Evolving Finite-State Machines Controllers for the Simulated Car Racing Championship

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

Abstract

Autonomous vehicles have many practical applications, but the development of such controllers is a difficult task. This work presents a finite state-machine model with evolved parameters as a suitable solution for a self-driving car, and the comparison of two different configurations in The Open Racing Car Simulator. This approach enables a clear division of behaviors in states, providing an easy way to test different configurations and simplifying the search for better controllers by allowing changes in selected states. A 5-state and a 3-state drivers were evolved through genetic algorithm and compared to each other and to AUTOPIA, the current state of the art controller for the Simulated Car Racing Championship. Results showed that the proposed model has potential for racing, even surpassing one of AUTOPIA's marks, and provide insights on developing and configuring.

Keywords: finite-state machine, genetic algorithm, TORCS, simulated car racing

Author's Contact:

gnramos@unb.br

1 Introduction

Automation of day to day tasks is an endeavor that has moved a large amount of scientific resources in the recent history [Terzic et al. 2008; Bejczy 2012]. One of the more desirable goals is the advent of self-driving cars, which should drive safely and efficiently, within traffic laws [Carbaugh et al. 1997; Barth et al. 2013], bringing hope to current issues of traffic in urban roads by aiming at shorter response times, better fuel consumption, and lower levels of pollution. Such drivers, however, face several difficult challenges such as perception, navigation and control [Luettel et al. 2012].

Another critical issue is driver development, since testing is a frequent task and real driving systems are expensive. Creating and testing solutions can be aided by realistic car racing games which can closely simulate the roads, other vehicles, and their complex interactions [Caldeira et al. 2013]. Games also present a well defined environment which may be used not only for applications of machine learning results, such as neuroevolution [Stanley et al. 2005; Cardamone et al. 2010] or human pose recognition [Shotton et al. 2011], but also for comparing AI solutions for specific problems such as path planning [de Freitas et al. 2012], controlling human non-playable character [Lucas 2008], car racing [Loiacono et al. 2010], among others.

The Open Racing Car Simulator (TORCS) is a modern, modular, highly-portable multi-player, multi-agent car simulator [Wymann et al. 2013], one of the most advanced racing games available, and frequently used as a platform for comparing driver solutions in the Simulated Car Racing Championship (SCR) [Loiacono et al. 2010; Loiacono 2012]. This paper uses a finite-state machine (FSM) controller to exploit the advantages of a divide-and-conquer approach by considering the task's various situations as distinct states (racing, getting unstuck, getting back on track, etc.).

FSM is a well-known mathematical model with widespread applications such as controlling air conditioning systems [Bernaden 1999], highway surveillance systems [HyunKim et al. 2013], and even simulated car racing [Perez et al. 2009]. Such solutions are implemented as software and configured according to a specified set of parameters, and defining the best possible configuration is usually a huge combinatorial problem for which exhaustive systematic searches become unfeasible. Thus, several machine learning approaches have been applied to it (heuristic algorithms [Quadflieg et al. 2014], evolutionary algorithms [Nallaperuma et al. 2014], modular fuzzy architectures [Onieva et al. 2012], sensory-to-motor couplings [Butz and Lonneker 2009], among others). One of the most successful optimization methods is Genetic Algorithm (GA), which is has been applied to a myriad of problems such as non-linear systems identification [Wang et al. 2003], biomedicine prosthesis development [Wang et al. 2009], agent behavior forecasting [Bharathi et al. 2007], improving classifiers [Pedrycz and Succi 2005], and others.

This work uses a GA for searching for optimal parameter settings for the proposed FSM based driver, using TORCS as test bed. The rest of this paper is structured as follows: Section 2 introduces the TORCS/SCR, finite-state machines, and Genetic Algorithms concepts involved; Section 3 details the proposed controller model. Section **??** describes the evaluation and validation process; and Section **??** presents concluding remarks.

2 Background

Scientific Research has been greatly aided by the evolution of simulators, which greatly simplify and reduce initial development costs. These tools have been widely used in several areas, such as medical education [Zhang et al. 2007], aiding decision making [Levy et al. 2002], aviation industry [Duncan and Feterle 2000], and automotive research and development [Xue et al. 2011]. In the field of Machine Learning, simulations are specially helpful for training and learning through experimentation.

