Neurocomputing 88 (2012) 87–99

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Generalization of TORCS car racing controllers with artificial neural networks and linear regression analysis

Kyung-Joong Kim*, Jun-Ho Seo, Jung-Guk Park, Joong Chae Na

Department of Computer Engineering, Sejong University, 98 Gunja-Dong, Gwangjin-Gu, Seoul, Republic of Korea

ARTICLE INFO

Available online 17 January 2012

Keywords:
Game
Artificial neural network
Linear regression analysis
Modular architecture
Control

ABSTRACT

Designing controllers for simulated cars is a challenging task because there are numerous sensor inputs and a lot of actuators to be controlled. Although it is possible to use domain-expert knowledge on the car racing, it is not trivial to represent the knowledge into controllers and tune the parameters for them. Therefore, it is natural to adopt machine learning approach into the knowledge-oriented controllers to enhance their performance and minimize tedious parameter tunings. In this paper, we try to enhance our own heuristic controllers using machine learning models which decide the appropriate parameters of the heuristic controllers given the current sensory inputs. At first, we predict the desired speed only using the equations derived by the linear regression analysis for the both curved and straight track segments. Because the decision on reducing speed before the corner is more complex than the one in the straight line and the corners, it is necessary to use a non-linear model such as artificial neural networks. Secondly, the linear regression and artificial neural networks are specialized to predict desired speed in different situations. Experimental results on TORCS-based car racing simulations show that the combination of the two machine learning algorithms with the heuristic outperforms other alternatives (heuristic only, heuristic+linear regression, and heuristic+artificial neural networks).

1. Introduction

Game is one of the most important problems used in artificial intelligence research because they provide with simple and abstract models of real world. For example, iterated prisoner's dilemma (IPD) games have been widely used to find good decision strategies in economy, politics, and foreign relations. In the game, there are only two choices (cooperation or defection) but it successfully simulates a lot of realistic dilemma situations. Since the beginning of the artificial intelligence, several board games have been widely used as a research platform and finally, some of them are nearly solved. For example, Deep Blue team by IBM defeated the world chess champion with a specially designed super computer. In the board games, each player has finite number of decision choices and there is no hidden information between players.

Recently, the game AI research has expanded their territories to other game genres like real-time simulations, video games, and role-playing games [1]. They're quite different with the traditional board games. It is not a turn-based game and each player has to respond to the game as quickly as possible. Unlike the board games, players have to decide the degree of controls. Also, there is hidden information between players and random factors make games unpredictable. They have raised several new research challenges asking for new computational methodologies dealing with uncertainty, large amount of data, real-time decision making, and adaptation. Because it is an early stage of development, the AI for the games is still dependent on human's expertise.

Since 2008, international game AI competitions have changed their basic platform from 2D point-to-point car racing to realistic 3D car racing simulator, TORCS (The Open Racing Car Simulator) [2] (Fig. 1). It provides with realistic 3D car racing simulation and opens their source codes to public. Competition organizers have developed patch programs for the TORCS to allow several AI controllers run together in the simulator. In the competition, TORCS continuously sends simulated car's sensory information to clients program connected through network. Each client gets the data and decides the values of several actuators (steering wheel, accelerator, brake and gear). Their decisions are returned to the TORCS server and the next time step is simulated for the car racing environments. In this way, the car racing is simulated realistically with several AI controllers.

It is not a trivial task to control the simulated car because the controller should determines the values of five actuators (steering wheel, gear, clutch, break, and acceleration) based on seventy

* Corresponding author. Tel.: +82 2 3408 3838.
E-mail addresses: kimkj@sejong.ac.kr (K.-J. Kim), sejunho@hanmail.net (J.-H. Seo), prfirst@sju.ac.kr (J.-G. Park), jcna@sejong.ac.kr (J.C. Na).

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2011.06.034

nine sensors available. In the competition, the three tracks used in the competition day are unknown before the event so that it is not possible to optimize their controller to them. Also, in the racing, they see only 200 m around the car and it is not possible to get the whole track data structure directly from the TORCS. Because of the complexity of the task, there has been several computational intelligence approaches for the controllers [3]. For example, fuzzy controllers have been used to determine the desired speed value of the simulated car and their fuzzy rules are optimized using genetic algorithms [4,5].

In our previous work, we developed our own heuristic controller placing the main focus on driving as safe as possible [6]. Although it is important to drive as fast as possible in the competition, it has to minimize damage caused by bumping the tracks and other cars. If the damage is larger than pre-defined threshold, the car is automatically dropped out from the race. Our basic driving algorithm was modified to improve safety by incorporating memorizing failures, speed adaptation zone, and artificial neural networks to predict corner difficulty. Experimental results on three tracks showed that it was able to minimize damages successfully. In this work, we focus on speed and safety together. Although the basic driving module and speed adaptation zone is similar to the previous one, we newly adopt linear regression analysis and artificial neural networks to adjust desired speed module. Also, we test the proposed method on fourteen tracks.

Fig. 1. TORCS (http://torcs.sourceforge.net/).

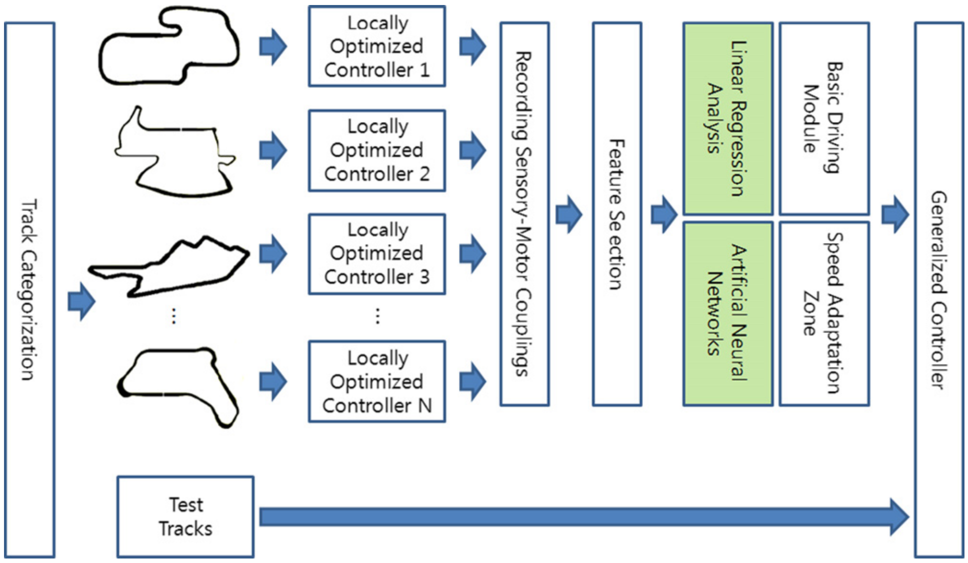
In the modular approach, there are several parameters to be tuned and usually, optimal parameter values are not the same for different tracks. It means that the best parameters on track *A* are not optimal on another track *B*. It is natural that each track requires different playing strategy due to the difference of track shapes. If designers choose parameters optimized for one track, it is likely to result in performance degradation on unseen tracks because of poor generalization capability. In this paper, we propose to use artificial neural networks and linear regression analysis to learn the different playing strategies of several controllers optimized for different tracks. From the TORCS package, we collected nineteen tracks and classified them into five groups based on the difficulty of driving. One representative track was chosen from each group to generate locally optimized controllers. The behavior of the controllers was monitored and they're used as training samples for regression analysis and artificial neural networks to predict the desired speed. Fig. 2 summarizes the basic idea of our proposed method.

