Car Setup Optimization Competition
Software Manual

January 2010

Luigi Cardamone, cardamone@elet.polimi.it
Daniele Loiacono, loiacono@elet.polimi.it
Pier Luca Lanzi, lanzi@elet.polimi.it

Politecnico di Milano, Dipartimento di Elettronica e Informazione, Italy

Abstract

This manual describes the competition software for the Car Setup Optimization competitions held at GECCO-09 and EvoStar-2010.

Please cite this manual as: Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi. "Car Setup Optimization. Competition Software Manual", Technical Report 2010.1, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, 2010.

1 Introduction

This manual describes the competition software for the Car Setup Optimization competitions, so far held at GECCO-2009 and EvoStar-2010. A description of the competitions, including the rules and regulations, can be found at `http://cig.dei.polimi.it/`.

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. The programmed bots available in TORCS have several parameters that need to be optimized for each track (e.g., rear wing angle, suspension spring). This is an ideal testbed for evolutionary based optimization algorithm, but the architecture of TORCS has three major drawbacks. First, since there is no separation between the bots and the simulation engine, the bots have full access to all the data structures representing the track and the car. As a consequence, each optimization approach can focus on different parameters or exploit some specific knowledge on the game data structures to achieve a better performance. Accordingly, a fair comparison among evolutionary optimization approaches is difficult since different approaches might focus on different parameters. Second, the TORCS architecture makes it difficult to use existing optimization frameworks and it restricts the choice of the programming language: the optimization process is currently performed inside the bot, which has to be written in C++ and compiled as a loadable module of the main TORCS application.

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 makes it possible to change the parameters of the car and of the controller policy during the race, while in TORCS they are loaded once at the beginning of each race. Finally, the competition software creates a physical separation between the optimizer and the race server building an abstraction layer which (i) gives complete freedom of choice regarding the programming language used for bots, (ii) restricts the access only to a set of parameters defined by the designer, and (iii) prevent the possibility of exploiting specific domain knowledge to improve the optimization process.

The architecture of the competition software is shown in Figure 1. The game engine is the same as the original TORCS, the main modification are in a new *server-bot*, called `optserver`, which manages the connection between the game and a client bot using UDP. An optimization process involves a server-bot and a client. At the beginning of the process, the client identifies itself to the *server-bot* establishing a connection and the sever-bot sends to the client two basic information: the number of parameters to optimize and the number of simulation tics that are available to the

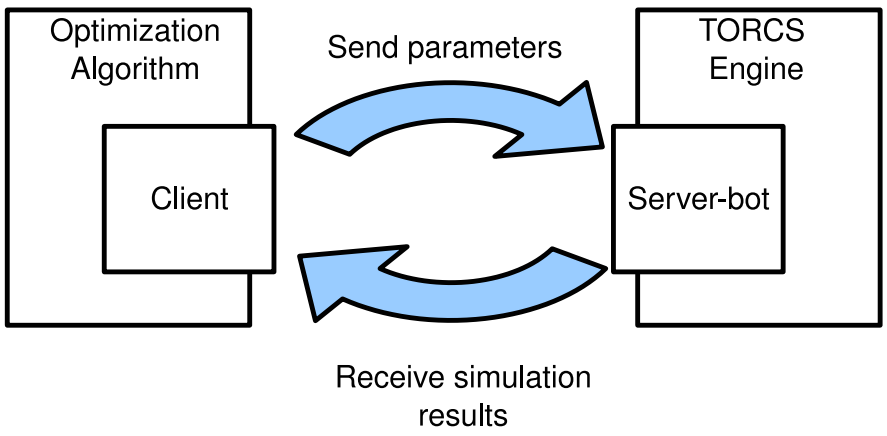

Figure 1: The architecture of the competition software.

whole optimization process. Then the simulation starts and the individuals submitted by the client are evaluated. For each evaluation, the client sends to the server an evaluation request consisting of (i) a list of values corresponding to the parameters to optimize and (ii) the length of the evaluation expressed as the number of simulation tics. As soon as the evaluation ends, the server-bot sends back to the client four performance measures: the distance raced by the driver, the top speed achieved by the driver, the damage suffered by the driver, and the the best lap-time achieved by the driver during the evaluation. The CPU time and the amount of the simulation game tics available for the optimization process are limited: each optimization process have up to two hours of CPU time and 1 millions of game tics (corresponding approximately to 5 hours of simulated time) to find the best parameter setting.

When the overall simulation time expires, the client must send to the server the best solution found. The final solution is the only result taken into account for the purposes of the competition. During the competition, the final solutions produced by each competitor will be compared in order to asses the best optimization technique. More details can be found at `http://cig.dei.polimi.it/`.

