Comparing the Performance of Finite-State Machines with Different Numbers of States on TORCS

Bruno H. F. Macedo
Gabriel F. P. Araujo
University of Braslia
Technogy Faculty (FT)

Gabriel S. Silva
Matheus C. Crestani
Yuri B. Galli
University of Braslia
Technogy Faculty (FT)

Guilherme N. Ramos
University of Braslia
Professor at *Department of Computer Science* (CIC)

Abstract

Alterar cabecalho do programa para aceitar portugues

This work presents two different approaches developed to control a self-driving car in the racing environment simulated by TORCS. The problematic revolves around the representation of the complex behaviour that describes a pilot during a race, whose proposed solutions are two different finite-state machines; while one method treats the problem with a more partitioned approach, the other tries to generalize the possible situations. Furthermore, in order for the final controller to attain competitive status, i.e. parameter fine tuning, the incorporation of a genetic algorithm was performed. Evolving the driver using this method initially required the definition of a set of parameters, where the optimization lied effectively. Validation occurred through exhaustively testing the configurations of the two approaches, comparing them along with artificial intelligences provided by the platform and also State of the Art implementations.

Keywords: finite-state machines, computer games, TORCS, SCRC, artificial intelligence, genetic algorithm, self-driving car

Author's Contact:

{name1,yurigalli}@gmail.com
name2@zzzz.vvvv.vvv
Colocar os e-mails. Devem ser os academicos?

1 Introduction

Automation of day to day tasks is an endeavour that has moved a large amount of scientific resources in the recent history. One specific example is target of studies around the globe by a lot of universities, companies and industries, which is the automation of vehicles, more specifically, automobiles. The objective of such attempts is the development of artificial intelligences capable of driving a car safely, with traffic law enforcement, real-time decision making, efficiency and, in addition, resource economy - as with gas or even time. The practical applications of such controllers in autonomous vehicles are truly numerous, for example, the researches pursued by DARPA [DAR 2015].

The Simulated Car Racing Championship (SCRC), using the platform TORCS (The Open Racing Car Simulator), has brought an excellent environment for benchmarking AI approaches for the problem of autonomous car controllers. Notwithstanding, the optimum behaviour of a controller is a complex matter in its full extent; for this purpose, the strategy adopted to deal with it was to divide the problem into smaller portions, i.e., less complicated subproblems, in order to implement a finite-state machine that admittedly covers all necessary behaviours. Later on this paper, each of those subproblems are treated as the states of the referenced finite-state machine, which individually incorporate different but complementary parts of the integral behaviour.

One way to enhance the performance of the created controllers is by comparing each and every possible set of parameters or configurations it could assume, but that choice is not always possible due to time and space complexities. For that reason, although hand-coded methods might present satisfactory outcomes in questions of such intricacy similar to the one discussed in this paper, they will hardly ever outperform the ones that are computer-aided on account of the incisive and indefatigable pace that the latter perform the task, which results in more possibilities tested. Considering this, after an initial structure of the controller was designed, a method of computer-aided fine tuning was assimilated to it, which was a genetic algorithm.

Apart from this Introduction, this paper is structured as follows: Section 2 introduces TORCS, the working environment used in the context of this task, along with the competition, SCR Championship, that currently represents the utmost metric to evaluate the performance of the controller proposed, and what is already being done at this context in related works, highlighting the strategies that are standing out - State of the Art - and analyzing the characteristics responsible for it; Section 3 then explains the proposal of the two developed controllers, clarifying their behaviour and structure, and briefly describing how the were augmented by the computer-aided method of a genetic algorithm; Section 4 describes how the validation process occurred, through the methodology, the experiments and the results achieved, including their correspondent analysis; Section 5 provides conclusions about the acquired results, which establish the comparison between the finite state machines with few and with moderate number of states, pointing out prospects about what is yet to be done in future works to improve those results.

2 TORCS Environment and SCRC

The Open Racing Car Simulator (TORCS) [Wymann et al. 2014] is a widely used platform for benchmarking AI, renowned for its highly credible physics modeling engine and yet user-friendly interface for car racing simulation. One of the many other qualities of this open-source simulator is its portability, concerning multi-platform environments - such as different operational systems and architectures - and multi-programming language ones alike.

2.1 TORCS as platform, interface and environment

TORCS deals with a robotic simulation in the format of car racing. It serves both as an ordinary AI game and a research platform [Loiacono et al. 2010], providing through the interface with its competitions a complete sensor-based interaction system in which the developer is able to interpret received parameters of the car - such as speed in X, Y and even Z axes - and construe them in order to control the car through programming its actuators, some of which are acceleration and steering. This behaviour is widely embraced when it comes to artificial intelligence methods - a set of inputs to result in a set of outputs.