Car simulators model several elements of a vehicular dynamics, including inertia, suspension types, differentials, friction, aerodynamics, and others [Wymann et al. 2013]. These models represent an approximation of real systems, and the reality gap (differences between model and real system results [Whitton and Moseley 2012]) that stems from the simplifications made have to be considered, so simulations cannot completely replace experimentation on actual cars. However, current advanced simulators provide such realistic experiences that they are ideal for most of the development phase.

2.1 TORCS & SCR

The Open Racing Car Simulator[1] is a platform that is renowned for its highly credible physics modeling engine and yet user-friendly interface with very customizable environment for car racing simulations [Wymann et al. 2013; Loiacono et al. 2013], and has been widely used in Artificial Intelligence (AI) for developing and comparing solutions [Loiacono et al. 2010]. The engine of this simulator considers factors such as collision, traction, aerodynamics, and fuel consumption and provides several circuits, vehicles, and controllers [Loiacono et al. 2010; Loiacono 2012], enabling all kinds of possible in-game situations. Additionally, it is open source, with an active community, making it possible and encouraging modifications of its source code to better suit specific needs. It has been used as a standard platform for simulated racing since 2007 [Loiacono 2012].

The TORCS software[2] represents a stand-alone application in which pieces of program code may be used to drive the simulated car robots, which represent every in-game opponent. The specific

---

[1] http://torcs.sourceforge.net/
[2] http://arxiv.org/pdf/1304.1672.pdf

robots that are loaded and compiled with the game as artificial intelligences are denominated "bots" inside the environment, and, consequently, as there is no separation layer between them and the simulation engine, they retain full access to the information concerning the structure and the status of the race. There is a great diversity concerning types of cars and tracks available at TORCS, and also around 100 bots to race against. The tracks differ in categorization according to their variety of ground, which affects the dynamics of the cars in execution time. The cars feature particular characteristics of tire and wheel properties, including aerodynamics and others. The players may also play the part of developers by coding their own robots to the game, which become what are called "controllers".

The Simulated Car Racing Championship (SCR) is a competition between controllers built on TORCS [Loiacono et al. 2013]. It was the first simulated car racing championship organized as a joined event of major scientific conferences: IEEE Congress on Evolutionary Computation, the ACM Genetic and Evolutionary Computation Conference, and the IEEE Symposium on Computational Intelligence and Games [Loiacono et al. 2010], and has been accepted by researchers as suitable platform for evaluation and comparison of controllers [Wymann et al. 2013].

Controllers are ranked according to performance during the championship, which consists of several races on different tracks divided into legs, spread through the conferences [Loiacono et al. 2010]. They are scored using the Formula 1 point system[3]. The software for SCR extends the TORCS architecture by structuring it as a clientserver application, by incorporating real time processing and by physically separating the driver code and the race server through a sensors/actuators model abstraction layer [Loiacono et al. 2010]. These changes provide an even more interesting environment for researchers by enabling any kind of controller implementation (as long as it can communicate via UDP connections) and defining a clear interface between controller and simulated car, which can be easily adapted for testing the controller with a different simulator or even a real car (provided the proper adaptations).

SCR controllers have been developed using AI techniques, such as neural networks, fuzzy logic, potential fields, and genetic algorithms [Loiacono 2012]. In the competition, the race track is initially unknown and it several drivers incorporate machine learning procedures to improve their performance [Loiacono et al. 2010] advantage, which can essentially be *online* or *offline*. Online systems learn by moving about the environment and observing the results while offline ones learn solely by simulating actions within an internal model [Mitchell 1997].

Mr. Racer, which has won the last three championships (2011 to 2013), employs several heuristics and black-box optimization methods in a a modular structure in order to reproduce human-like mechanisms [Quadflieg et al. 2014], it applies a Covariance Matrix Adaptation Evolution Strategy (CMA-ES), to evolve parameters offline.

