Designing Robust Robotic Car Controllers based on Artificial Neural Network

Jun-Ho Seo, Jung-Guk Park, Jung-Hyun Lee, and Kyung-Joong Kim
Department of Computer Engineering, Sejong Univ
98 Gunja-Dong, Gwangjin-Gu, Seoul, Republic of Korea
kimkj@sejong.ac.kr

ABSTRACT

There are several factors to be considered in the design of controllers for simulated car racing competition: Speed, safety and adaptation. In this paper, we propose an experience-based design method for safe driving which improves its driving skills from failures. This imitates human's learning mechanisms which exploit a huge number of failures to get stable performance. In the context of simulated car racing competition, the failures mean that car has accidents or is stuck. Our system provides with methods to design controllers for safe driving by analyzing, memorizing, predicting failures from its experience. At first, our analysis shows that the accidents occur frequently when the car enters a corner with high speed. Based on this analysis, we decide to recognize the corner before entering it and reduce speed for safe turn. Secondly, our system remembers the position where the car fails to drive successfully. In the next lap, the car reduces its speed at the position by recalling memory. Although this is a useful approach to minimize failures incrementally but there is no guarantee that the failure will occur in the same place. Finally, we use a computational intelligence method (neural network) to predict the failures from sensory information. Experimental results show that the combination of the three strategies can improve the design of controllers for simulated car racing.

Keywords

Artificial Neural Network, Fault-Tolerant, Robotic Car

1. INTRODUCTION

"Learning from failures" is one of important skills for our survival [1]. It is not possible to grasp difficult techniques at one shot and we need to repeat them as many as possible resulting in failures. From the experience of failures, we can learn to avoid the failures and finally do the target behavior without errors. For example, we need a lot of failures in learning to take a bicycle, riding a car, playing the tennis, and playing the piano.

In the context of car racing competition, the failure means that the car is stuck or has accidents (Figure 1). It gives damages to the car and takes much time to escape from such situations. Even for successful controllers, there are some failures. In the course of competition, if the car makes a failure in the first lap, there is high possibility of the same mistakes in the next lap. This is not true for human because he can learn from the failures and try to avoid them. This is natural mechanism for human to deal with complex skills but our system has no idea on the failures.

There are several approaches to deal with the failures in machine learning domains. The first one is to make systems robust to the failures [3]. The fitness of controller is dependent on the robustness to the failures. This requires much computational cost to find good solutions and is available to only known faults. The second approach is to model the normal status of the system and realize the occurrence of failures from the continuous self modeling [4]. This approach can deal with unknown faults but requires much computational cost in the operation of systems.

Figure 1. The TORCS-based simulated car racing competition

We approach to develop mechanisms to handle the faults and exploit them to improve the performance of the controller for simulated car racing. Our basic driving program is not perfect and could generate several failures in the warming-up stage newly proposed in the 2010 competitions. Our idea is to use the information to avoid the same mistakes in the next lap. Simple method is to remember the position of failure and reduce the speed in the place in the next lap. Also, it is possible to learn generalized models to predict the situation where the failure occurs. This is similar to the data-driven approach to drive the car [5]. It has been known that the bumping to the wall occurs frequently in the corner with high speed driving. Based on those facts, we defined speed adaptation zone before the corner to minimize the bumping. Figure 2 summarizes the proposed approaches to deal with failures for the simulated car racing.

