Toxic Racing - A Torcs Controller

Andreas Hadjipieris, Juan Buhagiar, Sarantos Tzortzis
https://github.com/sarantinio/Torcs-Racing.git

University of Amsterdam
December 22th, 2017

Abstract. In this paper we describe three different controllers to race in the TORCS Platform. Using a neuro-evolution approach, with specific fitness functions, we obtained interesting results. Firstly, we obtained a networks that races tracks successful on its own. Secondly, we obtained a network that competes with opponents and lastly, we obtained a model to obstruct other cars from finishing the race. Overall, we conclude that neuro-evolution is a good fit to obtain intelligent agents for driving.

1 Introduction

The open racing car simulator (TORCS) is a state of the art car simulator that is frequently used in research for testing intelligent racing agents [1]. Our focus is to develop three different intelligent agents that control cars in the TORCS platform given the current state. We investigate three scenarios, firstly, we are given a scenario where we need to race through tracks without any opponents. Secondly, we have to solve the same problem with opponents on the track. Lastly, we are given two cars on the same track to control instead of one.

2 Related Work

Modern games today have become very complex and realistic, this is due to the increasingly advanced techniques used to program the artificial intelligence in games. Many approaches have been developed in this area ranging from rule based approaches to machine learning approaches [2]. With the recent improvement in computation, genetic algorithms have become an increasingly researched topic [3]. More specific to our problem, neuro-evolution is a topic that is greatly used in the research community [4]. Neuro-evolution is a combination of Genetic Algorithms and Neural Networks combining two state-of-the-art methods used for Machine Learning. There are several implementations of Neuro-evolution, one which has been widely used and appraised is the NEAT algorithm [5]. Other research has been conducted on Online Reinforcement and heuristic based methods [6], [7].

Another topic which is continuously being utilized in modern games is Swarm-intelligence, which allows multiple agents to base decisions as a group and form strategies. Initial research was done one simple rule based swarms but this developed into more complex models that allow for more intelligent behaviour. Ant-colony optimization and Particle swarm optimization are two techniques that have obtained favorable results [8], [9].

3 Approach

Once a race is started on the TORCS platform, we receive the car state every few milliseconds. Every-time we receive the car state, we must react with a response indicating how to control the car. We are given 17 inputs from the car state and we must respond with 3 commands to control the car.

The approach chosen is one of neuro-evolution, which utilizes neural networks as required in the assignment documentation. We have identified a genetic approach because it is the most appropriate one in this case, as it does not require any training data. This is an important factor because the network must learn how to steer, accelerate etc. Also, the network would have to to learn to stay on the track and avoid possible collisions with other cars. The latter would require a huge dataset which we do not have.

We identified the NEAT algorithm as the most suitable algorithm for our project since it is proven success on the TORCS platform. We will try to teach the network how to race round a track using the data from car states. We are able to tune the generation of ANNs using hyperparamters of the NEAT algorithm such as the fitness function, number of layers etc. To further improve our implementation we have also identified and tested extensions to the NEAT algorithm such as HyperNEAT and ES-HyperNEAT. We have tried to train the models on different tracks.

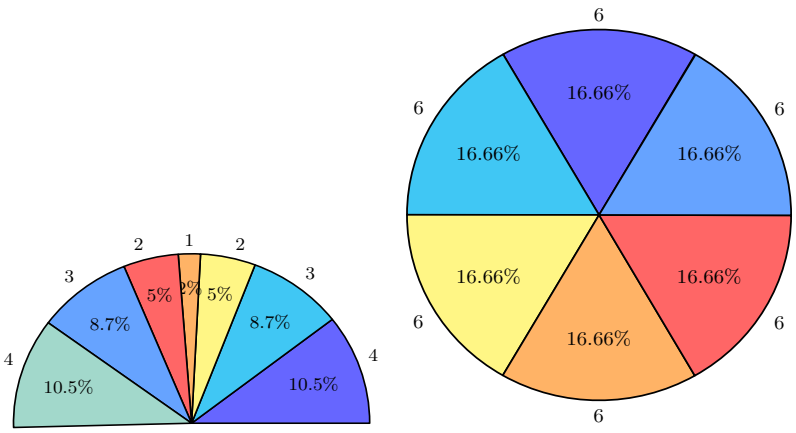

Fig. 1. Left: Grouping of track sensors **Right:** Grouping of opponent sensors

We have decided to use the Albourg track as it has many turns and is difficult enough for the models to learn in a reasonable time.

We have chosen some inputs based on our intuition and experience for the neural network. We are using the distance from the center, the speed in the x direction, the angle to the track axis, the set of grouped averaged sensors of the track and grouped averages for opponents sensors, Figure 1 shows how we decided to group the sensors. Specifically in order to make the network less complex so it can create strong rules we grouped the front(track) sensors in groups of 4-3-2-1-2-3-4 and so we did for the opponents we put them in groups of 6-6-6-6-6-6.

We have also applied some hard coded rules for when a car has unexpectedly crashed. Once the controller realizes that it crashed by using the front sensors, it will try to reverse until it is back on the road. Once the car is back on the road the Neural network controller is given back control.

