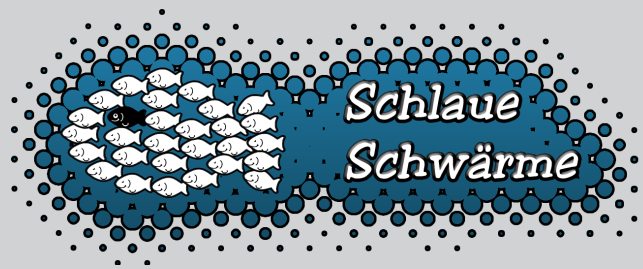UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fachgruppe Algorithmen und Komplexität
Projektgruppe Schlaue Schwärme

Schlaue Schwärme

March 1, 2009

User's Guide

⟶ Alexander Klaas

⟶ Andreas Cord-Landwehr

⟶ Christoph Raupach

⟶ Christoph Weddemann

⟶ Daniel Warner

⟶ Daniel Wonisch

⟶ Kamil Swierkot

⟶ Marcus Märtens

⟶ Martina Hüllmann

⟶ Peter Kling

⟶ Sven Kurras

Contents

1 Introduction

This document shall help you to use the RobotSwarmSimulator to simulate robot swarms and to get useful information by the visualization and statistics modules.

For this purpose the document is divided in several sections. It will be usefull if you just start with the "Getting Started!" section and play a little bit with the simulator and continue with reading the more detailed information on input parameters, input-file specifications, generating simulations and getting statistics.

2 Getting Started

For a first test of RobotSwarmSimulator you can simply run it in generation mode, and use the generated scenario with one of our default algorithms. All example files can be found in subdirectory `ProjectFiles` of the install directory. Here we will present a short example how to simulate the behavior of the CenterOfGravity algorithm (COG).

1. Change to the directory that contains the RobotSwarmSimulator binary.

2. Run the following command:
   `RobotSwarmSimulator --generate --distr-pos 20 --algorithm COGRobot`

   This will generate a simulation specification in form of the following files:

   - `newrandom.swarm` – contains information about the simulation process
   - `newrandom.robot` – contains information about the robots
   - `newrandom.obstacle` – contains information about the obstacles

   The last two files are referenced in the `.swarm` file. Thus, to load the simulation, you only need to load the `.swarm` file. Note that all these files are simple text files that can be edited by hand.

3. Run the command:
   `RobotSwarmSimulator --project-file newrandom --output mylogs`

   This will start the simulation process. There are various keyboard shortcuts that can be used to control the simulation. Press `h` for an overview. You can quit the simulation by pressing `q`.

4. The previous step will generate various output files in the subdirectory `mylogs`, mainly produced by the statistics module of the RobotSwarmSimulator. You may directly use Gnuplot to analyze these files.

This simple example can be used as base for further test runs. Please look at the following chapters to get specifications of the input–files and the user's inferface.

3 Running the RobotSwarmSimulator with Parameters

The RobotSwarmSimulator exists as executable for Linux, Mac OS X and Windows. Each execution of the RobotSwarmSimulator needs specific parameters that are mandatory. By running the RobotSwarmSimulator you have to specify at least, which kind of execution you want. There are two main options:

- By adding the parameter `--generate` the generation mode is started to create new input–files. This will generate three different files having the suffixes `.swarm`, `.robot` and `.obstacle`. The most important of these is the `.swarm` file, which references the other two files.

- Using the parameter `--project-file` ⟨`your_input_file`⟩, the common simulation mode is started. In this case, <your input file> should be the name of a file having the suffix `.swarm` (e.g. generated with generation mode).

There other command line options that may be used to show help or about messages and to customize the generation or simulation mode. Start the RobotSwarmSimulator with the parameter `--help` to get an overview. A typical session may look like this:

```
./RobotSwarmSimulator --help
./RobotSwarmSimulator --project-file <path_to_testdata>/testfile_2 \\
                      --output <path_to_output_files> \\
                      --history-length 10
```

All of the options listed in Listing 1 can be used. Further information for this parameters can be found in the following sections. The definition of most of the parameters can be found in Table 1.

