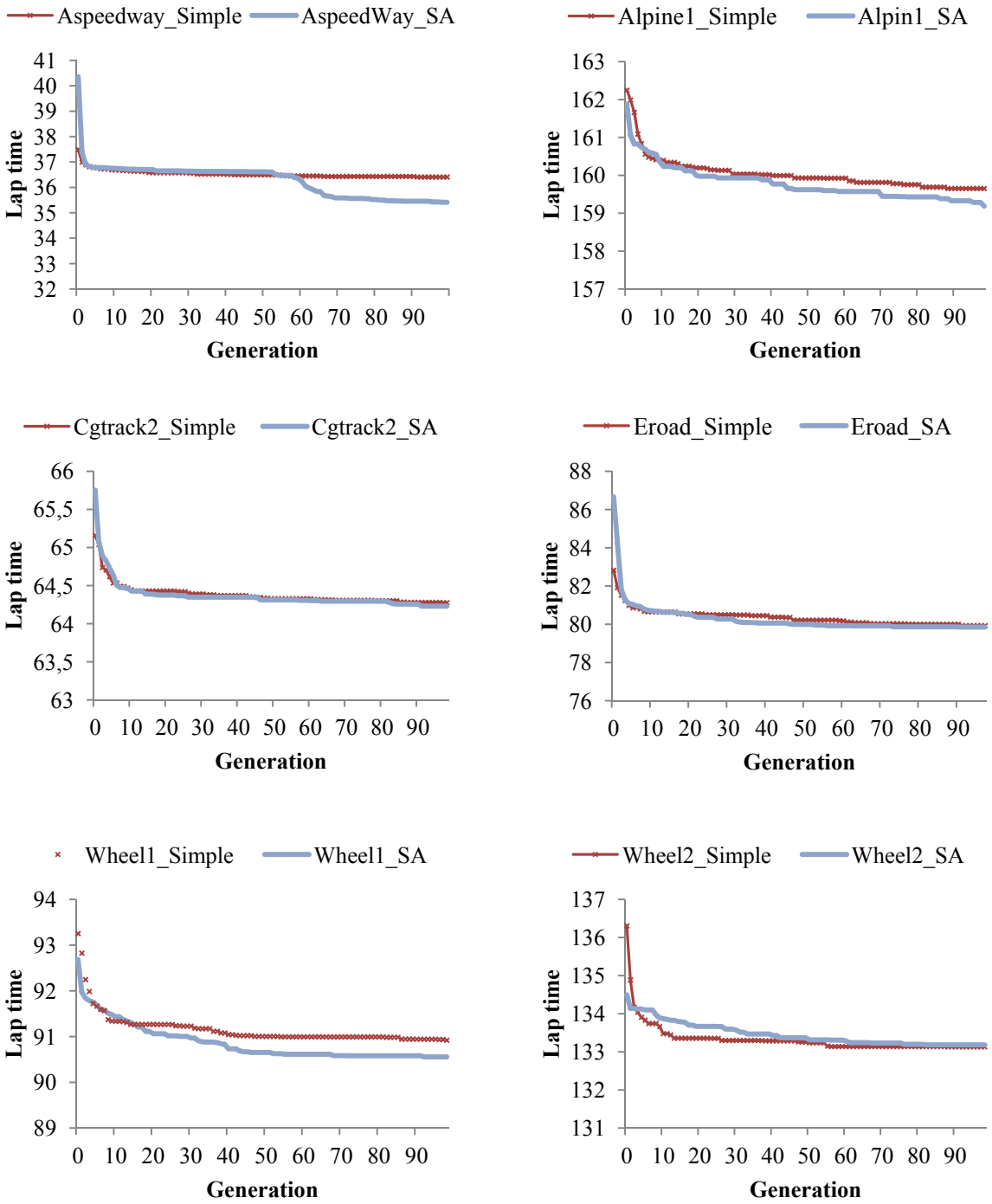


Figure 12. Variation for the average performance of the simple ES and the self-adaptive ES.

