TORCS Training Interface: An auxiliary API for developing TORCS drivers

Clara Caldeira
University of Brasília,
Institute of Exact Sciences,
Department of Computer Science
claramcaldeira@gmail.com

Claus Aranha
University of Tsukuba,
Graduate School of Systems and Information Engineering,
Department of Computer Sciences
caranha@cs.tsukuba.ac.jp

Guilherme N. Ramos
University of Brasília,
Institute of Exact Sciences,
Department of Computer Science
gnramos@unb.br

***Abstract*—Autonomous vehicles have many practical applications, and the process of developing controllers for them can be aided by the use of simulators. Among several available such softwares, TORCS stands out for being one of the most advanced and for being used as benchmark for comparison of controllers in international competitions. In order to aid the inherently iterative process of developing controllers for this platform, we present *TORCS Training Interface*, a free open source solution that provides greater agility in setting up and running simulations.**

I. Introduction

Digital games in general offer a great test bed for experimentation and study of Artificial Intelligence (AI), and there has been a growing interest in applying AI in several problems present in them, regardless of genre [1]. One of the standard uses is the control of non-playable characters (NPCs), whose behavior should be interesting and believable for the player, a task that has become increasingly difficult to do [2]. Thus, games can be seen as an excellent tool for evaluating how AIs handle unforeseeable situations and adapt to the player.

Electronic games also present a well defined environment, which may simulate extremely complex situations. Since input and output are constrained by the game's characteristics, these games provide the ideal test bed for comparing solutions. This evaluation may be done individually, such as in the *Mario AI Championship*[1], or collectively, as in the *Simulated Car Racing Championship*[2].

The increasingly realistic simulations used in games can be readily applied in other industries. For example, the control of a humanoid robot and that of a human non-playable character [1] or autonomous vehicle control. Self-driving cars bring the promises of improved traffic in urban roads by aiming at shorter response times, better fuel consumption, and lower levels of pollution. Other possible benefits include more accurate driving, more independence and mobility for individuals who cannot drive for themselves, and less accidents. However, the development of such drivers face a number of difficult challenges, including perception, navigation and control [3].

Creating and testing solutions for these problems can be greatly aided by car racing games, because these can realistically simulate the roads, other vehicles and their interaction. *The Open Racing Car Simulator* (TORCS) is one of the most advanced racing games available, and is frequently used when developing artificial intelligence techniques around the world [4], [5]. Not only does it provide a very realistic experience with its physics engine and detailed graphics, it also stimulates research with features like platform independence, a command-line AI interface, and a GPL license.

The AI controller in TORCS handles a car by receiving the input data from the sensors and sending commands to its actuators. This input/output data involved in the simulation is similar to an abstraction of the data collected by the sensors of the robot Stanley [6], the winner of the DARPA Challenge for autonomous driving cars in 2005. Clearly, this indicates that a racing game can be used as a tool for aiding AI research, enabling several tests and development before moving on to real applications.

TORCS has been used as a platform for an annual championship of simulated racing since 2007 [5]. The competition is composed of three legs of competitions between controllers submitted by researchers from around the world. Among the submissions, there are several examples of computational learning and artificial intelligence techniques, such as neural networks, fuzzy logic, potential fields, and genetic algorithms [5]. This competition provides an opportunity for the scientific community to perform a straightforward comparison among these approaches in complex environments containing multiple continuous variables [4].

TORCS provides a very customizable environment, sophisticated 3D graphics, a powerful physics simulation platform [7], which takes into consideration factors such as collision, traction, aerodynamics, and fuel consumption. It also provides a variety of circuits, vehicles, and controllers leading to a wide set of possible in-game situations [4], [5]. Additionally, since it is open source, there is the possibility of modifications to better fit the specific needs.

This work approaches the problem of the high costs involved in repeatedly testing TORCS controllers, a necessary step in developing a driver which is also a bottleneck in the process. To this end, we propose the *TORCS Training Interface* (TTI), to aid researchers by facilitating the configuration and execution of multiple simulations.

[1] http://www.marioai.org
[2] http://cig.ws.dei.polimi.it

II. How TORCS simulations work

The TORCS software is composed of a racing simulator and a graphical interface to interact with it whose basic purpose is to simulate a single race, configured by the user. The race can be between AI controllers only, to be observed by the user, or he/she can take part by controlling one of the cars. To improve research in artificial intelligence, later versions of TORCS provides a command-line interface, bypassing the need to display the simulation graphics and, thus, reducing considerably the time necessary to get the results.

