Thesis proposal

"The Gazelle Adaptive Racing Car Pilot"

By
Kholah Albelihi
Department of Computer and Information Sciences
Indiana University South Bend
E-mail Address: kalbelih@iusb.edu

Date: October 5, 2013

Advisor
Dr. Dana Vrajitoru, Ph.D.
Department of Computer and Information Sciences

Committee
Dr. Yi Cheng, Ph.D.
Department of Mathematical Sciences

Dr. Hossein Hakimzadeh, Ph.D.
Department of Computer and Information Sciences

Table of Contents

1. Introduction

In this thesis we propose to conduct a study on various methods that can be applied for successfully driving a car in a simulated environment in the presence of opponents.

1.1. The Importance of Autonomous Cars

Nowadays, the interest in developing autonomous vehicles increases day by day with the purpose of achieving high levels of safety, performance, sustainability, and enjoyment. Driverless cars are ideal to use in crowded areas, on highways, and because they ease the flow of the cars. The autonomous cars also reduce the opportunity of occurring accidents which are usually caused by an oncoming car or people who are crossing the street while the drivers don't pay attention to their presence.

There are many research centers founded around the world for developing smart systems for driverless cars. These automotive research centers are supported by the leading automobile companies and universities such as the Center for Automotive Research at Stanford University (CARS) [13]. CARS has a network of more than 80 affiliated industry partners like Ford Motor Company, General Motors, BMW of North America, Mercedes-Benz Research & Development North America, Allstate Roadside Services...etc. [13]. The CARS center brings together industrial interests and academia by attracting the researchers who have the passion to work on the automotive research which is supported by the affiliated industry partners.

As an attempt to simulate autonomous cars, the simulated car racing competitions have arisen recently. This category of computer games involves computational and artificial intelligence [7]. The importance of such competitions comes from the fact that they are a perfect environment for testing the application of autonomous driving techniques [7]. Thus, simulated car racing competitions offer a structure to "test learning, adaptability, evolution and reasoning features of algorithms under investigation" [6]. The simulation offers a realistic platform for car racing in real time.

In this proposal we present an adaptive racing car controller developed within TORCS (The Open Racing Car Simulator) [4]. The TORCS system visualizes racing cars with complex graphics based on physics principles. The program offers a server which implements the race combining multiple cars, and the setup for the user to develop a client is a module that can be written by the user [2] supplying the actions of an individual car. The client module that we developed for this thesis is called Gazelle. We submitted the Gazelle controller to the TORCS completion is organized by the Genetic and Evolutionary Computation Conference in 2013.

1.2. The TORCS Simulator Environment

The TORCS (The Open Racing Car Simulator) is a popular car racing simulator written in C++ [6]. TORCS is commonly used for academic purposes, because it is similar to the commercial car racing games, and it is considered to be a fully customized environment [6]. It has a powerful physics engine and a 3D graphics engine; together they enable visualizing the car racing uninterruptedly in real time [6]. It also provides the capability to develop and build new controllers for cars. The TORCS attracts a wide community of developers and users, and it is the

platform for popular competitions which are organized every year as a part of various international conferences [2].

In this environment, each car is controlled by a controller. The controller can access the current state of the car in the race, consisting of information about the track, the car, and the opponents [7]. Based on this information the controller makes decisions to modify the following control units:

- the steering wheel with values in the range [-1, +1] for a change in direction: -1 corresponds to -45^{o} while +1 to 45^{o};
- the gas pedal [0, +1] for accelerating; 0 corresponds to losing the speed;
- the brake pedal [0, +1] for decelerating;
- the gearbox with possible values in the set {-1,0,1,2,3,4,5,6} for choosing the gear.[2]

The system works in a client-server model. The race application is a server, while each car controller is a client exchanging information with it.

The remainder of the proposal is organized in the following way. In chapter 2 we cite a few previous papers, works, and other materials that are relevant to our controller. In chapter 3 we discuss the procedural methods and the learning methods that we have already used and will develop further to improve the driver algorithm that we started from. While chapter 4 discusses the experimentation methodology that we use to evaluate the controller's performance.