Listing 1: RSS Helpline

```
1  localhost:~$ ./RobotSwarmSimulator --help
2
3  General options:
4   --help              shows this help message
5   --version           shows version of RobotSwarmSimulator
6   --about             tells you who developed this awesome piece of software
7
8  Generator options:
9   --generate                switch to generator mode
10  --seed arg (=1)           seed for random number generator
11  --robots arg (=100)       number of robots
12  --algorithm arg (=NONE)   name of algorithm or lua-file
13  --worldfile arg (=newrandom) world-file for output
14  --robotfile arg (=newrandom) robot-file for output
15  --obstaclefile arg (=newrandom) obstacle-file for output
16  --add-pos-handler         add position request handler for testing
17  --add-vel-handler         add velocity request handler for testing
18  --add-acc-handler         add acceleration request handler for testing
19  --distr-pos arg (=0)      distribute position in cube [0;distr-pos]^3
20  --min-vel arg (=0)        distribute velocity in sphere with radius
21                            min-vel
```

```
22   --max-vel arg (=0)              distribute velocity in sphere with max-vel
23   --min-acc arg (=0)              distribute acceleration in sphere with radius
24                                   min-acc
25   --max-acc arg (=0)              distribute acceleration in sphere with radius
26                                   max-acc
27   --distr-coord                   distribute robot coordinate-systems uniformly
28
29   Simulation options:
30   --project-file arg       Project file to load
31   --output arg             Path to directory for output
32   --history-length arg (=25) history length
```

3.1 General options

--**help** Lists all possible options including a short description.

--**version** This option shows the version information of your RobotSwarmSimulator.

--**about** Get information about the developer team, contact information and more.

3.2 Generator options

--**generate** Switch to generator mode. This is necessary for the further options of this section.

--**seed arg** Sets the seed for the random number generator for robot generation. If not set the seed is 1. An unsigned integer value is expected.

--**robots arg** The number of robots to be generated. The default number is 100. An unsigned integer value is expected.

--**algorithm arg** The name of the algorithm the robots should use. If not set the algorithm SimpleRobot is used. This is only a stub without any functionality. Also the name of a lua-file can be given. Extensionsion .lua is mandatory for lua-files.

--**worldfile arg** The name of the worldfile that shall be generated. Default is newrandom. File-name without extension is expected.

-**robotfile arg** The name of the robotfile that shall be generated. Default is newrandom. File-name without extension is expected.

--**obstaclefile arg** The name of the obstaclefile that shall be generated. Default is newrandom. File-name without extension is expected.

--**add-pos-handler** Causes the generated files to contain a position request handler with reasonable default values. If you need a more sophisticated position request handler, you have to edit the generated .swarm file yourself.

--**add-vel-handler** Causes the generated files to contain a velocity request handler with reasonable default values. If you need a more sophisticated velocity request handler, you have to edit the generated .swarm file yourself.

--add-acc-handler Causes the generated files to contain a acceleration request handler with reasonable default values. If you need a more acceleration request handler, you have to edit the generated `.swarm` file yourself.

--distr-pos arg Distributes the position of robots uniformly at random in the cube $[-arg/2, +arg/2]^3$ If not set, all robots are at position zero.

--min-vel arg The robots will be generated with a velocity distributed uniformly between the given minimum and maximum velocity (see `---max-vel`). This parameter defaults to 0.

--max-vel arg The robots will be generated with a velocity distributed uniformly between the given minimum and maximum velocity (see `---min-vel`). This parameter defaults to 0.

--min-acc arg The robots will be generated with a acceleration distributed uniformly between the given minimum and maximum velocity (see `---max-acc`). This parameter defaults to 0.

--max-acc arg The robots will be generated with a acceleration distributed uniformly between the given minimum and maximum velocity (see `---min-acc`). This parameter defaults to 0.

--distr-coord arg Generates uniformly distributed coordinate–systems for the robots. If this option is not given, all robots will have the same global coordinate system (defined by a unit matrix). If not set, all velocities are zero.