In this paper, we try to enhance our own heuristic controllers using machine learning models which decide the appropriate parameters of the heuristic controllers given the current sensory inputs. At first, we predict the desired speed only using the equations derived by the linear regression analysis for the both curved and straight track segments. Because the decision on reducing speed before the corner is more complex than the one in the straight line and the corners, it is necessary to use a nonlinear model such as artificial neural networks. Secondly, the linear regression and artificial neural networks are specialized to predict desired speed in different situations. Experimental results on TORCS-based car racing simulations show that the combination of the two machine learning algorithms with the heuristic outperforms other alternatives (heuristic only, heuristic + linear regression, and heuristic + artificial neural networks).

2. Backgrounds

2.1. Car racing competition

In this competition, each controller gets sensory information from the TORCS server and sends their decisions to the simulator. There are seventy nine sensors available and seven actuators to be controlled by the program (Fig. 3).


Track Categorization
Locally Optimized Controller 1
Locally Optimized Controller 2
Locally Optimized Controller 3
Locally Optimized Controller N
Recording Sensory-Motor Couplings
Feature Selection
Linear Regression Analysis
Artificial Neural Networks
Basic Driving Module
Speed Adaptation Zone
Generalized Controller
Test Tracks
Fig. 2. An overview of the proposed method.

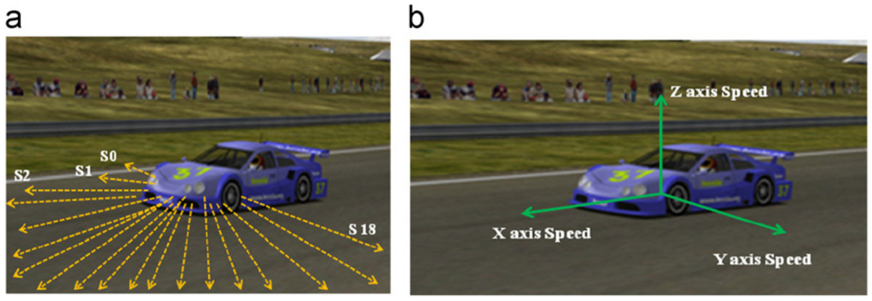


Fig. 3. Illustration of (a) track and (b) speed sensors.

Competition organizers provide clients with the API that accesses the sensory information and sends actuator values to the server. The rule of the competition is simple. If the number of competitors is larger than eight players, there is another preliminary league called solo racing. In the racing, each player race on the unseen tracks alone and the distance raced is used to rank them. Finally, the best eight players race together in the same unseen tracks and the final score is measured based on F1 system used in real car racing competition. In 2010, there are small changes in the competition rules. They added a new stage called "warming-up" and players are allowed to collect information from unseen tracks. The maximum distance of range finder sensors are extended from 100 to 200 m.

The competitions have been hosted by several international conferences. For example, in 2010, there were three car racing competitions by CIG, GECCO and WCCI (World Congress on Computational Intelligence). For each competition, they used different styles of tracks: motor speedways (GECCO), technical (F1-like) tracks (WCCI), and dirt (involving non-asphalt stretches and bumps) tracks (CIG). The final champion of 2010 was decided based on the sum of scores collected from the three competitions. The winner of the 2010 championship was AUTOPIA by Onieva et al. who also won the 2009 championship. In 2008, the winners of the CIG and WCCI used evolutionary neural networks to control the simulated car. Computational intelligence algorithms have been widely used for TORCS-based car racing competitions (Table 1).

Onieva et al. proposed a parameterized modular architecture consisting of desired speed, gear control, low level gas & brake control, steering control, and opponent modifier modules [4]. In the gear control, they used simple heuristics: the gear control increases the current gear if the rpm value is higher than a certain value and decreases it if the rpm value is lower than a certain value. In the desired speed module, it estimates the appropriate speed given environmental sensors using hand-coded fuzzy rules. If the desired speed is larger than the current speed, the car press the acceleration pedal and vice versa. For steering module, they calculated a weighted average from nine track sensors' values. Finally, the opponent modifier adjusts the brake, acceleration, and steering values based on opponent sensors.

Butz et al. developed the controller named as COBOSTAR and won CEC and CIG competitions in 2009 [11]. In their approach, they used different strategies for on-track and off-track situations. There were fourteen parameters for the on-road optimization and the covariance matrix adaptation (CMA) evolutionary strategy was used. All parameters were optimized on various available tracks in TORCS. They reported that different CMA optimization runs yield radically different parameter settings, each of which is optimized for the track trained on. They tested the optimized parameters on the other tracks and the most general parameter set was finally chosen.

Table 1
Summary of TORCS-based car racing research.

Year	Authors	References	Techniques
2010	Munoz et al.	[7]	NN
	Cardarmone et al.	[12]	ENN
		[13]	GA
	Quadflieg et al.	[14]	ES
2009	Onieva et al.	[4]	Fuzzy
		[5]	Fuzzy + GA
	Butz et al.	[11]	CMA-ES
	Cardarmone et al.	[15,16]	ENN
		[8]	KNN, ENN
	Perez et al.	[17]	Fuzzy, DT, MOEA
	Quadflieg et al.	[3]	ES
	Munoz et al.	[3]	EA
		[9]	NN
	Bernardi et al.	[3]	Rule
	Szymaniak	[3]	GP
	Ebner et al.	[18]	GP
	Hoorn et al.	[10]	MOEA
2008	Perez et al.	[19]	Rule+GA
		[20]	GA
	Kinaird-Heether et al.	[2]	Rule+CA
	Tan et al.	[2]	ES
	Gowrisankar	[21]	ENN
	Cardamone	[21]	ENN
	Simmerson	[2]	ENN

*ENN=Evolutionary Neural Network, NN=Neural Network, GA=Genetic Algorithm, ES=Evolutionary Strategy, KNN=K-Nearest Neighbor, DT=Decision Tree, MOEA=Multi-Objective Evolutionary Algorithm, GP=Genetic Programming.

2.2. *Learning approach*

Human learns new skills by observing other's behaviors and this is more efficient than learning from scratch. Similarly, the learning by imitation has been applied to create new controllers from the behaviors of human players or other controllers. The first step is "demonstration by experts": Human players or other controllers drive the simulated cars through TORCS. In the demonstration, the TORCS server records all the sensory information of the cars and the decisions made by the players. The next step is "learning from the training samples." It is a kind of supervised learning problems to predict the decisions of the players given the sensory information.

There are several works on the imitative learning for the simulated car competitions. Hoorn et al. proposed to solve the imitation learning using multi-objective evolutionary algorithms, with objectives relating both to traditional progress-based fitness (playing the game well) and similarity to recorded human behavior (behaviors like the recorded player) [10]. Munoz et al. used data obtained from two controllers (the winner of the WCCI 2008 simulated car racing competitions and hand coded controller that performs a complete lap in all tracks) and one human player [9]. They tried to generate new controllers from the mix of the data. They reported that it was very complicated to learn the human behavior in a video game but it was possible to learn a behavior from non-human controllers. In case of the mixed data, the results showed that the controllers did not work. Cardamone et al. proposed to use high-level information about the track ahead of the car and predict high-level actions [8]. They reported that it was possible to develop drivers with performance that in some cases are only 15% lower than the performance of the fastest driver available in TORCS. Munoz et al. trained neural networks with data retrieved from a human player to predict trajectory and speed [7].

