

SimGT2003

Thomas Röfer

Contents

1	Introduction	1
2	Getting Started	2
3	Views	3
3.1	Scene View	3
3.2	Robot View	3
3.3	Information Views	3
3.3.1	Image Views	4
3.3.2	Field Views	4
3.3.3	Xabsl2 Views	5
4	Scene Description Files	6
5	Console Commands	6
5.1	Initialization Commands	6
5.2	Global Commands	7
5.3	Robot Commands	8
6	Examples	11
6.1	Recording a Log File	11
6.2	Replaying a Log File	12
6.3	Remote Control	12

1 Introduction

SimGT2003 is based on SimRobot [1], a kinematic robotics simulator. In fact, only a so-called controller has been added to SimRobot that provides the same environment to robot control code that it will also find on the real robots. Therefore, SimGT2003 shares the user interface with SimRobot. This user interface is documented in the online help file that comes with SimRobot. In addition, the scene description language that is used to model the simulation scenes is explained in the help file. Hence, these descriptions are not repeated here.

SimGT2003 is the second Windows tool of the GermanTeam besides RobotControl. While RobotControl focuses on *interaction*, SimGT2003 has its strength in *automation*. The main input channel of SimGT2003 is a console window that is much harder to use than the mouse-enabled interface of RobotControl, but the text based approach to command input also provides the possibility to use script files, which is the key feature to automate a lot of processes. Therefore, SimGT2003 can speed up the development, because—once configured—no further user intervention is required after the start of the program. Therefore, there is no waste of time for opening log files, setting debug keys, switching solutions, and connecting to robots. Besides, SimGT2003 still seems to be more stable than RobotControl, mainly because of its simpler concept. Each process layout has its own set of views, and message handling is not dependent on Windows idle time. The approach requires a lot less synchronization, which also makes SimGT2003 faster than RobotControl. On the other hand, there are a lot of things that cannot be done with SimGT2003, e. g. creating color tables, setting camera parameters, all kinds of OpenGL visualizations, etc. And, in fact, if very different tasks have to be performed in a row as, e. g., during a contest, the mouse-enabled interface of RobotControl is much more comfortable.

2 Getting Started

SimGT2003 can either be started directly from the Windows Explorer (from *T:\GT2003\Bin*), from Microsoft Developer Studio, or by starting a scene description file¹. In the first case, a scene description file has to be opened manually, whereas it will already be loaded in the latter two cases. When a simulation is started for the first time and no layout has been patched into the Windows registry, only the editor window will show up in the main window. Select *Simulation|Start* to run the simulation. The *Tree View* will appear. A *Scene View* showing the soccer field can be opened by double-clicking *WORLD GT2003*. However, the scale of the display will not be appropriate. After selecting *View|Zoom|4x* and *View|Perspective Distortion|Level 1* the field will fit into the window. In addition, the presentation can be simplified by reducing the *View|Detail Level*. Please note that there also exist keyboard shortcuts and toolbar buttons for most commands.

After starting a simulation, a script file may automatically be executed, setting up the robots as desired. The name of the script file is part of the scene description file. Together with the ability of SimRobot to store the window layout, the software can be configured to always start with a setup suitable for a certain task.

Although any object in the *Tree View* can be opened, only displaying certain entries in the object tree makes sense, namely the *WORLD*, the objects in the group *robots*, and all *VALUE*s of objects starting with *VIEW*.

¹This will only work if SimGT2003 was started at least once before.

3 Views

3.1 Scene View

The *Scene View* appears if the *WORLD* is opened from the *Tree View*. As stated above, a 4x-zoom and a level 1 perspective distortion are ideal for displaying the field. The view can be rotated around two axes, and it supports several mouse operations:

- If a robot is clicked between its forelegs (on the field plane), it can be dragged to another position.
- If a robot is clicked between its hind legs, it can be rotated around its body center, i. e. the middle between its forelegs.
- If an active robot (see below) is double-clicked, it is the currently selected robot, i. e. the robot console commands are sent to.
- The ball can be dragged around. Note that its *click position* is on the field plane.