In the GECCO leg of the 2013 competition[4], *AUTOPIA*'s performance stands out. It implements a fuzzy architecture with gear, steering and speed control modules which are optimized offline by a genetic algorithm and an online learning mechanism for landmarking lane exit points to avoid leaving the track [Onieva et al. 2012].

Not surprisingly, other participants also use combinations of offline and online learning and modular approaches [Loiacono et al. 2010; Perez et al. 2009; Cardamone et al. 2009]. Modularity has the clear advantage of independent development and optimization[Quadflieg et al. 2014; Onieva et al. 2009], and one of the simplest models for implementing different behaviors is a finite-state machine.

2.2 Finite-State Machines

A FSM is a mathematical model with a finite number of states that can transition from one to another, and a FSM whose output values

are determined solely by its current state is a Moore machine [Ajzerman et al. 1971]. FSMs have been widely used in AI application [Millington and Funge 2006], mostly due to its inherent characteristics of flexibility, modularity, and intuitive behavior, among others [Buckland 2005]. Since the FSM is a widely known model to handle robot engines problems it seems to be highly applicable to the controller.

States are usually implemented with hard-coded rules concerning a specific situation [Buckland 2005], which in turn demands small amounts of processor time, and can be easily implemented in different manners. The different driving behaviors can then be coded in parallel with less effort, and easily translated into different states of a FSM. Such approach has successfully been used in SCR [Loiacono et al. 2010; Perez et al. 2009].

The proper configuration of states and transitions, however, are a more complex problem. Several possible solutions exist, and machine learning techniques can readily be applied.

2.3 Genetic Algorithms

GAs are a particular kind of genetic optimization mechanism, inspired by evolutionary algorithms inspired by Darwin's theory of natural selection [Holland 1984]. They are probabilistic search procedures designed to work on large spaces [Goldberg and Holland 1988], and have been successful applications in many areas [Wang et al. 2009; Bharathi et al. 2007; Stanley et al. 2005; Pedrycz and Succi 2005].

GAs work by generating successor solutions by repeatedly mutating and recombining parts of the best currently known solutions, replacing a fraction of the population by offspring [Mitchell 1997]. This is interesting because it works with little information on the problem's domain, the only requirements related specifically to the problem are a representation of a solution for a problem and a function for evaluating it's quality (fitness).

In the context of a self-driving car controller, a solution can be seen as a representation of the driver's parameters and the fitness a measure of its performance, considering speed, safety, fuel consumption, or whatever is the developer's interest. Considering SCR, the usual fitness is the distance driven during the given simulation time [Loiacono et al. 2010].

3 FSMDriver

Driving a car can be intuitively divided into several behaviors, and thus provide a straightforward implementation as a finite-state machine model. Such model's configuration parameters can be optimized through a genetic algorithm, and the TORCS/SCR platform provides a standard measure of driver quality, and its sensor/actuator interface enable it to be readily replaced by actual robotic prototypes (or other more advanced models) for further testing. This work proposes to use these ideas for evolving a controller for a car, the FSMDriver, as one more step towards autonomous vehicles.

3.1 Driving Behaviors

A `State` in the FSM model considered here defines a racing behavior, deciding the driver's output according to the given sensor input as defined by SCR's API. The FSM implements a transition function that, at every game tick, analyses the input and decides which state is appropriate, triggering a change if necessary. The state then processes the input and defines the proper output.

The next step for implementing the FSM model is defining its states. Intuitively, `Driving` could be a definite solution but it obviously involves several distinct situations which should be divided, simplifying the development.

Considering the testing environment, it is clear that two antagonistic situations that require specific behaviors: `Racing`, for situations where the car is within track limits, facing the right direction; and

---

`Recovery`, for when the car is outside track limits or facing the right direction, or unable to move.

These can be further divided into two more specific behaviors for implementation:

Straight Line: for racing straight ahead;

Curve: for racing through a curve;

Out of Track: for recovering from leaving the track or facing the wrong direction; and

Stuck: for recovering from being unable to move forward.