Another credibility factor for this platform is its non-punctual cars, each of which possess a body with sensors and actuators, as mentioned earlier, and interact with other cars in the environment by a life-like collision system. Nevertheless, TORCS is still a simulator, and its limitations, along with the defined racing environment and the modeled robotic car, are more than likely to affect any results obtained. This is an inherited characteristic of any real-life problem simulation, what in academia is denominated *reality gap* [Whitton and Moseley 2012, ch. 8], and it stems from the simplifications attained concerning the car models, the technical features of the tracks, and so forth.

2.2 The SCR Championship

The Simulated Car Racing Championship (SCRC) [Cardamone et al. 2014] is an example of a well-known competition which utilizes TORCS as interface. Being an event joining three competitions held at major scientific conferences, such as *IEEE Congress on Evolutionary Computation* [CEC], *Genetic and Evolutionary Computation Conference* [GEC] and *IEEE Conference on Computational Intelligence and Games* [CIG], it is an accepted metric of evaluation in the fields of Evolutionary Computation and Computational Intelligence regarding Games.

The SCRC features encapsulation with its interface towards TORCS, meaning there is an abstraction layer between the two and, in addition, some of the information about the racing execution remains hidden from the controllers, such as the geometrical format of the track and its category. SCRC and TORCS communicate through UDP packages, and each player receives information regarding the sensors of the car, some of which were already presented earlier on this section.colocar imagens ilustrativas

A brief description of the sensors is as follows: the driver can use a set of sensors to acquire relative position, speed, and car condition such as rpm, damage and gas. Useful information about the car during a race can be mined from the data provided by them; for example, there are two sets of sensors labeled *track* and *trackpos* that inform the position of the racer according to a desired direction and according to the track track axis, respectively, and these two sets can be combined to inform a approximated location of the car.

The complete sensorial input information can be found at the Simulated Car Racing Championship Competition Software Manual [SCR 2013], within section 7.7. Noise can be incorporated to the aforementioned sensors, option that is inexorably present during the actual competition and will be dealt with in the section Future Works.

Race tracks differ from each other concerning sensorial input if they are part of different group types, which are categorized into **Road**, **Dirt** and **Oval**. The races from the SCRC select the track types randomly, and adopt the following structure:

- *Warm-up* - when each driver is able to explore the track and deduce information from it at will, for a limited time on their own. This encourages the application of online learning techniques that can be useful in a race on the same track;
- *Qualifiers* - a stage placing each driver in a race alone against the clock to select the best ones to compete in the *Actual Race*;
- *Actual Race* - when finally the eight best drivers from the *Qualifiers* race against each other.

The reason why TORCS presents itself as a satisfactory AI benchmark, in combination with SCRC, is because even though there is an infinity of possibilities on how the sensorial input received from the server can be translated into the behaviour of the actuators, they can all be compared in a race, which has a robust and steady scoring and evaluational system. In other words, there are many different approaches concerning how to teach the racer encoded by the developers to drive in a racing competition only with the information given by the sensors, and the metric to that issue is the performance on the race itself.

2.3 Related Works and State of the Art

By controller, let it be understood that the subject is the programming code that in fact controls the car/driver/racer within racing environment. Some examples of awarded controllers and their driving methods will be presented in this subsection. They are what can be called the State of the Art among TORCS, and it is very common among them the incorporation of machine learning methods, along with other evolving techniques using artificial intelligence. Instinctively, as the nature of the problem comprises evolution by experience, learning procedures tend to enhance performance and competitiveness. Essentially, there are two ways of evolving controllers: Online Learning and Offline Learning, the first meaning that improvements are achieved during the actual race execution time and the latter that it is done before the competition, on the account of the developers themselves and with their own resources.

The current champion of the SCR Championship is the controller *Mr. Racer* [Quadflieg et al. 2014], and it has proven to be the State of the Art by winning at least the last three competitions that happened. The authors of this implementation learn parameters offline through Covariance Matrix Adaptation Evolution Strategy (CMA-ES), use regression and low-pass filtering to reduce noise impact, distinguish normal asphalted roads from dirt-based ones for behavioral separation and implement an authentic opponent-handling method. Their Online Learning consists on the track model selection to categorize into dirt or asphalt, choice of databased sets of parameters that best fit the track and the tuning of a target speed for all its corners.