3.2 Robot View

A *Scene View* containing a single robot can be opened by double-clicking a *VEHICLE* in the sub-tree *robots* of the *Tree View*. In such a view, the robot is displayed centered, and it can be zoomed to fill the entire window. This allows seeing more details of the robot, e. g. the state of its LEDs. The view supports four different mouse actions:

- If the back of the robot is clicked, this simulates an activated back switch of the robot. A second click will deactivate the switch.
- If the back area of the top surface of the head is clicked, this simulates an activated back head switch. Again, a second click will deactivate the switch.
- If the front area of the top surface of the head is clicked, this simulates an activated front head switch. It is deactivated by a second click.
- A double-click in the window throws the robot on its side. Note that this can only be seen in the global *Scene View*. In the robot view, the fact that the robot felt down is visualized by hiding its tricot. A second double-click will bring the robot back on its feet.

3.3 Information Views

In SimGT2003, *information views* are used to display debug drawings. These are generated by the robot control program, and they are sent to SimGT2003 via *message queues*. In SimGT2003, the views are defined in the source code. They are instantiated separately for each robot. All views in the current code are defined in `T:\GT2003\Src\Platform\Win32\SimRobot\RobotConsole.cpp`.

There are three kinds of views related to information received from robots: *image views*, *field views*, and *Xabsl2 views*. Field and image views display debug drawings received from the robot, whereas the Xabsl2 views print text information sent by the Xabsl2 behavior control on the robot.

3.3.1 Image Views

An image view displays information in the system of coordinates of a camera image. It is defined by giving it a name and by listing the debug drawings that will be part of the view. The identifiers of all debug drawings are defined in class *Drawings*. Two special elements can be part of an image view that are not debug drawings: *image* and *colorClassImage*. They either show the camera image or the segmented camera image, respectively, and must be the first entries in the list, because they will occlude anything drawn before.

Note that only information can be drawn that is actually sent by the robot, i. e. the corresponding debug requests must have been set. To receive images, either the debug keys *sendImage* or *sendJPEGImage* must have been activated. To display a certain debug drawing *XYZ*, the debug key *send_XYZDrawing* must be set.

For instance, the view *image* is defined as:

```
IMAGE_VIEW(image)
  Drawings::image,
  Drawings::horizon,
  Drawings::linesImageProcessor,
  Drawings::perceptCollection,
  Drawings::selfLocatorLines
END_VIEW(image)
```

To display all this information, console commands (cf. Sect. 5) such as the following are also required:

```
dk sendJPEGImage every 100 ms
dk send_horizonDrawing every 100 ms
dk_send_linesImageProcessorDrawing every 100 ms
dk send_selfLocatorLinesDrawing every 100 ms
dk sendPercepts every 100 ms
```

3.3.2 Field Views

A field view displays information in the system of coordinates of the soccer field. It is defined similar to image views. Two special elements can be part of a field view that are not debug drawings: *fieldPolygons* and *fieldLines*. The field polygons are green, skyblue and yellow areas visualizing the field and goal areas. The field lines are the field boundary and all lines. If used, the field polygons must be the first entry in the list of drawings, because they will occlude anything drawn before.

For instance, the view *worldState* is defined as:

```

FIELD_VIEW(worldState)
  Drawings::fieldPolygons,
  Drawings::fieldLines,
  Drawings::selfLocatorMonteCarlo,
  Drawings::worldState,
  Drawings::perceptCollectionOnField
END_VIEW(worldState)

```

To display all this information, console commands (cf. Sect. 5) such as the following are also required:

```

dk send_selfLocatorMonteCarloDrawing every 500 ms
dk sendPercepts on
dk sendWorldState on

```

Please note that the Monte-Carlo drawing is sent less often, because it is pretty large.