2. Literature Review

The work in this thesis is based on the EPIC controller as presented by Guse and Vrajitoru in [2]. This paper presents the EPIC controller, which is the previous version of the Gazelle code. EPIC was submitted to the GECCO 2009 competition [9]. In this competition, cars driven by code submitted by the competitors run against each other in a race. Beside the car status, the controllers are provided information about the angle with the track's center line, the free distance ahead within 45 degrees of the car direction, and information about close opponents. The paper describes a car driver based on two components: determining the target angle for turning in each frame, and determining the target speed in the next frame. The controller calculates the target angle based on the target direction in an efficient way. It also provides a sharp turn detecting system which allows adjusting the target speed for an approaching sharp turn to keep the car inside the track. The system also adjusts the target angle if it determines that it might lead the car out of the track [2]. EPIC depends on the principle of calculating the free available distance ahead to determine the target angle. However, this controller lacks a component to handle opponents, and the movement along the track requires more fluency. So, we started improving the EPIC code to achieve these desirable goals.

Many approaches can be found in the literature for track prediction with the purpose of optimizing the performance. Such predictions help the controller to make early decisions on adjusting the steering angle and the target speed, in order to keep the car inside the track. Such an approach allows the controller to minimize the damage to the vehicle and to reduce the time required to complete the race.

One popular approach of track prediction depends on calculating the distance ahead, such as the one used in the EPIC controller. It calculates the free available distance ahead of the car to determine the target angle. Another approach is "the track segmentation", in which the track is divided into pieces and these pieces are classified as pre-defined types of polygons. Then the controller reconstructs a full track model from these polygons, as presented in [8].

Another controller based on the track segmentation principle is presented by Onieva et al. [6]. Their controller was submitted to TORCS Car Racing Competition 2009 [9]. The architecture of the controller consists of five simple modules that control gear shifting, steer movements, and pedals positions [6]. In addition, the target speed is adjusted by the "TSK fuzzy system". As the authors pointed out, "Fuzzy rule-based systems are considered one of the most important applications of the fuzzy set theory suggested by "Zadeh [10]". When the car is inside the track, the target speed is calculated based on certain rules [6]. The most important aspect of this work is the opponent modifier. It controls the driving behavior in situations when an opponent is nearby by adjusting the steering controller and the braking controller immediately. However, this approach doesn't take into consideration the factor of the opponent's speed. In general, this paper provides an important contribution for detecting the track mode and handling the opponents for autonomous cars.

Another paper [7], also written by Onieva et al. in 2012, presents a driving controller called AUTOPIA for the simulated car racing competition. It provides a full driving architecture including six separate main tasks: gear control, pedal control, steering control, stuck situation manager, target speed determination, opponent modifier, and learning module [7]. The performance of the controller was tested in two efficient ways: it was run over several tracks

with and without opponents. Several measures of performance were reported, such as participating in international competitions and running the car on several tracks once alone and another time with opponents. The controller was submitted as a participant to the 2010 Simulated Car Racing Competition, in which it won laurels in the end as the authors claimed [7]. The paper provides a simple and a powerful architecture especially for the opponent modifier. It deals with opponents in all directions in a simple approach. When an opponent is present within unallowable distances, heuristic rule sets are applied for pedal control and steering control [7].

Furthermore, many learning approaches are presented to find the optimal path the car should follow to reduce the time required to complete the race. Finding the optimal path could be accomplished by shortening the distance covered by the car and avoiding unnecessary turns.

"The evolutionary learning approach" is presented in a paper by Kim, Na et al. [3]. It presents an optimized algorithm which was used for an autonomous car controller using "self-adaptive" evolutionary strategies (SAESs) [3]. Kim, Na et al. developed additional rules and parameters to enhance the performance of their previous model, and they applied new learning approaches to those rules and parameters [3]. This work is well-experimented and it provides learning approaches that are able to derive the parameters used to determine the target speed in an efficient and easy to generalize way. Yet, it lacks an opponent handling system.

Another controller using the evolutionary learning system is presented by Quadflieg et al. in [8]. The controller is based on the track segmentation principle. It was submitted to TORCS Car Racing Competition 2010 [8]. This controller uses a simple evolutionary learning approach which enables planning the path ahead for the car [8].