Param. set	Alpine1	ASpeedWay	CgTrack2	Eroad	Wheel1	Wheel2	Sum of diff.	Stddev.
Optimized	158.49	35.27	64.05	79.54	90.17	133.03	0	0
Wheel1_100_8	**160.49**	**37.25**	**65.15**	**78.33**	**90.69**	**134.91**	**6.28**	**1.14**
Wheel2_20_1	159.93	36.75	65.02	80.29	91.68	133.19	6.31	0.49
Alpine_20_6	160.47	37.13	64.43	79.19	92.06	134.53	7.25	0.89
Alpine_80_9	159.85	37.05	65.68	80.00	91.42	134.55	8.00	0.43
Alpine_100_3	158.89	36.75	64.95	81.19	92.21	134.59	8.04	0.54

Table 3. Lap times of the track-dependent parameter sets and the 5 most general parameter sets.

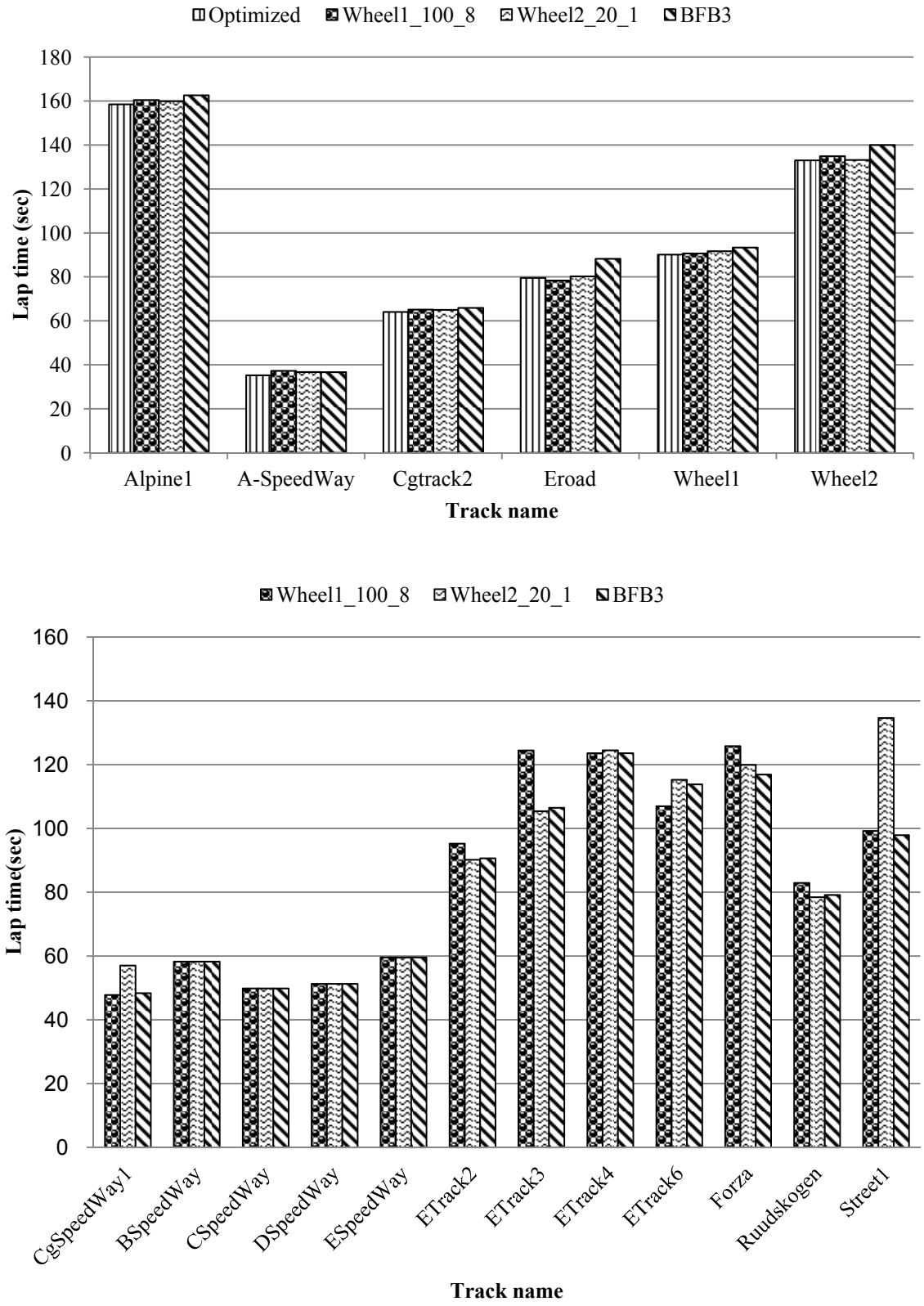


Figure 13. Comparison of the BFB3, track-dependent controllers and two generalized controllers at the six training tracks and other tracks.