3.3.3 Xabsl2 Views

A single Xabsl2 view is part of each set of views. The information displayed is configured by the console commands *xis* and *xos* (cf. Sect. 5.3). In addition, the debug key *sendXabsl2DebugMessages* must have been set, and a Xabsl2 behavior that matches the one loaded by the console command *xlb* (cf. Sect. 5.3) must be active on the robot.

```

# set behavior control solution to Bremen Byters
sr BehaviorControl Bremen-Byters-soccer

# load behavior of the Bremen Byters
xlb bb

# request Xabsl2 debug messages
dk sendXabsl2DebugMessages on

# show some symbols
xis ball.seen.distance on
xis ball.time-since-last-seen on
xos head-control-mode on

# set output symbol
xos head-control-mode search-for-ball

```

4 Scene Description Files

The language of scene description files is documented in the online help file of SimRobot. However, there are some facts that are special in SimGT2003:

- At the top of a scene description, just below *WORLD*, the instruction *REMARK* can be used to specify the name of the script that will be executed when the simulator is started. A script file contains commands as specified below, one command per line. The default location for scripts is *T:\GT2003\Config\Scenes*, their default extension is *.con*. If no file is specified, SimGT2003 will use *T:\GT2003\Config\Scenes\console.con* if it exists.
- Near the end of a scene description file, there is a group called *robots*. It contains all *active* robots, i. e. robots for which processes will be created.
- Below the group *robots*, there is the group *extras*. It contains *passive* robots, i. e. robots which just stand around, but which are not controlled by a program. Passive robots can be activated by moving their definition to the group *robots*.
- Below that, there is the group *balls*. It contains the balls, i. e. normally a single ball, but it can also contain more of them if necessary, e. g. for the ball challenge in 2002.

A lot of scene description files can be found in *T:\GT2003\Config\Scenes*. Please note that there are two types of scene description files: the ones required to simulate one or more robots (about 20 kb in size), and the ones that are sufficient to connect to a physical robot or to replay a log file (about 1 kb in size).

5 Console Commands

Console commands can either be directly typed into the console window or they can be executed from a script file. There exist three different kinds of commands. First, two commands can only be used in a script file that is executed when the simulation is started. Second, *global commands* change the state of the whole simulation, or they are always sent to all robots. Third, *robot commands* only have an impact on the set of currently *selected robots*.

5.1 Initialization Commands

sc [gameManager] (<a.b.c.d> | <a.b.c> <d> {<d>}). Starts a wireless connection to real robots. The syntax is very similar to the one of the start-script of the router). The command will start the router in the background and will display its messages in the console window. It should only be used once. It will add new robots to the list of available robots (named by the least significant byte of their IP-addresses), and for each of these robots, a full set of views is added to the *Tree View*. Please note that physical robots only send debug drawings on demand, so the views will remain empty until the drawings are requested

by the appropriate debug keys. When the simulation is reset or SimGT2003 is exited, the router will be terminated.

sl <name> <file>. Replays a log file. The command will instantiate a complete set of processes and views. The processes will be fed with the content of the log file. The first parameter of the command defines the name of the virtual robot. This name can be used in the *robot* command (see below), and all views of this particular virtual robot will be identified by this name in the *Tree View*. The second parameter specifies the name and path of the log file. If no path is given, *T:\GT2003\Config\Logs* is used as default. Otherwise, the full path is used. *.log* is the default extension of log files. It will be automatically added if no extension is given.

Please note that the backslash character has to be doubled to be recognized by the system, e. g. write *sl AIBO1 t:\\logs\\hallo* to load the log file *t:\logs\hallo.log*.

When replaying a log file, the replay can only be stopped by halting the simulation, i. e. by pressing the *start/stop* button. To avoid the loss of log data during the replay, select the *simulation time mode*, i. e. execute the command *st on* (see below).