3.3 Simulation options

--project-file arg Specifies the project file use. Use this parameter to load a simulation specified by a `.swarm` file. Not that you may ommit the file extension `.swarm`. Mandatory for simulation.

--output arg Specifies a directory to be used for files generated by the simulation (e.g. statistics files for Gnuplot). The given name is interpreted relative to the current directory and will create the directory if necessary. If ommitted, all generated files will be stored in the current directory.

--history-length arg Sets the history length i.e. the length of the ringbuffer that stores past simulation states. Standard is 25 (which should be a good choice in most cases). An unsigned integer is expected.

4 Using the Simulator-Interface

During the simulation it is possible to interact with the simulation in different ways. The following hot-keys are supported while simulation:

| Space | Start/Stop |

| Q | the Quit RobotSwarmSimulator |

| F1 | Help |

| G | Show Center of all gravity of Swarm |

| V | Show velocity vectors |

| B | Show acceleration vectors |

| K | Show global coordinates system |

| W |, | S | In the corresponding camera mode use | W | for up and | S | for down

Arrow-Keys left, right, before, behind

Mouse Mouse for spinning

| + |, | − | increase/ decrease simulation-speed by constant

| * |, | / | double/ half simulation-speed

| C | Change camera

5 Robot Definitions by LUA-files

There are two ways to define a new robot. One way is hardcoded in the RobotSwarmSimulator   the other way is to define the robots by scripting their behavior in the Lua scripting language. For example Listing 2 shows you how to formulate the COG-algorithm in Lua.

TODO yeah, how can I do this?

Listing 2: COG algorithm in Lua

```lua
function main()
    robots = get_visible_robots();
    center = get_position(get_own_identifier());
    for i = 1, #robots do
        center = center + get_position(robots[i]);
    end
    center = center / (#robots+1);
    add_position_request(center);
end
```

6 Statistics

A simulation run results in three output files of statistics:

- `gnuplot_20091224_184129_ALL.plt` (GNUPlot-configuration file)
- `output_20091224_184129_ALL.plt` (according statistic data)
- `output_20091224_184129_DATADUMP_FULL.plt` (complete data dump)

The file-names results from current date (year, month, day), the current time (hour, minute, second), followed by description of screened object subset (e.ġ. `ALL, MASTERS,`...). For each entry in STATISTICS_SUBSETS there will be created one such data pair.

A Visualization with GNUPlot

Anzeige mit GNUPlot Zur Anzeige der Statistiken eines abgeschlossenen Simulations-
durchlaufes muss die entsprechende GNUPlot-Konfigurationsdatei mit GNUPlot geöffnet
werden: Unter Unix-System kann man hierfür nach Aufruf von `$ gnuplot` die interak-
tive GNUPlot-Shell verwenden (`> load ▪gnuplotfile.plt▪`). Weitere Befehle mit `>`
`help`. Alternativ kann man sich die Ausgabe direkt anzeigen lassen mittels Shellkom-
mando `$ gnuplot -persist ▪gnuplotfile.plt▪`, wobei das `-persist` bewirkt, dass
das Anzeigefenster auch nach Erstellen der Ausgabe geöffnet bleibt. Weitere Flags siehe
`$ man gnuplot`.

Konfiguration der Anzeige Der Inhalt der Datei `output_20091224_184129_ALL.plt`
könnte auszugsweise so aussehen:

```
# time   avg_spd   minball_x  (...)
     0      9.2       20.5
     1     11.3       21.5
     4      9.1       21.9
     6      7.5       22.3
```

Jede Zeile enthält die statistischen Daten zu einem bestimmten Zeitpunkt. Der Beginn
einer neuen Spalte wird hierbei durch (mindestens) ein Leerzeichen gekennzeichnet. '#'
leitet eine Kommentarzeile ein.

