Simulated Car Racing Championship
Competition Software Manual

April 2013

Daniele Loiacono, daniele.loiacono@polimi.it
Luigi Cardamone, luigi.cardamone@polimi.it
Pier Luca Lanzi, pierluca.lanzi@polimi.it

Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Italy

Abstract

This manual describes the competition software for the Simulated Car Racing Championship, an international competition held at major conferences in the field of Evolutionary Computation and in the field of Computational Intelligence and Games. It provides an overview of the architecture, the instructions to install the software and to run the simple drivers provided in the package, the description of the sensors and the actuators.

1 Introduction

This manual describes the competition software for the Simulated Car Racing Championship, an international competition held at major conferences in the field of Evolutionary Computation and in the field of Computational Intelligence and Games.

The goal of the competition is to design a controller for a racing car that will compete on a set of unknown tracks first alone (against the clock) and then against other drivers. The controllers perceive the racing environment through a number of sensors that describe the relevant features of the car surroundings (e.g., the track limits, the position of near-by obstacles), of the car state (the fuel level, the engine RPMs, the current gear, etc.), and the current game state (lap time, number of lap, etc.). The controller can perform the typical driving actions (clutch, changing gear, accelerate, break, steering the wheel, etc.). A description of the championship, including the rules and regulations, can be found at http://cig.sourceforge.net/

The championship platform is built on top of The Open Racing Car Simulator (TORCS) a state-of-the-art open source car racing simulator which provides a full 3D visualization, a sophisticated physics engine, and accurate car dynamics taking into account traction, aerodynamics, fuel consumption, etc.

In the remainder of this manual, we provide an overview of the architecture, the instructions to install the software and to run the simple drivers provided in the package, the description of the sensors and the actuators.

2 The Architecture of the Competition Software

The Open Racing Car Simulator (TORCS) comes as a stand-alone application in which the bots are compiled as separate modules that are loaded into main memory when a race takes place. This structure has three major drawbacks. First, races are not in real-time since bots execution is blocking: if a bot takes a long time to decide what to do, it will block all the others. Second, since there is no separation between the bots and the simulation engine, the bots have full access to all the data structures defining the track and the current status of the race. As a consequence, each bot can use different information for its driving strategy. Furthermore, bots can analyze the complete state of the race (e.g., the track structure, the opponents position, speed, etc.) to plan their actions. Accordingly, a fair comparison among methods of computational intelligence is difficult since different methods might access different information. Last but not least, TORCS restricts the choice of the programming language to C/C++ since the bots must be compiled as loadable module of the main TORCS application which is written in C++.

The competition software extends the original TORCS architecture in three respects. First, it structures TORCS as a client-server applications: the bots are run as external processes connected to the race server through UDP connections. Second, it adds real-time: every game tic (roughly

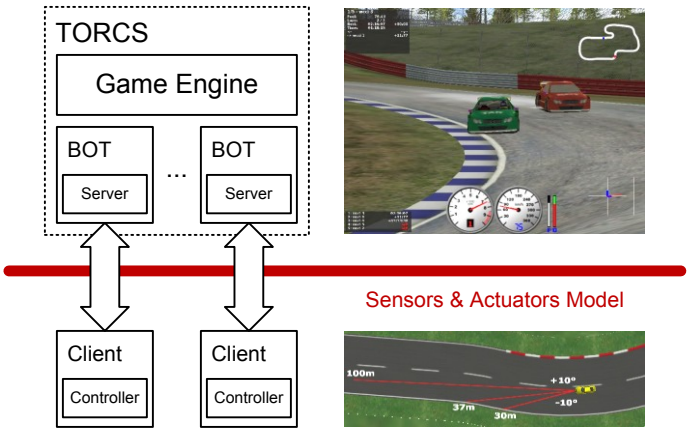

Figure 1: The architecture of the competition software.

corresponding to 20ms of simulated time), the server sends the current sensory inputs to each bot and then it waits for 10ms (of real time) to receive an action from the bot. If no action arrives, the simulation continues and the last performed action is used. Finally, the competition software creates a physical separation between the driver code and the race server building an abstraction layer, a *sensors and actuators model*, which (i) gives complete freedom of choice regarding the programming language used for bots and (ii) restricts the access only to the information defined by the designer.