Another renowned controller is *AUTOPIA* [Onieva et al. 2012]. According to the founders of the competition [SCR 2013] and the authors of *Mr. Racer* themselves, it is a competitive match, with the potential to even be the best one available, but since no entries were received from them in a while, their means of winning a competition were somewhat restrained. Nevertheless, assessing its performance is worthwhile, and its description is the implementation of a modular Fuzzy Architecture, whose division contains gear, steering and speed control. Their controller is optimized by means of a genetic algorithm for Offline Learning, and by means of landmarking the lane exit points for further speed reduction for Online Learning.

These and other controller exemplifications served as criteria for the analysis and development of the approach presented in this paper. Aspects incorporated and adapted feature modularity, offline learning through genetic algorithms, online learning through landmarking and choosing sets of parameters for different categories of tracks, etc. Adicionar o quł mais fizemos no projeto como pincelada inicial para fazer o gancho com a seo Controller Structure.

3 The FSMDriver

Some of the main aspects discussed to outline a controller for TORCS were sufficed by the concept of finite state machines (FSM) [Millington and Funge 2006], the most important one being the goal of reaching an autonomous driving behaviour in a car race.

3.1 Five-state FSM

According to Mat Buckland in *Programming Game AI By Example* [Buckland 2005]:

> *"A finite state machine is a device, or a model of a device, which has a finite number of states it can be in at any given time and can operate on input to either make transitions from one state to another or to cause an output or action to take place. A finite state machine can only be in one state at any moment in time."*

This architecture was chosen in order to transform the problem of complex driving into smaller problems of situations found within the racing environment. Initially, the design of the finite-state machine proposed comprised the following states:

- *Straight Line*;
- *Approaching Curve*;
- *Curve*;
- *Out of Track*;
- *Stuck*.

Essentially, for this method, normal behaviour covered *Straight Line*, *Approaching Curve* and *Curve*, as the controller was located inside the track boundaries and no recovery actions needed to be considered, whereas exception behaviour consisted of *Out of Track* and *Stuck*, situations in which such conduct was expected. Here, a consideration needs to be taken into account: the real first model of the finite-state machine did not have an *Approaching Curve* state, but, as the interpretations of the demeanour concerning the *Straight Line* and the *Curve* were so different from one another, a preparation had to be established so as to smoothen the transitions between them.

If the controller was currently in *Straight Line*, it would be expected of him to simply go as fast as he could, with no steering changes whatsoever; when in *Approaching Curve* state, he would reposition himself in relation to the track and recalibrate his speed in order to make better curves, which is done by steering the car towards the direction of the sensor with the biggest read value and braking until a proportionally calculated target speed is achieved; or, when in situations of *Curve*, the pilot would stop braking while maintaining the steering direction towards the sensor pointing the biggest distance value read - which represents the direction of the curve, prepared by the approaching curve state. For the exception states, even though the expected deportment is well known, e.g. a stuck controller should maneuver the car out of the current situation and proceed to normal race conduct, the implementations vary among developers. The strategy chosen for the exception states was used on both proposals as a matter of regularity of comparison, and is explained on the next subsection.

One big problem about this way of treating the matter is that the function responsible for choosing which state is more appropriate for each situation would more than often be overcharged, and, in some cases, rather different sets of parameters received by it would result in the same classification among the states. Thus, in order to minimize the dependency of the driving performance in relation to the function in charge of the transition between states, a project decision was made to reduce the number of states.

3.2 Three-State FSM

The very nature of the initial architecture gave rise to the new approach. As mentioned within this very section, three of the states were innately part of a common bigger state, thereby *Straight Line*, *Approaching Curve* and *Curve* summed into *Inside Track*, resulting in the new controller with only three states, which were:

- *Inside Track*;
- *Out of Track*;
- *Stuck*.

Inside Track, therefore, is how the car, desirably, will spend most part of the time. The controller assumes a position of increasing the speed while steering towards the sensor with the biggest read value. This state also brakes, if necessary, when approaching turns.

Out of Track is when, for any unknown reason, the car is found outside of the track limits. In this case, the proper behaviour is to try to return to the lane. In road tracks, the outside track normally has a different terrain, sometimes dirt-based, meaning that skidding frequently occurs, and in an effort to avoid this, a control system to brake when the car begins skidding above a threshold was implemented.