3 Installing the Competition Server

To implement an optimization framework inside TORCS, we developed two modules that communicate using a client/server paradigm. The main goal of the server is to receive a string of parameters from the client, plug in such parameters into the car setup and send back the results of the simulation. The server runs inside TORCS as a specific bot driver called `optserver`. 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 1.3.1 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-1.3.1.tar.bz2`, with "`tar xfvj torcs-1.3.1.tar.bz2`" which will create the directory `torcs-1.3.1`. Then, run the commands:

```
$ cd torcs-1.3.1
$ ./configure
```

Download the package `optpatch.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. Copy the package `optpatch.tgz` in your base TORCS directory (where you run `./configure`) and run

```
$ tar xfvz optpatch.tgz
```

This will create a new directory called `optpatch`. Enter the `optpatch` 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 performed `./configure`) and run

```
$ 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 → Practice → Configure Race → Accept.

If everything has been installed correctly, you should find ten instances of the `optserver` 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 1.3.1 Windows installer from http://torcs.sourceforge.net or directly from here and install it. Then, download the file `optpatch-win.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 → Practice → Configure Race → Accept.

If everything has been installed correctly, you should find ten instances of the `optserver` 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 Java Client

The Java client 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.

4.1 Running the Java Client

First, unpack the package `optclient-java.zip` to create the directory `optclient-java` containing both the compiled code and the sources. To launch the Java client with a simple optimization algorithm, go to the directory `bin` and type,

```
$ java ga.SimpleGA
```

This command launches the class SimpleGA that for default connects to *localhost:3001*. To change the ip address and the port the client connects to, modify the following line in the Main method of SimpleGA.java and then re-compile the client:

```
ServerCommunication server = new ServerCommunication("localhost",3001);
```

The port is an integer between 3001 and 3010, depending on which instance of the `optserver` bot is used: "`optserver 1`" listens on port 3001, "`optserver 2`" listens on port 3002 and so on. If the message "Starting..." is displayed and no exception is raised, the client started correctly and it is waiting for the server.

To compile the client, go to the directory `src` and type,