In der Datei `gnuplot_20091224_184129_ALL.plt` findet nun die Formatierung der Daten
für die Anzeige statt und kann beliebig geändert werden:

```
# statistics of the simulation
#=================================
set title ' SCHLAUE SCHWÄRME '
set xrange []
set yrange []
set grid
set pointsize 0.5
set xlabel 'time'
set ylabel ''
plot 'output_(...)_ALL.plt' using 1:2 title 'avg_spd' with linespoints, \
     'output_(...)_ALL.plt' using 1:3 title 'minball_x' with linespoints
```

Die Angaben haben folgende Bedeutung:

- `set title '`*Titel*`'` erzeugt einen zentrierten Titel am oberen Teil der grafischen
  Darstellung.

- `set xrange [min:max]` begrenzt den horizontal sichtbaren Bereich zwischen den Werten *min* und *max*, `set xrange []` führt diese Skalierung auf Grundlage der vorliegenden Daten automatisch durch.

- `set grid` blendet ein Gitternetz ein.

- `set pointsize multiplikator` skaliert die Größe der Punkte (falls eine Darstellung gewählt wird, welche Punkte beinhaltet) gemäß *multiplikator*.

- `xlabel 'Label'` beschriftet die X-Achse.

Diese Angaben beinhalten also *nur* grafische grafische Aspekte, die z.B. für die Verwendung in Dokumentationen angepasst werden können. Die Einstellungen der anzuzeigenden Daten finden erst daraufhin statt:

- `plot 'Dateiname.plt'` leitet den tatsächlichen Vorgang der Darstellung ein. Hierbei werden die Daten aus der angegebenen Datei gelesen. Durch `using 1:2` werden die erste und die zweite Spalte gegeneinander dargestellt, wobei der erstgenannte Spaltenindex, also `time`, die X-Koordinate und folglich `avg_spd` die Y-Koordinate definiert. `title 'avg_speed'` fügt für die definierte Funktion einen Legendeneintrag mit dem Titel `avg_speed` hinzu. Der Zusatz `with linespoints` beschreibt die Darstellung durch mit Linien verbundene Zeichen an den Datenpunkten. `linespoints` kann auch ersetzt werden durch:

  - `lines` verbindet jeden Datenpunkt mit einer Linie
  - `dots` zeigt einzelne Punkte an (sinnvoll bei vielen Datenpunkten)
  - `points` fügt ein Zeichen an der Position des Datenpunktes ein. Die Größe kann durch den Befehl `set pointsize` geändert werden
  - `linespoints` erstellt gleichzeitig *points* und *lines*
  - `impulses` erzeugt eine vertikale Linie ausgehend von der X-Achse zum Datenpunkt

  Zum Plotten einer weiteren Funktion in derselben Darstellung wird die gleiche Struktur durch ein Komma getrennt angehangen (auch aus verschiedenen Quelldateien). Dies ermöglicht auf einfache Art, dass die Darstellung der Statistiken unterschiedlicher Simulationsdurchläufe in einer Grafik dargestellt werden können.

