Machine learning in gear changing

Alexandru Petruţ Becheru
Teacher Coordinator: Cătălin Stoean

Abstract

In the past few years applying computational intelligence to games has gained a lot of interest. Car racing simulation games are a great test bed, as researchers can directly see the practicability of their work. This paper is an initial approach to optimizing the basic controller for the game TORCS. We did this by using an artificial intelligence algorithm: Hill Climbing. It was used in finding proper values for gear changing that will make the car faster. A gear changing value represents the rpm value of the motor where the shift is done, i.e. like shifting from 3rd to 4th gear at 2500 rpm. After several laps of a circuit the algorithm was able to determine several sets of values, which in some cases made the car faster by 30%. Just to make things more real we used 2 tracks that resemble a motorway and a city road, and we also introduced several competitors within the same race.

1 Introduction

Computers have made our life better in such many ways that they became part of our daily routine. In the past when someone had to design something he or she was obliged to make a lot of prototypes and test them, which is very time consuming and expensive. Nowadays programs like CAD are able to simulate tests, and aid the users to design the product faster. The accuracy of some programs makes them a necessity to have for every researcher.

Computational intelligence games simulate the real world, or at least some aspect of it, as good as they can. This is done with the scope of easily testing new ideas, we can perceive it as the first feedback necessary to determine the potential of one's idea. Another part that makes these games receive such attention is that one can have fun while working. This translates in better motivated and productive researchers.

Today, cars are a big part of our life, whether we like or not. Every boy and more and more girls dream to have their own car. Some of them dream to race one another, others don't just want to drive, they want to enhance their cars performance. Games made those dreams easier, cheaper and healthier. But what about those people who really want to make a change in the car design, the real car, not just those that powerful graphic cards generate on our monitors? They also have their games, not as fancy looking as the others, but which simulate races to their best. Race simulation is not just about how a car turns brakes and accelerates, it is about why, when, and in which circumstances it does that and many other. Driving on two different roads is not the same, one might need to brake harder and accelerate more, change gear faster and use different techniques of driving.

TORCS [11] (The Open Race Car Simulator) is a multi-platform racing simulator used in artificial intelligence racing as a research platform. If offers a universe of 50 cars, 20 circuits

and the possibility to create your own to suit your needs. It has several modules implemented such like damage control, aerodynamics, wheel control; where one can even specify things like the rigidity of the springs. In 2007 at its 10th birthday *Linux Journal* proclaimed it as the most capable open source simulator [12].

The following section will present the previous work done in this research field and the motivation. The software and the approach to the problem are also put into light. Finally the experimental results together with the concluding regards will close this paper.

2 Previous work and motivation

On top of the TORCS platform there are competitions held, one of which is the IEEE *Simulated Car Racing Championship* [13]. In 2011 the competition took place during 3 conferences: EVO held at Torino, ACM GECCCO held at Dublin, IEEE CIG held at Seul; each consisting of 3 races. Its purpose is to encourage the development of intelligent agents capable of driving cars on unknown circuits. It improves the TORCS platform by offering a package of software to aid the development of intelligent agents. A car can be driven by a set of sensors and actuators.

One approach to creating an intelligent agent is learning from another agent. At first Jorge Muñoz et al. [3] tried to imitate the behavior of a human agent. But as he concludes, it is very complicated to learn the human behavior, because of several causes: neither the human makes the same action in the same circumstances, nor performs all actions in a proper way. Creating a pattern for learning is difficult in this situation. Next, they tried to imitate the behavior of a non-human agent, the 2008 winner of the Simulated Car Racing Championship. Although they demonstrated that it was possible to imitate that agent with good results, they had a problem in gear changing learning.

Evolving a rule based agent with the aid of genetic algorithms was tried by Diego Perez et al. [5]. They succeeded in creating about 120 rules, of which they reduced to a subset of 10 to 20 rules depending on the circuit. Their algorithm showed that after a few generations the lap time was reduced and the damage to the car was almost inexistent. In this study gear changing did not take part in the learning process.

Other many approaches have been used to create intelligent drivers, like using Ant Colony optimization algorithm by Luis delaOssa et al. [1], or developing fuzzy rules by Diego Perez et al. [6] just to name a few.

In studying all of the above, gear changing either was not a issue of interest or it became a problem. Gear control module is one of the five basic modules in the architecture of the TORCS Racing Engine according to E. Onieva et al. [4], so the aim of the current research was to find out if proper gear changing can reduce the lap time.