Name	S_1	S_2	S_3	S_4	S_5
Value	5.89	1.47	-1.43	2.69	2.08
Name	D_1	D_2	D_3	D_4	O_1
Value	11.43	-10.38	0.30	0.00	-0.09

(a) Wheel1_100_8.

Name	S_1	S_2	S_3	S_4	S_5
Value	1.61	0.70	3.12	1.07	1.38
Name	D_1	D_2	D_3	D_4	O_1
Value	2.74	-1.83	3.06	-0.28	1.63

(b) Wheel2_20_1.

Table 4. Values of the parameter set

The values of the parameter set Wheel1_100_8 are given in Table 4. Even though we used the 10 parameters, only four parameters are meaningful. First, S_2 and D_2 of the straight track segment are never used in the desired speed module because S_1 is larger than S_2 (see **Figure 6**). Second, in a SAZ, only S_3 is used and *prob* is always set to 0 because S_3 is negative. That is, the controller with the parameter set Wheel1_100_8 does not require the prediction of an ANN. Consequently, the controller with Wheel1_100_8 only uses four parameters (S_1, D_1, S_3 and O_1). Similarly, the second best parameter set Wheel2_20_1, given in Table 4, uses 6 parameters (S_1, D_1, S_3, S_4, D_3 and O_1) and the value of *prob* is still not used. Although Wheel2_20_1 does not use four parameters (S_2, S_5, D_2 and D_4), it corresponds with our design intention because it still requires the prediction of an ANN.

To test the generality of the two parameter sets, we compared the four controllers: the generalized controller with Wheel1_100_8, the generalized controller with Wheel2_20_1, the optimal track-dependent controller, and the BFB3. The performance of each controller was measured upon the elapsed time to complete a single lap and the comparisons were carried out for the 6 tracks, Alpine1, ASpeedWay, Cgtrack2, Eroad, Wheel1 and Wheel2 (see Figure 13). Both the controllers with Wheel1_100_8 and Wheel2_20_1 performed in a manner which was worse on average than the track-dependent controller, but they were always better than the BFB3.

Although these generalized parameter sets are not dependent on a particular track, it may bring out slightly lower performance on several tracks because these parameter sets are still subordinate to the 6 tracks. Figure 13 shows the lap times on twelve TORCS tracks other than the 6 tracks used in the ES. Our controller is not always better than the BFB3 on some tracks which were not considered in our research. On only one track (CgSpeedWay1), the controller with Wheel1_100_8 had the best record among the three controllers. On three tracks (Etrack2, Etrack3 and Ruudskogen), Wheel2_20_1 was the best. On four tracks (BSpeedWay, CSpeedWay, DSpeedWay and ESpeedWay), the lap times of the controllers were almost similar. On the other four tracks, the BFB3 was the best.