Ergebnis

Weitere Informationen zu GNUPlot

[1] offizielle Homepage: `http://www.gnuplot.info/`
[2] Grundkurs: `http://userpage.fu-berlin.de/~voelker/gnuplotkurs/gnuplotkurs.html`
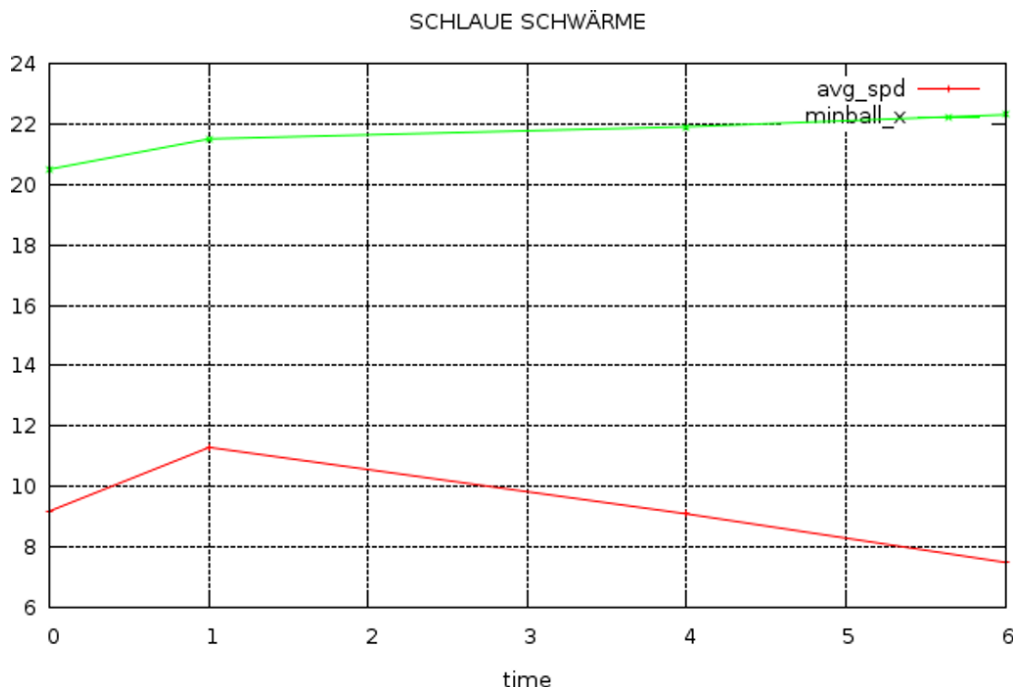
Figure 1: Example output from GNUplot

B Input-file Specifications

There are exactly four kinds of input files for the RobotSwarmSimulator. This includes the project specification files and also the Lua-script-files that define the robot behaviour.

1. The main projectfile containing information about the model. The extension of this type of file is ”'.swarm”'.

2. A file containing robot information. The extension of this file is ”'.robot”'.

3. A file containing obstacle information. The extension of this file is ”'.obstacle”'.

4. Lua file that describes the robot behavouir. The extension of this file is ”'.lua”'.

B.1 Main projectfile

The following specifications hold only for the main projectfile (with extension `.swarm`):

- A comment begins with a '#'.

- A line is a comment line (beginning with a '#'), an empty line or a line containing a variable followed by an equal sign followed by a *quoted* value of this variable. Example:

```
VAR_1="value"
VAR_2 = "value"
VAR_3= "value"
VAR_4 ="value"
```

- a variable name has to be of the following form: `[A-Z0-9_]`$^{+}$

B.1.1 Variables

The main project file contains the variables defined in Tables 1 and 2.

Also the following should be considered:

- The order of the variables in the main project file is not important.
- If a variable does not appear in the main projectfile, then its default value will be used if such a default value does exist (otherwise an exception will be thrown while loading the main project-file).

Variable name	Possible Values	Description	Default
PROJECT_NAME	String	Name of the project	–
COMPASS_MODEL	Still needs to be specified by the ASG-Team. For instance NO_COMPASS	Compass model	FULL_COMPASS
ROBOT_FILENAME	For instance robot_file. The extension of the file must not be appended in this variable.	Filename of the robotfile	same as project file
OBSTACLE_FILENAME	For instance obstacle_file. The extension of the file must not be appended in this variable.	Filename of the robotfile	same as project file
STATISTICS_SUBSETS	A concatenation of none or more of the following strings: {ALL}, {ACTALL}, {INAC-TALL}, {MASTERS}, {ACTMASTERS}, {IN-ACTMASTERS}, {SLAVES}, {ACTSLAVES}, {INACTSLAVES}	Defines the subsets of all robots for which to calculate individual statistical data. E.g. "{ALL} {MASTERS}" will produce statistical information on all robots as well as on *masters only*	NONE
STATISTICS_TEMPLATE	One of the following: "ALL", "BASIC" or "NONE"	Identifies the set of informations to calculate for each subset.	ALL
STATISTICS_DATADUMP	Either "FULL" or "NONE"	Whether or not detailled information (e.g. all robots positions at each event) should be streamed to a file during simulation.	NONE
ASG	SYNCHRONOUS, ASYNCHRONOUS or SEMISYNCHRONOUS	Type of ASG	SYNCHRONOUS
ASYNC_ASG_SEED	unsigned int	Seed for asynchronous ASG, only set if ASG=ASYNCHRONOUS	-
ASYNC_ASG_PART_P	double	Participation Probability for asynch ASG, only set if ASG = ASYNCHRONOUS	-
ASYNC_ASG_TIME_P	double	parameter governing the timing of asynch ASG, only set if ASG = ASYNCHRNOUS. The lower this is the more often events happen.	-
ROBOT_CONTROL	see section B.1.4	RobotControl to use	-