3. Proposed methods

Fig. 4 shows our heuristic controller with artificial neural network and linear regression analysis. In the desired speed module, it decides the target speed to be maintained by the car. If the current speed is larger than the target speed, it controls brakes. On the other hands, if the current speed is smaller than the target speed, it accelerates. The desired speed is calculated differently based on the current observation on the tracks. If the sensory information tells that the track is straight, the target speed is the maximum. If the car is in the area where speed adaptation is required, it asks the target speed to artificial neural networks. In other cases, the desired speed is calculated from the equations derived from linear regression analysis.

3.1. Heuristic controller

Our controller was constructed following the philosophy of modular architecture proposed by Onieva et al. [4]. However, the implementation of each module is not the same with the original proposal. Unlike the original proposal, we added new modules for speed adjustment and failure handling, which reduces the speed of the car at corners and stuck positions, and each module was implemented using different algorithms. If the car is in stuck, it uses different steering and low level gas & brake control strategies. In normal situation, the controller calculates the desired speed and gear levels. The gear control increases the current gear if the rpm value is higher than a certain value and decreases it if the rpm value is lower than a certain value. The speed adjustment module is activated to reduce the speed of the car if the car is entering to corners. If the current speed of the car and the desired speed are different, the low-level gas & brake control do actions to reduce the difference. ABS & TCL module protects the car from slipping. The direction of the car is controlled by the steering module.

3.1.1. Steering control

The 19 track sensors are used to measure the distance between the car and the track edge. By default, the sensors sample the space in front of the car every 10°, spanning clockwise from $+\pi/2$ up to $-\pi/2$ with respect to the car axis. From 2010 car racing competition, it is allowed to customize the position of the track sensors by users. In this work, we customized the configuration of the track sensors. Unlike the default, there are a lot of track sensors in front side of the car. The basic idea of the steering is to control your car to go to the largest free distance.

We first check whether the car is in the straight or corner track using the center track sensor. If the value of the center track sensor is greater than 70 m, it is regarded as "straight" and we steer the car toward the direction of the track axis. Angle is between the car direction and the direction of the track axis. TrackPos is the distance between the car and the track axis. In this case, the target steering value is defined as

$$T_{Steering} = 0.1(angle - trackPos)/(\pi/4).$$

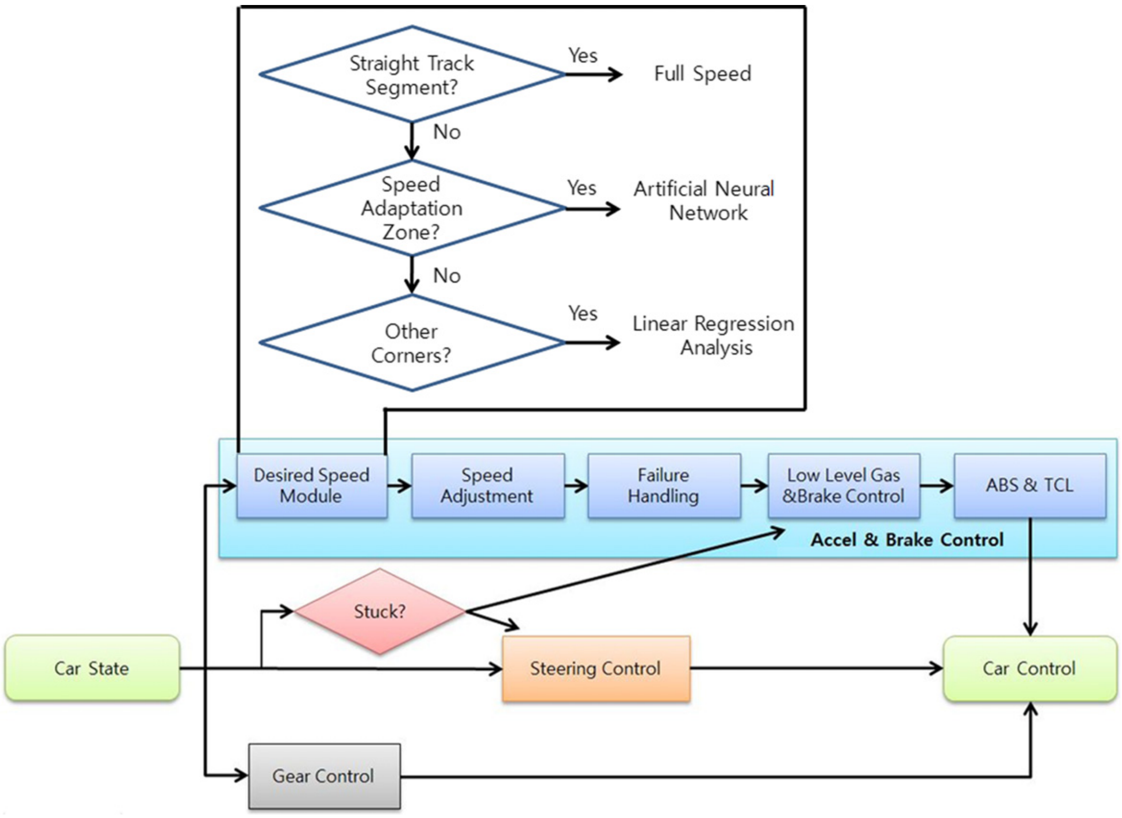


Fig. 4. Basic driving module with artificial neural networks and linear regression analysis.

```
// steerLock = $\pi/4$

Procedure Get_Target_Steer( ){
  index = argmax(track[2], track[3], …, track[16]);
  If ( Track[9] > 70 ) // in straight track
    $T_{Steering}$ = 0.1 x(angle – trackPos)/steerLock;
  Else                // in corner track
    $T_{Steering}$ = w[index] – 0.3 x trackPos;
  Return $T_{Steering}$;
}
```

Fig. 5. Steering algorithm.

Otherwise, the track is “curved” and we steer the car toward the direction of the *track* sensor with the largest free distance. The target steering value in the curve is defined as

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

where *index* indicates the *track* sensor number with the maximum value (largest free distance) except the center, right, and left *track* sensors, and *w* is a vector of weight values for the *track* sensors tuned manually. Fig. 5 gives a full description of the “steering control” module.

3.1.2. Desired speed calculation and speed adjustment

Based on the value of the center *track* sensor (CT), we categorized current situation into two groups. If the CT is larger than 150 m, it means that there is no obstacle in front of the car and the maximum speed is desired. If the CT is smaller than 150 m, the desired speed is proportionate to the CT. The bending of track is proportionate to $|Tmax - \mathrm{CT}|$ but inverse proportionate to $|Tmin - \mathrm{CT}|$, where $Tmax = \max(track\ [1],\ track\ [17])$ and $Tmin = \min(track\ [1],\ track\ [17])$. Consequently, the desired speed is computed as follows:

$$Desired\_Speed = \frac{\beta \times CT \times |\mathrm{Tmax} - CT|}{(\mathrm{Tmin} - CT)},$$

where β is a parameter to be tuned.

After the desired speed is computed, the acceleration and brake signal (*A&B*) is adjusted by the difference between the current speed (*speedX*) and the desired speed (*Desired_Speed*) as follows:

$$A\&B = \frac{2}{(1 + \exp(speedX - Desired\_Speed))} - 1.$$

The acceleration and brake signal is in -1 (full brake) ~ 1 (full acceleration). Fig. 6 gives a full description of the basic module.