The architecture of the competition software is shown in Figure 1. The game engine is the same as the original TORCS, the main modification in a new *server-bot*, called `scr_server`, which manages the connection between the game and a client bot using UDP. A race involves one server-bot for each client; each server-bot listens on a separate port of the race server. At the beginning, each client-bot identifies itself to a corresponding *server-bot* establishing a connection. Then, as the race starts, each server-bot sends the current sensory information to its client and awaits for an action until 10ms (of real time) have passed. Every game tic, corresponding to 20ms of simulated time, the server updates the state of the race which is sent back to the clients. A client can request a race restart by sending a special action to the server.

3 Installing the Competition Server

To provide a very accessible interface to TORCS we developed two modules to run TORCS in a client/server architecture. The server has been developed by providing a specific bot driver called `scr_server` that, instead of having its own intelligence, sends the game state to a client module and waits for a reply, i.e., an action to be performed by the controller. So to begin the competition, we first need to install TORCS and the competition server-package provided in this bundle.

3.1 Linux Version

Download the all-in-one TORCS 1.3.4 source package from SourceForge (`http://sourceforge.net/projects/torcs/`) or directly from here. To compile the server you will need:

- Hardware accelerated OpenGL (usually provided by your Linux distribution)

- GLUT 3.7 or FreeGlut (better than GLUT for full screen support)
- PLIB 1.8.5 version
- OpenAL
- libpng and zlib (usually provided by your Linux distribution)
- FreeALUT

Unpack the package `torcs-1.3.4.tar.bz2`, with "`tar xfvj torcs-1.3.4.tar.bz2`" which will create the directory `torcs-1.3.4`.

Download the package `scr-linux-patch.tgz` containing the patch for the TORCS sources from the CIG project page at http://sourceforge.net/projects/cig/ or as a direct download from here. Unpack the package `scr-linux-patch.tgz` in your base TORCS directory (where you unpacked `torcs-1.3.4.tar.bz2`). This will create a new directory called `scr-patch`. Enter the `scr-patch` directory and run the script `do_patch.sh` with "`sh do_patch.sh`" (run `do_unpatch.sh` to revert the modifications). Move to the parent directory (where you unpacked `torcs-1.3.4.tar.bz2`) and run

```
$ ./configure
$ make
$ make install
$ make datainstall
```

At this point you should be able to check whether the competition software has been properly installed by executing the command "`torcs`"; then, from the main window select

Race → Quick Race → Configure Race → Accept.

If everything has been installed correctly, you should find ten instances of the `scr_server` bot in the list of "Not Selected Player" on the righthand side.

Further information about the installation process are available here. Additional information are available at http://www.berniw.org/ (TORCS → Installation – from left bar).

3.2 Windows Version

It is possible to compile TORCS on Windows from sources but it can be rather challenging. Therefore, we provide the binary distribution of the competition software for Windows. In this case, to install the competition software first download the TORCS 1.3.4 Windows installer from https://sourceforge.net/projects/torcs/files/torcs-win32-bin/ and install it. Then, download the file `scr-win-patch.zip` from the CIG project page at http://sourceforge.net/projects/cig/ or as a direct download from here. Unzip the package in the TORCS main directory. During the unpacking, you will be asked to overwrite some existing files, answer yes for all the files. At this point you should be able to check whether the competition software has been properly installed by launching `wtorcs.exe` from the installation directory or from the start menu; then, from the TORCS main window select,

Race → Quick Race → Configure Race → Accept.

If everything has been installed correctly, you should find ten instances of the `scr_server` bot in the list of "Not Selected Player" on the righthand side.

3.3 Mac OsX

We do not provide support for Mac OsX since it is not supported by the TORCS developers.

4 The C++ Client

The C++ client for the competition is a stand-alone console applications that can be compiled from the sources. The package can be downloaded from the CIG project page at http://sourceforge.net/projects/cig/ or as a direct download from here.