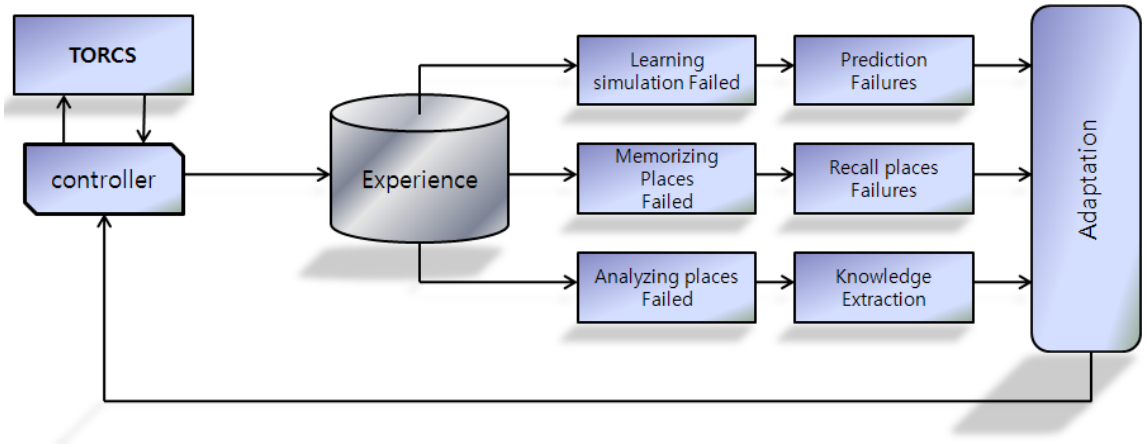


Figure 2. The overview of the proposed method (The controller continuously stores the sensory information from the environments (called as experience) and failures. Our system analyzes the failure history and extracts knowledge from them. For example, our system reveals that the accidents frequently occur in the corners with high speed and we decide to reduce speed before the corner. The next approach is to memorize the place where the accidents occur and recall this knowledge to reduce speed at the same position in the next lap. Finally, we used artificial neural networks predict the possibility of accidents based on sensory information. From the outputs of the three strategies, we finally change the behavior of controllers and adapt to the failures.

COBOSTAR, the winner of the CEC2009 competition proposed a similar strategy to remember the car crashing point and reduce speed 100m before this location [10]. Compared to the COBOSTAR, we applied two other approaches to deal with the car crashing appropriately and proposed to use adaptive speed adaptation strategy instead of ad-hoc passive strategy. Also, we adopted several new features of 2010 competition.

2. RELATED WORKS

Cardamone *et al*. developed a new sensor called look ahead which returns the bended angle of track segments [5]. Using the high-level information, they can generate useful sensory datasets from successful controllers. They can predict the next action of the car drivers using k-nearest neighbor and artificial neural network from the sensory inputs. They reported that this approach can generate successful controller although previous works failed because of low-level sensory inputs.

Ebner *et al.* used genetic programming to evolve two symbolic expressions which control the steering angle and acceleration/deceleration of the car [6]. Genetic programming is well-known techniques to evolve programs and provides with interpretable equations. The evolved equations can give insight to the designers who develop controllers for simulated car racing because they're represented with human understandable symbols.

Munoz *et al*. trained the neural controllers from the datasets generated by two well-known controllers and human [7]. They reported that it is very complicated to learn the human behavior because neither the human makes the same action in the same circumstances, nor performs all actions in the proper way and the human makes a lot of mistakes that have to resolve. This is interesting that human plays the game not perfectly while doing several mistakes and changing his behaviors over time.

Perez *et al.* evolved fuzzy rules controlling the cars in order to optimize lap times, damage taken and out of track time. They optimized fuzzy rules and fuzzy sets for inputs and outputs of controllers using genetic algorithm. They reported that the use of fuzzy sets for input sensors are quite useful.

Onieva *et al*. is the winner of the 2009 simulated car racing championship [9]. They are classified as "mainly programmed category." They adopted a complete modular architecture that each module is in charge of managing one basic aspect of the driving (Gear, desired speed, low-level gas & brake, steering, and opponent modifier). A simple TSK fuzzy controller is in charge to determine the target speed. In their work, many of the features/parameters are hand-tuned.

Butz *et al*. developed COBOSTAR which won the CEC-2009 competitions. COBOSTAR adopts several strategies to deal with failures [10]. For example, they tried to develop different strategies for off-track situation. In the off-track, distance sensors to the track borders are not available, steering becomes much more error-prone and wheel slippage is much stronger.

They also added a crash-point strategy, remembering crash points on the road. If a crash occurred in the first lap somewhere and if furthermore this crash appeared not to be caused by an opponent, then the relative point on the track with respect to the start was remembered and 100 meters before this location the strategy was converted into a passive strategy, leading to a much slower driving style. While this strategy seemed to have big potential, the definition of the passive strategy was very ad-hoc- the car simply drove very slow around the crash area.