The *Simulated Car Racing Championship* offers a basic agent that can drive a car on a circuit; though it is a pretty slow one it keeps the car from going off track. The idea was to use this agent to run several times on the circuit with different sets of gear changing values given by the AI algorithm, trying to get better lap times. After each lap the agent will send to the AI algorithm the feedback consisting in time of the lap, the AI algorithm based on this result will generate a new set of gear changing values based on which the agent will race again.

At first we had to develop a program which had to automate the running and the collecting of feedback from TORCS, also this program had to ensure the data transmission between the AI algorithm and TORCS, finally it had to manage all the data obtained by writing it to a Microsoft Excel document so that it can be viewed and analyzed with ease.

Next step was to employ the AI algorithm: Hill Climbing (HC). Its parameters had to be adjusted in order to better fit with our scenario. Having the software developed we tested the agent on 2 separate tracks with and without other cars. Unfortunately due to the hardware problems, we could just add 2 other cars, which thankfully was enough to get some conclusions.

3 Tools

In this section you'll find the most important piece of software that we used. Apart from those mentioned in the following subsections we used Microsoft Excel, and Dev-C++ [14]. The obtained software incorporates the AI algorithms and the interface with the TORCS server [15].

3.1 TORCS

TORCS [10][11] is one of the most popular car racing simulator for academic purposes, developed by the *University of Würzburg* and *Politecnicio di Milano*, licensed under GPL (General Public License), so it is free to use. It presents the following advantages according to E. Onieva et. al. [4]:

- Besides being a good car racing game (Fig. 1), it offers a fully customisable environment, like the ones typically used by computational intelligence researchers for benchmarking purposes.
- It features a sophisticated engine (aerodynamics, fuel consumption, traction, etc.) as well as 3d graphics engine for the visualisation of the races.
- It was not conceived as a free alternative to commercial racing games, but it was specially devised to make it as easy as possible to develop your own controller.

It is a fully customisable environment offering the possibility to develop you own cars and circuits [2].

Fig. 1 Torcs game screenshot

3.2 Simulated Car Racing Championship

As previously mentioned, the Simulated Car Racing Championship [12] offers a platform on which racing competition are held. This software package is multi-platform, working on both Microsoft Windows and major Linux distributions. Controllers can be developed both in C++ and JAVA.

The environment and the car's status are perceived by a number of sensors [2]. There are sensors which only give feedback, current car ranking, distance to the opponents etc. Other sensors which are called actuators don't only give feedback but by modifying their value we can control the car, i.e. by modifying acceleration, one can drive faster or slower.

The architecture of the race is client-server based, this way the competition is fair and data can be compared, and if one controller is stuck, the others can run without a problem:

- The client is the game of TORCS, utilized as an interface for the user so he can receive the data and see the race if he wishes. On top of this, there is a communication module that sends data back and forward to server.

- On the server side we can find the engine of TORCS, but there is no visualization here, only the physics engine. During a race, the controller on the client part will receive information concerning the environment and the status of the car from the server at every 10ms. If the controller wishes to act in some way, i.e. change gear, he modifies the specific actuator and sends the data to the server.

3.3 AutoIt

AutoIt [16] is a BASIC (Beginner's All-purpose Symbolic Instruction Code) like programming language, conceived to automate the GUI (Graphical User Interface) of Microsoft Windows, but in can be used for general programming. It simulates the presence of a user by controlling the mouse and the keyboard, thus making possible the manipulation of program windows and the automatisation of user programmed tasks.

It was initially developed to automate the configurations necessary for millions of computers before being shipped to clients. Over time other modules were added for better interoperability with other programs which run on the Microsoft Windows platform, i.e. Microsoft Excel. It also has the possibility to create GUI's by using the following components: edit boxes, check boxes, list boxes, buttons, status bars and combo boxes. This feature makes it more user friendly.