Artificial neural networks (ANN) are also used as a learning system. In [5], a controller presented by Mun˜oz, et al. was submitted to the 2010 Simulated Car Racing Championship. It is "a human-like controller" using neural networks [5]. It adopts the principle of track segmentation. The controller builds a model of the tracks using the neural networks to predict the trajectory the car should follow and the target speed [5]. "The neural networks are trained with data retrieved from a human player, and are evaluated in a new track" [5]. The AANs are trained to reach the optimal path the car should take to behave similarly to the human player. This work shows a satisfying result of predicting the trajectory in new tracks; however, the target speed is most likely slower than the human's in the same tracks because of the absence of an opponent overtaking component, as the authors mentioned [5].

A different controller suggested by Chaudhary and Sharma in [1] generates the optimal racing line using artificial neural networks. The controller choses the optimal racing line within a scope angle of 15 degrees that gives the maximum possible speed in every point on the path.

Overall, most of the works succeed in building either a track prediction system or an opponent-handling system. It is challenging to deal with opponents while the car is traveling on a specific target angle and at a specific target speed. Sometimes, the presence of opponent requires adjusting the steering angle and modifying the speed, either accelerating or decelerating. Thus, most of the papers focus on improving track prediction systems regardless of the presence of the opponents.

We will compare our model with both the Epic controller described earlier in this section, and with a Simple Driver controller provided by the TORCS engine as part of the client code. The Simple Driver is a very simple controller providing basic modules for steering control and

accelerating/brake control. It keeps the car in the middle of the track as much as possible, and it applies a simple recovery policy if the car is stuck.

3. Proposed Methods

We will discuss more in details the procedural and learning methods that we use to improve the EPIC algorithm that we started from. In the procedural methods, we will describe the units that we add to enhance the performance of the Gazelle driver. As part of the discussion of the learning methods, we will describe some algorithms that we would like to use to improve the procedural driver automatically.

3.1. Procedural methods

The TORCS engine provides the following information to the controllers: a car status containing current speed, angle with the centerline of the road, distance from the center of the road, and more; an array of sensors detecting the distance to the road border in a 5 degree increment in a range of [-45, 45] degrees around the car's direction of movement; and array of opponent sensors with information about opponents present within a 200m radius of the car in all directions.

The first goal of this thesis is to implement the Gazelle controller efficiently by improving the existing modules from the EPIC controller and by adding new components to deal with aspects not present in the EPIC driver. The EPIC driver is the starting module for the Gazelle driver. We will also add new modules to minimize the damage and deal with opponents.

The Gazelle Controller

The Gazelle controller consists of three components: the target direction unit, the target speed unit, and the opponent adjuster. The target direction unit controls the direction in which the car is moving. The target speed unit adjusts the speed based on the target direction, while the Opponent Adjuster adjusts the direction and speed based on the opponents' presence. We will describe each unit in more details as follows.

Target Direction Unit

The unit determines the target angle using the following guidelines:

- If the current direction of the car is close enough to the road centerline, there is enough distance straight ahead, and the car is safely inside the track, then the car can continues in the same direction.
- Otherwise, we start with the direction of the road centerline, and scan by 10 degrees in the direction in which the distance ahead increases, until we find an angle at which it decreases, or we reach the maximal turn angle of 45^{o}.

Figure 1 (source: [2]) shows this scanning process of searching for a good path of movement.

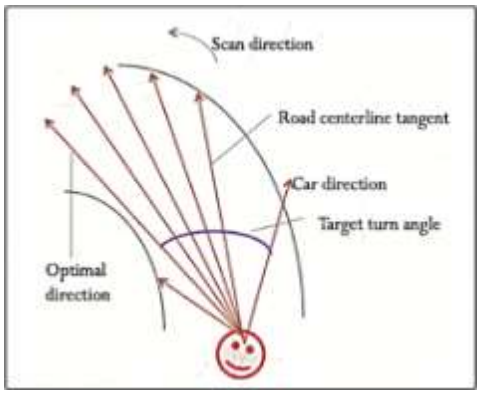

Figure 1: *The scanning process [2]*

- If the car is too close to the border of the road or gets outside, we add a direction change to move it back inside.
- If the current turning angle is good enough, we maintain it for movement continuity. This is an addition to the Gazelle controller to improve the fluency of the car's movement.