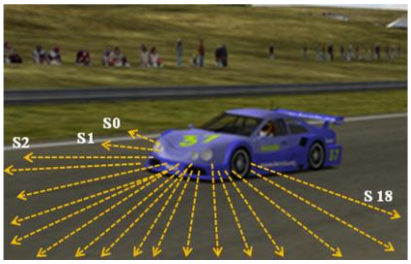
The idea proposed by COBOSTAR is quite similar to one of our components called "memorizing failures." However, we tried to define the passive strategy adaptively considering the width of tracks. If the track is wide, the car reduces the speed as small as possible, but in case of narrow track, the car significantly reduces its speed. Also, we have two other strategies to deal with crashes. In the first strategy, we defined speed adaptation zone before

corners. In that area, the car adaptively changes the speed to the target speed which is proportionate to the width of tracks. In the second strategy, the car predicts the probability of car crash using multi-layered neural network and exploit it to change its speed.

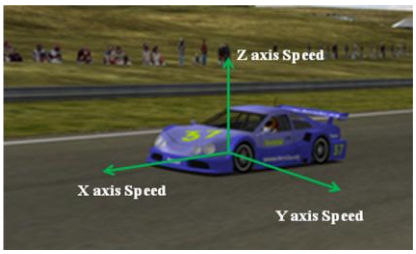
In this section, we summarized the papers published in IEEE Symposium on Computational Intelligence in Games 2009. It is a mixture of hand-tuned programs and completely learned programs without expert knowledge. Interestingly, there are two papers about imitating other player's driving skills based on the behavior of others. Unlike the two papers, in our work, we record the behavior of our program to improve its ability to avoid car crashes.

3. PROPOSED METHODS

There are several changes in the rules of 2010 simulated car racing competitions. 1) The range of the proximity sensors have been increased from 100m to 200m. 2) The position of the range finders can be customized by users. 3) New focus sensor provides accurate sensing of the road ahead. 4) Noise is introduced to range and proximity sensors. 5) Drivers now control the clutch. 6) A new warm-up stage allows drivers to learn about track properties before the qualifying. In this work, we adopted our controller to the new rules.



(a) 19 Distance Sensors



(b) 3-dimensinal speed of the car

Figure 3. The basic sensors of simulated cars

Table 1 and Figure 3 show the basic sensors from TORCS engine [13]. There are 19 sensors and designers can configure the position of the sensors.

Table 2 summarizes the configuration of the 19 track sensors. Sensor 0 and Sensor 18 are used to calculate the width of the track. Sensor 1, 9, and 17 are located in the center of the car. The steer weight was hand-tuned and used in a steering algorithm (Algorithm 1: Steering). It improves the cornering behavior of the car.

3.1 Basic Driving Module

this module, there are several components charging one aspect of the driving. This is similar to the modular architecture proposed by [9]. Figure 4 shows the architecture of the basic driving module. Car state means the sensory input to the controller and car control is a set of control signals to the car. The failure handling module changes only the speed of the car by subsumes the outputs of the acceleration and brake module.

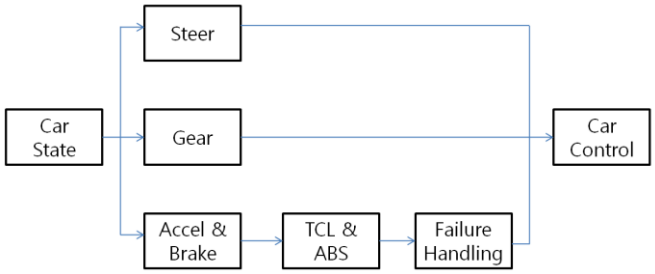


Figure 4. An overview of the basic driving module

Table 1. Description of the available sensors

Name	Range(unit)	Description
angle	[-π, + π] (rad)	Angle between the car direction and the direction of the track axis
Track	[0, 200] (m)	Vector of 19 range finder sensors: each sensor returns the distance between the track edge and the car within a range of 200 meters.
trackPos	(-∞, +∞)	Distance between the car and the track axis. The value is normalized w.r.t to the track width: it is 0 when car is on the axis, -1 when the car is on the right edge of the track and +1 when it is on the left edge of the car. Values greater than 1 or smaller than -1 mean that the car is outside of the track.
speedX	(-∞, ∞)(km/h)	Speed of the car along the longitudinal axis of the car.
distFromStart	[0, +∞)(m)	Distance of the car from the start line along the track line.