Table 1: Variables in the main project file

Variable name	Possible Values	Description	Default
MARKER_REQUEST_HANDLER_TYPE	element from {STANDARD,NONE}	Type of Marker Request Handler to use	{NONE}
TYPE_CHANGE_REQUEST_HANDLER_TYPE	element from {STANDARD,NONE}	Type of Type Change Request Handler to use.	{NONE}
POSITION_REQUEST_HANDLER_TYPE	element from {VECTOR,NONE}	Type of Position Request Handler to use	{NONE}
VELOCITY_REQUEST_HANDLER_TYPE	element from {VECTOR,NONE}	Type of Velocity Request Handler to use	{NONE}
ACCELERATION_REQUEST_HANDLER_TYPE	element from {VECTOR,NONE}	Type of Acceleration Request Handler to use	v
MARKER_REQUEST_HANDLER_SEED	integer	Seed for Marker Request Handler to use	{NONE}
TYPE_CHANGE_REQUEST_HANDLER_SEED	integer	Seed for Type Change Request Handler to use.	-
POSITION_REQUEST_HANDLER_SEED	integer	Seed for Position Request Handler to use	-
VELOCITY_REQUEST_HANDLER_SEED	integer	Seed for Velocity Request Handler to use	-
ACCELERATION_REQUEST_HANDLER_SEED	integer	Seed for Acceleration Request Handler to use	-
MARKER_REQUEST_HANDLER_DISCARD_PROB	element from interval [0, 1]	Discard probability for Marker Request Handler to use	-
TYPE_CHANGE_REQUEST_HANDLER_DISCARD_PROB	element from interval [0, 1]	Discard probability for Type Change Request Handler to use.	-
POSITION_REQUEST_HANDLER_DISCARD_PROB	element from interval [0, 1]	Discard probability for Position Request Handler to use	-
VELOCITY_REQUEST_HANDLER_DISCARD_PROB	element from interval [0, 1]	Discard probability for Velocity Request Handler to use	-
ACCELERATION_REQUEST_HANDLER_DISCARD_PROB	element from interval [0, 1]	Discard probability for Acceleration Request Handler to use	-
POSITION_REQUEST_HANDLER_MODIFIER	list of vector modifiers (see B.1.3)	List of vector modifiers for Position Request Handler to use	-
VELOCITY_REQUEST_HANDLER_MODIFIER	list of vector modifiers (see B.1.3)	List of vector modifiers for Velocity Request Handler to use	-
ACCELERATION_REQUEST_HANDLER_MODIFIER	list of vector modifiers (see B.1.3)	List of vector modifiers for Acceleration Request Handler to use	-

Table 2: Variables in the main project file

B.1.2 Activation Sequence Generators

There are two Activation Sequence Generators (ASGs). A synchronous ASG and an asynchronous ASG.

To use the synchronous ASG one only needs to set the variable `ASG=SYNCHRONOUS`. No further variables need to be set. To use the asynchronous ASG one needs to set `ASG=ASYNCHRONOUS`. Furthermore one needs to set the following variables:

- `ASYNC_ASG_SEED`: Seed for the ASG

- `ASYNC_ASG_PART`: The higher this is, the more robots are activated for each event. Goes from 0.0 to 1.0.

- `ASYNC_ASG_TIME_P`: The lower this is the smaller is the time difference between events. Be careful with very high values as buffer overflows might happen. Goes from 0.0 to 1.0.