As Figure 2 shows, after the target angle is computed, we identify three types of situations on the road:

- Straight: if the road is straight ahead of the car and the target angle is between 0^o and 10^o.
- Fast Curve: if the upcoming curve is small enough and its angle is between 10^o and 15^o.
- Medium Curve: if the angle of the upcoming curve is between 15^o and 30^o.
- Slow Curve: if the upcoming curve is wide and the target angle is greater than 30^o.

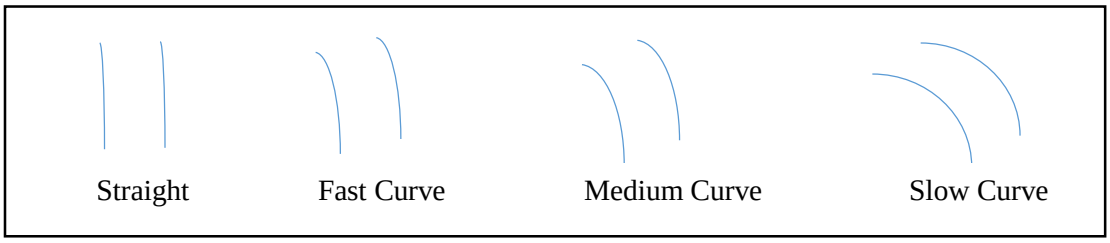

Figure 2: *The curve types.*

We differentiate between the curves in order to adjust the target speed in the next module. Thus, the straight and fast curves allow the controller to drive at the maximum speed, while the slow curves require adopting the minimum safe speed to keep the car inside the track.

Target Speed Unit

The target speed is computed once we know the target angle. The unit determines the speed using the following guidelines:

- If we are going almost straight or on a fast curve, the distance ahead is large enough, and no sharp turn is coming ahead, we aim for a modularity high speed called "Sunday Driver".
- Otherwise the target speed is first scaled with the cosine of the target angle for the change in direction and with the available distance in the aimed direction.
- The resulting target speed is scaled afterwards by a factor depending on the sharpest turn in the road detected ahead, 20 degrees left and right of the aimed direction. The purpose of this is to anticipate situations where the speed needs to be reduced.

Opponent Adjuster Unit

We put more efforts into building a component for dealing with opponents because the car's performance can be optimized by handling the opponents properly. As we mentioned above, most of the controllers we discussed before don't handle the opponents well or at all. Neither the Simple Driver, the controller provided as an example by the TORCS competition, nor the EPIC controller can deal with the opponents.

In our opponent adjuster, if an opponent violates chosen tolerance values of closeness as determined by the opponent sensors in each direction, then the gas/brake control and steering control will be modified to avoid the collision the following way:

- If there is an opponent at a distance of 200m or less, then a test will determine if it violates the safe distance (the tolerance values) in each of the available sensor directions.
- If there is an opponent in the front of the car, on the sides, or in the rear of the car within an unallowable space, the following flags are turned on, causing a reaction of the respective modules:

- A **Brake** flag for an opponent in the front. This flag takes care of the sensors in the range of -40^o to 40^o [6]. If an opponent is found within unallowable and its speed is close to ours, the car should brake immediately by modifying the brake/accelerate value to the half of the current speed. The tolerance values are shown in Table 1 and were adopted from [6].

Table 1: *Opponents adjuster over the gas & brake action [6]*

Orientation of the Opponent Sensor	Tolerance Value
$\pm 40^o$	6 m
$\pm 30^o$	6.5 m
$\pm 20^o$	7 m
$\pm 10^o$	7.5 m
0^o	8 m

- A **Steering** flag for an opponent in the front or on the side, it takes care of the opponent sensors in the range of -100^o to 100^o, also adopted from [6]. An overtaking manoeuver requires to modify the steering angle if the opponent violates the tolerance values. The tolerance values are shown in Table 2.

Table 2: *Opponent sensors tolerances for overtaking [6].*

Orientation of the Opponent Sensor	Tolerance Value
0^o, $\pm10^o$	20 m
$\pm20^o$	18 m
$\pm30^o$	16 m
$\pm40^o$	14 m
$\pm50^o$	12 m
$> \pm50^o$	10 m

- An **Accelerating** flag for an opponent at the rear of the car driving at an equal or higher speed than ours. Increments values are summarized in Table 3.