4. The approach

The approach is based on the 3 main objectives. First is to build a stable and reliable platform which will facilitate the integration of the AI algorithms with TORCS and Simulated Car Racing Championship software. The second is to adjust the algorithms to our scenario. Last but not least is to get enough data so we could establish if gear changing does count in the lap time

4.1 TORCS and AI algorithm integration

The bridge between the algorithm and TORCS was coded in AutoIt. To facilitate the testing we introduced the concept of *series* and *generations*. A *generation* is one race with a fix number of laps, in our case a generation equals a circuit lap. A *series* is a number of *generations*, after several tests we concluded that a series should have 200 "generations". If this number was smaller then we may not always reach the vicinity of good results, but if it was bigger, it would make no sense hence we already reached it. It offers the following facilities:

- A graphical interface, where we can program the testing and get information about the state of the test shown in Fig. 2. It is composed of a window with 2 functionalities:

    a. Running options and management, in the top half. The *Command* pane contains buttons for the management of the tests: *Start* for starting the tests, *Stop Generation* stops the tests after the current generation end, *Stop Series* stop the tests after the current series/lap. The *Options* pane is where we set the algorithm, the number of generations and series, and the name for the excel file where the data will of the tests will be saved. Also here there is a *Send Options* button, which we must press if we want to save the changes we made.

b. Running state, in the bottom half. Here we receive data about the elapsed time from the beginning of the test, the estimated time until the end of the test, the current number of generations and series, results for each series, push button messages.

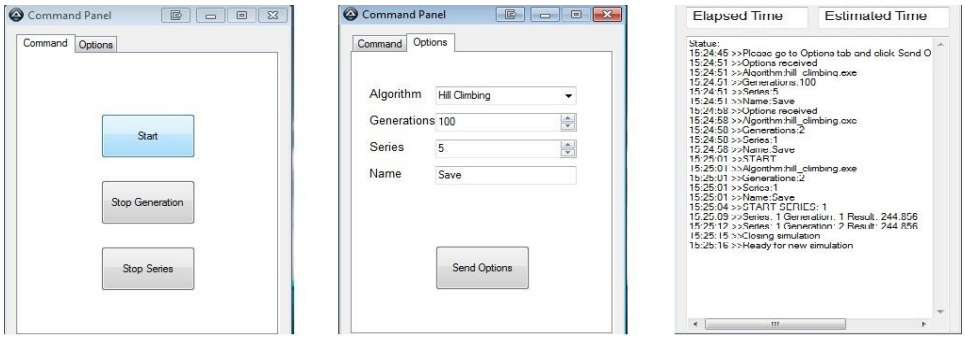

Fig. 2 Command Panel, Options Panel, Status Panel (from left to right)

- The automatisation of the TORCS platform and the Simulated Car Racing Championship software that comes on top, meaning that the bridge is able to start the game and race, close the game without any human intervention
- Data management which consist in the following:

  a. Retrieve the data obtained after running on a track, in our case the elapsed time. This is done by reading an XML file.

  b. Transfer data from the controller to the algorithm and back. This was done by reading and writing data to a file, and by starting and pausing the algorithm and the game in an alternative way. When a race is running the algorithm waits for it to finish, next the game is paused and the time is retrieved from the race and passed to the algorithm. Then the algorithm starts, reads the new data and after doing the necessary calculations passes the new set of values to the controller, and pauses.

  c. Saving the data of the test into an Excel file. There we can find the time and the set of values for each series. At the end the best time, the average time, and the worst time will be retrieved and written here.

- The automatisation of the algorithm which means that they can be started, paused and closed.

4.2 Developing the AI algorithm

The algorithm [7][9] needed to be adjusted for the current scenario, this was done during 10000 test laps. We wanted to make it as good as they can be in reaching the best results in a faster way, but at the same time having them consistent. The tests were run on a computer with 2.8 GHz dual core computer, having 4 GB of ram and a graphics card with 256 MB of shared memory. The software simulated a lap of the circuits in about 30 seconds.

4.2.1 The mutation

Within HC one set of gear values is modified to another set of gear values by mutation. We started with uniform distribution (Fig. 3) but it did not conduct to the expected results, the

consistency of getting in the vicinity of the best result was low. We then tested new distributions, and discovered one that has a bigger space of search and reaches a consistency where almost every time the algorithms come in a close vicinity of the best result that we found (Fig. 4).