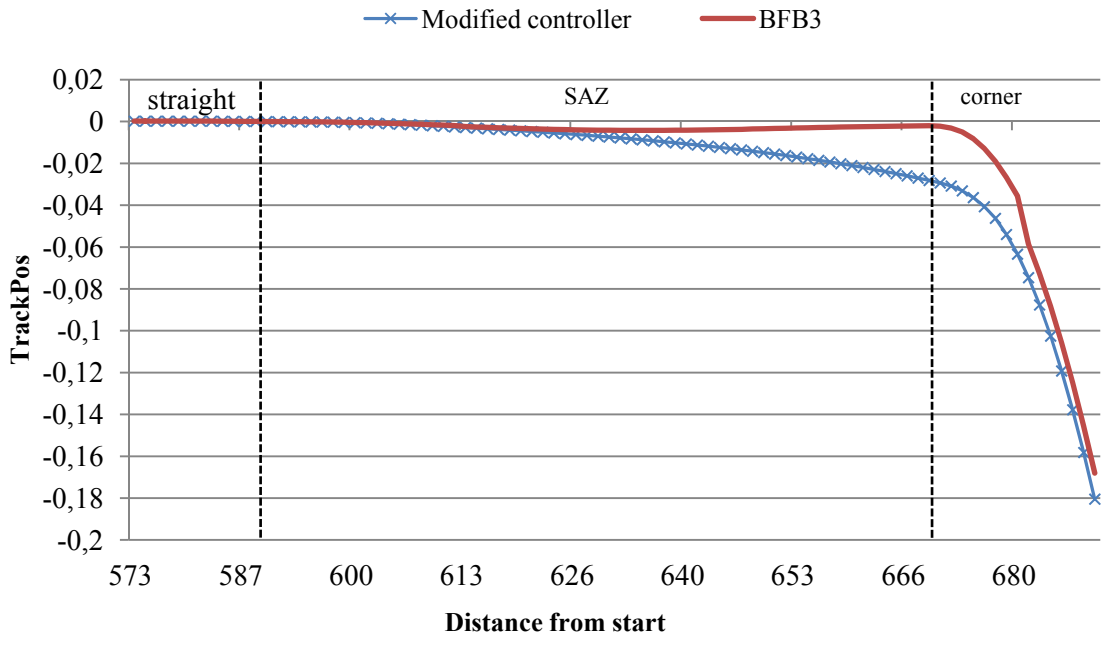


Figure 14. Variations for the values of *trackPos* by two different steering control modules.

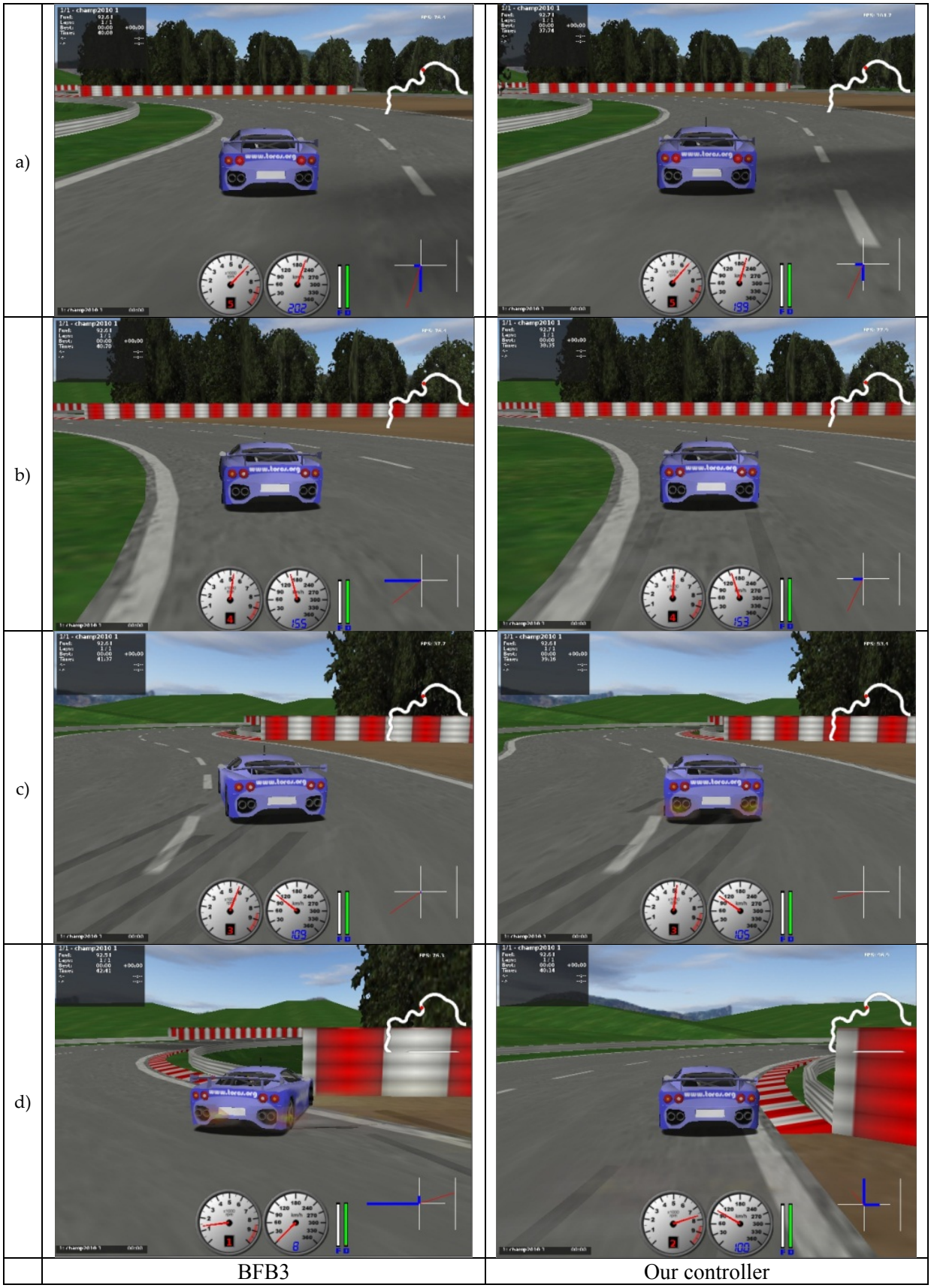


Figure 15. Two series of screenshots at Eroad.