The racing simulation is executed in a client-server architecture. Each client is assigned to the controller of one of the racing cars. Data about the vehicle's sensors is received from the simulator through an UDP connection. Values for the actuators are sent in the same fashion. This process is executed in parallel for each client, and the information exchange occurs continuously until the end of the simulation.

The controlling client can collect data about the circuit it is running on from the vehicle's sensors, and information about its performance in the race. It is possible to evaluate different race configurations, a standard necessity in developing AIs with techniques such as genetic algorithms and neural networks.

The client has the option of restarting a race whenever desired, which makes it possible to run iterative evaluations. However, in the latest version (1.3.4), the one being used here, this option only works for the current race being simulated, meaning that the training process will be limited to the behavior of the client in that specific circuit. There is no option for the client to change the track, or any other race configuration e.g. number of laps.

Features like the length of the evaluation period, the set of tracks used and the starting order greatly influence the results. Longer and more diversity-filled evaluations provide more complete and thorough information on the controller's behavior, despite increasing the costs involved in testing. These costs are specially relevant when such evaluations need to be done repeatedly.

TORCS in text-based interface receives as a parameter a `XML` file which contains the settings for the race to be simulated, such as length in distance or number of laps, the track, the participating controllers and their order. It then executes the corresponding simulation and exits as soon as it is finished.

The completion of a lap in text interface in the latest version of TORCS takes time in the order of seconds in a mid-range model notebook computer. Although this is a significant improvement on real time racing/simulation, it is still too slow for the repeated evaluations required in training processes based on iterative evaluations.

Additionally, any change in the configurations in terms of length, selected track or quantity of cars involved must be modified manually in a specific file in `XML` format or in the graphical interface of the game.

III. TTI

The required manual intervention and the reduced flexibility in the repetitive tests involved in the development of drivers for TORCS increases the effort involved in using it as a research platform. Providing services that solve, or at least improve on these problems will reduce the time spent with these manual labors, and improve the productivity of research efforts using this platform.

With this in mind, we decided to develop a tool which works as an interface between the TORCS server and a computational evolution algorithm (or any other approaches based on iterative evaluation processes), offering useful functionalities to interact with the simulation software.

An open application programming interface (API) providing utilities for automating processes related to running TORCS simulation should reduce the effort spent in manual and repetitive tasks, as well as the replicated development of project specific scripts. The maintainers of the simulator itself have recently added resources intended to ease this process, such as a text based interface for test execution, and the possibility of requesting a race restart. However, we believe that there is still is room for significant improvement of the available resources, giving us the motivation for the current work.

In this section we first describe the API, explaining how it works and how it differs from currently available mechanisms used to interact with the simulator. Then, we discuss a few points identified as having potential for optimizations that could result in considerable reductions in the overall costs of automatized simulations, or in better information they are able to provide.

A. How TTI works

TTI's goal is to facilitate the process of developing an AI for TORCS, trying to speed up the simulation iterations and providing a simple way to schedule various tests. Every controller has a collection of parameters which have to be set, and which may change depending on the race's features. The evaluation of a controller's performance is ultimately defined by the simulated race's result, in terms of time and distance, even if the parameter values are updated during the race. Thus, it is essential to have several races to properly assess one or more controllers.

The *TORCS Training Interface* works as an interface for the simulator (see Fig. 1). It receives input parameters to configure the simulation and the types of results that will be returned. The tool manages all the interaction between the clients (controllers) and the server (simulator), from running the races to the collecting the data available.

TTI is implemented as a shared library for Unix environments (`.so` format). It works work alongside the TORCS server, sharing some of its resources. We simplify the process of executing a race, which originally consists of:

1) Creation or adjustment of a `XML` file containing the race configurations
2) Call of the server by a text-based interface
3) Individual call of each client in its own process and configuration of the port used by each client to communicate with the server;

In contrast to these three steps, the interface in our tool needs only a single method call. Its parameters are an identifier

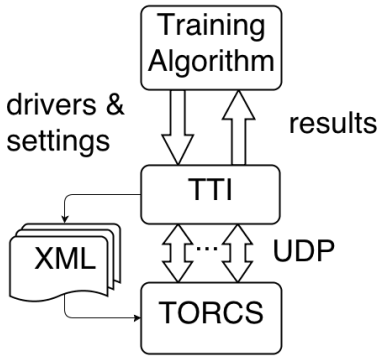

Figure 1. Interaction of TTI with TORCS and with a training algorithm.

for the track (track's name), the number of laps, and a collection of 1 to 10 controllers, which is the range of cars allowed in a single race. The tool returns a collection of data describing the results for each controller in the race.