Table 3: *Opponent sensors increments for overtaking [6].*

Orientation of the Opponent Sensor	Increment Value
0^o, $\pm10^o$	$\pm0.20^o$
$\pm20^o$	$\pm0.18^o$
$\pm30^o$	$\pm0.16^o$
$\pm40^o$	$\pm0.14^o$
$\pm50^o$	$\pm0.12^o$
$> \pm50^o$	$\pm0.10^o$

Trouble Spots Register

This component was added in order to avoid the accidents caused by mistakes in predicting the right steering angle, leading the car out of the track. In TORCS competitions, the race starts with a warming level which allows drivers to learn the track, then the actual race takes place in the second level. Thus, we introduced the "Trouble Spots Register" detecting and storing places in the track where the car gets out of the road starting from the warming level. In the subsequent lapses of the circuit, to avoid repeating these mistakes, we use a method decelerating the speed whenever the car is close to a trouble spot, by an amount inversely proportional to the distance to the trouble spot.

A list of "trouble spots" on the road will be stored by the Gazelle driver in a persistent memory space in order to be accessible at later points during the race. To achieve this, the last position of the car on the road is stored in each frame. Then when the code detects that the car got out of the road, this position is added to the list.

In each frame, the current position of the car is compared to the trouble spots. If we are close enough to one of them, the speed will be adjusted as mentioned above. The closer we are to the trouble spot, the closer the speed will approach the safe one.

The issue arises from the fact that visibility of the driver is limited to 200m ahead and that it's difficult to break down the speed fast enough if the situation requires it. For this reason we adopted the approach of a sharp turn on a road combined with the troupe spots detector.

3.2. Learning Methods

In this part of the research, we aim to optimize the performance of the procedural driver automatically using learning methods. As we mentioned before, our goal is to minimize the damage as much as possible, and to reach the maximum safe speed. These two goals can be achieved by reaching the ideal target angle and the ideal target speed. We need to enable the controller to learn during the racing time using learning algorithms. We will use two main algorithms for this purpose: Artificial Neural Networks and Hill-Climbing.

An Artificial Neural Network (ANN) is a learning method that is inspired by the way the human's biological nervous system processes information. Such a system is composed of a large number of connected neurons, the processing elements, in which components work together to solve a specific problem [14]. The ANN is a "layered structure" consisting of three main layers: the inputs layer, the hidden layer, the outputs layer [11]. The hidden layer uses the learning processing elements (neurons) to adjust the input values combined with a set of parameters in order to produce the optimal output solution.

ANNs can learn by examples and they can be used for pattern recognition or data classification and they are also appropriate for prediction or forecasting [14]. There are many applications of ANNs such as modelling and diagnosing the cardiovascular system, sales forecasting, industrial process control, customer research, data validation, risk management, target marketing, and credit evaluation [14].

We can implement an ANN in the Gazelle for the Target Angle Unit using the car state to represent the input layer. As Figure 3 shows, a hidden layer will process this input combined with parameters that predict the damage and the maximum safest angle. Based on these parameters, the ANN will output the optimal target angle. Another ANN is desirable also to be used in Gazelle controller to take care of the Target Speed Unit, taking the output of the first ANN as an input, then processing the input in the hidden layer to produce the optimal target speed.

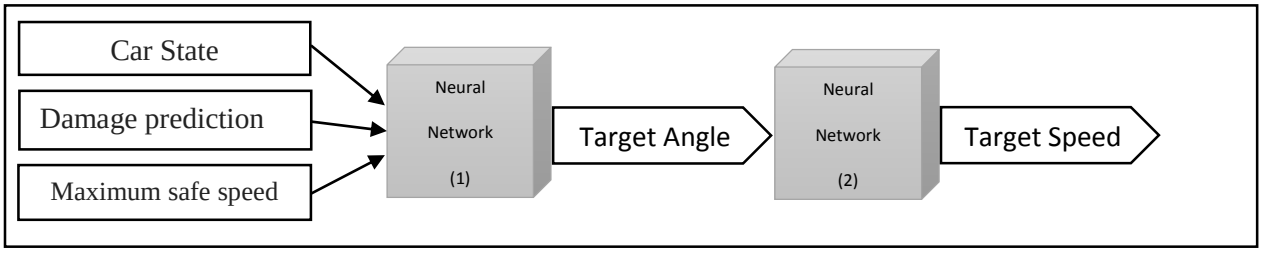

***Figure 3**: Implementation plan for using neural networks in the Gazelle Controller.*

We will use another learning method: the Hill Climbing (HC) algorithm which creates the first candidate solution and then produces the offspring using "a parameterless search operation". The search operation performs a loop in which the optimal solution at the current time is used to produce one child. If this new child is better than its parent, it replaces it. Then, the cycle starts all over again [12].The algorithm does not maintain a search tree: It looks for an appropriate path only from the current state and immediate future states. Hill climbing is widely

used in networking and communication, robotics, data mining and data analysis, and developing behaviors for game players [12].