3.1.3. Speed adaptation zone

Another strategy to improve the performance of the controller is to introduce a new category called the *speed adaptation zone (SAZ)* in addition to the straight track and the corner track. From our initial observation, we realized that the car accidents frequently occur before corners with high speed entering. Based on this observation, we set the front of a corner as SAZ and compute the desired speed differently from the straight track and the corner track. If $70 < \mathrm{CT} < 150$, the car is considered in the SAZ and the desired speed is computed as follows:

$$Desired\_Speed = \frac{\alpha \times (MAX\_SPEED - speedX) \times |\mathrm{Tmax} - CT|}{|\mathrm{Tmin} - CT|},$$

```
Procedure Get_Accel_Brake_Basic( ){
  Tmax = max(track[1], track[17]);
  Tmin = min(track[1], track[17]);

  If ( track[9] $\geq$ 150 ) {   // in straight Track
    Desired_Speed = MAXSPEED;
  }
  Else {   // in corner track
    Desired_Speed = $\beta$x track[9] x |Tmax - track[9]| / |Tmin - track[9]|;
  }
  Return 2/(1+exp(speedX - Desired_Speed)) - 1;
}
```

Fig. 6. Desired speed calculation and speed adjustment algorithm.

where α is a parameter to be tuned like β. Fig. 7 gives the description of controlling the acceleration and brake signal when the SAZ strategy is combined with the basic module.

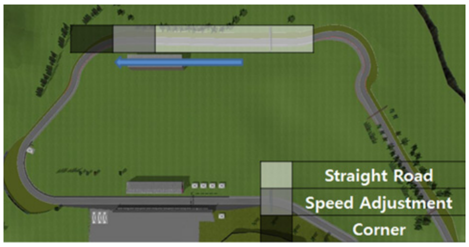
3.1.4. Failure handling

To improve the performance of the heuristic controller, we added to the basic module a new module called “Failure Handling.” This module utilizes the car racing property that the car rounds repeatedly the same track. The basic idea is to memorize the position where the car is stuck and reduce the speed when the car approaches the position in the next round. If the value of the *angle* sensor is not between $-\pi/4$ and $+\pi/4$, it is regarded that the car is stuck. The “*distFromStart*” sensor was used to indicate the failure position. If the failure position is denoted by x, then the *speed reducing area* is between $x - 300$ m and $x + 20$ m. If the car is in the speed reducing area and the current speed of the car is greater than $(15 \times Track\_Width)$, we set the acceleration and brake signal as -1 (full brake) ignoring the value computed by the “Speed Adjustment” module. Fig. 8 gives the description of controlling the acceleration and brake signal in “Failure Handling” module.

3.2. Learning approach

Usually, there are a lot of tracks available. However, it is not always useful to use multiple tracks because it increases the number of training samples and learning time. Our approach is to group the tracks into several categories based on its difficulty. Our heuristic controller has two parameters alpha and beta to control desired speed for different situations (SAZ and corners). For each track, we run grid search for the alpha and beta to minimize lap time. Based on the record, we calculate average speed for each track. If the average speed is large, the track is regarded as an easy one. In our work, we set the group size as five. For each group, we choose one track and they are used to generate training samples with the best alpha and beta for the track. The training sample contains all the sensory information and the decision of the controllers. The last step is to merge the training samples from the five tracks and run machine learning algorithms. It is also possible to find the global best alpha and beta on the five tracks. The final generalized controller contains a neural network, equations from linear regression and the global best alpha and beta. Fig. 9 summarizes this process as a flowchart.

a



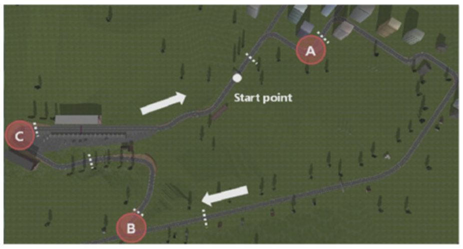
b

```
Procedure Get_Accel_Brake_SAZ ( ){
 $Tmax$ = max($track$[1], $track$[17]);
 $Tmin$ = min($track$[1], $track$[17]);

 If ( $track$[9] ≥ 150 ) { // in straight track
   $Desired\_Speed$ = MAXSPEED;
 }
 Else if (70 < $track$[9] && $track$[9] < 150){ // in SAZ
   $Desired\_Speed$ = α x (MAXSPEED - $speedX$) x |$Tmax$-$track$[9]| / |$Tmin$ - $track$[9]|;
 }
 Else {   // in corner track
   $Desired\_Speed$ = β x $track$[9] x |$Tmax$ - $track$[9]| / |$Tmin$ - $track$[9]|;
 }
 Return 2/(1+exp($speedX$ - $Desired\_Speed$))-1;
}
```

Fig. 7. Algorithm with SAZ. (a) An example of SAZ. (b) A pseudo code with SAZ.

c



d

```
// If the car is near the memorized failure position again, reduce speed
Procedure Get_Accel_Brake_Failure($current\_accel\_brake$){
   If ( the car is in a speed reducing area && $speedX$ > 15x $Track\_Width$ )
       Return -1; // Reduce Speed
   Else
       Return $current\_accel\_brake$; // Don't need to change the value
}
```

Fig. 8. Failure handling algorithm and example. (c) An example of the stuck list for memorizing failure. (d) A pseudo code for failure handling.

3.2.1. *Track categorization and training sample generation*

Our first approach for discovering the generalized models is to find the *generalized* parameters α and β for the selected tracks. The behavior of our heuristic controller is dependent on the parameters α and β. Because each track has different properties, the optimal parameters for each track might not be the same. To find the generalized parameters, we first computed the sum of the lap time on every sampled track for various α and β, and chose the parameters with the minimum sum as the generalized parameters. The parameters with the minimum sum can be found using an exhaustive search on some parameter candidates.

In this paper, we tried to extract generalized controller from several optimized controllers, each of which is optimized for the track trained on. The first step is to generate training samples that record all the sensory information and decisions of each controller on the track optimized (Table 2). From the mix of training samples from the multiple controllers, linear regression analysis and neural networks are used to discover the generalized models.

3.2.2. *Linear regression analysis*

From Section 3.2.1, we found the generalized parameters α and β for the five tracks. As we can see at experimental results section, the generalized controller did good performance on the sampled tracks, but it worked poor in some unknown tracks. To improve the performance on unknown tracks, we apply machine learning methods such as linear regression analysis.

Linear regression analysis is used to predict the desired speed given environmental sensors. In this analysis, 22 sensors (19 *track* sensors and 3 *speed* sensors) are used and their weighted sum is the output of the prediction. Linear regression algorithms search for the weight vectors for the linear equation of sensors. If the

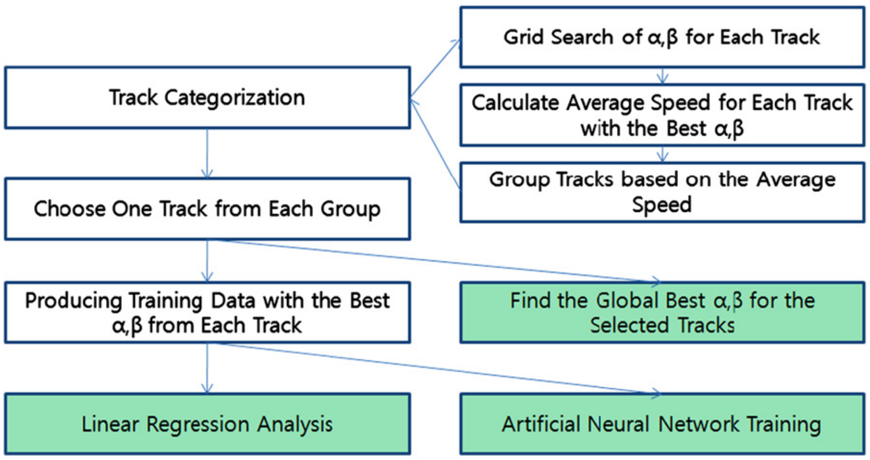

Track Categorization
Grid Search of α,β for Each Track
Calculate Average Speed for Each Track with the Best α,β
Group Tracks based on the Average Speed
Choose One Track from Each Group
Producing Training Data with the Best α,β from Each Track
Find the Global Best α,β for the Selected Tracks
Linear Regression Analysis
Artificial Neural Network Training

Fig. 9. Flowchart of learning approach (the shade box means models used in the generalized controller).