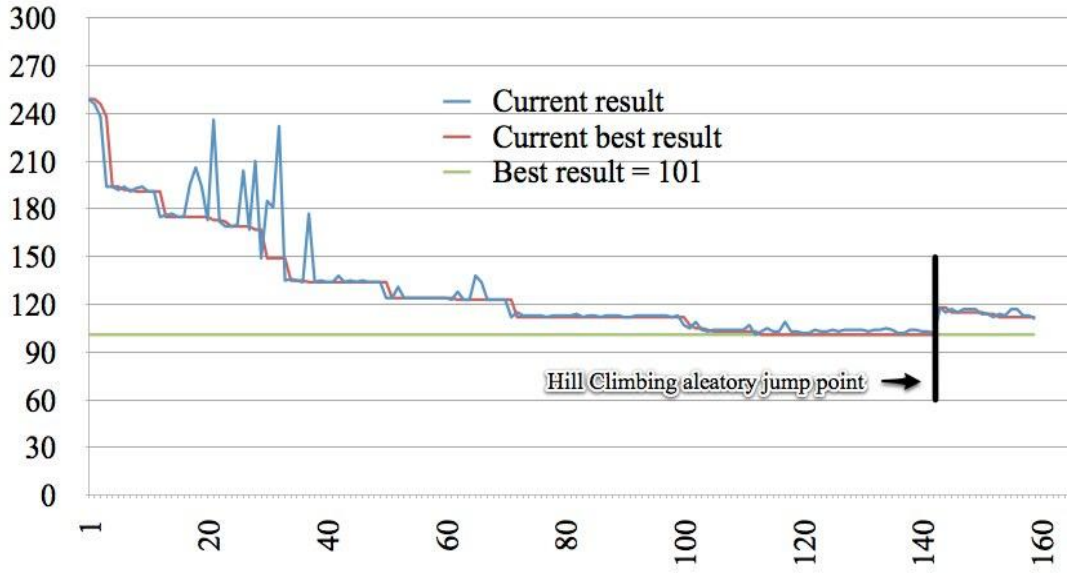

Fig. 3 Results obtained using HC with a uniform distribution

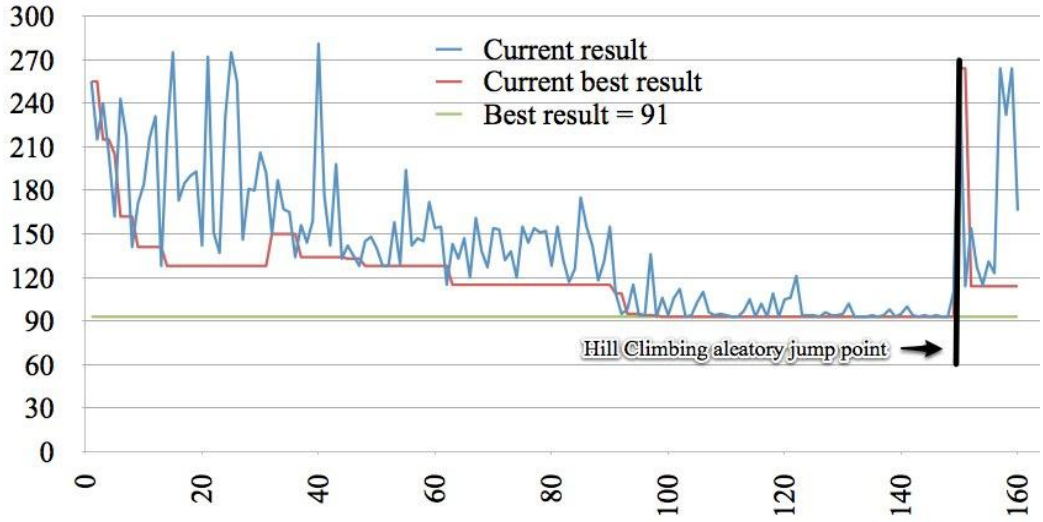

Fig. 4 Results obtained using HC with the proposed distribution

In Fig. 3 and 4, the horizontal axis represents number of generations, and the vertical axis represents the time for each lap. Restart point appears because of the random jumping of the Simulated Annealing algorithm that was used to generate these charts. One can see that the proposed distribution gives results which are lower than 100, while the uniform reaches its best at 100. Also by following the blue line it can be seen that the space of search is significantly bigger. The X parameter for the distribution is calculated the following way:

Algorithm 1 Proposed mutation

1: ***while*** *s≥1* ***do***

2: *u=rand(1,1000)*

3: *v=2*(u/1000)-1*

4: $s=2*v^2$

5: ***end while***

6: $X= sqrt((-2* log(s))/s)*v$

4.2.2 Hill Climbing

For a better understanding of the algorithms the following notions are necessary:

- vectorRpmOptim : is the vector of rpm values with the best result. The #Age represents the number of generations since it has not been modified, and the #Result represents the lap time associated with those values.
- vectorRpm: is the vector that is constructed out of the vectorRpmOptim and is due to be used on the next race. Also #Result attached represents the result associated with it.

