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

Bruno H. F. Macedo Gabriel S. Silva Yuri B. Galli
Gabriel F. P. Araujo Matheus C. Crestani Guilherme N. Ramos
University of Brasília

Abstract

This work presents two different approaches developed to control a self-driving car in the racing environment simulated by TORCS. The problematic is the understanding of the complex behavior that a pilot assumes during a race, and the proposals for dealing with it are two finite-state machines that reproduce this behavior. The evaluation of the methods proposed provided a comparison between finite-state machines with different numbers of states, and by the experimental results it could be inferred that using fewer states with this technique leads to improved performance in simulated races from TORCS.

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

Author's Contact:

{bhfmacedo}@gmail.com
gnramos@cic.unb.br

1 Introduction

Automation of day to day tasks is an endeavour that has moved a large amount of scientific resources in the recent history [Terzic et al. 2008] [Bejczy 2012]. One specific example is target of research 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 [Carbaugh et al. 1997], with traffic law enforcement, real-time decision making, efficiency and, in addition, resource economy - as with gas, pollution emission [Barth 2013] or even time. The practical applications of such controllers in autonomous vehicles are numerous, for example, the researches pursued by DARPA [1].

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 [Loiacono et al. 2010]. Even with the advent of computer simulations, finding the optimum behaviour of a car controller is a complex matter, so, many approaches have been suggested, such as heuristic algorithms [Quadflieg et al. 2014] and modular and fuzzy architectures [Onieva et al. 2012]. The strategy adopted in this work 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.

One way to enhance the performance of the controllers created is by evaluating 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. 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.

The rest of this paper is structured as follows: Section 2 introduces TORCS, the working environment used in the problematic presented, along with the competition, SCR Championship, that currently represents the metric to evaluate the performance of controllers proposed for this environment; the section also presents what is already being done at this context in related works. Section 3 then explains the proposal of the two developed controllers, clarifying their behaviour and structure, and briefly describing how they 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 results acquired, which establish the comparison between the finite state machines with few and with moderate number of states, pointing out prospects about what might be done in future works to improve those results.

[1] http://www.darpa.mil/our-research

2 The Simulation Environment

Multiple applications that require verification and evaluation have become dependent of expensive resources, which makes their development a delicate matter. Examples of systems whose tests involve costly supplies are medical education [Zhang et al. 2007], the aviation industry [Duncan and Feterle 2000] and automotive research and development [Xue et al. 2011]; because naturally no company or enterprise has expendable numbers of bodies for tests, or airplanes and cars to crash in order to safe-proof and improve. In these conditions computer simulations arise, virtually reproducing the environments of the said and of other important applications so that their testbeds practically eliminate the need of the valuable resources related to them.

Many other reasons justify the use of virtual simulations, varying from one application to another, and, because of this, countless companies are incorporating them in their educational and training process [Sim 2015] [Ope 2014]. A few number of reasons why a simulation was used in this work:

- control and operational management;
- planning actions;
- understanding a problem and how to react to it;
- training and learning through experimentation.

Simulations represent just an approximated model of the systems they try to imitate, but in most cases the effects of the estimates adopted in the attempt to do this do not come to the point that the model becomes invalid. For example, a simulation that reproduces a surgery environment may fail to accurately determine the tension of the skin of a patient into whom it wants to make an incision, but that does not mean that the whole process is compromised by just this approximation. Similarly, car racing simulations also incorporate flawed modeling - regarding friction, gravity, trajectory lines, and so on - but resorting to them may save some spare parts that would most likely be jeopardized in real-life tests. So, new studies, new researches and new automation methods are able to be evaluated through simulations and avoid costs in the development process.

2.1 TORCS

The Open Racing Car Simulator (TORCS) is a platform that is renowned for its highly credible physics modeling engine and yet user-friendly interface for car racing simulations [Wymann et al. 2014]. It is a modern, modular, multi-player and multi-agent car simulator; it also is an interface that is a widely used for benchmarking AI [Loiacono et al. 2010] due its high degree of modularity and portability, concerning multi-platform environments - such as different operational systems - and support to the programming languages C and C++. Artificial intelligent agents can be developed as modules inside TORCS where there are many possible levels of abstractions. For example, at the car level, there is and intelligent

control system for each car component; at the driver level, mid-level control systems for complex driving agents could be implemented using the partial simulation information given by a low-level API provided by TORCS.