Table 2
An example of training data.

Time	Input variables							Target variables	
	track [0] (T0)	*track*[1] (T1)	...	*track* [18] (T18)	*speedX* (SX)	...	*speedZ* (SZ)	*Desired speed* (DS)	0, SX ≤ DS 1, SX > DS
00:00:00	200	200	...	100	10	...	2	20	0
00:00:02	198	199	...	99	20	...	4	40	0
00:00:04	195	197	...	98	30	...	1	20	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

```
Procedure Get_Accel_Brake_LR ( ){
  If ( $track[9] \geq 150$ ) {  // in straight track
    $Desired\_Speed$ = MAXSPEED;
  }
  Else {  // in SAZ & corner track
    $Desired\_Speed = w_1 track[0] + \cdots + w_{19} track[18] + w_{20} speedX + w_{21} speedY + w_{22} speedZ$;
  }

  Return 2/(1+exp($speedX$ - $Desired\_Speed$ ))-1;
}
```

Fig. 10. A controller only with equations from linear regression.

linear regression is used, the desired speed is calculated with the following equation:

$$Desired\_Speed = w_1 Track[1] + \cdots w_2 Track[1] + w_{19} Track[18] + w_1 speedX + w_{21} speedY + w_{22} speedZ$$

Fig. 10 gives the description for the acceleration and brake signal modified with the linear regression analysis.

3.2.3. *Artificial neural network*

The neural network consists of one hidden layer and all neurons of the network have a sigmoid transfer function $f(x) = 1/(1+exp(-x))$. The network is fully connected and each neuron has a bias. The training rule is as follows:

$$W_{n+1} = W_n + \eta \frac{\partial E}{\partial W} + \zeta (W_n - W_{n-1})$$

where E is the sum of squared error with respect to weight vector W, the term η denotes learning rate and ζ represent momentum. The learning strategy employs on-line method during epochs. Fig. 11 gives the description for the acceleration and brake signal modified with the linear regression analysis and artificial neural network.

4. Experimental results and analysis

4.1. *Track categorization*

In the TORCS, there are nineteen tracks that have different properties. For each track, we did the exhaustive parameter search on α and β (Table 3). The parameters are chosen from {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0}. It yields different

```
Procedure Get_Accel_Brake_LR_ANN ( ){
 If ( $track[9] \geq 150$ ) {  // in straight track
    $Desired\_Speed$ = MAXSPEED;
 }
 Else if ($70 < track[9]$ && $track[9] < 150$){ // in SAZ
   $Prob$ = ANN(Sensors);
   $Desired\_Speed = (\alpha - Prob) \times (MAXSPEED - speedX) \times |Tmax - track[9]| / |Tmin - track[9]|$;
 }
 Else {  // in corner track
   $Desired\_Speed = w_1\, track[0] + \cdots + w_{19}\, track[18] + w_{20}\, speedX + w_{21}\, speedY + w_{22}\, speedZ$ ;
 }

 Return 2/(1+exp($speedX$ -  $Desired\_Speed$ ))-1;
}
```

Fig. 11. A controller with linear regression and artificial neural networks.

Table 3
The optimal parameters for each track (the best lap time with zero damage).

Track name	Length	α	β	Lap time	Average speed	Group
Alpine2	**3772.91**	**2.0**	**5.0**	**116.06**	**30.41**	**A**
ETrack2	3147.41	5.0	3.5	88.58	35.53	A
Alpine1	6390.86	3.5	5.0	147.17	39.72	A
Street1	3822.60	2.0	3.0	94.83	40.31	A
E Road	**3383.50**	**2.0**	**3.5**	**82.45**	**41.04**	**B**
Ruud	3352.55	2.5	3.5	78.90	42.49	B
ETrack6	4441.24	4.5	3.0	102.22	43.45	B
CGTrack1	2056.97	3.0	3.5	47.27	43.52	B
ETrack3	**4799.26**	**1.0**	**3.0**	**106.78**	**44.95**	**C**
Wheel1	4257.60	4.0	3.5	92.74	45.91	C
Wheel2	6205.44	3.0	3.5	134.77	46.05	C
CGTrack2	**3184.97**	**3.5**	**3.5**	**63.65**	**50.04**	**D**
E-Speed	3049.78	3.5	3.5	59.05	51.65	D
Forza	5784.09	3.0	3.5	106.58	54.27	D
A-Speed way	1917.04	5.0	4.5	35.17	54.51	D
ETrack4	**7041.56**	**5.0**	**3.5**	**122.53**	**57.47**	**E**
C-Speed	3293.98	3.5	3.5	49.29	66.83	E
D-Speed	3427.40	3.5	3.5	50.73	67.57	E
B-Speed way	3999.00	3.5	3.5	57.89	69.08	E

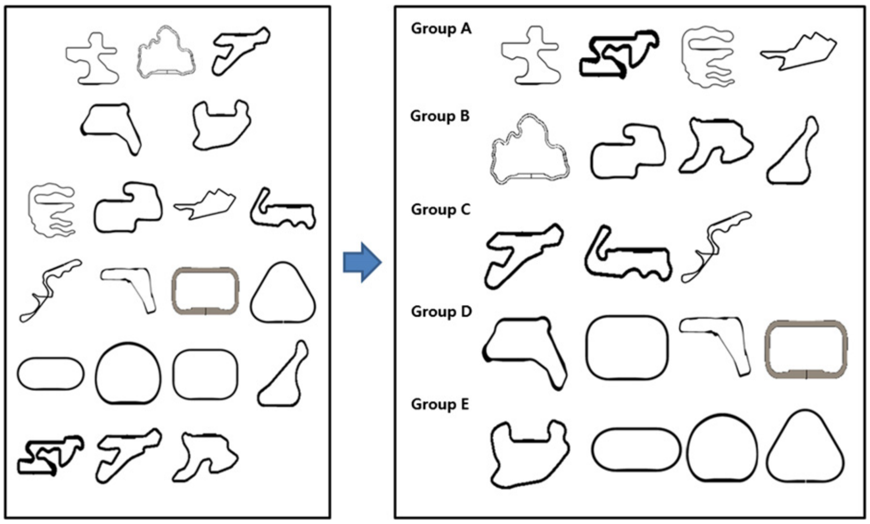


Fig. 12. Categorization of tracks based on their difficulty.

parameter settings, each of which is optimized for the track trained on. In the table, the tracks are sorted by the average speed in ascending order. If the average speed is low, it means that the track is too difficult to drive. They are categorized into five groups based on the difficulty of driving. If a track is grouped into A, it is one of the most difficult tracks in the TORCS. Fig. 12 shows the tracks grouped into five categories.

Figs. 13 and 14 shows training and test tracks respectively.