This algorithm had to be adjusted to our needs and improved by introducing an random jump (line 6 – 10 of Algorithm 2). If the best result is not improved within 20 generations the algorithm randomly generates a new set of values for vectorRpmOptim. This avoids getting stuck on a local optimal, thus making our search area bigger.

The mutation strength has a value of 500, but if the best result remains unmodified for 10 generations, it is compressed to 250 (line 11 to 15 of Algorithm 2). At first we chose the value of 500 explore the search space better. If that fails we concentrate in slightly improving the current result by changing the value to 250.

The probability for a gear value to change depends on the last result (line 18 of Algorithm 2). A random number is generated between 1 and 1000; if this is smaller that the last result, then the value will be changed. So if the result is higher, then the chance of change increases. The results range from 90 to 300, so in the worst case scenario with the result being 300, there should be an average of 3 gear values changed out of 10. This way we make sure that a consistent change is made, but this is not large enough to make the search chaotic.

Algorithm 2 Hill Climbing

1: vectorRpm#Result ← 0

2: vectorRpmOptim ← vector_of_randomly_generated_values[1,8000]

3: vectorRpmOptim#Restult ← 1000

4: ***while*** *currentExecution ≤ maximum_number_of_executions* ***do***

5: *currentExecution ← currentExecution +1*

6: ***if*** *vectorRpmOptim#age > 20* ***then***

7: *vectorRpmOptim#Age ← 0*

8: *vectorRpmOptim ← vector_of_randomly_generated_values[1,8000]*

```
 9:          vectorRpmOptim#Result ← 1000
10:          vectorRpm#Result ← 0
11:      else if ( vectorRpmOptim#Age > 10 ) and ( vectorRpmOptim#age ≤ 20 ) then
12:          mutationStrength ← 500
13:      else
14:          mutationStrength ← 250
15:      end if
16:      vectorRpmOptim#Age ← vectorRpmOptim#Age+1
17:      for i ← 1, i ≤ vectorRpm#Length do
18:          if randomly_generated_number[1,1000] ≤ vectorRpm#Result then
19:              deviation=mutationStrength*X
20:              vectorRpm[i] ← vectorRpmOptim[i]+deviation
21:          end if
22:      end for
23:      Algorithm is paused and a new race is run with the values from vectorRpm
24:      vectorRpm#Result ← Lap Time
25:      if vectorRpm#Result ≤ vectorRpmOptim#Result then
26:          vectorRpmOptim ← vectorRpm
27:          vectorRpmOptim#Result ← vectorRpm#Result
28:      end if
29: end while
```

4.3 Circuit choosing

We chose two tracks for our tests. The first one (Fig. 5 left side) resembles a major city artery with a width of 15m and medium to fast corners, the second one (Fig. 5 right side) looks a lot like a highway with a width of 30m and fast corners. The tracks length is not important as we will not compare the two of them directly. In choosing the tracks we also consulted [8].

Fig. 5 On the left you can see the city track, and on the right the highway track

5. Experimental results

In the following charts there are some results that spike up to the value 300, this could be explained either by the random jump in Hill Climbing, or by impossibility to run the race. The horizontal axis represents the number of laps / generations, and the vertical axis the lap time. In the charts where will present specific values of the gears, the horizontal axis represents the gears and the vertical axis represents their value. In the latter case there will be 10 gear values, 5 are for changing gear upwards, i.e. from 3rd to 4th, the other 5 are for changing gear downwards, i.e. from 2nd to 1st.

We needed to track the improvement to the lap time. At first we calculated the average of the results obtained from the 200 laps. Then by comparing the best result obtained during those 200 laps to the average we calculated the improvement.

On the city track Fig. 6 (read it form right to left), the results get better as the generations pass. Actually the improvement is 34% compared to the average of results. The best lap was 91.338 and the average 122.60538.