3.1 PID Controllers

We have three major controls we can give to the platform, these are steering, acceleration and brake. For steering and acceleration we have performed tests using proportional–integral–derivative controller (PID). This allows us to define a target values and the PID adjusts the variables accordingly to move towards this target. We have used this constantly for acceleration and steering throughout all controllers.

3.2 NEAT Algorithm

We are required to use neural networks for our controllers, since we do not have a big dataset to perform supervised learning, we have identified NEAT [5] as the appropriate method of searching the search space of all possible neural networks. Neuro-evolution of augmenting topologies or NEAT is a genetic algorithm used to evolve artificial neural network by altering weighting paramters and network structures. We have defined different experiments where we change fitness functions, NEAT parameters and modifications in the input and output formats. After many generations, we have saved the most effective neural network depending on the fitness function. For each scenario, we have defined a specific fitness function, these are defined below.

The inputs of the network are the object sensors data, the speed, the distance from the center and in some cases the distance from opponents. The output of the network is the target speed, target position on track and the brake. The target speed is used in a PID loop to obtain this speed. The brake output from the network is fed directly to the control because it is easy to handle it as long as in most cases there is zero need for braking and never too much.

HyperNEAT Hypercube-based NeuroEvolution of Augmenting Topologies or HyperNEAT [10], is an extension of the NEAT algorithm that has proven to be effective in high dimension space. HyperNEAT uses a primary network to generate the weighting parameters and structure of the predictive secondary network.

ES-HyperNEAT Despite its novel capabilities, a significant limitation with the original HyperNEAT is that the user must literally place hidden nodes at locations within a two-dimensional or three-dimensional space called the substrate. This was solved with another iteration of the NEAT algorithm called ES-HyperNEAT [11], which decides on the placement and density of hidden neurons without any addition representation beyond the traditional HyperNEAT CPPN.

3.3 Part 1: Basic controller

The initial basic controller is trained to be able to race through the track as quickly as possible without incurring damage. We have used the NEAT algorithm described above to produce ANN that is evaluated by using a fitness function. The fitness function categorizes the models into three one group of crashed cars that did not finish the race giving it a -100000 fitness, a group of crashed cars that managed to finish giving them a -10000 + the distance raced as a fitness. Finally there is a group of cars that finished the race without crashing. The fitness function of this group is calculated by the average speed. This is done for 500 generation and then the fittest function of that generation is taken as the best current model.

3.4 Part 2: Controller that races other cars

For this part, apart from racing a track, we also have to deal with opponents and try to win the race. To find an ANN suitable for this problem we have used the best controller from part 1 as an initial network. We then use the NEAT algorithm as before but now include opponent sensors as an input and a apply a reward for having a better position in the race and avoiding to be close with the cars behind. The outputs of the function stays the same. This is repeated for another 500 generations and the best controller is selected.

3.5 Part 3: Team based racing

For the Part 3 we will be allowed to use two cars in the race and apply strategies such that one car is first in the race. To be able to do this we must have two control sequences, one that tries to win the race and the other to impede opponents from racing. Since we can only submit one controller we also have to classify which car will apply what strategy. This is done by looking at the race position and allowing the car with the highest position to try and win the race while the car with the lower position to impede the opponents. The controller to win the race will be the one obtained from part 2, while the one to impede the opponents will be trained with a fitness to reward when the back opponent sensors are activated.

4 Evaluation and Results

We have set up different tests to evaluate our methods.We have first compared the Part 2 controller by comparing the lap times obtained with that of the initial driver given with the assignment. Subsequently, we evaluate the last controller by looking at the lap times of the opponents with and without the car that tries to impede them from finishing the race.

	W/O opponents	**With 8 opponents**	**Average speed (Km/h)**	**Default Driver Time**
CG Track 2	1:23	1:24(5th)	140	2:27
CG Speedway 1	0:59	0:58(5th)	129	1:36
Ruudskogen	1:45	1:45(6th)	112	2:31
Alpine 1	3:35	3:26(5th)	107	4:49
Aalborg	1:45	1:50(4th)	92	2:18
C-Speedway	1:10	1:10(4th)	170	3:01
Dirt 6	2:46	2:30(6th)	71	3:01

Table 1. Table of lap times for our best performing driver with the default driver.

Table 1 shows the lap times obtained without opponents and with opponents, the average speed of the car and the default driver time. First by comparing the lap times of our controller(no oppononents) with the default car, we see major improvements. This can be of-course attributed to our fitness function which rewards high average

speed and keeping the car on the road by penalizing crashed and stuck cars.

Then, comparing the times of our controller when racing against the clock and with opponents, it is observed that both times are similar. Though, slighting improvements are observed to our controller when racing with opponents. This is due to ANN which trying to overpass the opponents by increasing speed when the sensors spot "enemies". We can also see that our controller tries to obtain a high average speed where possible. This can be attributed to the fitness function chosen which rewards faster cars.