4.1 Compiling for Linux

Unpack the client package `scr-client-cpp.tgz` creating the directory `scr-client-cpp`. Then, open a terminal in the directory where you unpacked the client and type `make` to compile it. The compilation process should end without any error or warning and you should now have an executable named `client` in you directory. To launch the client, type,

```
    $ ./client  host:<ip>  port:<p> id:<client-id> maxEpisodes:<me> \
  maxSteps:<ms> track:<trackname> stage:<s>
```

where `<ip>` is the IP address of the machine where the TORCS competition server is running (the default is `localhost`); `<p>` is the port on which the server-bot is listening, typical values are between `3001` and `3010` (the default is `3001`); `<client-id>` is your bot ID (the default is `SCR`); `<me>` is the maximum number of learning episodes to perform (the default value is `1`); `<ms>` is the maximum number of control steps in each episode. (the default value is `0`, i.e., unlimited number of steps); `<trackname>` is the name[1] of the track where the bot will race (default value is `unknown`); `<s>` is an integer that represents the current stage of the competition the bot is involved in: 0 is *Warm-up*, 1 is *Qualifying*, 2 is *Race*, and 3 is *Unknown* (default is *Unknown*). All the parameters are optional (if not specified, the default values are used instead).

4.2 Compiling for Windows

Unpack the client package `scr-client-cpp.tgz` creating the directory `scr-client-cpp`. The package is provided with a DevC++ project file (http://www.bloodshed.net/devcpp.html) but any C++ development tool/IDE may be used. To compile the client on Windows, uncomment the first two lines of `client.cpp` following the instructions provided in the same file. The package also contains the system library `WS2_32.lib` that is required for using the *WinSock* functions. The client should compile without any error or warning, producing the `client.exe` executable. To launch the client open an MS-DOS console and type:

```
client.exe host:<ip>  port:<p> id:<client-id> maxEpisodes:<me> \
  maxSteps:<ms> track:<trackname> stage:<s>
```

where `<ip>` is the IP address of the machine where the TORCS competition server is running (the default is `localhost`); `<p>` is the port on which the server-bot is listening, typical values are between `3001` and `3010` (the default is `3001`); `<client-id>` is your bot ID (the default is `SCR`);

[1] This trackname is *not* necessarily the name of the track in TORCS. It is a conventional name that can be used by the bot to store its own information specific for the different tracks.

<me> is the maximum number of learning episodes to perform (the default value is 1); <ms> is the maximum number of control steps in each episode. (the default value is 0, i.e., unlimited number of steps); <trackname> is the name[2] of the track where the bot will race (default value is unknown); <s> is an integer that represents the current stage of the competition the bot is involved in: 0 is *Warm-up*, 1 is *Qualifying*, 2 is *Race*, and 3 is *Unknown* (default is *Unknown*). All the parameters are optional (if not specified, the default values are used instead).

4.3 Customizing Your Own Driver

To write your own driver, the `BaseDriver` class provided in the client sources must be extended and these methods must be implemented:

- `void init(float *angles)`, the method is called before the beginning of the race and can be used to define a custom configuration of the *track* sensors (see Table 2): the desired angles (w.r.t. the car axis) of all the 19 range finder sensors must be set in the parameter `angles`.
- `string drive(string sensors)`, where `sensors` represents the current state of the game as perceived by your driver; the method returns a string representing the actions taken (see Section 6 for details regarding sensors and actuators);
- `void onShutdown()`, the method called at the end of the race, before the driver module is unloaded;
- `void onRestart()`, the method called when the race is restarted upon the driver request (this function should be used to free allocated memory, close open files, saving to disk, etc.).

In addition, the class attributes `stage` and `trackName` contain respectively the current stage of the race (*warm-up*, *qualifying*, *race* or *unknown*) and the name of the current track (both the *stage* and the *track name* must be specified using the corresponding command line option of the client). This information can be used to save useful information about the current track as well as to adopt different strategies in the different stages of the competition. As an example, the file `SimpleDriver.cpp` implements a very simple driver and it is used by default to build the client executable. Therefore, to build a client executable to run your own driver:

- **on Windows**, uncomment the first two lines of `client.cpp` and set the `__DRIVER_CLASS__` and `__DRIVER_INCLUDE__` definitions to the name of the implemented driver class and to the header file of the same driver class.
- **on Linux**, in the `Makefile` set `DRIVER_CLASS` and `DRIVER_INCLUDE` to the name of the implemented driver class and to the header file of the same driver class.

5 The Java Client

The Java client works similarly to the C++ version. It is a stand-alone console application that can be compiled from the sources. The package can be downloaded from the CIG project page at http://sourceforge.net/projects/cig/ or as a direct download from here.

[2]This trackname is *not* necessarily the name of the track in TORCS. It is a conventional name that can be used by the bot to store its own information specific for the different tracks.