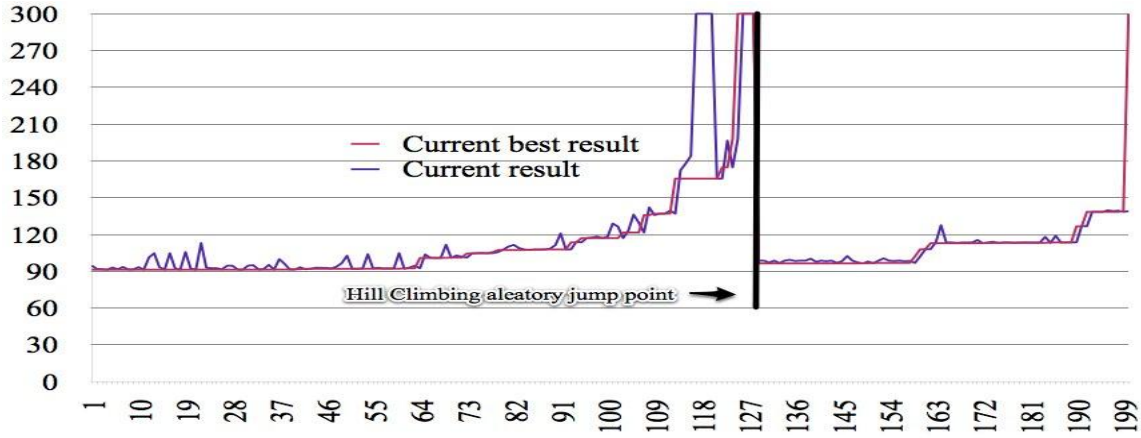

Fig. 6 HC results on the city track with no other cars

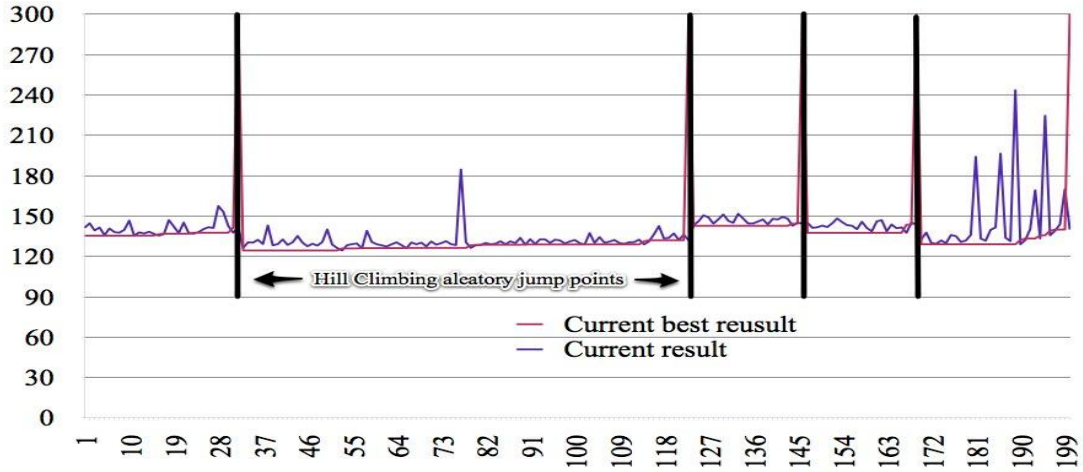

Fig. 7 HC results on the highway track with no other cars

On highway track in Fig. 7 (read from right to left) the improvement is not so obvious, 11% percent is not as good as the one before but we believe it's significant. One explanation could be that on the highway track, the controller doesn't manipulate the car so often as in the city track.

Next we introduced two other cars on both circuits .We realised that on the city circuit these had influenced the results in a bad manner, but the algorithm managed to improve the results. With our car alone on the circuit the best results came at about 90, with two other cars they were about 110. But considering that the average results even with one car was 120, this is still significant. On the highway circuit the results showed the oponents had no effect on the algorithm performance, the results were all in the range of 120 to 125. One might say that introducing other cars made the controller a little faster, but the difference is to small to say that for sure.