Table 2. The configuration of 19 track sensors and their steer weights

Sensor Number	Angle	Steer weight	Sensor Number	Angle	Steer weight
0	-90	-	10	5	-0.25
1	-1	-	11	10	-0.2
2	-45	1.3	12	15	-0.55

3	-30	0.8	13	20	-0.4
4	-25	0.9	14	25	-0.9
5	-20	0.4	15	30	-0.8
6	-15	0.55	16	45	-1.3
7	-10	0.2	17	1	-
8	-5	0.25	18	90	-
9	0(center)	0			

3.1.1 Steering

The basic idea of steering is to go to the direction of track sensors with the maximum distance. The sensor located in the center of the car is used to classify whether the car is in the straight or corner tracks. In Algorithm1 "Steer_weight[Index]" values correspond with table2 Steer weight.

Algorithm 1: Steering

```
// k: 0.3 (usually 0.1~0.5)
// steerLock = PI/4
Procedure Get_Target_Steer(){
  Index = argmax(Track[2], Track[3], …, Track[16]);
  If(Track[9] > 70 meters) {  // Car is in the straight track
    Target = 0.1 x (angle – trackPos)/steerLock;
  }
  Else { // Car is in the corner track
    Target = Steer_weight [Index]-k x trackPos;  }
  Return Target; }
```

3.1.2 Acceleration & Brake

If the value of the track sensor in the center of the car is larger than 150 meters, it is classified as a straight road. If not, it is a corner area. We have changed the parameters of TCL & ABS used in the source code provided by organizer as an example.

Algorithm 2: Acceleration and Brake

```
// β: 3
Procedure Get_Accel_Brake_Basic(){
  T[1] = Track[1];
  T[2] = Track[17];
  If(T[1] > T[2]) { max = 1; min = 2;}
  Else {max = 2; min = 1;}

  If(Track[9]>150 meters) {
    Target = MAXSPEED; // Straight Track
  }
  Else {
    Target = β x Track[9]  x |T[max]-Track[9]|/|T[min]-Track[9]|;
  }
  Return 2/(1+exp(speedX-Target))-1 }
```

3.2 Memorizing Failures

When the car is stuck, the current position of the car is memorized in the list structure. distFromStart sensor was used to indicate the failure position. If the position is labeled as *x*, speed reducing area is between *x*-300 meters and *x*+20 meters. If the car is in the area, the car has to adjust the speed to target speed(15 x Track_Width). In this case, the definition of stuck is as follows. If the car's angle is larger than 45 degree, we define the situation as stuck and memorize the position.

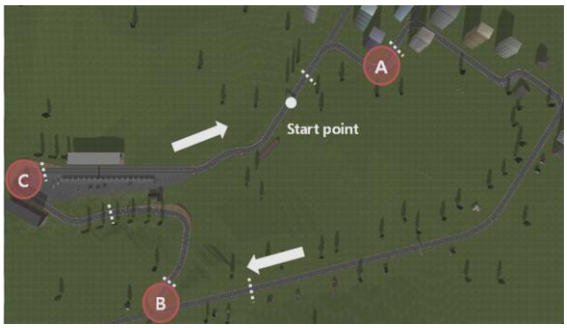


Figure 5. An example of the stuck list for memorizing failures.

Algorithm 3: Memorizing Failures

```
Procedure MEMORIZE_FAILURE(){
If(angle > 45){  // Car is stuck
     If(ISADDEDSTUCK(distFromStart)==FALSE)
       INSERTSTUCK(distFromStart);
}
}
Procedure ISADDEDSTUCK(distFromStart){
    Iterator it = stucklist.begin;
    While(it!=stucklist.end){
       If(it→front < distFromStart && it→end > distFromStart)
          Return TRUE;
       it=it→ next;
    }
    Return FALSE;
}
Procedure INSERTSTUCK(distFromStart){
   Stuckpoint sp;
   sp.front = distFromStart – 300;
   sp.end = distFromStart +20;
   Insert sp to stucklist;
}
// If the car is near the memorized position again, reduce speed
Procedure Get_Accel_Brake_S1(){
   If(ISADDEDSTUCK(distFromStart)==TRUE){
      If(speedX>15 x Track_Width) Return -1; // Reduce Speed
}}
```

3.3 Speed Adaptation Zone (SAZ)

From our initial analysis, we realized that the car accidents frequently occur before corners with high speed entering. Based on this analysis, we decide to set speed adaptation zone before the corner to adjust the speed. This makes the car reduce the speed before the corner. If the speed is lower than target speed in the zone, the car increases its speed to the goal. The target speed is proportionate to the width of the track.