4.2. Learning approach

The next step is to learn the generalized controller from the observed behavior of the optimized controllers. From the group A,

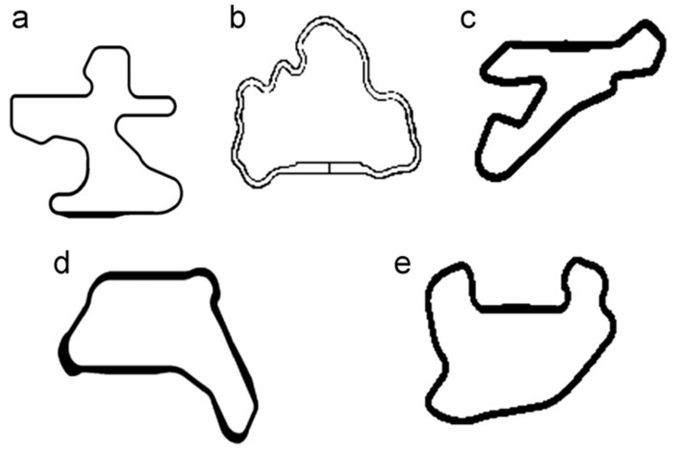


Fig. 13. Optimized controllers for the tracks are used to train machine learning models. (a) Alpine2. (b) ERoad. (c) ETrack3. (d) CGTrack2. (e) ETrack4.

Table 4
The number of training samples collected from the optimized controller on its track.

Tracks	The number of training samples	Percentage
Alpine2	6363	25.8
ERoad	4041	16.4
ETrack3	5156	20.9
CGTrack2	3243	13.1
ETrack4	5863	23.8
Total	24,666	

Table 5
The results of feature selection algorithm.

	Sensors
All	Track[0]~Track[18], speedX, speedY, speedZ
8	Track[1], Track[7], Track[8], Track[9], Track[10], Track[11], Track[17], speedX
6	Track[1], Track[8], Track[9], Track[10], Track[17], speedX
4	Track[1], Track[9], Track[17], speedX

Table 6
False positive rate, negative predictive value, and false discovery rate for different number of features.

The number of features	FPR	NPV	FDR
22	0.338578	0.714723	0.056823
8	0.414412	0.630847	0.069583
6	0.440529	0.604762	0.073929
4	**0.263059**	**0.560421**	**0.047237**

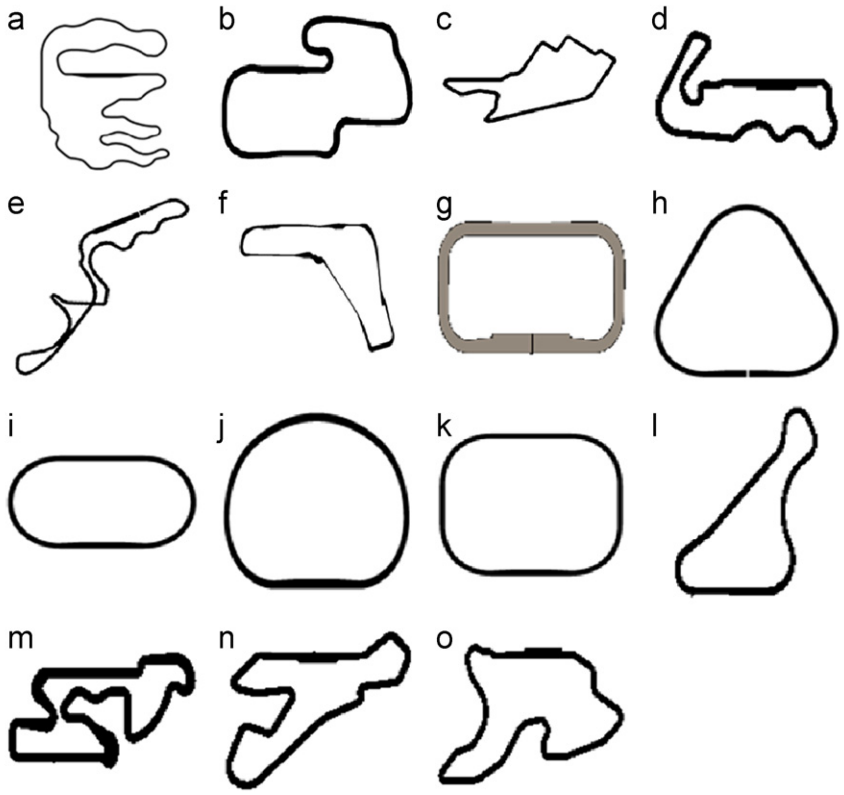


Fig. 14. Fourteen tracks for evaluation. (a) Alpine1. (b) Ruudskogen. (c) Street1. (d) Wheel1. (e) Wheel2. (f) Forza. (g) A Speed Way 1. (h) B Speed Way 1. (i) C Speed Way 1. (j) D Speed Way 1. (k) E Speed Way 1. (l) CG Track1. (m) E Track 2. (n) E Track 3. (o) E Track 6.

Table 7
Summary of results (LR=Linear Regression, NN=Neural Network).

(a) The performance of controllers optimized on one track

Track name	Group	Sum of lap time on the fourteen tracks	Sum of damage on the fourteen tracks
Alpine2	A	1236.19	53
Eroad	B	1395.164	77
ETrack3	C	1236.19	53
CGTrack2	D	1621.086	1996
ETrack4	E	1818.028	1784

(b) The performance of controllers generalized from the five controllers

	Average controller		Average controller+NN		Average controller+LR		Average controller+NN+LR	
	Lap time	Damage	Lap time	Damage	Lap time	Damage	Lap time	Damage
Alpine1	171.42	0	171.44	0	156.62	0	162.51	0
CGTrack1	48.71	0	48.71	0	47.36	3	47.77	3
A-Speed Way	38.90	5	38.84	4	36.97	0	37.4	0
B-Speed Way	58.28	11	58.28	11	58.28	11	58.28	11
C-Speed	49.84	12	49.84	12	49.84	12	49.84	12
D-Speed	51.27	9	51.27	9	51.27	9	51.27	9
E-Speed	59.59	13	59.59	13	59.59	13	59.59	13
ETrack2	94.65	0	94.66	0	89.87	0	90.59	0
ETrack6	106.56	0	103.45	0	110.06	74	106.51	0
Forza	110.18	0	110.17	0	112.01	3	111.37	0
Ruud	81.90	0	82.47	0	80.02	0	79.73	0
Street1	117.05	2602	98.39	0	101.45	407	99.23	42
Wheel1	95.84	0	95.78	0	92.51	111	93.34	0
Wheel2	139.40	0	139.13	0	135.00	3	136.71	0
Sum	1223.59	2652	1202.02	49	1180.85	646	1184.14	90