5.1 Running the Java Client

First, unpack the package `scr-client-java.tgz` to create the directory `scr-client-java` containing the source code. To compile the client, go to the directory `src` and type,

```
$ javac -d ../classes scr/*.java
```

To launch the Java client with a simple controller, go to the directory `classes` and type,

```
$ java scr.Client scr.SimpleDriver host:<ip> port:<p> id:<client-id> \
  maxEpisodes:<me> maxSteps:<ms> verbose:<v> track:<trackname> stage:<s>
```

where `scr.SimpleDriver` is the implementation of a controller provided with the software as an example (it can be replaced with a custom implementation); where `<ip>` is the IP address of the machine where the TORCS competition server is running (the default is `localhost`); `<p>` is the port on which the server-bot is listening, typical values are between `3001` and `3010` (the default is `3001`); `<client-id>` is your bot ID (the default is `SCR`); `<me>` is the maximum number of learning episodes to perform (the default value is `1`); `<ms>` is the maximum number of control steps in each episode. (the default value is `0`, i.e., unlimited number of steps); `<v>` controls the verbosity level, it can be either `on` or `off` (the default value is `off`); `<trackname>` is the name[3] of the track where the bot will race (default value is `unknown`); `<s>` is an integer that represents the current stage of the competition the bot is involved in: 0 is *Warm-up*, 1 is *Qualifying*, 2 is *Race*, and 3 is *Unknown* (default is *Unknown*). All the parameters are optional (if not specified, the default values are used instead).

5.2 Customizing Your Own Driver

The Java client is organized similarly to the C++ client. To write your own driver, the `Controller` interface must be implemented by providing the following methods:

- `public float[] initAngles()`. the method is called before the beginning of the race and can be used to define a custom configuration of the *track* sensors (see Table 2): the method returns a vector of the 19 desired angles (w.r.t. the car axis) for each one of the 19 range finders.
- `public Action control(SensorModel sensors)`, where `sensors` represents the current state of the game as perceived by your driver; the method returns the action taken (see Section 6);
- `public void shutdown()`, the method called at the end of the race, before the driver module is unloaded;
- `public void reset()`, the method called when the race is restarted upon the driver request (this function should be used to free allocated memory, close open files, saving to disk, etc.).

In addition, the class attributes `stage` and `trackName` contains respectively the current stage of the race (*warm-up*, *qualifying*, *race* or *unknown*) and the name of the current track (both the *stage* and the *track name* must be specified using the corresponding command line option of the client). This information can be used to save useful information about the current track as well as to adopt different strategies in the different stages of the competition. As an example, the file `SimpleDriver.java` implements a very simple driver.

[3] This trackname is *not* necessarily the name of the track in TORCS. It is a conventional name that can be used by the bot to store its own information specific for the different tracks.

6 Sensors and Actuators

The competition software creates a physical separation between the game engine and the drivers. Thus, to develop a bot it is not required any knowledge about the TORCS engine or the internal data structure. The drivers perceptions and the available actions are defined by a sensors and actuators layer defined by the competition designer. For this competition, the drivers inputs consists of some data about the car status (the current gear, the fuel level, etc.) the race status (the current lap, the distance raced, etc.) and the car surroundings (the track borders, the obstacles, etc.). The actions allow the typical driving actions.

6.1 Sensors

The bot perceives the racing environment through a number of sensor readings which provide information both about the surrounding game environment (e.g., the tracks, the opponents, the speed, etc.) and the current state of the race (e.g., the current lap time and the position in the race, etc.). Table 1 and Table 2 report the complete list of sensors available along with a description. Please notice that the readings provided by `opponents` sensors (Table 1) do not take into account the edges of the track, i.e., distances between cars are computed "as the crow flies" even if the paths cross the edges of the track.

6.2 Actuators

The bot controls the car in the game through a rather typical set of actuators, i.e., the steering wheel, the gas pedal, the brake pedal, and the gearbox. In addition, a *meta-action* is available to request a race restart to the server. Table 3 details the actions available and their representation.