5.2 Global Commands

call <file>. Executes a script file. A script file contains commands as specified here, one command per line. The default location for scripts is *T:\GT2003\Config\Scenes*, their default extension is *.con*.

cls. Clears the console window.

echo <text>. Print text into the console window. The command is useful in script files to print commands that can later be activated manually by pressing the *enter* key.

gc reset | **ready** | **playing** | **final** | **kickOff** (**blue** | **red**) [<blueScore> <redScore>]. Game control. The command is sent to all robots. The *kickOff*-command is interpreted according to the team color of each robot. *gc reset* resets the score counters.

help | **?**. Displays a help text.

jbc <button> <command>. Sets a joystick button command. The first parameter specifies the joystick button by its number between 1 and 32. Any text after this first parameter is part of the second parameter. The second parameter can contain any legal script command. The command will be executed when the corresponding button is pressed. While a joystick button is pressed, no changes in the walking direction of the robot will be accepted. A typical command to be assigned to a button is the executing of a special action, e. g. *jbc 1 mr unswBashOptimized* will try to kick the ball when button 1 is pressed.

jhc tilt | **pan** | **roll**. Set head axis to be controlled by the accelerator lever of the joystick. The other two axes will be controlled by the coolie head. By default, the pan axis is controlled by the accelerator lever.

robot ? | all | <name> {<name>}. Connects the console window to a set of *selected robots*.

All commands in next section are only sent to the selected robots. The command *robot ?* displays a list of all robot names. To select a single simulated robot, it can also be double-clicked in the *Scene View*. To select them all, type *robot all*.

st off | on. Switches the simulation of time on or off. Without the simulation of time, all calls to *SystemCall::getCurrentSystemTime()* will return the real time of the Windows PC. However, as the simulator runs slower than real-time, the simulated robots will receive less sensor readings than the real ones. If the simulation of time is switched on, each step of the simulator will advance the time by 8 ms. Thus, *SimGT2003* simulates real-time, but it is running slower. By default this option is switched off.

<text>. Comment. Useful in script files.

5.3 Robot Commands

ci off | on. Switches the calculation of images on or off. The simulation of the robot's camera image costs a lot of time, especially if multiple robots are simulated. In some development situations, it is a better solution to switch off all low level processing of the robots and to work with *oracled world states*, i. e. world states that are directly delivered by the simulator. In such a case there is no need to waste processing power by calculating camera images. Therefore, it can be switched off. However, by default this option is switched on. Note that this command only has an effect on simulated robots.

dk ? | (<key> off | on | <number> | every <number> [ms]). Sets a debug key. The GermanTeam uses so-called debug keys to switch several options on or off at runtime. Type *dk ?* to get a list of all available debug keys. Debug keys can be activated permanently, for a certain number of times, or with a certain frequency, either on a counter basis or on time. All debug keys are switched off by default.

hcm ? | <mode>. Sets the head control mode. Type *hcm ?* to get a list of all available head control modes.

hmr <tilt> <pan> <roll> <mouth>. Sends a head motion request, i. e. it sets the joint angles of the three axes of the head and the opening angle of the mouth. This will only work if the actual head control mode is *none*. The angles have to be specified in degrees.

log start | stop | clear | save <file>. Records a log file. *log start* starts or continues recording all data received from the robot. *log stop* stops the recording. *log clear* removes all recorded data from memory. *log save* stores the data recorded to the log file with the name specified. If the file already exists, it will be replaced. If no path is given, *T:\GT2003\Config\Logs* is used as default. Otherwise, the full path is used. *.log* is the default extension of log files. It will be automatically added if no extension is given.

mr ? | **<type>** [**<x>** **<y>** **<r>**]. Sends a motion request. This will only work if no *behavior control* is active. Type *mr ?* to get a list of all available motion requests. Walk motions also have to be parameterized by the motion speeds in forward/backward, left/right, and clockwise/counterclockwise directions. Translational speeds are specified in millimeters per second; the rotational speed has to be given in degrees per second.

msg off | **on**. Switches the output of text messages on or off. All processes can send text messages via their debug queues to the console window. As this can disturb entering text into the console window, it can be switched off. However, by default text messages are printed.

pr continue | **illegalDefender** | **obstruction** | **keeperCharged** | **ballHolding**. Penalize robot. The command sends one of the four penalties to all selected robots, or it signals them to continue with the game after a penalty.