Stuck represents any given situation that the car is unable to progress in the race. This is a delicate state, because it presents itself as difficult to identify and also due to its impact to the performance of the controller. In order to detect *Stuck* circumstances, the speed of the car is monitored throughout the race, during every game tick, if it lingers with a low speed for a determined period, then it is considered stranded, or stuck. When detected, this state activates the reverse gear of the car and turns it until its front is directed towards the correct axis of the track. The reason why *Stuck* is a sensitive state is because, when detected early, might indicate false positive, and when detected late, could lessen the efficiency of the controller. Thusly, detecting *Stuck* situations is crucial, and so is handling the car out of them.

In conclusion, each state in this manner of dealing with the process becomes, ideally, an independent problem, whose solution can be attacked separately. This way, they can all have individual sets of parameters susceptible to improvement, which will be discussed in the next section.

3.3 Search for Parameter Values - Genetic Algorithm

Due to the quantity of parameters to be tuned and the defined granularity, the search space becomes enormous and renders fundamental the use of a search algorithm, which optimizes the process of finding better configurations by being more incisive and saving resources such as computational time and space. In the present study, an evolutionary algorithm was chosen for this task.

A genetic algorithm [Holland 1984] is an evolutionary algorithm inspired by nature, in special by the concept of evolution through natural selection [Darwin 1859], whose main idea is that a set of solutions for a problem can be evolved like the population of a generic species in nature. The applications of genetics algorithms are present at many areas, extending from control engineering at non-linear system identification to biomedicine prosthesis development and even economy to forecast the behavior of agents.

In this context, a viable solution is called an individual and can be represented by a string of parameters; each individual then becomes an array that contains the information necessary to evaluation. The first population is instantiated randomly to its full extent, because the search space is too big for an initial attempt at a good possibility. This population passes through a fitness function that indexes a score to each individual. This function is responsible for assessing how good - or how adapted - the solution that is being evaluate is. After evaluating the population separately, a group of individuals is chosen as the parents of the next set of solutions, which will compose a new generation; there are countless ways of performing the selection of the parents to the new generation of offspring, and this work gave preference to picking the higher individuals on the scoring system, what is called Elitismreferencia para elitismo. Each pair of parents is submitted to crossover in order to generate two offspring solutions, and in the end of the process each offspring may present mutation - everything according to predefined rates. For this work, the crossover rate is 95%, while the mutation rate is 1%, which are values proposed by Nnez-Letamendia, reference.

For the Five-state FSM, 22 parameters required adjustment, which originated, in different quantities, from each state separately and also the *transition* function, as follows:

- *Approach Curve* has 4 parameters;
- *Transition Function* has 3 parameters;
- *Straight Line* has 4 parameters;
- *Out of Track* has 7 parameters;
- *Stuck* has 4 parameters.

However, for the Three-state FSM, only 19 parameters demanded adjustment, which are divided as follows:

- *Inside Track* has 8 parameters;
- *Out of Track* has 7 parameters;
- *Stuck* has 4 parameters.

Theoretically, as the search space for the latter case is smaller, finding a better controller was expected to happen first for it, but only by effectively testing both architectures could such a result be achieved. The source codes for both models presented in this section are available at the *GitHub* repository provided in the references [Git 2015]. Devemos fazer isso, certo? Reprodutibilidade.

4 Experimental Results

A veil was put over the Five-state FSM from the very beginning of the experimentation phase, which was its great dependency towards the transition function. Early superficial assessments of the performance of this first model indicated an overcharge concerning this function, which was verified by considerable changes in behaviour derived from adjustments in its parameters. For that reason, a second model with less states - the Three-state FSM - was designed regarding this characteristic and releasing part of the performance burden from the transition function, and the results from the comparison of these architectures are described and analyzed in the succeeding subsections.

4.1 Methodology

Once defined the models and which of their parameters required tuning, the genetic algorithm was applied to each one separately, in order to adjust their configurations for superior and competitive status. At the same time, the goal was to find general and versatile controllers with good results for any track and specific ones that were fittest to race in various tracks: road, dirt and oval alike. Therefore, due to the differences observed in the parameters of controllers evolved on dirt and road tracks, the evolution process took place separately for those two kinds of environments, which produced two contrasting sets of enhanced values for each model.

The *metric* chosen to evaluate the generated controllers was the combined sum of the distance raced by the car alone in the first 10.000 game cycles - also called *game tics* - of a list of mixed tracks. This value will henceforth be called the *fitness* of the controller, as it was used to determine whether he would remain in the evolution process.