The TORCS engine uses a discrete-time simulation, whose discretization is set to 0.002s of simulation time. The engine solves differential equations using Ëuler method, and all basic elements of the vehicular dynamics are handled, which are:

- basic properties of the vehicular system;
- mechanical details;
- dynamic and static friction;
- aerodynamic model.

Mass, rotational inertia of the car, engine, wheels, and other components, are included in the model of the vehicular system. The types of different suspension, links, and differentials, in the mechanical model. The profile for different ground types with each dynamic and static friction are also included. The aerodynamics modeling includes slipstreaming and ground effects. Nevertheless, the simulation engine can be replaced or easily modified as a result of the modularity of TORCS.

TORCS is a computer game that serves both as an ordinary car racing game and as a research platform, making it possible for everyone to create a pilot through coding. The interface with this platform occurs by means of a 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 control the car through programming its actuators, some of which are acceleration and steering.

Another credibility factor for this platform is its non-punctual cars, which interact with each other in the races by a life-like collision system. Nevertheless, TORCS is still a simulator, and its limitations, along with the defined racing environment and the modeled 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 made concerning the car models, the technical features of the tracks, and so forth.

2.2 The SCR Championship

The Simulated Car Racing Championship (SCRC) is an example of a well-known competition which utilizes TORCS as interface [Cardamone et al. 2014]. Being an event joining three competitions held at major scientific conferences, such as *IEEE Congress on Evolutionary Computation* [2], *Genetic and Evolutionary Computation Conference* [3] and *IEEE Conference on Computational Intelligence and Games* [4], it is an accepted metric of evaluation in the fields of Evolutionary Computation and Computational Intelligence regarding Games.

In the SCRC, some of the information about the racing execution remains hidden from the controllers, such as the geometrical format of the track and its category. The communication between this championship and TORCS is made through a client-server interface, with each player receiving information from the server regarding the sensors of the car, and in return providing actuator values that determine how the controller is supposed to drive the car. Figure 1 illustrates the data available at the TORCS and client - controller - layer of abstraction.

The complete sensorial input information can be found at the Simulated Car Racing Championship Competition Software Manual [SCR 2013]. Noise can be introduced in the sensors, option that is present during the actual competition.

Race tracks are categorized into *Road*, *Dirt* and *Oval* inside TORCS. The races from the SCRC take place in track types decided by the organization of the championship, information which is not provided to the participants and that may incorporate maps that are unknown to them. The competition adopts a structure that gathers a *Warm-up* stage, a *Qualifier* stage and a *Final* race, which are described in detail in a website of the competition [SCR 2013].

[2] http://www.cec2015.org/
[3] http://www.sigevo.org/gecco-2015/
[4] http://www.ieee-cig.org/

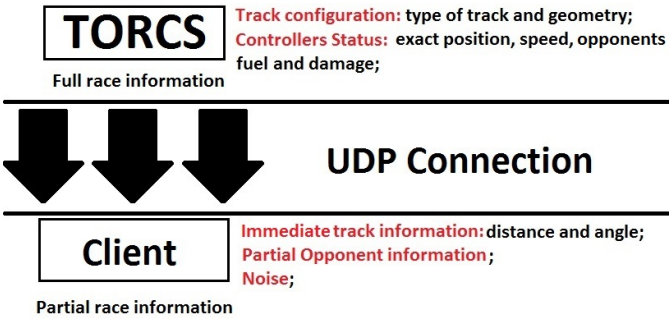

Figure 1: *Available data inside TORCS and at the client*

The reason why TORCS presents itself as a satisfactory AI benchmark, in combination with SCRC, is because even though there are multiple possibilities on how the sensorial input received from the server can be translated into the behavior 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

It is very common among some of the SCRC awarded controllers the incorporation of machine learning in their driving methods, along with other evolving techniques using artificial intelligence. As the nature of the problematic presented comprises evolution by experience, learning procedures tend to enhance performance and competitiveness. Essentially, there are two ways of evolving learning solutions: Online Learning and Offline Learning.