qfr queue | **replace** | **reject** | **collect <seconds>** | **save <seconds>**. Send queue fill request. This request defines the mode how the message queue from the debug process to the PC is handled.

queue is the default mode. It will insert all messages received by the debug process from other processes into the queue, and send it as soon as possible to the PC. If more messages are received than can be sent to the PC, the queue will overflow².

replace. If the mode is set to *replace*, only the newest message of each type is preserved in the queue³. On the one hand, the queue cannot overflow, on the other hand, messages are lost, e. g. it is not possible to receive 25 images per second from the robot.

reject will not enter any messages into the queue to the PC. Therefore, the PC will not receive any messages.

collect <seconds>. This mode sends messages to the PC for the specified number of seconds. After that period of time, no further messages will be sent until another queue fill request is sent.

save <seconds>. This mode collects messages for the specified number of seconds, and it will afterwards store them on the memory stick as a log file under *OPEN-R/APP/CONF/LOGFILE.LOG*. No messages will be sent to the PC until another queue fill request is sent.

sg ? | **<id>** { **<num>** }. Sends generic debug data. Generic debug data consists of an *id* and up to ten decimal numbers. Type *sg ?* to list all generic debug data ids.

so off | **on**. Switch sending of *oracled world states* on or off. *Oracled world states* are normally sent to all processes. This allows the modules calculating the world state to be switched off without a failure of the robot. However, the option can produce confusing results if

²Currently, the robot crashes if the queue overflows.

³Currently, this mode does not reasonably work together with debug drawings, because newer drawing commands replace the older ones.

parts of the world state are only sometimes calculated by the robot. Then, the world state sometimes results from the robot's own calculations and sometimes from the simulator. Therefore, sending oracled world states to the robots can be switched off. By default, it is switched on. Note that this command only has an effect on simulated robots.

- sr ?** | **<task>** (**?** | **<solution>**). Sends a solution request. This command allows switching the solutions for a certain task. Type *sr ?* to get a list of all tasks. To get the solutions for a certain task, type *sr <task> ?*.
- tr ?** | **<type>**. Sends a tail request. Type *tr ?* to see all available tail requests.
- xbb ?** | **unchanged** | **<behavior>** { **<num>** }. Selects a Xabsl2 basic behavior. The command suppresses the basic behavior currently selected by the Xabsl2 engine and replaces it with the behavior specified by this command. Type *xbb ?* to list all available Xabsl2 basic behaviors. Some basic behaviors can be parameterized by a list of decimal numbers, e. g. *xbb go-to-point 1600 0 0* to walk to position (1600 mm, 0 mm, 0°). Use *xbb unchanged* to switch back to the basic behavior currently selected by the Xabsl2 engine. The command *xbb* only works if a Xabsl2 behavior was loaded with the command *xlb* (see below).
- xis ?** | **<inputSymbol>** (**on** | **off**). Switches the visualization of a Xabsl2 input symbol in the *Xabsl2 View* on or off. Type *xis ?* to list all available Xabsl2 input symbols. The command *xis* only works if a Xabsl2 behavior was loaded with the command *xlb* (see below).
- xlb ?** | **<name>**. Load a Xabsl2 behavior. The command loads the symbols for the specified behavior and will send the compiled version of the behavior to the robot. The command must be executed before any other Xabsl2 command and the *Xabsl2 View* will work. Type *xlb ?* to list all available behaviors. Please note that the behavior loaded has to match the solution for *behavior control* selected on the robot. To use the *Xabsl2 View*, the corresponding debug key has to be set, i. e. *dk sendXabsl2DebugMessages on*.
- xo ?** | **unchanged** | **<option>**. Selects a Xabsl2 option. The command suppresses the option currently selected by the Xabsl2 engine and replaces it with the option specified by this command. Type *xo ?* to list all available Xabsl2 options. Use *xo unchanged* to switch back to the option currently selected by the Xabsl2 engine. The command *xo* only works if a Xabsl2 behavior was loaded with the command *xlb* (see above).
- xos ?** | **<outputSymbol>** (**on** | **off** | **?** | **unchanged** | **<value>**). Show or set a Xabsl2 output symbol. The command can either switch the visualization of a Xabsl2 output symbol in the *Xabsl2 View* on or off, or it can suppress the state of an output symbol currently set by the Xabsl2 engine and replace it with the value specified by this command. Type *xos ?* to list all available Xabsl2 output symbols. To get the available states for a certain output symbol, type *sr <outputSymbol> ?*. Use *xos <outputSymbol> unchanged* to switch back to the state currently set by the Xabsl2 engine. The command *xos* only works if a Xabsl2 behavior was loaded with the command *xlb* (see above).