7 Running the Competition Server

Once you have installed TORCS and the server-bot provided (either Windows or Linux version), you can start to develop your own bot extending one of the provided client modules. When you want to run your own bot you have to launch TORCS and start a race, then you have to launch the client extended with your own programmed bot and finally your driver bot will start to run in the race. In TORCS there are several race modes available, however the client-server modules supports only two modes:

- the *Practice* mode that allows a single bot at once to race
- the *Quick Race* modes that allows multiple bots to race against

However, before starting a race with TORCS, you need to configure the following things:

- you have to select the track on which you want to run the race
- you need to add a `scr_server x` bot to race participants and eventually other bots you want as opponents
- you have to define how many laps or how many kilometers that race will last
- you might want to select the desired display mode

In TORCS, all the above options are stored in a set of XML configuration files (one for each race mode). Under Linux configuration files are created after the game is launched for the first time and are located in `$HOME/.torcs/config/raceman/`, where `$HOME` is your home directory. Under Windows instead the configuration files are located in the `\config\raceman\` directory located under the directory where you installed TORCS.

7.1 Configuring TORCS Race Via GUI

The easiest way to configure the race options is using the TORCS GUI. Each race mode can be fully configured selecting from the main menu of TORCS:

Race → Quick Race [or Practice] → Configure Race.

Once you change the configurations of a particular race mode, all the changes are stored automatically by TORCS the corresponding configuration file.

Selecting track. In the first screen you can select any of the track available in the games and then click on *Accept* to move to the next screen.

Selecting bots. The second screen allows the selections of bot that will participate to the race. Notice that in the *Practice* mode only one bot is allowed, therefore in order to add a bot you have first to deselect the currently selected one (if any). First of all you have to make sure that one competition bot, `scr_server x`, is in the list of selected drivers (on the left of the screen). Then, in the *Quick Race* mode only, you can add other drivers to the race from the list on the right (representing all the bot drivers provided with the game). When adding bots pay attention to the car model they use: there are several types of car in TORCS with different features and you might want to be sure that only drivers with the same cars will race against. Notice that `scr_server` uses a `car1-trb1` and the others bot using the same car are:

- `tita 3`
- `berniw 3`
- `olethros 3`
- `lliaw 3`
- `inferno 3`
- `bt 3`

When you have selected all the drivers that will be in the race, you can click on *Accept* and move to the next screen

Setting race length and display mode. In the final configuration screen you can set the race length either as the distance to cover (in km) or as the number of laps to complete. Finally you can choose between two display modes option: *normal* or *results only*. The *normal* mode allows you to see the race either from the point of view of one bot driver or as an external spectator. In this display mode, the time speed can be accelerated up to four times the normal speed, that is you can see 1 minute of race in 15s. In the *results only* mode instead you will not see the race but only the lap times (in *Practice* mode) or the final result of the race (in *Quick Race* mode). However this mode allow you to run simulation much faster: time speed can be accelerated up to 20 times (or even more), that is one minute of race can be simulated within 3 seconds.

7.2 Configuring TORCS through Configuration Files

All the race settings described above can be configured also editing directly a configuration file. In TORCS each race type as its own XML configuration file. The settings of *Practice* are stored in `practice.xml` while the settings of *Quick Race* are in `quickrace.xml`.

Selecting track. To select the track, find the "Tracks" section inside the XML file, that will contain the following section:

```
<section name="1">
  <attstr name="name" val="TRACK-NAME"/>
  <attstr name="category" val="TRACK-CAT"/>
</section>
```

where you should (i) replace `TRACK-ROAD` with the category of desired track (i.e., `road`,`oval` or `dirt`); (ii) replace `TRACK-NAME` with the name of desired track (e.g., `aalborg`). For a complete list of the installed tracks in TORCS, you can see the list of all the directories organized under three main directories, `tracks/road/`,`tracks/oval/` and `tracks/dirt/`, where TORCS is installed. Under Windows you find them in your main torcs directory, under Linux the tracks directories could be found in `/usr/local/share/games/torcs/` or in different places depending on your distribution.

Selecting bots. To select bots you should modify the "Drivers" section inside the XML file. In particular in this section you should be able to find a list of the following elements:

```
<section name="N">
  <attnum name="idx" val="IDX"/>
  <attstr name="module" val="NAME"/>
</section>
```

where `N` means you are editing the Nth bots that will be in the race. The `IDX` is the index of the instance of the bot you want to add: for some bots provided with the game there are several instances (e.g., `bt` bot has several instances: `bt 1`, `bt 2`, $\cdots$); when a bot has only one instance `IDX` should be set to 1). The `NAME` should be replaced with the name of bot you want to add without the index of the instance (e.g., to add the `bt 7` bot, you should use as `NAME` simply `bt` and `7` as `IDX`). A list of available drivers can be found in the `drivers/` directory located in the same place where you have the `tracks` directory introduced before.