	Only Bots (easy)				**With Swarm (easy)**				**Only Bots (hard)**				**With Swarm (hard)**			
	Bot 1	Bot 2	Bot 3	Bot 4	Bot 1	Bot 2	Bot 3	Bot 4	Bot 1	Bot 2	Bot 3	Bot 4	Bot 1	Bot 2	Bot 3	Bot 4
E-track 4	2:16	2:19	2:20	2:22	2:00	2:21	+1 lap	+1 lap	2:14	2:19	2:31	2:40	2:17	2:27	2:34	+1 lap
Ruudskogen	1:17	1:24	1:26	0	1:17	1:24	+1 lap	+1 lap	1:24	1:25	1:26	1:32	1:24	1:30	1:42	+1 lap
Alpine 1	2:38	2:39	2:54	2:55	2:38	2:39	+1 lap	+1 lap	2:54	2:55	2:56	2:59	2:54	2:55	2:56	4:31
Aalborg	1:23	1:30	1:31	1:32	1:23	1:30	+1 lap	+1 lap	1:30	1:31	1:32	1:39	1:30	1:31	1:32	+1 lap
Corkscrew	1:39	1:40	1:43	1:45	1:39	1:40	+1 lap	+1 lap	1:44	1:45	1:46	1:50	1:44	1:49	1:52	+1 lap

Table 2. Swarm

Table 2 shows a comparison between lap times of opponents before and after the addition of the second model that tries to obstruct opponents from completing the race. In almost all tracks we can see that our approach to slow down opponents has been effective. In most cases, opponents are slowed down for a whole lap. On the other hand we have noticed that some opponents are not effected as they are always in front of the car and therefore it is not able to block them.

5 Discussion

The good results obtained from the first controller can be attributed to the different factors applied to our fitness function. The first part of the fitness function rewards cars that manage to drive a greater distance. We determined that this enables the controller to learn basic driving skills such as basic steering, accelerating and braking. The second part of the fitness function rewards higher average speeds which we have determined to be the cause behind the controller learning to accelerate to higher speeds. Finally, the last part of the fitness function which penalizes cars which crash and drive out of the track, gave insights to the network to keep the car on the track this is by adjusting the acceleration, brake and steering especially when a corner is closeby.

After several successful tests racing round tracks, we introduced the opponents to the equation. This also introduced more variables to the fitness function and more inputs to our network. We adjusted the fitness function to reward higher positions on the race and also reward when the car moves away from opponents that are activating the back sensors. We have also introduced penalties for crashing into opponents. This lead to the controller becoming more competitive by accelerating and steering in a more realistic pattern.

Lastly, we trained another model to be used when we are given two cars to control. This controller will be used to slow down other opponents from winning the race. We have done this by rewarding when the back opponent sensors are activated. This showed great results instantly as the controller quickly learned to obstruct opponents and slow them down. Most of the times leading to a full lap of delay for 2 opponents.

6 Conclusion and Future Work

From our models, we can conclude that the ES-HYPERNEAT algorithm is indeed a good tool to find neural networks for controlling cars in video games. We can say that the fitness function is amongst the most influential factors in the training process.

Our results show a good generalization from training on just one track. Our controller manages to race through all tracks tested in a reasonable time. We believe that to further improve our model we could have trained the network on different tracks to help the network generalize better.

Controller two cars to facilitate one car winning also obtained good results. Using such a simple fitness function we have already managed to obstruct opponents. Other work can be looked at such that the two cars can communicate and then can perform better obstructions to opponents. Nonetheless, our simple approach has obtained results which are sufficient for most cases.

Finally, we suggest that more research can be conducted in developing a fitness function that can teach the controller to follow the race line. The race line is the best path to follow when racing cars, this allows the car to steer as less as possible and accelerate as much as possible. This is one of the many optimization we can do to further our position on tracks.

References

1. C. G. C. D. R. C. A. S. Bernhard Wymann, Eric Espié, "TORCS, The Open Racing Car Simulator," http://www.torcs.org, 2014.
2. S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," *CoRR*, vol. abs/1410.7326, 2014. [Online]. Available: http://arxiv.org/abs/1410.7326
3. L. Davis, "Handbook of genetic algorithms," 1991.
4. D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.
5. K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.
6. K.-J. Kim, J.-H. Seo, J.-G. Park, and J. C. Na, "Generalization of torcs car racing controllers with artificial neural networks and linear regression analysis," *Neurocomputing*, vol. 88, pp. 87–99, 2012.
7. L. Cardamone, D. Loiacono, and P. L. Lanzi, "On-line neuroevolution applied to the open racing car simulator," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on.* IEEE, 2009, pp. 2622–2629.
8. M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.
9. J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning.* Springer, 2011, pp. 760–766.
10. K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial Life*, vol. 15, no. 2, pp. 185–212, 2009, pMID: 19199382. [Online]. Available: https://doi.org/10.1162/artl.2009.15.2.15202
11. S. Risi, J. Lehman, and K. O. Stanley, "Evolving the placement and density of neurons in the hyperneat substrate," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation.* ACM, 2010, pp. 563–570.