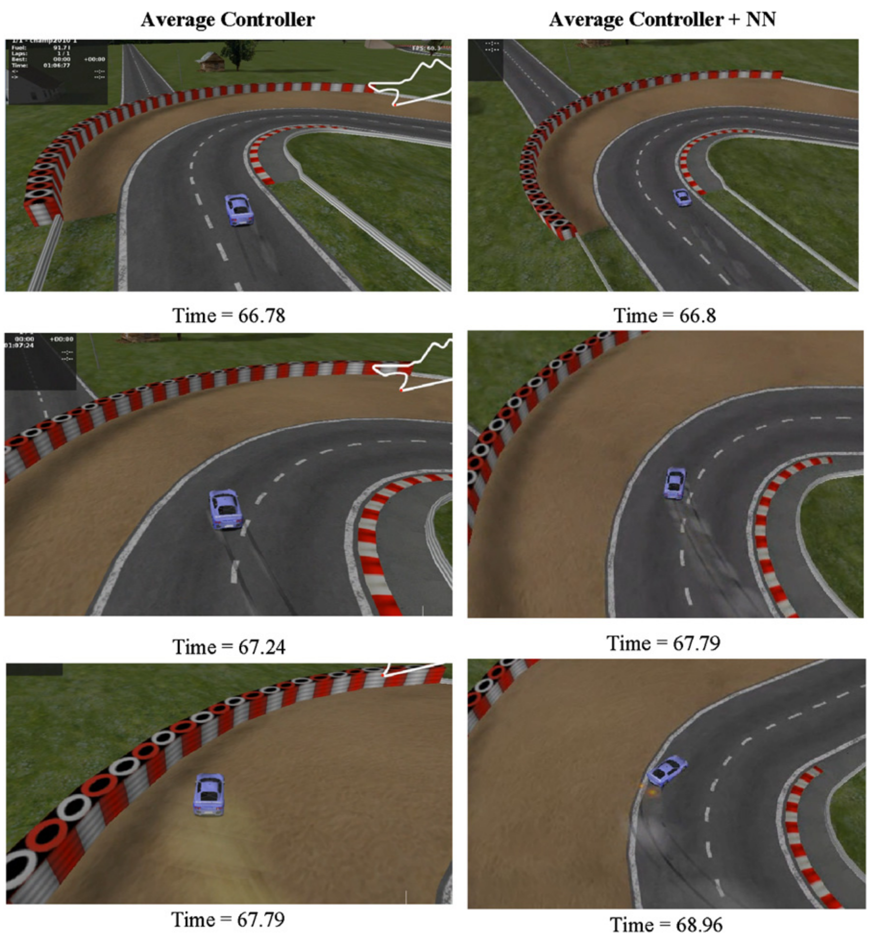


Fig. 15. Snapshots show the difference of behaviors by the average controller and plus NN (Street 1 track).

the controller ($\alpha=2.0$ and $\beta=5.0$) optimized for Alpine2 was chosen as a demonstrator because the track is the most difficult one in the group. ERoad, ETrack3, CGTrack2, and ETrack4 became the demonstrator because they are the most difficult one from each group.

Fig. 13 shows the five tracks selected. In this way, it is possible to get the mix of training samples from controllers optimized for tracks with different difficulty. Each controller drove on the track where it had optimized while it generated training samples on sensory input and target variables.

Table 4 summarizes the number of training samples from each controller. Because each track has different length, the number of samples from the track is not the same. However, they used the same sampling rate (50 samples/s). From the training samples, we ran the feature selection algorithm based on Pearson correlation analysis (Table 5). At first, we applied linear regression to predict the desired speed. For the analysis, WEKA software toolkit was used [22]. In the learning, we used a least median squared linear regression. The best linear equation is as follows and it uses only six features:

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

The number of input neuron was the same with the number of features. It had one hidden layer and the number of hidden neurons was equal to the number of input neurons. It was trained using standard back-propagation algorithms. Learning rate was 0.1 and momentum parameter was set as 0.01. The number of epoch was 200. The best neural network used only four input features (Track [1], Track [9], Track [17], and speedX). The accuracy of the network was 87.78% on the dataset. Table 6 shows

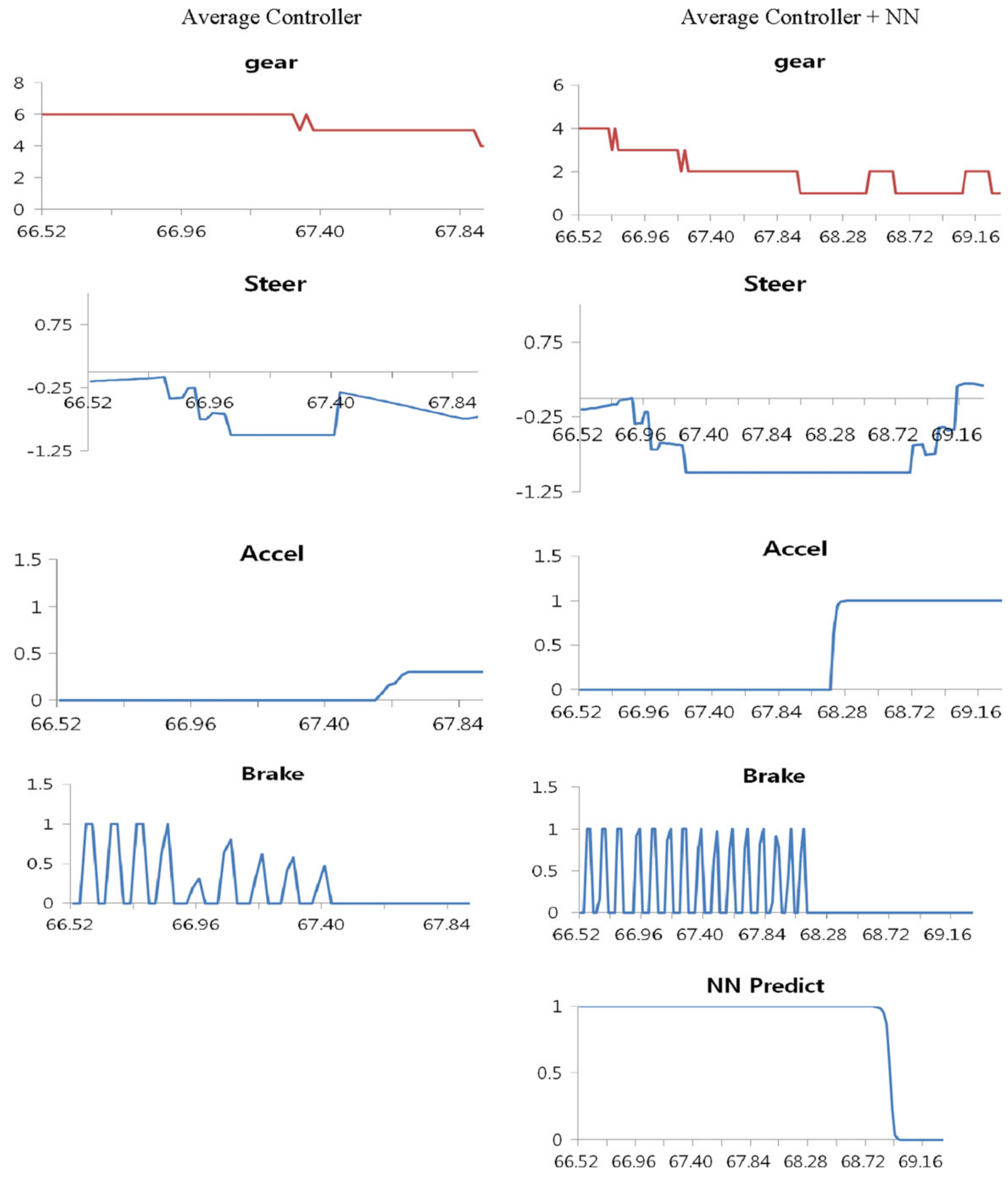


Fig. 16. The comparison of actuators' value of the two controllers for the situation depicted in Fig. 15.

that the performance of artificial neural networks for different number of features.

The performance of controllers optimized on one track was tested on the fourteen tracks in terms of the sum of lap time and damages (Table 7(a)). It shows that the controller optimized on Alpine2 (the most difficult track) performs the best among the five controllers. However, the performance was poor if the controller was optimized on the easy tracks (for example, CGTrack2 and ETrack4). If you optimize controllers with small number of tracks, it is a good strategy to use difficult one instead of easy one.