Each of these states implements its behavior as follows. `Straight Line` attempts to go as fast as possible parallel to the track axis by accelerating at full throttle, which improve the performance to track's variance, and changing gears according to RPM thresholds. `Curve` steers towards the direction towards the track sensor with the largest reading (see Figure 1) with 60% throttle, since it indicates at each state iteration a retiline path to exit the curve, braking in case it starts to slide in the X axis. `Out of Track` attempts to return to the track, facing the right direction, adjusting its speed and steering according to its current orientation concerning the track (see Figure 2). `Stuck` tries to get unstuck by using the reverse gear and hard steering.
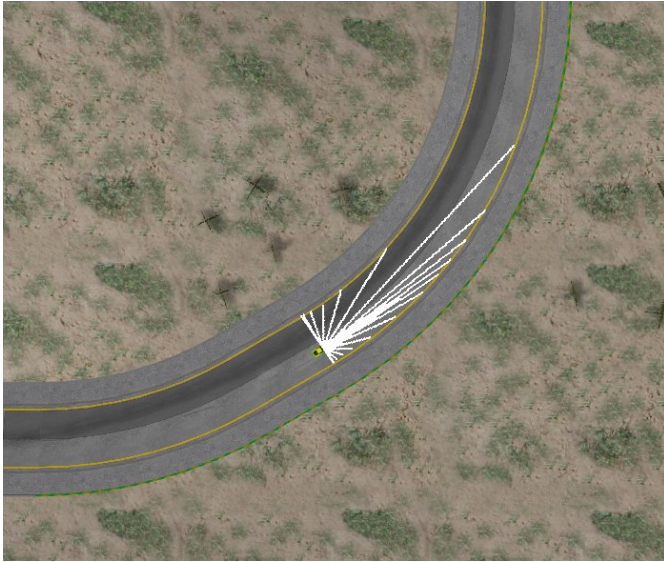

Figure 2: *Angle between car and track axis.*

Figure 1: *Sensor input in curve.*

3.2 FSMDriver5

Initial tests showed that transitioning from `Straight Line` to `Curve` was too swift, hampering performance because the car consistently entered curves at such high speeds that the controller was unable to turn and exited the track, so the additional behavior `Approaching Curve` was implemented. `Approaching Curve` positions the driver towards the outside of the incoming curve and tries to race at a speed proportional to the curvature, so less braking would be necessary in `Curve`. Thus, a 5-state FSM-Driver (FSMDriver5) model (illustrated in Figure 3) was ready for testing.

reviewed paragraph, please evaluate There are 19 range finders available in SCR [Loiacono et al. 2013] whose directions are defined by the driver, and these sensors provide the distance between the car axis and the edge of the track. When the car is outside of the track's boundaries, they read -1 value. For the Five-State driver, those directions were arbitrarily settled from -90 to 90 degrees in a 10 degree step, the 0 degree direction pointing toward the car axis, see Figure 4.

Through several observations it was concluded that the farther from a curve the car is, the greater the difference among the frontal and lateral sensors is. As the car approaches a turn this difference starts
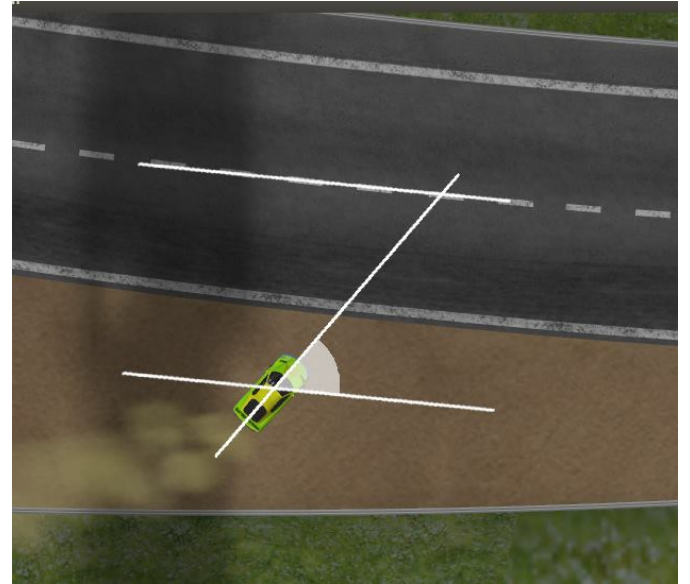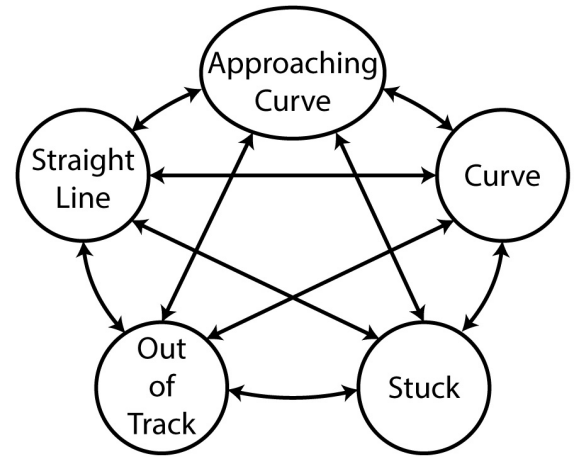