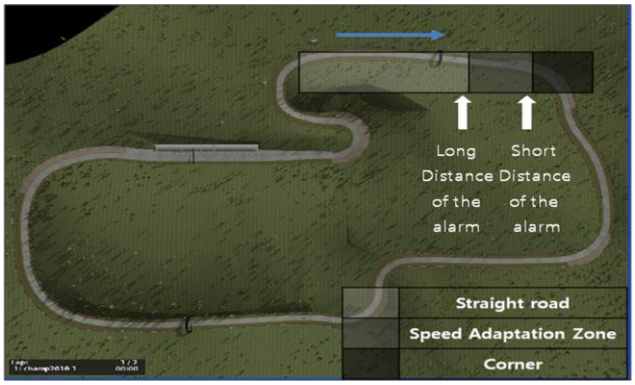


Figure 6. Speed adaptation zone before the corner (the area between two vertical arrows)

Algorithm 4: Acceleration and Brake with SAZ

```
Procedure Get_Accel_Brake_S2 (){
  T[1] = Track[1];
  T[2] = Track[17];
  If(T[1] > T[2]) { max = 1; min = 2;}
  Else {max = 2; min = 1;}

  If(Track[9]>150 meters) {
    Target = MAXSPEED; // Straight Track
  }
  Else if (Track[9]<150 && Track[9]>70){ // car is in the SAZ
    Target = 20 x Track_Width;
  }
  Return 2/(1+exp(speedX-Target))-1
}
```

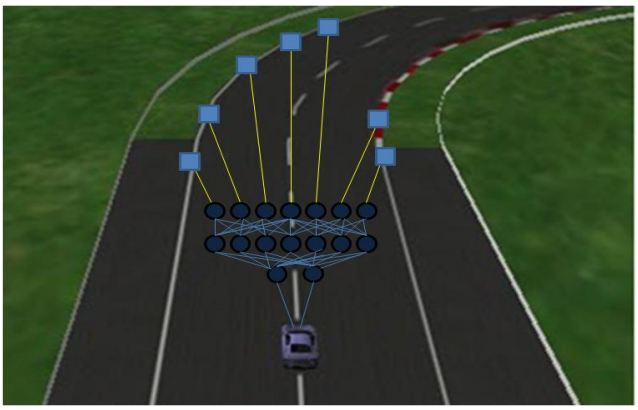
3.4 Train Multi-Layer Neural Networks to Predict Failures

In our early work, we tried to learn neural networks to predict whether the car is stuck or not. In fact, this was not working well and we changed the goal of neural networks to predict the degree of danger of the future tracks. If there is sharp corners, the neural networks has to return the value near 1. If the track is straight line, it has to return the value near 0. This information can be used to measure the level of danger and is closely related to the probability of actual failures. To learn this neural network, we collected data from several tracks by using the basic driving modules. Initially, we used all 19 track sensors as inputs for neural networks but it worked poorly. Instead of using all sensors, we selected the most informative eight sensors manually (speedX, Track[1], Track[17], Track[7]~Track[11]). It significantly improved the performance of the neural networks. The dataset includes the value of the eight sensors and the label (the corner is sharp or not).

Algorithm 5: Acceleration and Brake with MLP

```
// β: 3
Procedure Get_Accel_Brake_S3 (){
  T[1] = Track[1];
  T[2] = Track[17];
  Prob = MLP(speedX, Track[1], Track[17], Track[7]~Track[11]);
  If(T[1] > T[2]) { max = 1; min = 2;}
  Else {max = 2; min = 1;}

  If(Track[9]>150 meters) {
    Target = MAXSPEED; // Straight Track
  }
  Else if (Track[9]<150 && Track[9]>70){ // car is in the SAZ
    Target = (β-Prob) x Track[9]  x | T[max]-Track[9] |
                                  /| T[min]-Track[9] |;
  }
  Return 2/(1+exp(speedX-Target))-1
}
```