```
$ javac -d ../bin evaluator/*.java ga/*.java
```

4.2 Implementing Your Own Optimization Algorithm

The Java client is composed of two packages:

- evaluator - this package implements the communication with TORCS server

- ga - this package contains a basic genetic algorithm together with its fitness function

The package *evaluator* can be used as it is. The principal class of this package is *ServerCommunication* whose interface is:

- *public ServerCommunication(String ip, int port)* - the constructor that initialize the communication with the bot-server of TORCS
- *public int getParamNumber()* - returns the number of parameters to optimize
- *public long getRemainingTime()* - returns the amount of game tics before the overall simulation ends
- *public void setFitnessFunction* - set the fitness function to use for the simulation results
- *public Object launchSimulation(double[] values, int time)* - launch a simulation with the given parameters. The simulation runs for *time* game tics and then the method returns the fitness of the evaluation
- *public void saveBest(double[] values)* - send the given parameters to the server. These parameters will be saved internally by the server and considered as the best solution found from the optimization process. This function can be invoked at any time, but at least once before the end of the experiment
- *public void close()* - terminates the communication with the server

To implement your own optimization algorithm you can easily extend the *SimpleGA* class or build your classes from scratch and use the *ServerCommunication* to manage the evaluations.

To change the fitness function to use in the experiments it is necessary to modify the class *Fitness.java* in the *ga* package. The fitness is a formula that combines the 4 performance measure returned after an evaluation: *dist-raced*, *top-speed*, *damage* and *best-lap*.

5 Fitness Evaluation

The competition software creates a physical separation between the game engine and the optimization algorithm. This separation is implemented using a client/server architecture where the client runs an optimization algorithm and the server performs the fitness evaluation for a given set of parameters.

5.1 Initialization

During the initialization the client connect to the server and retrieve two values: *Total Simulation Time* and *Parameters Number*. The *Total Simulation Time* represents how many game tics the entire experiment will last. When the *Total Simulation Time* is over the optimization algorithm has no more possibilities to perform further evaluations.

The *Parameters Number* represents the number of parameters that the algorithm has to optimize. The parameters to optimize are 22 and are reported in Table 1. To increase the complexity of the optimization problem it is possible to extend the number of parameters adding some fake parameters as in GECCO-09 competition. The vector of parameters can follow the ordering of

Parameter	Section	Name	Unit	min	max
1	Gearbox/gears/2	ratio	SI	0	5
2	Gearbox/gears/3	ratio	SI	0	5
3	Gearbox/gears/4	ratio	SI	0	5
4	Gearbox/gears/5	ratio	SI	0	5
5	Gearbox/gears/6	ratio	SI	0	5
6	Rear Wing	angle	deg	0	18
7	Front Wing	angle	deg	0	12
8	Brake System	front-rear brake repartition	SI	0,3	0,7
9	Brake System	max pressure	kPa	100	150000
10	Front Anti-Roll Bar	spring	lbs/in	0	5000
11	Rear Anti-Roll Bar	spring	lbs/in	0	5000
12	Front Left-Right Wheel	ride height	mm	100	300
13	Front Left-Right Wheel	toe	deg	-5	5
14	Front Left-Right Wheel	camber	deg	-5	-3
15	Rear Left-Right Wheel	ride height	mm	100	300
16	Rear Left-Right Wheel	camber	deg	-5	-2
17	Front Left-Right Suspension	spring	lbs/in	0	10000
18	Front Left-Right Suspension	suspension course	m	0	0,2
19	Rear Left-Right Suspension	spring	lbs/in	0	10000
20	Rear Left-Right Suspension	suspension course	m	0	0,2
21	Front Left-Right Brake	disk diameter	mm	100	380
22	Rear Left-Right Brake	disk diameter	mm	100	380

Table 1: List of the parameters

Table 1 or can use a random mapping (as in GECCO-09 competition) so that the client can not assume any prior knowledge about the parameters to optimize. Although the parameters in Table 1 have different ranges the optimization algorithm inside the client must always assume a range of [0,1]. Then the server will map (uniformly) each value in the corresponding parameter range.

When the server correctly identifies the client, the race starts and the car remains blocked until an evaluation request arrives from the client.

5.2 Evaluations

After the initialization phase the optimization algorithm starts. Each time the algorithm needs an evaluation the following steps occur:

1. The client sends a list of parameters to the server . This list must have size equal to *Parameters Number* and all values must be in the [0,1] interval. Together with this list the client sends a value called *Simulation Time*

2. The server receives the list, maps those values to the real parameters of the car and loads the parameters in the simulation engine

3. The car starts driving in the track using a scripted policy

4. After *Simulation Time* game tics the evaluation terminates, and the car stops

5. The server sends back the performance of the evaluation

6. The client use the values of performance to build the fitness

Each evaluation is performed for *Simulation Time* game tics. It is important to accurately choose this parameter: if the interval is too short the car will drive only in a little part of the track and the fitness can be very noisy; if the interval is too big the *Total Simulation Time* can expire after after not so many evaluations. Since each evaluation can have a different *Simulation Time* the client can adopt a dynamic policy for deciding this interval.

The performance of each evaluation are measured through 4 values: *dist-raced*, *top-speed*, *damage* and *best-lap*. These values are returned to the client that choose which one to use for the fitness function. The fitness could be also a combination of such values (eg. $fitness = topspeed - 2 \cdot damage$). If the *Simulation Time* is short it may be possible that the car is not able to complete an entire lap: in that case the value of *best-lap* will be -1. Finally, it is important to remark that each evaluation starts in that point of the track where the previous evaluation ended. This can be a source of noise when the fitness is evaluated.

5.3 Termination

When the *Total Simulation Time* expires the server will not accept any further evaluation and sends to the client a time-over message. At this point the client must send to the server the champion, i.e. the set of parameters that reached the highest fitness during the overall experiment. To prevent a timeover in a crucial phase of the algorithm, the client can monitor the remaining simulation time and for example decide to stop earlier. The remaining time can be monitored taking in account the sum of the *Simulation Time* of each evaluation with respect to the *Total Simulation Time*.

6 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 optimization algorithm extending the provided client modules. When you want to run your optimization algorithm you have to launch TORCS and start a race, then you have to launch the client extended with your optimization algorithm and finally the optimization experiment will start. To get a more robust experiment it is necessary to launch TORCS with some command line arguments as it is explained in section 6.5. 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 `optserver x` bot to race participants
- 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.

6.1 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 in 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, `optserver 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). However, for the optimization experiments, adding other drivers is not well suited since the fitness noise can drastically increase. 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.

6.2 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`.

6.3 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 optserver  1...
```

The OS terminal should report `Waiting for request on port 3001`. This means that the server-bot `optserver` 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.

6.4 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. To run TORCS in text-mode version it is enough to add the "-T" command line option to TORCS executable:

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

In text-mode, TORCS runs automatically a *Quick Race*, using the XML configuration file (as explained in a previous section) to setup the race. Therefore, you would configure properly your experiment through the `quickrace.xml` file (either using the GUI or directly editing it). Please notice that in the current version it is not possible to specify a configuration to be used from TORCS for the race setup. Accordingly, to perform a batch series of different experiments you have to overwrite the configuration file between an experiment and the next one.

6.5 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 indiduals 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 (or sent to boxes in some cases). 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:

```
$ torcs -nofuel -nodamage -nolaptime (Linux)
```

```
$ wtorcs -nofuel -nodamage -nolaptime (Windows)
```

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

6.6 Time Constraints

Between an evaluation and the successive the optimization algorithm performs some computation. The server measures this time as the interval between the last message of results and the successive message of evaluation. If the total amount of computation time exceed a certain threshold (2 hours of CPU time) the server produces an error and the simulation is ended.

7 Further Information and Support

Further information about the competitions is available at http://cig.dei.polimi.it . All the competition software is available at http://sourceforge.net/projects/cig/.

To report bugs, problems, or just for help, send an email to racingcompetition@gmail.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
- http://www.sigevo.org/gecco-2009, the Genetic and Evolutionary Computation Conference (GECCO-2009) Montréal, Canada, July 8-12 2009
- http://dces.essex.ac.uk/research/evostar/index.htm, EvoStar-2010, Istanbul, Turkey, April 7-9 2010.