Figure 3: *FSMDriver5 state diagram.*

to drop until it reaches a minimum value, when the car is taking the turn. The variance among the range finders was taken into account for telling which state of the `Racing` behavior would better fit with the current segment of the track, as shown in Algorithm 1.

finish this

Initially, random configurations were set and manually adjusted until an acceptable behavior was achieved, i.e. such a configuration in which a car could successfully complete a race. In the beginning, the `Recovery` states (`Out of Track` and `Stuck`) were constantly triggered, and therefore were first to be adjusted for improved behavior. Eventually their settings were such that the `Racing` behaviors could be focused on. These too were manually and arbitrarily set through trial and error, considering the driver's quantitative performance racing (time and damage) and its qualitative skill (visual analysis of tests).

After several iterations, this model performed better than some of the less capable robots available in TORCS, showcasing the FSM model's potential for controlling a car. However, testing also revealed that the transitions between states required were the bottleneck for improved racing. Not only some of the triggers needed a more detailed analysis to avoid eventual erratic behavior, but the sheer number parameters considered for transitioning, each with its triggers (see Figure 3), and their impact on the model's behavior during a complete race caused the whole model to be thought over.
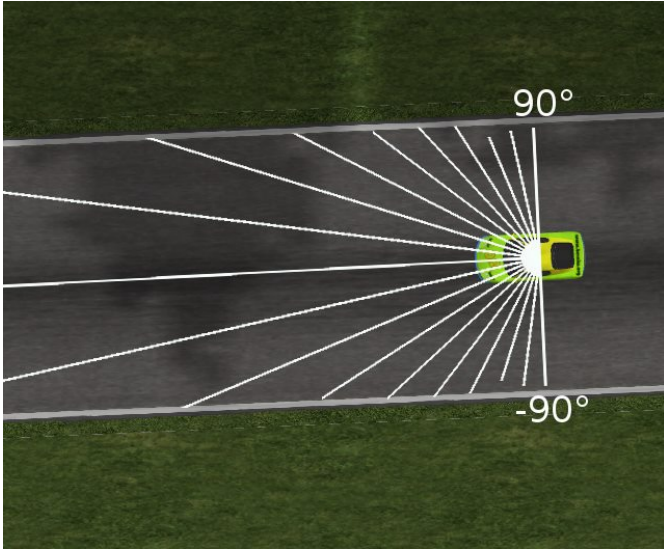
Figure 4: *FSMDriver5 range finders.*

Algorithm 1 FSMDriver5 Transition

if car is stuck **then**
    $state \Leftarrow Stuck$
else
    if ($variance < MAX\_STRAIGHTLINE\_VAR$) **or** ($variance < MIN\_STRAIGHTLINE\_VAR$ **and** the current state is not $StraightLine$) **then**
        $state \Leftarrow StraightLine$
    end if
else if ($variance > MAX\_APPROACHING\_CURVE\_VAR$ **and** current state is not curve) **or** ($variance > MIN\_APPROACHING\_CURVE\_VAR$ **and** current state is not Approaching Curve) **then**
    $state \Leftarrow ApproachingCurve$
else if $variance > 0$ **then**
    $state \Leftarrow Curve$
else
    $state \Leftarrow OutOfTrack$
end if

Analyzing FSMDriver's transitions occurrences within a race, it was clear that the Recovery behavior had two distinct situations, which were acceptably handled by the current configuration, and that the major issue was frequent triggering between the Racing states, which inevitably resulted in the car leaving the track and jeopardizing its performance.