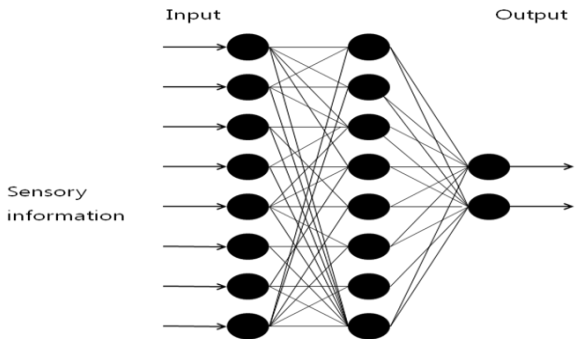


Figure 7. The architecture of neural network and the use of MLP to predict the degree of danger (sigmoid function is used)

3.5 The Integration of the Modules

Because we use multiple strategies to handle failures, we need an integrated algorithm to determine the final acceleration/brake value. If there is no dangerous factor, the output from the basic module is returned. If there are several (more than one) outputs, the one with small value is used. For example, the car is in the speed adaptation zone and also near the failure position memorized. In this case, the target speed is determined as one with low value. The car has to reduce the speed to the lowest target speed if the car satisfies multiple failure conditions.

Algorithm 6: Integrated Algorithm

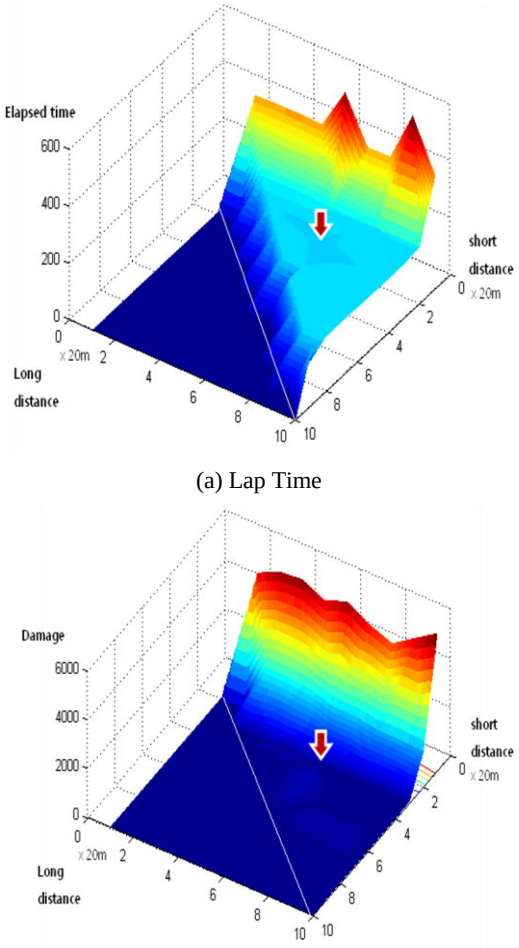
S1=MAX_SPEED;
S2=MAX_SPEED;
S3=MAX_SPEED;
If(Car is near the Failure Position Memorized){
  S1 = Get_Accel_Brake_S1(); // Memorized Position
}
If (Car is in SAZ){
  S2 = Get_Accel_Brake_S2(); // SAZ
  S3 = Get_Accel_Brake_S3(); // MLP
}
If(Car is not near the Failure Position && Car is not in SAZ){
  S = Get_Accel_Brake_Basic();
  Return S;
}
Else { Return Min(S1, S2, S3); }

4. EXPERIMENTAL RESULTS

For neural network training, we used backpropagation algorithm [11]. The training samples were collected from six tracks (Street1, Ruddskogen, Wheel1, Wheel2, E-track3, and Cgtrack2). We collected the data by running the car with the basic driving module for two or three laps for each track. It was sampled per 0.02 seconds. The inputs to the neural networks were normalized between 0 and 1. To set the parameters of the backpropagation algorithm, we tested several learning rate parameters (Figure 10). Based on this test, we set the rate as 0.1. The training accuracy of the neural network is 95% with full datasets. In the SAZ, we defined the area between 70 meters and 150 meters before corner. This was determined from exhaustive tests of several combinations of the two values (start and end position of the area) (Figure 8).