It is possible to define ‘‘average controller’’ that minimizes the sum of lap time on the five tracks. The average performance of the controller is better than any controller optimized to single track. In our work, the average controller was defined as $\alpha=4.0$ and $\beta=3.0$. In Table 7(b), it summarizes the lap time and the damage of the average controller and the proposed method on the fourteen unseen tracks. It shows that the average controller reduced the sum of lap time than the Alpine2 controller but was poor in terms of damage. The hybrid of the average controller and NN significantly reduced the damage with small improvement on the lap time. On the other hands, the combination of the average controller and the LR significantly reduced the sum of lap time but suffered from the high damage. The last one was the average controller with NN and LR. The results were promising because it reduced both the sum of the lap time and the damages.

Fig. 15 shows some snap shots of the cars controlled by the average controller and the plus neural networks on Street1 track. In this curve, the average controller failed to avoid crashes. However, the controller with the neural network reduced the speed and turned its directions safely. Fig. 16 shows the actuator's values and response from the neural network while driving the corner depicted in the Fig. 15.

5. Conclusion and future works

It is quite challenging to design controllers for simulated cars because it requires much expert knowledge on car racing. In this work, we tried to combine our heuristic controllers with two machine learning approaches. It is true that the optimized controller play well in the track trained on but there is often performance degradation on unseen tracks. In this paper, we applied the machine learning to generate the generalized controller using the training samples from several optimized controllers. In details, we used neural networks and linear regression analysis to model the common properties in the training samples. Experimental results show that the combination of two methods with our basic controller performed the best on the fourteen unseen tracks.

It is interesting to compare the generalization ability of different machine learning algorithms for this kind of problems. For example, it is interesting to use evolutionary neural networks to predict the desired speed instead of the standard neural networks. It might be interesting to use training samples from human players together with the data from the controllers.

Acknowledgments

Co-Author Kyung-Joong Kim was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0012876). Co-Author Joong Chae Na was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0007860). The authors would like to thank Mr. Jung-Hyun Lee for his help to implement the system.

References

[1] S.M. Lucas, G. Kendall, Evolutionary computation and games, IEEE Comput. Intell. Mag. 1 (1) (2006) 10–18.

[2] D. Loiacono, et al., The WCCI 2008 simulated car racing competition, in: Proceedings of the IEEE Symposium on Computational Intelligence and Games, 2008, pp. 119–126.

[3] D. Loiacono, et al., The simulated car racing championship, IEEE Trans. Comput. Intell. AI Games 2 (2010) (2009) 131–147.

[4] E. Onieva, D.A. Pelta, J. Alonso, V. Milanes, J. Perez, A modular parametric architecture for the TORCS racing engine, in: Proceedings of the IEEE Symposium on Computational Intelligence and Games, 2009, pp. 256–262.

[5] E. Onieva, J. Alonso, J. Perez, V. Milanes, T. de Pedro, Autonomous car fuzzy control modeled by iterative genetic algorithms, in: Proceedings of the IEEE International Conference on Fuzzy Systems, 2009, pp. 1615–1620.

[6] J.-H. Seo, J.-G. Park, J.-H. Lee, K.-J. Kim, Designing robust robotic car controllers based on artificial neural network, in: Proceedings of the International Conference on Convergence & Hybrid Information Technology, 2010, pp. 183–190.

[7] J. Munoz, G. Gutierrez, A. Sanchis, A human-like TORCS controller for the simulated car racing championship, in: Proceedings of the IEEE International Conference on Computational Intelligence and Games, 2010, pp. 473–480.

[8] L. Cardamone, D. Loiacono, P.L. Lanzi, Learning drivers for TORCS through imitation using supervised methods, in: Proceedings of the IEEE Symposium on Computational Intelligence and Games, 2009, pp. 148–155.

[9] J. Munoz, G. Gutierrez, A. Sanchis, Controller for TORCS created by imitation, in: Proceedings of the IEEE Symposium on Computational Intelligence and Games, 2009, pp. 271–278.

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

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

[12] L. Cardamone, D. Loiacono, P.L. Lanzi, Learning to drive in the open racing car simulator using on-line neuroevolution, IEEE Trans. Comput. Intell. AI Games 2 (3) (2010) 176–190.

[13] L. Cardamone, D. Loiacono, P.L. Lanzi, A.P. Bardelli, Searching for the optimal racing line using genetic algorithms, in: Proceedings of the IEEE International Conference on Computational Intelligence and Games, 2010, pp. 388–394.

[14] J. Quadflieg, M. Preuss, O. Kramer, G. Rudolph, Learning the track and planning ahead in a car racing controller, in: Proceedings of the IEEE International Conference on Computational Intelligence and Games, 2010, pp. 395–402.

[15] L. Cardamone, D. Loiacono, P.L. Lanzi, Evolving competitive car controllers for racing games with neuroevolution, in: Proceedings of the Annual Conference on Genetic and Evolutionary Computation, 2009, pp. 1179–1186.

[16] L. Cardamone, D. Loiacono, P.L. Lanzi, On-line neuroevolution applied to the open racing car simulator, IEEE Cong. Evol. Comput. (2009) 2622–2629.

[17] D. Perez, G. Recio, Y. Saez, P. Isasi, Evolving a fuzzy controller for a car racing competition, in: Proceedings of the IEEE Symposium on Computational Intelligence and Games, 2009, pp. 263–270.

[18] M. Ebner, T. Tiede, Evolving driving controllers using genetic programming, in: Proceedings of the IEEE Symposium on Computational Intelligence and Games, 2009, 279–286.

[19] D. Perez, Y. Saez, G. Recio, P. Isasi, Evolving a rule system controller for automatic driving in a car racing competition, in: Proceedings of the IEEE Symposium on Computational Intelligence and Games, 2008, pp. 336–342.

[20] Y. Saez, D. Perez, O. Sanjuan, P. Isasi, Driving cars by means of genetic algorithms, Lect. Notes Comput. Sci. 5199 (2008) 1101–1110.

[21] J. Togelius, The Simulated Car Racing Competition@CIG-2008: Events Reports, SIGEVOlution. 3 , 2008.

[22] WEKA, ⟨http://www.cs.waikato.ac.nz/ml/weka/⟩.

Kyung-Joong Kim received a B.S., an M.S., and a Ph.D. in Computer Science from Yonsei University in 2000, 2002, and 2007, respectively. He worked as a post-doctoral researcher in the Department of Mechanical and Aerospace Engineering at Cornell University in 2007. He is currently an assistant professor in Department of Computer Science and Engineering at Sejong University. His research interests include artificial intelligence, game, and robotics.

Jun-Ho Seo received a B.S. in Software Engineering from Sejong University in 2011. He is currently a M.S. candidate in Department of Computer Science and Engineering at Sejong University. His research interests include design and analysis of algorithms, and multi-threading.

Jung-Guk Park received the B.S. degree from Sejong university. He has studied machine learning and neural networks. His research interests are generalization and training scheme of neural networks.

Joong Chae Na received a B.S., an M.S., and a Ph.D. in Computer Science and Engineering from Seoul National University in 1998, 2000, and 2005, respectively. He worked as a visiting postdoctoral researcher in the Department of Computer Science at the University of Helsinki in 2006. He is currently an assistant professor in Department of Computer Science and Engineering at Sejong University. His research interests include design and analysis of algorithms, and bioinformatics.