Setting race length and display mode. To change race length and display mode you have to modify the "Quick Race" or "Practice" section (depending on which race type you want to setup). In particular you should change the following lines:

```
...
<attnum name="distance" unit="km" val="DIST"/>
...
<attnum name="laps" val="LAPS"/>
...
<attstr name="display mode" val="MODE"/>
...
```

where `DIST` should be either the desired race length in km or 0 if the number of laps is used as race length. Accordingly, `LAPS` should be either the desired number of laps or 0 if the distance is used as race length. Finally `MODE` is either `normal` or `results only`.

7.3 Start to Race!

Once you configured properly TORCS you are ready to run your own bot. From the main menu of TORCS select:

Race → Quick Race [or Practice] → New Race.

You should see that TORCS screen should stop reporting the line

```
Initializing Driver scr_server 1...
```

The OS terminal should report `Waiting for request on port 3001`. This means that the server-bot `scr_server` is waiting for your client to start the race. After the race is started, it can be interrupted from the user by pressing `ESC` and then by selecting `Abort Race` from the menu. The end of the race is notified to the client either if it has been interrupted by a user or if the distance/lap limit of the race has been reached. Please notice that it the `Quit Game` option is chosen in the game menu, instead of the `Abort Race` option, the end of the race will not be notified correctly to the clients preventing them from performing a clean shutdown.

7.4 Running TORCS in text-mode

It is possible to run TORCS without graphics, i.e. without any GUI to launch the race. This run mode could be useful when you plan to run an experiment (or a series of experiments) in a batch mode and you do not need to use the GUI to setup the experiment. Using the "-r" command line option it is possible to specify the race configuration file an to run TORCS in text-mode, as follows:

```
C:\> wtorcs.exe -r race_config.xml (on Windows)
$ torcs -r race_config.xml (on linux)
```

TORCS will run automatically the race defined by the `race_config.xml` file, that can be configured either using the GUI or directly editing it (as explained in the previous section).

7.5 Disabling Fuel, Damage and Laptime Limit

To performs very long experiments in TORCS it is necessary to disable some features that can stop or alter the simulation. Fuel consumption and damage should be disabled for two reasons: first, they increase the noise in the evaluation process because two individuals with a different amount of fuel or damage have different performance; second if the fuel is low or the damage too high the car is removed from the race. The laptime limit removes a car from a race if it takes to much to complete a lap. This situation can happen if with a particular configuration of parameters the car performances are very poor.

To disable these features it is possible to run the patched version of TORCS with these command line arguments:

```
C:\> wtorcs.exe -nofuel -nodamage -nolaptime (on Windows)
$ torcs -nofuel -nodamage -nolaptime (on Linux)
```

Of course each of this arguments can be used alone or in combination with the others.

7.6 Time Constraints

In the development of your driver, please keep in mind that the race is in real-time. Accordingly, the server has a timeout on the client answers: your driver should perform an action (i.e., return an action string) by **10ms** in order to keep in sync with the server. If your bot is slower, you would probably loose the sync with the server and so it is up to you to find out how to avoid that this will happen. It is also possible to specify a custom timeout through the following command line option:

```
C:\> wtorcs.exe -t <timeout> (on Windows)
$ torcs -t <timeout> (on Linux)
```

where `timeout` is the desired timeout (measured as nanoseconds).

7.7 Noisy sensors

By default, the range finders in the sensor model are *not* noisy. However, during the competition noisy range finders will be used (according to the specification in Table 1 and Table 2). To enable noisy range finders, it is possible to use the following command line option:

```
C:\> wtorcs.exe -noisy (on Windows)
$ torcs -noisy (on Linux)
```