Table 3. The number of samples collected from the tracks

Track Name	# of Samples
Street1	13521
Ruddskogen	9433
Wheel1	5983
Wheel2	7454
E-Track3	13188
Cgtrack2	8833
Total	58412



(a) Lap Time

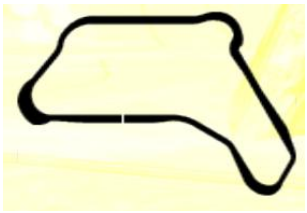
(b) Damage

Figure 8. The exhaustive test of the combination of start and end position of speed adaptation zone. (Long distance means the start position and the short distance means the end position) (Long distance must be larger than the short distance) (The arrow indicates where minimizes the time and damages (between 70 meters and 150 meters).

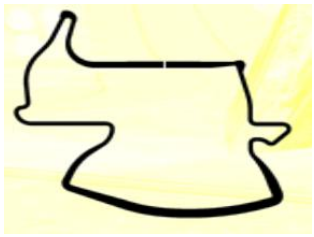
We evaluated the lap time and the damage for several combination of the strategies used. We used three different tracks (Street1, CGTrack2, and Aalborg) (Figure 9). We repeated the experiments for five times(each time runs 5laps). It showed that the combination of the all strategies (B+S1+S2+S3) shows minimum lap time with zero damage. S2 is quite useful to minimize the damage of driving but slow down the lap time of the car. The introduction of S3 improves the lap time by increaing the prediction ability of the danger level of corners.

B : Basic driving module
S1 : Memorizing failures
S2 : Speed adaptation zone
S3 : Predicting the degree of danger using MLP

Street1



CGTRACK2



Aalborg

Figure 9. Tracks used to evaluate the strategies used

Table 4. The summary of experimental results

Map	repeat	Basic		B+S1		B+S1+S2		B+S1+S2+S3	
		lap time	damage	lap time	damage	lap time	damage	lap time	damage
Street1	1	480.54	2600	464.1	765	477.9	0	461.79	0
	2	480.54	2600	464.1	765	477.9	0	461.79	0
	3	480.54	2600	464.1	765	477.9	0	461.79	0
	4	480.54	2600	464.1	765	477.9	0	461.79	0
	5	480.54	2600	464.1	765	477.9	0	461.79	0
CGTrack2	1	310.75	16	310.75	16	338.36	0	320.91	0
	2	310.75	16	310.75	16	338.36	0	320.91	0
	3	326.38	20	310.75	16	338.36	0	320.91	0
	4	310.75	16	310.75	16	338.36	0	320.91	0
	5	310.75	16	310.75	16	338.36	0	320.91	0
Aalborg	1	446.11	2582	446.11	2582	471.51	0	451.65	0
	2	446.11	2582	446.11	2582	471.51	0	451.65	0
	3	459.34	3275	446.11	2582	471.51	0	451.65	0
	4	446.11	2582	446.11	2582	471.51	0	451.65	0
	5	446.11	2582	446.11	2582	471.51	0	451.65	0