According to Tom Mitchel in *Machine Learning* [Mitchel 1997]:

> *"Systems that learn by moving about the real environment and observing the results are typically called online systems, whereas those that earn solely by simulating actions within an internal model are called ofline systems."*

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 the last three competitions that happened from 2011 to 2013. The authors of this implementation employ several heuristics and black-box optimization methods in order to reproduce the mechanisms to which human racing drivers resort, doing so by means of a modular structure. *Mr. Racer* uses a Covariance Matrix Adaptation Evolution Strategy (CMA-ES), to evolve parameters offline.

According to the founders of the competition [SCR 2013] and the authors of *Mr. Racer* themselves, *AUTOPIA* [Onieva et al. 2012] is another competitive controller, with the potential to even be the best one available. *AUTOPIA* implements a modular Fuzzy Architecture, whose division contains gear, steering and speed control; and it 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 [SCR 2013] served as criteria for the analysis and development of the approach presented in this paper. Aspects incorporated and adapted from them feature modularity, offline learning through genetic algorithms, online earning through landmarking and choosing sets of parameters for different categories of tracks, etc. Aspiring to design a controller

capable of incorporating these features, the design of a model was proposed and is presented in the succeeding section.

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 autonomous driving behavior in a car race.

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 that describe the situations found within the racing environment.

3.1 The Design of the Behavior States

Initially, the design of the finite-state machine proposed comprised the following states:

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

Essentially, for the first method, normal behavior 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 behavior consisted of *Out of Track* and *Stuck*, situations in which such conduct was expected.

The main difference between the method described and the second one is the way they deal with normal behavior, while one separates it into *Straight Line*, *Approaching Curve* and *Curve*, the other treats it as a unique conduct in the form of the *Inside Track* state. Therefore, the modified finite-state machine was composed by only three states, which were:

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

3.1.1 Normal Behavior

If a controller using the first method 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 proportionally to the curvature of the approaching curve in order to achieve higher speeds once inside it. For example, if there is a sharp left turn nearby, the controller sets a target steer that the car needs to obtain before entering it, while bringing the car further to the right of the lane; that way, when the left turn comes, the car can proceed with a less abrupt change of steer, which results in a higher speed. Otherwise, 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.

Farthest sensor in a turn:
Inside Track, therefore, is how the car, desirably, will spend most part of the time, if the second method is being employed. The controller calculates a *target speed* based on how far the car is from the farthest edge of the track, then, it assumes a position of increasing the speed until it reaches this velocity while driving towards the sensor with the biggest read value. The distance of this method conveys the greater length the car may advance with little or sometimes without significant steer changes. This state also brakes, if necessary, in situations that the car assumes a speed higher than the one calculated to be the target.

Insert Picture

Figure 2: *Proper picture.*

3.1.2 Exception Behavior

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 policies chosen for the exception states were used on both proposals equally as a matter of regularity of comparison.

Out of Track is when, for any unknown reason, the car is found outside of the track limits. In this case, the proper behavior 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. Since there are many ways that a car may be out of the track, reentering it in an efficient way might require many different angles as well, so, every time the car exceeds the inside boundaries of the lane, a proper returning angle is calculated. Out of Track desired return angle:

Insert Picture

Figure 3: *Proper picture.*

.

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.

3.2 Five-State FSM

The first method was named "Five-State FSM", naturally because it is a finite state machine with five states. The real first model of the finite-state machine did not have an *Approaching Curve* state, but, as the demeanour of *Straight Line* and *Curve* were so different from one another, a preparation had to be established so as to smoothen the transitions between them. Figure **??** represents the states diagram of the Five-State FSM.