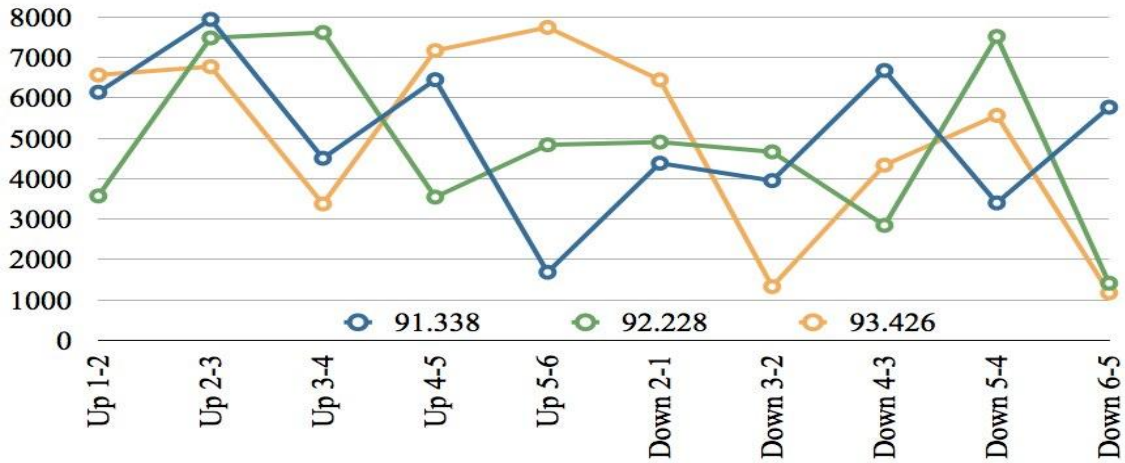

Fig. 8 Gear values for city track, no other cars

Up 1-2	Up 2-3	Up 3-4	Up 4-5	Up 5-6	Down 2-1	Down 3-2	Down 4-3	Down 5-4	Down 6-5	Result
6139	7942	4504	6449	1683	4382	3951	6677	3399	5768	91.338
3571	7484	7619	3543	4837	4905	4663	2843	7520	1412	92.228
6566	6773	3374	7171	7745	6446	1328	4335	5566	1173	93.426

Table 1 Gear values for city track, no other cars

Up 1-2	Up 2-3	Up 3-4	Up 4-5	Up 5-6	Down 2-1	Down 3-2	Down 4-3	Down 5-4	Down 6-5	Result
4214	2887	5946	7001	4233	1932	7144	4532	2564	2904	124.41
6252	1962	6561	4373	6591	2214	4559	2409	2421	6276	124.072
4214	2887	5946	7001	4233	1932	7144	4532	2564	2904	124.41

Table 2 Gear values for the highway track with 2 other cars

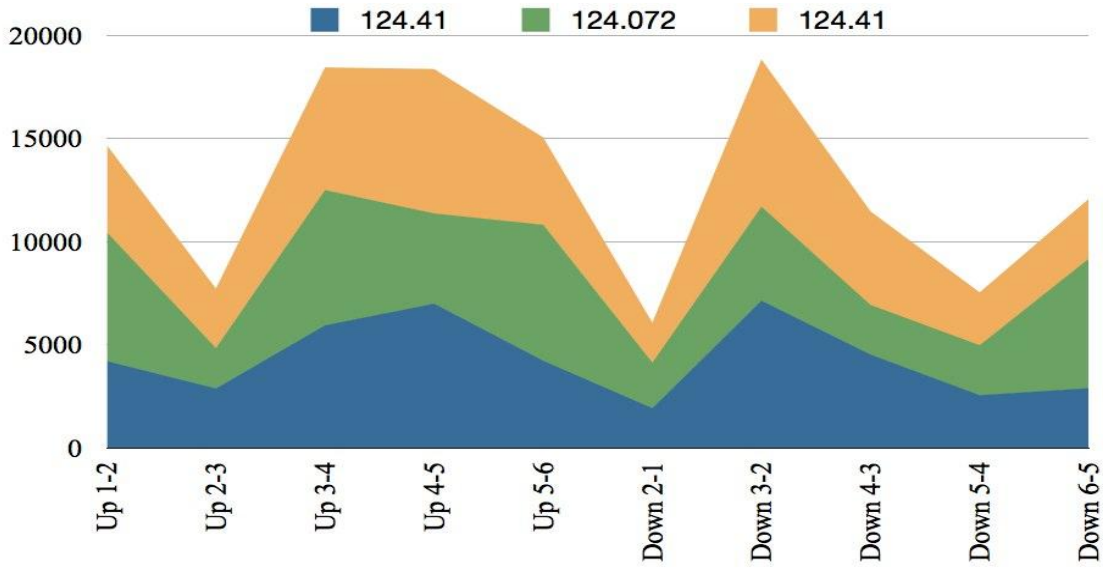

Fig. 9 Gear values for the highway track with 2 other cars

After this, we looked upon the set values which gave the best results. On the city track with no other cars, the values appeared to be absolutely random, as shown in Fig. 8 and Table 1. On the other hand on the highway track, the values of the gears look to be correlated, as shown in Fig. 9 and Table 2.