The experiments concerning Oval Tracks would repeatedly provide inconclusive results, so they were neglected in this evaluation process. Thus, in order to find the best set of parameters for a general track, the two finite-state states were evolved in three different sets of tracks, one with 4 Dirt Tracks, one with 4 Road Tracks and another with the 4 of each type. The evolution process for each set of tracks consisted on 600 generations of 30 individuals and culminated in one controller; in other words, At the end of the experiments, there were six evolved controllers, one specific for Road Tracks for each model, one specific for Dirt Tracks for each model, and also one evolved in a mixed manner for each model.

The validation occurred through testing the six produced pilots in a predefined set of tracks different from those in which they were evolved, lest they would evidence too track-limited parameters. On behalf of comparison, the results from the AUTOPIA controller were incorporated in the analysis, since it can be considered the State of the Art, displaying one of the best performances for the SCR Championship, and should provide satisfactory basis for appraisal.

4.2 Results

4.3 Analysis

5 Conclusions

5.1 Conclusions

5.2 Future Works

Trocar de topicos para texto:

- Since at the SCRC the controller is allowed to perform a warm-up before the race it is possible to acquire the track data, not only mapping critical section such as sharp curves and points where the car go out the track to improve the result of the controller at the race itself. Also a warp-up stage would supply the information about environment where the car is, including the type of track,road or dirt, which determine a set of parameters best fitted to which occasion.

- One important task to be accomplish is the opponent treatment, routines to reduce collisions, avoid the controller to be surpassed and surpass the opponents is a fundamental issue. Ignoring the opponent would make the driver to face unexpected collisions ending stuck, considering the worst event.

References

BUCKLAND, M. 2005. State-driven agent design. In *Programming Game AI By Example, 2nd ed.*, W. Publishing, Ed. Wordware, Plano, TX, ch. 2, 43–45.

CARDAMONE, L., LOIACONO, D., AND LANZI, P. L. 2014. *Simulated Car Racing Championship - Competition Software Manual - `http://arxiv.org/pdf/1304.1672.pdf`.*

2015 IEEE Congress on Evolutionary Computation (CEC2015). `http://www.cec2015.org/`. Accessed: 2015-05-14.

2015 IEEE Conference on Computational Intelligence and Games (CIG 2015). `http://www.ieee-cig.org/`. Accessed: 2015-05-14.

2015. Darpa. Website: `http://www.darpa.mil/our-research`.

DARWIN, C. R. 1859. On the Origin of Species. In *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life.*, J. Murray, Ed.

2015 Genetic and Evolutionary Computation Conference (GECCO 2015). `http://www.sigevo.org/gecco-2015/`. Accessed: 2015-05-14.

2015. GitHub repository for the FSMDriver. Website: `https://github.com/bruno147/fsmdriver`.

HOLLAND, J. H. 1984. Genetic Algorithms and Adaptation. In *Adaptive Control of Ill-Defined Systems*, vol. 16 of *NATO Conference Series*, 317–333.

LOIACONO, D., LANZI, P. L., TOGELIUS, J., ONIEVA, E., PELTA, D. A., BUTZ, M. V., LNNEKER, T. D., CARDAMONE, L., PEREZ, D., SEZ, Y., PREUSS, M., AND QUADFLIEG, J. 2010. The 2009 Simulated Car Racing Championship. *IEEE Transactions on Computational Intelligence and AI in Games* (June).

MILLINGTON, I., AND FUNGE, J. 2006. Decision making. In *Artificial Intelligence for Games, 2nd ed.*, Elsevier, Ed. Morgan Kaufmann Publishers, San Francisco, CA, ch. 5, 309–311. Section 3.

ONIEVA, E., PELTA, D., GODOY, J., MILANÉS, V., AND RASTELLI., J. P. 2012. An Evolutionary Tuned Driving System for Virtual Car Racing Games: The AUTOPIA Driver. *International Journal of Intelligent Systems, Wiley-Blackwell.*

QUADFLIEG, J., PREUSS, M., AND RUDOLPH, G. 2014. Driving as a human: a track learning based adaptable architecture for a car racing controller. *Genetic Programming and Evolvable Machines - Springer Science+Business New York.*

2013. SCR Championship State of the Art. `http://pt.slideshare.net/dloiacono/gecco13scr?related=1`.

WHITTON, N., AND MOSELEY, A. 2012, ch. 8. Authentic contextual games for learning. In *Using Games to Enhance Learning and Teaching: A Beginner's Guide, 2nd ed.*, N. Whitton and A. Moseley, Eds. Routledge, New York, NY.

WYMANN, B., ESPIÉ, E., GUIONNEAU, C., DIMITRAKAKIS, C., COULOM, R., AND SUMNER, A., 2014. TORCS, The Open Racing Car Simulator. `http://www.torcs.org`.