3.3 FSMDriver3

To reduce the model's complexity, and considering the intuitive division of behaviors proposed, a new model where the defined Recovery behaviors were maintained and the Racing behaviors were joined into a new state called Inside Track. Figure 5 illustrates this FSMDriver3 model.

Inside Track implements a very straightforward idea, get to an arbitrary target speed towards the largest open space inside track limits. The target speed is proportional to the largest sensor reading, and braking is activated if the current speed is greater than the target. Gear changing works based on RPM threshold as before. Overall, this implementation works for straight lines as well as curves.

review In order to have a more detailed view of upcoming track segments the initialization of the range finders changed in the Three-State FSM. Here a normal distribution was applied in this intuit, focusing sensorial information ahead the car, as shown in Figure 6.
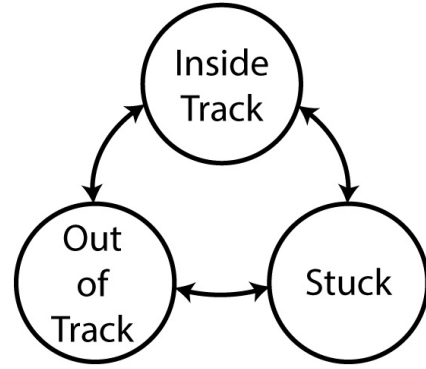
Dividing Racing into three other states reduces the need of a more

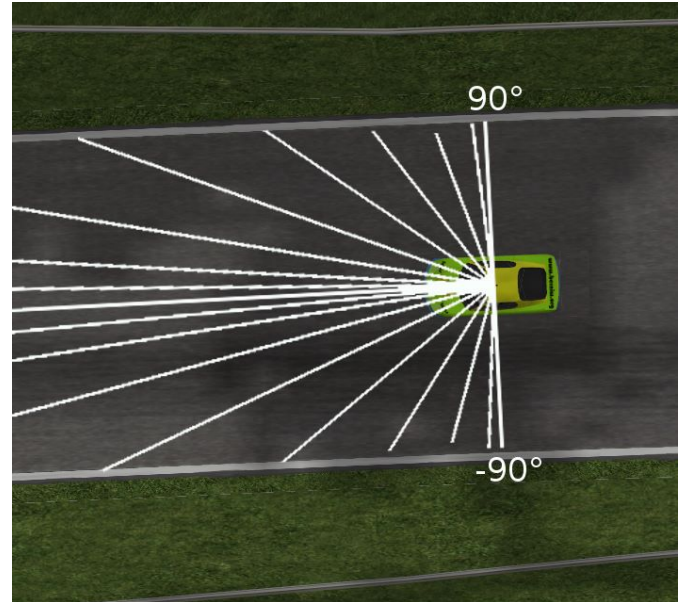
Figure 5: *FSMDriver3 state diagram.*

Figure 6: *FSMDriver3 range finders.*

elaborated transition function. The transition function can then be reimplemented in a simpler fashion, as shown in Algorithm 2:

Initially, configurations resembling FSMDriver5's parameters were set and manually adjusted until a behavior similar to the original driver was achieved. As expected, this process was quicker than for FSMDriver5 and soon a configuration better than some robots was available.

With the finished models[5], the issue at hand became to set the models' parameter to such values as to maximize their performance, a large combinatorial problem to which a well known tool was applied.

3.4 Evolving Parameters

In order to apply a genetic algorithm to the task of evolving the drivers' parameters, a model of the solution must be defined. In this context, a solution is called an individual and was represented

---

Algorithm 2 FSMDriver3 Transition

if car is stuck **then**
    $state \Leftarrow Stuck$
else
    if car is within track limits **then**
        $state \Leftarrow InsideTrack$
    else
        $state \Leftarrow OutOfTrack$
    end if
end if
if the current state is not $state$ **then**
    $CurrentState \Leftarrow state$
end if

by a string of bit parameters, that internally to the algorithm are interpreted as float numbers.