B. Running on multiple tracks

The first point which we aimed to improve through the proposed tool was the management of server instances. Because the text interface executes only a single race and interrupts immediately after it is completed, two options were considered. The first and simplest, although less efficient, is to call a new instance for every race. The second one, possible only if the same track set is used for every iteration, consists of keeping an instance active for each track and only to ask the restart of the race at the beginning of each iteration. This second option, however, doesn't allow changes on the amount of controllers in a race or its length, since the parameters cannot be changed during the execution. It is only possible to change the controller's order by adjusting the ports used by each of them in the connection.

The execution in multiple tracks requires the call of at least one instance of the server for each track. It is possible to keep them running continuously, but the parallel execution of the server instances might lead to delays affecting the resulting behavior of the controllers.

There are a number of possibilities when it comes to the ordering in a the race. We can select the ordering by rotation, i.e., shifting of the positions in one unit n times where n is the amount of controllers, order inversion, exhaustive evaluation of all possible combinations, or, as is often used in real life competitions, an individual classification round to determine the order of the race. Clearly, the more evaluations are performed, more complete is the resulting information and larger are the associated costs. It is up to the user to determine which is the best option, and thus our tool allows each ordering scheme to be selected through a configuration parameter.

C. Early termination

A work by Haasdijk et al. [8] suggested that in some cases the evaluation process should be interrupted early. To this end, an estimate is made on the probability of an individual being good enough to join the current population at different points throughout its evaluation. If it is determined that the individual's quality is significantly inferior in comparison to the worst individual of the population, continuing with the process is likely a waste of time, specially when the algorithm uses an elitist selection process. The results presented indicate that this interruption has a good influence on the achieved results when the evaluation process is expensive.

To this end, our tool allows the user to chose a minimum distance to be covered in sub-intervals of the evaluation. This allows for quickly disposing of individuals which do not move at all, drive too slowly, go in the wrong direction, or are unable to get unstuck efficiently. This minimizes the time spent evaluating poor controllers.

D. Comparisons as metrics

Storing the best known values for distance or lap-time for a circuit/car configuration can be used as a quality measure for the training algorithm. This comparison provides a quick assessment of the client's performance, which can be used, for example, to stop the current simulation and proceed to the next test configuration.

These values could also be used for estimating an optimal trajectory in a given track would allow for an evaluation based in how much the actual resulting path taken adheres to it. Usually, it consists of a compromise between that of shortest length and the one which minimizes curvatures, i.e., maximizes curve radius. The shortest path is that which is traveled fastest by a body of constant speed while the maximum curvature allows for a vehicle to achieve the highest possible average speed. It is known that the time a car takes to complete a lap is inversely proportional to its average speed and directly proportional to the length of the path taken. Minimizing the time, therefore, can be understood as maximizing the average speed and minimizing the length. Because the two paths differ significantly, there is an intermediary one which offers the best combination [9].

This is a simplistic model, since the actual ideal path takes into consideration physical properties of the vehicle such as its acceleration and breaking efficiency, and opponent interaction in each point, e.g. overtaking. Still, it can make for a useful tool, even as a complement, for estimating controller quality. Its math is extremely simple given the ideal trajectory for the track and the information provided on the quality is, in relation to the associated cost, quite relevant and complete.

Choosing a specific controller as basis for comparisons could be done amongst those included in the TORCS distribution or among the best seen in the competitions, all of which are freely available. This option makes for relevant information about how close individuals are, in quality terms, to the best of the previously developed. Values above 100% would indicate surpassing. This strategy could consider simply the time resulting or it could choose a more sophisticated analysis of the decisions taken by the controllers. This last one is specially useful when it comes to evaluating complex specific behaviors which are difficult to determine or measure, such as overtaking [5]. It is desirable that resulting drivers are skillful enough to outstrip adversaries, doing so without spending too much time, in the best possible point and without losing significantly in performance. Like the choice of precluding as much as possible to be overtaken, which is another complex behavior, it could be easier and faster to make comparisons

Table I. COMPARISON OF TTI WITH CURRENT PROCESSES

	With TTI	Without TTI
Server and client instance calls and settings handling	Automatically managed and configurable	Manual, possibly aided by custom scripts
Simulations on multiple tracks	Automatically managed and configurable	Only with frequent manual interventions
Acquired data comprehensiveness	Possibly high	Limited
Code modifications	Only the driver object and derived classes	Any class and source file of the module

of the decisions taken by the individual under evaluation and those of another which is considered skilled in it.