Figure 4 instantiates the relations between the states and the transition function through two-headed arrows, which is done because

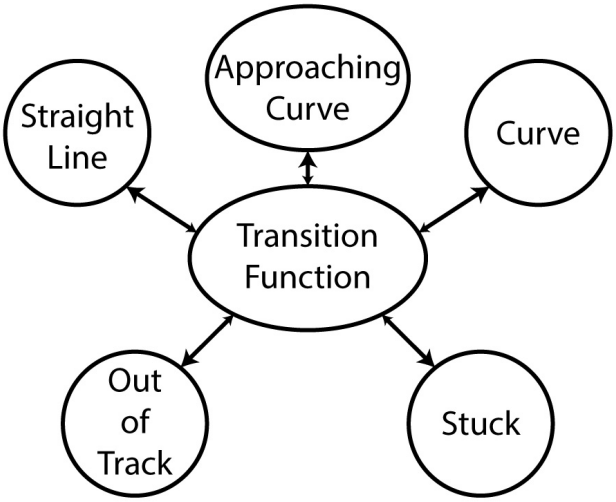

Figure 4: *States diagram of the Five-State FSM*

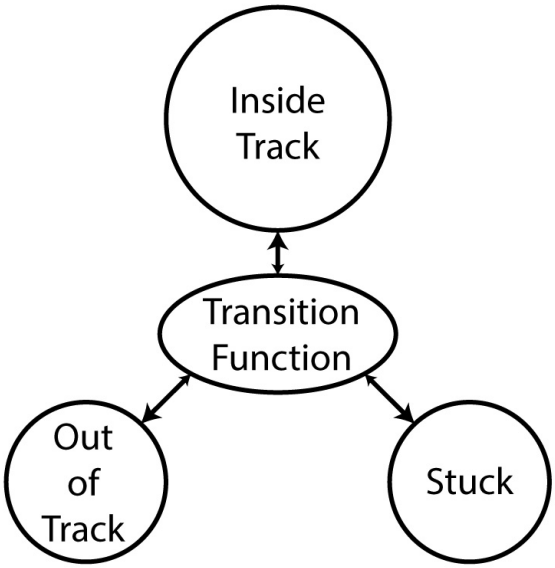

Figure 5: *States diagram of the Three-State FSM*

the function chooses which state the controller is supposed to be, but the states also communicate with it through the actuators that determine indirectly which will be the next state. This way, the states provide a type of feedback.

One big problem about this approach 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.

In addition, the angles - with relation to the car axis - of all the 19 range finder sensor have to be initialized in a manner that includes as much information of the track as possible. For instance, if all the angles were to be initialized pointing only to one side of car, the data concerning the other side would be neglected. For the Three-State FSM, the angles were instantiated using steps of 10 degrees, which results in angles ranging from -90 to +90 degrees - 0 meaning the direction that indicates the front of the car.

3.3 Three-State FSM

The finite state machine with less states was designed from a derivation of the previous one, by adapting the normal behavior and simplifying it to form the Three-State FSM. The exception behavior remained the same, since it should not affect the car as much if the adaptation was well performed. Figure 5 shows the modified states diagram of the Three-State FSM.

Even though the Three-State FSM also has a transition function, it does not receive the heavy burden of decision present in the other approach. It mainly checks if the car is *Stuck*, if so, it manages this situation, if not, it checks if the car is *Out of Track* and deals with it as well. Otherwise, the car will be at the *Inside Track* state, which represents most of the racing time - justifying it being bigger in Figure 5, and also most of the work required from the controller to handle the driving behavior.

Besides the states, there is also a learning module that is called whenever the *Out of Track* state is requested. This module records both speed and position from the state of the car in the vicinity of the departure from the track, and retaining these information allows the controller to slow down in subsequent laps when approaching the critical points highlighted by the learning module. The implementation of this procedure consists on replacing the speed recorded from the landmarked position by a slower one on future occurrences, which is done by starting to break in these next occasions. This module constitutes the Online Learning technique of the controller, and in order to maximize the efficiency of the information received from the track, the vector of sensors in the Three-State FSM was initialized according to a normal distribution, i.e., the sensors are more densely distributed in front of car and less on the sides.

Each state in the two ways described to deal 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.

3.4 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.