Name	Range (unit)	Description
angle	[-π,+π] (rad)	Angle between the car direction and the direction of the track axis.
curLapTime	[0,+∞) (s)	Time elapsed during current lap.
damage	[0,+∞) (point)	Current damage of the car (the higher is the value the higher is the damage).
distFromStart	[0,+∞) (m)	Distance of the car from the start line along the track line.
distRaced	[0,+∞) (m)	Distance covered by the car from the beginning of the race
focus	[0,200] (m)	Vector of 5 range finder sensors: each sensor returns the distance between the track edge and the car within a range of 200 meters. When `noisy` option is enabled (see Section 7) sensors are affected by i.i.d. normal noises with a standard deviation equal to the 1% of sensors range. The sensors sample, with a resolution of one degree, a five degree space along a specific direction provided by the client (the direction is defined with the *focus* command and must be in the range [-90,+90] degrees w.r.t. the car axis). Focus sensors are not always available: they can be used only once per second of simulated time. When the car is outside of the track (i.e., pos is less than -1 or greater than 1), the focus direction is outside the allowed range ([-90,+90] degrees) or the sensors has been already used once in the last second, the returned values are not reliable (typically -1 is returned).
fuel	[0,+∞) (l)	Current fuel level.
gear	{-1,0,1,$\cdots$ 6}	Current gear: -1 is reverse, 0 is neutral and the gear from 1 to 6.
lastLapTime	[0,+∞) (s)	Time to complete the last lap
opponents	[0,200] (m)	Vector of 36 opponent sensors: each sensor covers a span of 10 degrees within a range of 200 meters and returns the distance of the closest opponent in the covered area. When `noisy` option is enabled (see Section 7), sensors are affected by i.i.d. normal noises with a standard deviation equal to the 2% of sensors range. The 36 sensors cover all the space around the car, spanning clockwise from -180 degrees up to +180 degrees with respect to the car axis.
racePos	{1,2,$\cdots$,N}	Position in the race with respect to other cars.
rpm	[0,+∞) (rpm)	Number of rotation per minute of the car engine.
speedX	($-\infty$,+∞) (km/h)	Speed of the car along the longitudinal axis of the car.
speedY	($-\infty$,+∞) (km/h)	Speed of the car along the transverse axis of the car.
speedZ	($-\infty$,+∞) (km/h)	Speed of the car along the Z axis of the car.

Table 1: Description of the available sensors (part I). Ranges are reported with their unit of measure (where defined).

track	[0,200] (m)	Vector of 19 range finder sensors: each sensors returns the distance between the track edge and the car within a range of 200 meters. When `noisy` option is enabled (see Section 7), sensors are affected by i.i.d. normal noises with a standard deviation equal to the 10% of sensors range. By default, the sensors sample the space in front of the car every 10 degrees, spanning clockwise from -90 degrees up to +90 degrees with respect to the car axis. However, the configuration of the range finder sensors (i.e., the angle w.r.t. to the car axis) can be set by the client once during initialization, i.e., before the beginning of each race. When the car is outside of the track (i.e., pos is less than -1 or greater than 1), the returned values are not reliable (typically -1 is returned).
trackPos	$(-\infty,+\infty)$	Distance between the car and the track axis. The value is normalized w.r.t to the track width: it is 0 when car is on the axis, -1 when the car is on the right edge of the track and +1 when it is on the left edge of the car. Values greater than 1 or smaller than -1 mean that the car is outside of the track.
wheelSpinVel	$[0,+\infty]$ (rad/s)	Vector of 4 sensors representing the rotation speed of wheels.
z	$[-\infty,+\infty]$ (m)	Distance of the car mass center from the surface of the track along the Z axis.

Table 2: Description of the available sensors (part II). Ranges are reported with their unit of measure (where defined).

Name	**Range**	**Description**
accel	[0,1]	Virtual gas pedal (0 means no gas, 1 full gas).
brake	[0,1]	Virtual brake pedal (0 means no brake, 1 full brake).
clutch	[0,1]	Virtual clutch pedal (0 means no clutch, 1 full clutch).
gear	-1,0,1,$\cdots$,6	Gear value.
steering	[-1,1]	Steering value: -1 and +1 means respectively full right and left, that corresponds to an angle of 0.366519 rad.
focus	[-90,90]	Focus direction (see the *focus* sensors in Table 1) in degrees.
meta	0,1	This is meta-control command: 0 do nothing, 1 ask competition server to restart the race.

Table 3: Description of the available effectors.

8 Further Information and Support

Further information about the championship and the competition software is available at http://cig.sourceforge.net/.

To report bugs, problems, or just for help, send an email to scr@geccocompetitions.com.

Additional information is also available from the following websites:

- http://www.torcs.org, The Open Racing Car Simulator main website
- http://www.berniw.org/, Bernhard Wymann's page with a lot of information about TORCS