FSMDriver5 has, overall, 22 parameters from states and the transition function:

in progress

Transition:

> **MAX_STRAIGHTLINE_VAR** upper boundary for classifying a straight line
>
> **MIN_STRAIGHTLINE_VAR** lower boundary for classifying a straight line
>
> **MAX_APPROACHIN_VAR** upper boundary for classifying an approaching curve
>
> **MIN_APPROACHIN_VAR** lower boundary for classifying an approaching curve

Approaching Curve:

> **TARGET_POS** desired percentage position
>
> **BASE_SPEED** lowest speed allowed during turns
>
> **nome do param3** utilidade do param3
>
> **nome do param4** utilidade do param4

Straight Line:

> **LOW_GEAR_LIMIT** threshlod to bound low gears
>
> **LOW_RPM** threshlod of rpm to delimit the change of low gears
>
> **AVERAGE_RPM** threshlod to decrease high gears
>
> **HIGH_RPM** threshlod to delimit the change of high gears

Out of Track:

> **MAX_SKIDDING** defines the threshold to start to break to avoid skidding
>
> **NEGATIVE_ACCEL_PERCENT** dictate how much to release the acceleration pedal to avoid to skidding, its defined with the axis speed of car
>
> **VELOCITY_GEAR_4** threshold to change the gear to 4
>
> **VELOCITY_GEAR_3** threshold to change the gear to 3
>
> **VELOCITY_GEAR_2** threshold to change the gear to 2
>
> **MAX_RETURN_ANGLE** upper boundary to angle of return to track
>
> **MIN_RETURN_ANGLE** lower boundary to angle of return to track

Stuck:

> **STUCK_SPEED** defines the threshold to monitorate if its in stuck

> **MINIMUM_DISTANCE_RACED** just to avoid to enter in stuck at the beginning of race when the car is stopped.
>
> **MAXIMUM_NUMBER_OF_TICKS_STUCK** utilidade do param3
>
> **MAXIMUM_NUMBER_OF_TICKS_IN_SLOW_SPEED** utilidade do param4

FSMDriver3, on the other hand, has the same 11 for `Out of Track` and `Stuck` as FSMDriver5, plus 6 more:

Inside Track:

> **LOW_GEAR_LIMIT** threshlod to bound low gears
>
> **LOW_RPM** threshlod of rpm to delimit the change of low gears
>
> **AVERAGE_RPM** threshlod to decrease high gears
>
> **HIGH_RPM** threshlod to increase high gears
>
> **SPEED_FACTOR** utilidade do param5
>
> **BASE_SPEED** utilidade do param6

The GA also requires a fitness function to evaluate the solutions, this work will consider the distance raced, which is also the standard metric for SCR.

References

AJZERMAN, M. A., GUSEV, L. A., ROZONOER, L. I., SMIRNOVA, I. M., AND TAL, A. A. 1971. *Logic automata and algorithms*. Math. Sci. Eng. Academic Press, New York, NY.

BARTH, M., BORIBOONSOMSIN, K., AND WU, G. 2013. The potential role of vehicle automation in reducing traffic-related energy and emissions. In *Connected Vehicles and Expo (ICCVE), 2013 International Conference on*, 604–605.

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*.

BERNADEN, A. 1999. Variable air volume control using a finite state machine. In *Control Conference (ECC), 1999 European*, 31–36.

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.

BUTZ, M., AND LONNEKER, T. 2009. Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, 317–324.

CALDEIRA, C., ARANHA, C., AND RAMOS, G. 2013. Torcs training interface: An auxiliary api for developing torcs drivers. *XII Simpósio Brasileiro de Jogos e Entretenimento Digital*, 13.

CARBAUGH, J., GODBOLE, D., AND SENGUPTA, R. 1997. Tools for safety analysis of vehicle automation systems. In *American Control Conference, 1997. Proceedings of the 1997*, vol. 3, 2041–2045 vol.3.

CARDAMONE, L., LOIACONO, D., AND LANZI, P. L. 2009. Learning Drivers for TORCS through Imitation Using Supervised Methods. *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*.