Genetic Algorithms [Holland 1984] are evolutionary algorithms 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 in many areas, extending from control engineering for non-linear systems identification [Wang et al. 2003] to biomedicine prosthesis development [Wang et al. 2009] and even economy to forecast the behavior of agents [Bharathi et al. 2007].

In this context, a viable solution is called an individual and can be represented by a string of parameters. The first population is instantiated randomly to its full extent due to the lack of information concerning how to effectively evaluate an individual. This population passes through a fitness function that indexes a score to each individual, and 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 Elitism [ELI 2006]. 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 takes place in 95% of the reproduction, while the mutation rate assumes the rate of 1% [Grefenstette 1986].

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].

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 the models - and which of their parameters required tuning - have been defined, the genetic algorithm was applied to each approach 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. Additionally, as the *Stuck State* is only triggered in very specific situations, it was not evolved with the controller and its parameters were hand tuned.

Pursuing an unbiased choice of parameters, the Online Learning module described in Subsection 3.3 was turned off during the evolution progress as it seizes the responsibility for the behaviour of the car during the race and could interfere with natural selection. The validation then 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.

Four road tracks, witch were used in evolution, are selected from the TORCS standard track set. They are *Spring*, the longest track available in torcs with more curves than any other track, *Wheel 2*, the most difficult track with their close and hard curves, *E-Track 3*, a fast track with curves that will get dexterity of the controller, *Forza*, this is a fast track and the pattern of their curves are usually found in others tracks.

Six road tracks are selected for evaluate the controllers, they are available in TORCS too, were used in 2008 competitions and were used for evaluate the *Autopia* controller (referencia autopia-2009). Ruudskogen, Street-1 and D-Speedway were used at IEEE World Congress on Computational Intelligence; and CG Speedway 1, E-Track3 and B-Speedway were used at Computational Intelligence and Games Symposium. (texto copiado da referencia autopia-2009)

Four dirt tracks are selected from the TORCS standard track set. They are *Dirt2, Mixed1, Mixed2, Dirt6*. *Dirt2* is a track difficult with close curves, *Mixed1* is a easy track with few curves, *Mixed2* have many curves with medium diffitulty, *Dirt6* is the longest dirt track available with hard curves.

4.2 Results

TABLE DOES NOT FIT IN A SINGLE COLUMN!

Table 1: *Distance covered racing alone for 10,000 game tics*

Driver	Street-1	D-speedway	CG-speedway	Dirt-1
FSM3(road)	7357.06	13047.5	8974.96	4587.12
FSM3(dirt)	2149.77	2450.84	1951.93	3525.84
FSM3(mixed)	6704.88	8545.64	6481.52	4854.00
FSM5(road)	3822.76	3427.11	4114.66	2145.49
FSM5(dirt)	1267.83	2936.82	4114.66	1072.92
FSM5(mixed)	3822.99	3427.06	4114.8	2145.75
AUTOPIA	7091.8	15612.3	8970.4	?

split table in time and ticks table

4.3 Analysis

review this section and add more details, just throwing ideas In general the three states controller outperformed by far the five states one. FSM3 presented better results in all the tested tracks. One of the causes of this overwhelming difference in performance may be the complexity of the transition function. The FSM5 presents a very complex transition that takes in account the sensors variance to decide if the car is in a straight line, approaching a turn or in a turn. On the other hand, the FSM3 has only a single state for handling normal driving situations and it is very easy to say if the controller is inside or outside the track only by checking the track sensor.

As expected, the controller evolved only in road tracks is the fastest one. This tracks provides an environment susceptible to high speeds since its curves are smoother than the dirt tracks ones, the friction experimented by the car is higher in road tracks allowing it to brake without losing control. As the friction is higher steering is more accurate in road tracks. No critical skidding. Result was an aggressive driver with high base speed.