We also compared the steering strategies of our controller and the BFB3 in SAZs. Figure 14 shows the variations of the *trackPos* values with our modified steering control module and the original steering control module of the BFB3 at a part of Wheel1 (573m~688m). We used the generalized parameter set Wheel1_100_8 for our controller. The track segment between 588m and 667m is considered as an SAZ in both controllers. Unlike the variation of the BFB3, our controller moved the car gradually in the same direction as the upcoming corner in the SAZ to prevent it from skidding. This shows that the BFB3 is more likely to be stuck at the sharp corner due to the drastic variation of the *trackPos* value.

Figure 15 shows a series of screenshots of the BFB3 (to the left) and our controller (to the right) at a corner of the track "Eroad". In Figure 15 b), our controller moved the car to a position closer to the inner side of the track than the BFB3 and it skidded relatively less than the BFB3 (Figure 15 c)). As a result, the BFB3 bumped the car against a track fence but our controller passed the corner smoothly (Figure 15 d)).

5. Conclusion and Future Works

In this paper, we proposed our new controller on the basis of the established controller, the BFB3 [17]. For this work, we modified the original desired speed module and the steering control of the BFB3 and utilized the curvature information. In addition, we defined additional rules and new parameters to improve the desired speed module and the steering control module. Moreover, we optimized the parameters using the simple ES and the self-adaptive ES method, and extracted the most general parameter set from the partial search space of the ES. As a result, our proposed controller can always drive faster than the BFB3 when it comes to the 6 tracks (Alpine1, AspeedWay, Cgtrack2, Eroad, Wheel1 and Wheel2). In addition, it reduces the time and effort for tuning the parameters of the manually designed controller.

Although this generalized parameter set is not dependent on a particular track, it may bring about slightly low performance on several tracks except for the 6 tracks. This is because the generalized parameter set is still subordinate to the set of the 6 tracks. However, we have supposed that if the set of tracks is extended enough, it will be able to obtain the more generalized parameter set through our method. In this work, we use if-then rules to represent the control mechanisms for the autonomous car, but there are several alternatives. For example, a type-1 and type-2 fuzzy logic controller can handle uncertainty and vagueness in the area of robotic control [22][23]. In particular, the type-2 fuzzy logic controller is promising in relation to the design of accurate control systems, but it requires additional parameters to be optimized [24][25][26].

6. Acknowledgements

Joong Chae Na, the corresponding author, was supported by the Basic Science Research Programme through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012-0003214). The co-author, Kyung-Joong Kim, was supported by the Basic Science Research Programme and the National Research Program for Brain Science through the National Research Foundation of Korea (NRF) and funded by the Ministry of Education, Science and Technology (2012-0001749, 2012-0005799).

7. References

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

[2] S. Haufe, M.S. Treder, M.F. Gugler, M. Sagebaum, G. Curio, B. Glankertz (2011) EEG potentials predict upcoming emergency brakings during simulated driving. Journal of Neural Engineering. Available: http://iopscience.iop.org/1741-2552/8/5/056001.

[3] P.V.D. Haak, R.V. Lon, J.V.D. Meer, L. Rothkrantz (2010) Stress assessment of car-drivers using EEG-analysis. Proceedings of the 11th International Conference on Computer Systems and Technologies. pp. 473-477.

[4] M.-I. Toma, D. Datcu (2012) Determining car driver interaction intent through analysis of behavior patterns. IFIP Advances in Information and Communication Technology 372: 113-120.

[5] C. Tran, M.M. Trivedi (2010) Towards a vision-based system exploring 3D driver posture dynamics for driver assistance: Issues and possibilities. IEEE Intelligent Vehicles Symposium. 179-184.

[6] F. Wang, M. Yang, R. Yang (2009) The intelligent vehicle coordination of the cybernetic transportation system. International Journal of Advanced Robotic Systems 6: 53-58.