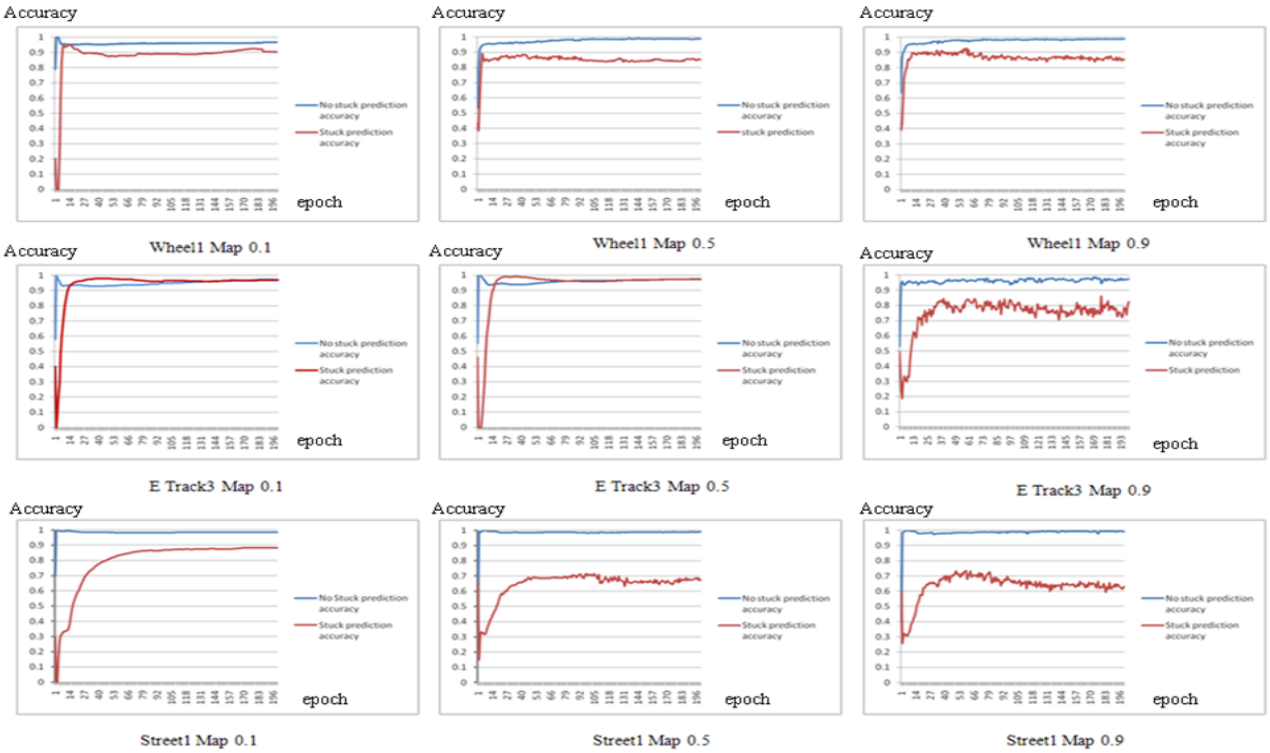


Figure 10. The accuracy of MLP by changing the learning rate of BP (0.1, 0.5 and 0.9) for three tracks.

5. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a method to handle with the failure in the simulated car racing. Our controller has basic driving module and several strategies to deal with failures: 1) Memorizing the position where the car is stuck 2) Reduce speed in the speed adaptation zone before corner 3) Predict the level of danger of corners using multi-layer neural networks. It shows that the combination of basic driving module and the three strategies performs well by minimizing damages and lap times.

It is interesting to makes maps of the track while driving cars like robotics algorithm called SLAM (Simultaneously Localization and Mapping) [12]. There are several parameters in our algorithm, we need to optimize them using machine learning approaches. Currently there are several parameters hand-tuned.

6. ACKNOWLEDGMENTS

This research 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)

7. REFERENCES

[1] A. Carmeli, "Social capital, psychological safety and learning behaviors from failure in organizations," *Long Range Planning*, vol. 40, pp. 30-44, 2007.

[2] http://cig.dei.polimi.it/?page_id=134

[3] K.-J. Kim, A. Wang, and H. Lipson, "Automated synthesis of resilient and tamper-evident analog circuits without a single point of failure," *Genetic Programming and Evolvable Machines*, vol. 11, no. 1, pp. 35-59, 2010.

[4] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, pp. 1118-1121, 2006.

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

[6] M. Ebner, and T. Tiede, "Evolving driving controllers using genetic programming," *IEEE Symposium on Computational Intelligence in Games*, pp. 279-286, 2009.

[7] J. Munoz, G. Gutierrez, and A. Sanchis, "Controller for TORCS created by imitation," *IEEE Symposium on Computational Intelligence in Games*, pp. 271-278, 2009.

[8] D. Perez, G. Recio, Y. Saez and P. Isasi, "Evolving a fuzzy controller for a car racing competition," *IEEE Symposium on Computational Intelligence in Games*, pp. 263-270, 2009.

[9] E. Onieva, D. A. Pelta, J. Alonso, V. Milanes, and J. Perez, "A modular parametric architecture for the TORCS racing engine," *IEEE Symposium on Computational Intelligence in Games*, pp. 256-262, 2009.

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

[11] B. D. Ripley, *Pattern Recognition and Neural Networks*, 1996.

[12] H. Durrant-Whyte, and T. Bailey, "Simultaneous localization and mapping (SLAM): Part I the essential algorithms," *Robotics and Automation Magazine*, vol. 13, pp. 99-110, 2006.

[13] TORCS software manual http://sourceforge.net/projects/cig/files/Championship%202010%20Manual/1.0/manual.pdf/download