6. Concluding regards

We started work on this paper with the scope to find out if gear changing learning could improve the lap time. As the results show there is no doubt that lap time is influenced by gear changing values. The data obtained depicts big improvements 10% to 30 %, but considering that the used controller is a slow one, we can't tell the margin of improvement that may be achieved in a good controller. One thing is certain, from now one every controller designer should take in consideration gear changing learning.

Regarding the algorithm, we recommend testing and improving it. But we believe that other algorithms could give good results, so in a further research we shall try adapting other algorithms to this scenario. This poses another question for future development: Is there a way that we can create an algorithm that can return the best sets of values for gear changing, by serving it with the tracks proprieties?

Another thing that we learned during these tests is that there is no universal set of values that will give very good results no matter the track. But we can imagine that in the future there will be intelligent gear boxes that will adapt to the track proprieties and probably even to the driving mode.

References

[1] Luis delaOssa, José A. Gámez and Verónica López. *Improvement of a car racing controller by means of Ant Colony Optimization algorithm. IEEE Symposium on Computational Intelligence and Games, 2008.*

[2] Daniele Loiacono, Luigi Cardamone and Pier Luca Lanzi. *Simulated Car Racing Championship 2010 Competition Software Manual. Politecnico di Milano, Dipartimento di Elettronica e Informazione, Italia, may 2010. IEEE Symposium on Computational Intelligence and Games, 2011.*

[3] Jorge Muňoz, German Gutierrez and Araceli Sanchis. *Controller for TORCS created by imitation. IEEE Symposium on Computational Intelligence and Games, 2009.*

[4] E. Onieva, D. A. Pelta, J. Alonso, V. Milanés and J. Pérez. *A Modular Parametric Architecture for the TORCS Racing Engine. IEEE Symposium on Computational Intelligence and Games, 2009.*
[5] Diego Perez, Yago Saez, Gustavo Recio and Pedro Isasi. *Evolving a rule system controller for automatic driving in a car racing competition. IEEE Symposium on Computational Intelligence and Games, 2008.*
[6] Diego Perez, Gustavo Recio, Yago Saez and Pedro Isasi. *Evolving a Fuzzy Controller for a Car Racing Competition. IEEE Symposium on Computational Intelligence and Games, 2009.*
[7] David L. Poole and Alan K. Mackworth. *Articial Intelligence Foundations of Computational Agents*. *Cambridge University Press, 2010*. *http://artint.info/html/ArtInt.html*
[8] Mike Preuss, Jan Quadflieg, Günter Rudolph. *TORCS Sensor Noise Removal and Multi-objective Track Selection for Driving Style Adaptation*, *IEEE Symposium on Computational Intelligence and Games*, 2011.
[9] Cătălin Stoean and Ruxandra Stoean. *Evolutie si inteligenta articiala. Paradigme moderne si aplicatii*. *Editura Albastra, 2010.*
[10] Bernhard Wymann, T.O.R.C.S. *Manual installation and Robot tutorial*. *http://berniw.org/*
[11] The Open Race Car Simulator. *http://torcs.sourceforge.net/*
[12] Linux Journal regards toward TORCS. *http://www.linuxjournal.com/node/1000435*
[13] Simulated Car Racing Championship. *http://cig.ws.dei.polimi.it/*
[14] Dev-C++. *http://www.bloodshed.net/devcpp.html*
[15] Link for downloading the software developed by us. *http://inf.ucv.ro/~cstoean/diploma.html*
[16] AutoIt. *http://www.autoitscript.com/site/autoit/*

ALEXANDRU PETRUȚ BECHERU
University of Craiova
Faculty of Exact Sciences
Department of Computer Science
Str. A.I. Cuza, Nr. 13, Craiova
ROMANIA
E-mail: alex.becheru@inf.ucv.ro