CARDAMONE, L., LOIACONO, D., AND LANZI, P. 2010. Learning to drive in the open racing car simulator using online neuroevolution. *Computational Intelligence and AI in Games, IEEE Transactions on 2*, 3 (Sept), 176–190.

DE FREITAS, L. G., REFFATTI, L. M., DE SOUSA, I. R., CARDOSO, A. C., CASTANHO, C. D., BONIFÁCIO, R., AND RAMOS, G. N. 2012. Gear2d: An extensible component-based game engine. In *Proceedings of the International Conference on the Foundations of Digital Games*, ACM, New York, NY, USA, FDG '12, 81–88.

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.

GOLDBERG, D., AND HOLLAND, J. 1988. Genetic algorithms and machine learning. *Machine Learning 3*, 2-3, 95–99.

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

HYUNKIM, D., KWON, J., CHEN, K., AND CHOI, K. 2013. Finite state machine for vehicle detection in highway surveillance systems. In *Frontiers of Computer Vision, (FCV), 2013 19th Korea-Japan Joint Workshop on*, 84–87.

LEVY, D., CHALOUPKA, F., GITCHELL, J., MENDEZ, D., AND WARNER, K. 2002. The use of simulation models for the surveillance, justification and understanding of tobacco control policies. *Health Care Management Science 5*, 2, 113–120.

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).

LOIACONO, D., CARDAMONE, L., AND LANZI, P. L. 2013. Simulated Car Racing Championship: Competition Software Manual. *ArXiv e-prints* (Apr.).

LOIACONO, D. 2012. Learning, evolution and adaptation in racing games. In *Proceedings of the 9th Conference on Computing Frontiers*, ACM, New York, NY, USA, CF '12, 277–284.

LUCAS, S. M. 2008. Computational intelligence and games: Challenges and opportunities. *International Journal of Automation and Computing 5*, 1, 45–57.

LUETTEL, T., HIMMELSBACH, M., AND WUENSCHE, H.-J. 2012. Autonomous ground vehicles – concepts and a path to the future. *Proceedings of the IEEE 100*, Special Centennial Issue (May), 1831–1839.

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.

MITCHELL, T. M. 1997. *Machine Learning*. McGraw-Hill.

NALLAPERUMA, S., NEUMANN, F., BONYADI, M. R., AND MICHALEWICZ, Z. 2014. Evor: An online evolutionary algorithm for car racing games. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, ACM, New York, NY, USA, GECCO '14, 317–324.

ONIEVA, E., PELTA, D., ALONSO, J., MILANES, V., AND PEREZ, J. 2009. A modular parametric architecture for the torcs racing engine. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, 256–262.

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.

PEDRYCZ, W., AND SUCCI, G. 2005. Genetic granular classifiers in modeling software quality. *Journal of Systems and Software 76*, 3 (June), 277–285.

PEREZ, D., RECIO, G., SAEZ, Y., AND ISASI, P. 2009. Evolving a fuzzy controller for a car racing competition. *Computational Intelligence and Games (CIG)*.

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 15*, 4, 433–476.

SHOTTON, J., FITZGIBBON, A., COOK, M., SHARP, T., FINOCCHIO, M., MOORE, R., KIPMAN, A., AND BLAKE, A. 2011. Real-time human pose recognition in parts from single depth images. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Washington, DC, USA, CVPR '11, 1297–1304.

STANLEY, K. O., BRYANT, B. D., AND MIIKKULAINEN, R. 2005. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation 9*, 6, 653–668.

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., WANG, W., AND ZHANG, E. 2009. Method of human-imitating posture generation for mechanical hand prosthesis. In *Reconfigurable Mechanisms and Robots, 2009. ReMAR 2009. ASME/IFToMM International Conference on*, 428–433.

WHITTON, N., AND MOSELEY, A., Eds. 2012. *Using games to enhance learning and teaching: a beginner's guide*. Routledge, New York.

WYMANN, B., ESPIÉ, E., GUIONNEAU, C., DIMITRAKAKIS, C., COULOM, R., AND SUMNER, A., 2013. TORCS, the open racing car simulator, v1.3.5. 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*.