IV. WHAT TTI DOES

In this section we make more focused comparisons of TTI features with the processes currently used for developing TORCS drivers, noting in which points it is beneficial and what limitations it introduces in a development process. Table I shows the difference TTI makes in four relevant points.

Handling server and client instances is complicated, specially when dealing with multiple clients. Because evaluating a group of drivers at once is faster than doing so individually and the ability of the driver in interacting with other cars is taken into consideration, racing multiple clients alongside each other is important. Communication ports have to be consistently configured since the race doesn't start until there are established connections in all participating ports, drivers using the wrong one are unable to connect and same port conflicts lead to similar problems. TTI provides automatic management of instances, allowing some adjustments with configuration parameters.

Working with multiple tracks is also greatly simplified when using our API, only a collection containing their names is needed. Otherwise, keeping individual `XML` files or repeatedly manually modifying one would be necessary, as would be restarting every client instance at each step, which is difficult to handle manually and very prone to cause problems as stated previously.

We consider that the information extracted from a step of evaluation would be higher in accuracy when options such as comparisons, multiple track or ordering are active. These provide much relevant information about a driver's behavior.

The biggest limitation introduced by TTI is related to the freedom the developer has in modifying the code in the client module. Since all communication with the server happens through a `UTP` connection, it is only necessary that the transactions stay consistent. However, because TTI deals directly with driver objects, all relevant control-related modifications must be done at that class or at a derived one, without overriding non virtual methods or attributes.

V. CONCLUSION

Artificial Intelligence can be applied to solve several problems, and the development of solutions can be aided by the use of simulators. TORCS stands out as one of the most advanced car racing simulators, and is a benchmark tool for comparing AIs in international competitions. In order to aid the process of developing controllers for this platform, we present the *TORCS Training Interface*, a free open source solution that provides greater agility.

TTI is used to automatically run a sequence of various simulations in TORCS, speeding up the evaluation a controller and enabling the developers to use the simulation results as part of the AI training process. We observed significant ease provided by the tool, as discussed in section IV.

By developing a tool with this functionality, we relieve other researchers from the burden of having to implement these basic features over and over again in their clients. The API of this project provides a fully configurable way to access the simulation features, so that it can be adapted to specific projects, removing the need of spending redundant time dealing with these issues. This time can now be better employed on advancing the controller itself, the result of which could be an extremely competitive driver for TORCS as a game or even an AI to handle an autonomous vehicle.

TTI was developed in `C++` for Unix, limiting the projects which could potentially use this tool, but its resources could be implemented in other languages and adapted to other environments. Nevertheless, this choice of programming language means the interface could readily be incorporated by *The Open Racing Car Simulator* project. Additionally, one could work on similar projects directed towards other games, aiming to provide better interfaces for artificial intelligence techniques which need to use them for training.

REFERENCES

[1] S. M. Lucas, "Computational intelligence and games: Challenges and opportunities," *International Journal of Automation and Computing*, vol. 5, no. 1, pp. 45–57, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11633-008-0045-8

[2] ——, "Computational intelligence and ai in games: a new ieee transactions," *IEEE Trans Computat Intell Artif Intell Games*, vol. 1, no. 1, pp. 1–3, 2009.

[3] T. Luettel, M. Himmelsbach, and H. J. Wuensche, "Autonomous ground vehicles - concepts and a path to the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831–1839, 2012.

[4] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lönneker, L. Cardamone, D. Perez, Y. Sáez *et al.*, "The 2009 simulated car racing championship," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 2, pp. 131–147, 2010.

[5] D. Loiacono, "Learning, evolution and adaptation in racing games," in *Proceedings of the 9th conference on Computing Frontiers*. ACM, 2012, pp. 277–284.

[6] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[7] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship: Competition software manual," *ArXiv e-prints*, Apr. 2013.

[8] E. Haasdijk, A. Atta-ul Qayyum, and A. E. Eiben, "Racing to improve on-line, on-board evolutionary robotics," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011, pp. 187–194.

[9] L. Cardamone, D. Loiacono, P. L. Lanzi, and A. P. Bardelli, "Searching for the optimal racing line using genetic algorithms," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 388–394.