The HC Algorithm can be efficient to use for predicting a good path that the car should take in order to optimize its performance. We will use HC in the Target Angle Unit and the Target Speed Unit to improve the car's performance.

4. Experimentation Methodology

We establish a set of tests to measure the performance of the Gazelle controller comparing it to previous work. We will compare the newly developed methods with two existing models: the *Simple Driver* provided by the TORCS software, and the *Epic Driver* that was developed before. We need to choose a number of tracks that we'll test them on, and determine what conditions we're going to run this in.

For the **tracks**, there are three main categories: road tracks, dirt tracks, and oval tracks. There are 21 road tracks, 8 dirt tracks, and 9 oval tracks to choose from. We choose three of the road tracks, one dirt track, and one oval track. Of the oval tracks, *E-Track 5* looks the most interesting. Of the dirt tracks, *Dirt 4* looks like it has a good variety of curves. For the road track, we choose three of these tracks: *Forza*, *Alpine 2*, and *E-road*.

Figure 4: *The Alpine2 track on the right, and the car is travelling on the same track on the left.*

As Figure 4 shows, the Alpine 2 track is a road track; its shape has many curves of all kinds: fast, medium and slow curves. Such a road enables us to test the performance more efficiently. We can also notice the material of the road on the left, which looks like asphalt. TROCS interacts with the road's material and the behavior of the cars on the road depends on it. It makes the tracks made of asphalt allows the car to travel more fluently than the dirt road.

We set the number of **lapses** to five, five lapses would be good enough to have an accurate comparison. At the end of the five lapses, the program itself outputs some information, such as the total time and the damage. We will also add some other **measures** that would be good indicators of performance: the number of times the car gets out of the road, the total time spent outside the road, and the total distance covered by the end of the race. The more distance is covered in one lapse, the less efficient the driver is.

Then we will run the three drivers, Simple, Epic, and Gazelle, on the five tracks and store these measures for all of them. We will take one track and discuss the results of the experiment as a sample of how these measures will be used in the thesis. We chose the E-Road from the road track category and the results are shown in Table 4.

Table 4: *The total statistics of running the three controllers individually on E-Road in TORCS.*

Description	**Simple Driver**	**Epic**	**Gazelle**
The number of times the car gets out of the road	0	0	33
The total time spent outside the road	0	0	4099
The total distance covered by the car from the beginning of the race	16328.5	16328.5	12949.6
The maximum distance covered by the car from the start line along the track line	3260.42	3260.42	3260.37
The damage of the car	0	0	10149
Total time	17:37:13	7:28:51	2:13:20
Lapses	5	5	3

We can have more precise statistics by calculating the average of each measure per lapse as shown in Table 5.

Table 5: *The average statistics for each lapse of running the three controllers individually on E-Road in TORCS.*

Description	**Simple Driver**	**Epic**	**Gazelle**
The number of times the car gets out of the road	0.0	0.0	11.0
The total time spent outside the road	0.0	0.0	1366.3
The total distance covered by the car from the beginning of the race	3265.7	3265.7	4316.5
The maximum distance covered by the car from the start line along the track line	652.1	652.1	1086.8
The damage of the car	0.0	0.0	3383.0
Total time for each lapse	0:03:31	0:01:30	0:00:44

As Table 5 shows, out of the three drivers, the minimum time per lapse was achieved by the Gazelle controller. This is due to the Target Direction Unit. The target direction allows the car to adjust the required steering angle to the minimum angle to achieve the safest maximum speed and as a result, the Gazelle succeeded in achieving the best time. However, taking a smaller target angle required more distance to be covered by the car making it take a less efficient trajectory. Also, the number of times the car gets out of track is higher for Gazelle and, accordingly, the total time the car spent out of the track is potentially higher than for the two other controllers. Thus, higher damage happens as a result of the collision with the outer walls of the track when the car gets out of the track.