Dirt tracks provides a more difficult environment for the pilot to fit in. Suddenly brakes result in unwanted behavior more often, skidding is way more common in this situation . The driver evolved here tends to drive in a low speed so it can keep itself inside the boundaries of the track. Speed driving results in high damage and even change it please "death". The result obtained was a very careful driver with a low base speed, and an early brake policy (car starts to brake far before the turn). This passive driving achieved the smallest distance covered both for the FSM3 and FSM5.

find better names than "dirt controller", "mixed controller", "road controller" The driver evolved in a mix set of tracks combines characteristics from both. It drives in a considerable speed as the first one, but also has the preventive brake policy from the second one. This behavior achieved better results than the "dirt behavior" in all the tracks tested and outperformed the "road behavior" in all the dirt tracks. *The "mixed controller" performed better than the "dirt controller" even in dirt tracks.* One of the reasons is that the "mixed driver" tries to reach higher speeds even though this means leaving the track in some turns. The time spent trying to get back to the track is compensated by the speed of the car. In addition, the "mixed" is better than the "road" in dirt tracks. The aggressive behavior of the "road driver" makes it take several damage when leaving the track and hitting walls, this results in premature ending of the race (max damage reached).

still missing lots of information!

5 Conclusions

5.1 Conclusions

5.2 Future Works

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

BARTH, M. 2013. The potential role of vehicle automation in reducing traffic-related energy and emissions. *International Conference on Connected Vehicles and Expo (ICCVE)*.

BEJCZY, A. K. 2012. New challenges for application of robotics and automation in space research. *7th IEEE International Symposium on Applied Computational Intelligence and Informatics*.

BHARATHI, R., KUMAR, M. J., SUNITHA, D., AND PREMALATHA, S. 2007. Optimization of combined economic and emission dispatch problem a comparative study. *International Power Engineering Conference*.

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.

CARBAUGH, J., GODBOLE, D., AND SENGUPTA, R. 1997. Tools for safety analysis of vehicle automation systems. *Proceedings of the American Control Conference*.

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

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.

DUNCAN, J. C., AND FETERLE, L. C. 2000. The use of personal computer-based aviation training devices to teach aircrew decision making, teamwork, and resource management. In *Proceedings of the IEEE 2000 National Aerospace and Electronics Conference*, 421–426.

2006. *Watchmaker Framework for Evolutionary Computation - Chapter 3. Selection Strategies and Elitism - http://watchmaker.uncommons.org/manual/ch03s06.html*.

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

GREFENSTETTE, J. J. 1986. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics SMC-16*, 1 (Jan).

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., LÖNNEKER, T. D., CARDAMONE, L., PEREZ, D., SÁEZ, Y., PREUSS, M., AND QUADFLIEG, J. 2010. The 2009 Simulated Car Racing Championship. *IEEE Transactions on Computational Intelligence and AI in Games 2* (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.

MITCHEL, T. M. 1997. Reinforcement learning. In *Machine Learning*, W. Publishing, Ed. McGraw-Hill Science/Engineering/Math, Plano, TX, ch. 13, 385.

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 27*, 3, 217–241.

2014. *7 Reasons to Justify an Operator Training Simulator: https://www.honeywellprocess.com/library/news-and-events/presentations/Hon-EMEA14-Operator-Training-Simulator.pdf*.

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.

2015. *Why use simulation? - Justifying Simulation (accessed: 2015): https://www.promodel.com/pdf/Justifying%20Simulation.pdf*.

TERZIC, I., ZOITL, A., AND FAVRE, B. 2008. A survey of distributed intelligence in automation in european industry, research and market. *IEEE International Conference on Emerging Technologies and Factory Automation*.

WANG, Q., SPRONCK, P., AND TRACHT, R. 2003. An overview of genetic algorithms applied to control engineering problems. In *International Conference on Machine Learning and Cybernetics*, vol. 3, 1651–1656.

WANG, S., HONG, J., LIU, Z., AND WANG, W. 2009. Method of human-imitating posture generation for mechanical hand prosthesis. *ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots*.

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`.

XUE, H., WU, H., MENG, X., AND LIU, W. 2011. Simulation research and programming of torque fluctuation of automotive steering system. In *Second International Conference on Mechanic Automation and Control Engineering*, 4385–4387.

ZHANG, F., DU, Z., SUN, L., AND JIA, Z. 2007. A new novel virtual simulation system for robot-assisted orthopedic surgery. *IEEE International Conference on Robotics and Biomimetics*.