[7] D. Meyer-Delius, C. Plagemann, W. Burgard (2009) Probabilistic situation recognition for vehicular traffic scenarios. IEEE International Conference on Robotics and Automation. pp. 459-464.

[8] M. Eilers, C. Moebus, (2010) Learning of a Bayesian autonomous driver mixture-of-behaviors (BAD MoB) model. 1st International Conference on Applied Digital Human Modeling.

[9] D. Loiacono et al., (2008) The WCCI 2008 simulated car racing competition. Proceedings of the IEEE Symposium on Computational Intelligence and Games. pp. 119-126.

[10] E. Onieva, D.A. Pelta, J. Alonso, V. Milanes, J. Pere (2009) A modular parametric architecture for the TORCS racing engine. IEEE Symposium on Computational Intelligence and Games. pp. 256-262.

[11] E. Onieva, J. Alonso, J. Perez, V. Milanes, T. de Pedro (2009) Autonomous car fuzzy control modeled by iterative genetic algorithms. IEEE International Conference on Fuzzy Systems. pp. 1615-1620.

[12] M.V. Butz, T.D. Lonneker (2009) Optimized sensory-motor couplings plus strategy extensions for the TORCS car racing challenge. IEEE Symposium on Computational Intelligence in Games. pp. 317-324.

[13] N.V. Hoorn, J. Togelius, D. Wierstra, J. Schmidhuber (2009) Robust player imitation using multiobjective evolution. Proceedings of the 11th Conference on Congress on Evolutionary Computation. pp. 652-659.

[14] J. Munoz, G. Gutierrez, A. Sanchis (2010) A human-like TORCS controller for the simulated car racing championship. IEEE International Conference on Computational Intelligence and Games. pp. 473-480.

[15] L. Cardamone, D. Loiacono, P. L. Lanzi (2009) Learning drivers for TORCS through imitation using supervised methods. IEEE Symposium on Computational Intelligence and Games. pp. 148-155.

[16] J. Seo, J. Park, J. Lee, K. Kim (2010) Designing robust robotic car controllers based on artificial neural network. International Conference on Convergence & Hybrid Information Technology. pp. 183-190.

[17] K. Kim, J. Seo, J. Park, J. C. Na (2012) Generalization of TORCS car racing controllers with artificial neural networks and linear regression analysis. Neurocomputing 88: 87-99.

[18] J. Quadflieg, M. Preuss, O. Kramer, G. Rudolph (2010) Learning the track and planning ahead in a car racing controller. IEEE International Conference on Computational Intelligence and Games. pp. 395-402.

[19] A.M. Lopes, E.J. Solteiro Pires, M.R. Barbosa (2012) Design of a parallel robotic manipulator using evolutionary computing. International Journal of Advanced Robotic Systems 9:1-13.

[20] C.-C. Lin, K.-C. Chen, and W.-J. Chuang (2012) Motion planning using a memetic evolution algorithm for swarm robots. International Journal of Advanced Robotic Systems 9:1-9.

[21] X. Wang, T. Lu, P. Zhang (2012) State generation method for humanoid motion planning based on genetic algorithm. International Journal of Advanced Robotic Systems 9:1-8.

[22] L. Astudillo, O. Castillo, L.T. Aguilar, R. Martínez-Soto (2007) Hybrid control for an autonomous wheeled mobile robot under perturbed torques. International Fuzzy Systems Association World Congress. pp. 594-603.

[23] R. Martínez-Soto, O. Castillo, L.T. Aguilar (2009) Optimization of interval type-2 fuzzy logic controllers for a perturbed autonomous wheeled mobile robot using genetic algorithms. Information Sciences 179(13): 2158-2174.

[24] N.R. Cázarez-Castro, L.T. Aguilar, O. Castillo (2010) Fuzzy logic control with genetic membership function parameters optimization for the output regulation of a servomechanism with nonlinear backlash. Expert Systems with Application 37(6): 4368-4378.

[25] O. Castillo, P. Melin, A.A. Garza, O. Montiel, R. Sepúlveda (2011) Optimization of interval type-2 fuzzy logic controllers using evolutionary algorithms. Soft Computing 15(6): 1145-1160.

[26] O. Castillo, R. Martinez-Marroquin, P. Melin, F. Valdez, J. Soria (2012) Comparative study of bio-inspired algorithms applied to the optimization of type-1 and type-2 fuzzy controllers for an autonomous mobile robot. Information Sciences 192: 19-38.