The Simple Driver and Epic achieved less damage for five lapses compared to the Gazelle. The Simple Driver & EPIC both completed the five lapses with no damage, while Gazelle couldn’t complete the race to the end and it did only three lapses because the damage was too high. The game rules state that the race will be terminated early for any car for which the damage reaches a given threshold. We hope that further improvements to the procedural methods and the application of the learning methods will remedy this aspect.

5. Conclusions

In the thesis, we hope to accomplish a well-developed and efficient algorithm for Gazelle. Such algorithm can improve the learning methods using neural networks and hill climbing. We need to lead the racing car to achieve the efficient path and the efficient speed and to minimize the damage caused either by opponents or by getting out of track. For this we will be using procedural and learning methods to enable the controller to make a good decision can be made at every frame of the track.

This work was a part of our participation in an international completion (GECCO-13). It was accepted and qualified to be a part of the final race. And it accomplished the eighth rank as the champion's organizers announced [15]. This participation inspired us to continue developing our controller to be a part of the next year's competitions for simulated car racing.

Our ongoing effort to develop the Gazelle controller aims to predict an appropriate path and speed for the racing car in each frame of the track based on the available information from the server and based on the knowledge that the car has built using the learning methods. In fact, using neural networks could lead the controller to use more accurate equations based on previous data derived during the learning process. We expect that the more the networks are trained, the more precisely they will predict the driving information. We may also use hill-climbing methods to refine the learning process.

Working with such a project helps us to improve our skills with programming using C++. It also introduces us to game programing and new learning methods such as neural networks and hill climbing. We hope this thesis satisfies our passion to make Gazelle controller a self-training controller and achieve higher orders in the next international champions.

6. References

[1] N. Chaudhary and S. Sharma (2013). Race Car Strategy Optimization under Simulation. April 2013, 1-7.

[2] C. Guse and D. Vrajitoru (2010). The Epic Adaptive Car Pilot. *In Proceedings of the Midwest Artificial Intelligence and Cognitive Science Conference*, April 17-18, South Bend, IN, 30-35.

[3] T. S. Kim, J. C. Na, et al. (2012).Optimization of an Autonomous Car Controller using a Self-Adaptive Evolutionary Strategy. *Published in International Journal of Advanced Robotic Systems,* vol. 9, 73, 1-15.

[4] D. Loiacono, L. Cardamone, et al. (2013). Simulated Car Racing Championship Competition Software Manual, April 2013.

[5] J. Mun˜oz, G. Gutierrez, et al. (2010). A Human-like TORCS Controller for the Simulated Car Racing Championship. *Published in IEEE Symposium on Computational Intelligence* and *Games (CIG),* 473 - 480.

[6] E. Onieva, D. A. Pelta, et al. (2009). A Modular Parametric Architecture for the TORCS Racing Engine. *In proceeding of IEEE Symposium on Computational Intelligence and Games*, 2009, Madrid, Spain, 256-262.

[7] E. Onieva, D. A. Pelta, et al. (2012). An Evolutionary Tuned Driving System for Virtual Car Racing Games: The AUTOPIA Driver. *Published in International Journal of Intelligent Systems 27, 3*, Oct 2012, Madrid, Spain, 217-241

[8] J. Quadflieg, M. Preuss, et al. (2010). Learning the Track and Planning Ahead in a Car Racing Controller. *Published in IEEE Conference on Computational Intelligence and Games (CIG'10),* 395-402

[9] G., Raidl, (2009). *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO).* Montreal, Canada: ACM SIGEVO.

[10] L. A. Zadeh, (1965). Fuzzy Sets, *Information and Control, vol. 8*, 338–353.

[11] R. Lopez, (2012). OpenNN: Open Neural Networks Library Software Manual.

[12] W., Tomas (2009). Global Optimization Algorithms: Theory and Application.

[13] The Center for Automotive Research at Stanford
(http://me.stanford.edu/groups/design/automotive/)

[14] Neural Network (http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html)

[15] 2013 Simulated Car Racing (GECCO-2013).
(http://www.slideshare.net/dloiacono/gecco13scr)