6 Examples

This section presents some examples of script files to automate various tasks:

6.1 Recording a Log File

To record a log file, the robot shall send images including the camera matrix and odometry data. The script connects to a robot and configures it to do so. In addition, it prints several useful commands into the console window, so they can be executed by simply setting the caret in the corresponding line and pressing the *enter* key. As these lines will be printed before the messages coming from the router, one has to scroll to the beginning of the console window to use them. Note that the file name behind the line *log save* is missing. Therefore, a name has to be provided to successfully execute this command.

```
# connect to a robot
sc 172.21.3.201

# suppress messages
msg off

# disable everything but sensor data processor and head control
sr SensorDataProcessor Default
sr ImageProcessor disabled
sr SelfLocator disabled
sr BallLocator disabled
sr PlayersLocator disabled
sr RobotStateDetector disabled
sr BehaviorControl disabled
sr HeadControl GT2002

# stop motion
mr normal 0 0 0
hcm none
hmr 0 0 0 0

# queue realtime mode, send JPEG images and odometry
qfr replace
dk sendJPEGImage on
dk sendOdometryData on

# print some useful commands
echo hcm searchForLandmarks
echo hcm searchForBall
echo hcm none
echo hmr 0 0 0 0
```

```

echo log start
echo log stop
echo log save
echo log clear

```

6.2 Replaying a Log File

The example script shown was used to test the LinesImageProcessor2/LinesSelfLocator pair. It instantiates a robot named *LOG1* that is fed by the data stored in the log file *T:\GT2003\Config\Logs\myLogFile.log*.

```

# replay a log file
sl LOG1 myLogFile

# suppress messages
msg off

# simulation time on, otherwise log data may be skipped
st on

# configure modules. Important: sensor data processor disabled
sr SensorDataProcessor disabled
sr ImageProcessor LinesImageProcessor2
sr SelfLocator Lines
sr BallLocator Default
sr PlayersLocator GT2001
sr RobotStateDetector disabled
sr BehaviorControl disabled
sr HeadControl disabled

# request some drawings
dk send_horizonDrawing on
dk send_linesImageProcessorDrawing on
dk send_selfLocatorMonteCarloDrawing on
dk send_selfLocatorLinesDrawing on

# print useful commands
echo sg linesSelfLocator 0.02 0.01 2 150 10 2 200 0.5 0

```

6.3 Remote Control

This script demonstrates joystick remote control of the robot.

```

# connect to a robot

```

```
sc 172.21.3.201

# suppress messages
msg off

# switch off everything but motion
sr ImageProcessor disabled
sr SelfLocator disabled
sr BallLocator disabled
sr PlayersLocator disabled
sr BehaviorControl disabled

# stop motion
mr normal 0 0 0
hcm none
hmr 0 0 0 0
tr noTailWag

# queue realtime mode, send JPEG images
qfr replace
dk sendJPEGImage on

# use accelerator lever to control head pan
jhc pan

# assign actions to joystick buttons
jbc 1 mr unswBashOptimized
jbc 2 mr leftHeadKick
jbc 3 mr rightHeadKick
jbc 4 mr sit
jbc 5 mr scratchHead
jbc 6 tr wagHorizontalFast
jbc 7 tr noTailWag
```

References

- [1] Simrobot homepage. <http://www.tzi.de/simrobot>.