B.1.3 Vector Modifiers

The list of vector modifiers is a (not necessarily nonempty) list, i. e.

$$(\texttt{VECTOR\_MODIFIER\_1});(\texttt{VECTOR\_MODIFIER\_2});...$$

The order of the elements of this list is important. If no Vector Modifier shall be used for the corresponding Request Handler, then use `VECTOR_MODIFIERS=""`.

An element `VECTOR_MODIFIER_k` of the Vector Modifier list is a tuple, defined as follows:

`VECTOR_MODIFIER_k=(VECTOR_MODIFIER_TYPE,VECTOR_MODIFIER_PARAM_1,VECTOR_MODIFIER_PARAM_2,..)`

The number and types of paramters like `VECTOR_MODIFIER_PARAM_1`, `VECTOR_MODIFIER_PARAM_2,...` depends on the corresponding type of the Vector Modifier. Currently there are the following types of Vector Modifiers:

- VectorDifferenceTrimmer

- VectorTrimmer

- VectorRandomizer

I. e. the value of `VECTOR_MODIFIER_TYPE` needs to be `VECTOR_DIFFERENCE_TRIMMER`, `VECTOR_TRIMMER` or `VECTOR_RANDOMIZER`.

VectorDifferenceTrimmmer   If VECTOR_MODIFIER_TYPE is equal to VECTOR_DIFFERENCE_TRIMMER, then the following parameters are expected:

1. length of type double

I. e. an element of the VECTOR_MODIFIERS-list of type VECTOR_DIFFERENCE_TRIMMER may look like: (VECTOR_DIFFERENCE_TRIMMER,5.2).

VectorTrimmer   If vector modifier type equals VECTOR_TRIMMER, then the following parameters are expected:

1. length of type double

I. e. an element of the VECTOR_MODIFIERS-list of type VECTOR_TRIMMER may look like: (VECTOR_TRIMMER,10.0).

VectorRandomizer   If vector modifier type equals VECTOR_RANDOMIZER, then the following parameters are expected:

1. seed of type unsigned int
2. standard derivation of type double

I. e. an element of the VECTOR_MODIFIERS-list of type VECTOR_DIFFERENCE_TRIMMER may look like: (VECTOR_DIFFERENCE_TRIMMER,1,0.5).

B.1.4 RobotControl

The RobotControl variable defines the class which should be used to control the robots (and in particular to control the views of the robots). Currently one of the following classes has be chosen:

1. UNIFORM_ROBOT_CONTROL
2. ROBOT_TYPE_ROBOT_CONTROL

Each class is explained in detail below. Note that each class expects certain class specific parameters.

UniformRobotControl   This class assigns each robot the same view type. The concrete view type needs to be defined using a VIEW variable. The possible values for this variable (view types) are definied below (see B.1.5). E.g. you may to assign each robot global view to the world using ROBOT_TYPE_ROBOT_CONTROL="GLOBAL_VIEW".

RobotTypeRobotControl   This class assigns each robottype the same view type. Therefore robots with different robot types may have different view types. Currently there are two robot types:

1. MASTER
2. SLAVE

To specify which view type should be used by each robot type, there must be variables of the form *RobotType*_VIEW.
The value of each variable has to be a view type (see B.1.5). Note that the view type parameters are also distinguished using the *RobotType* prefix. E.g. you may specify

```
MASTER_VIEW="CHAIN_VIEW"
MASTER_CHAIN_VIEW_NUM_ROBOTS="5"
```

to set the view for master robots to a chain view allowing the robots to see five neighbor robots. Note that exactly one view type should be defined for each robot type.

B.1.5 ViewTypes

The view type of a robot defines its vision model. Whenever a view type is expected you may use one of the following values:

1. GLOBAL_VIEW
2. COG_VIEW
3. CHAIN_VIEW
4. ONE_POINT_FORMATION_VIEW
5. SELF_VIEW

Each view type is explained in detail below.

GLOBAL_VIEW   Allows robots to see literally everything. There are no parameters expected.

COG_VIEW   View model meant to be used for center of gravity algorithms, i.e. every robot can see every other robots position, velocity and acceleration. The coordinate-system and id of each robot is not visible. There are no parameters expected.

SELF_VIEW   View model which allows robots to access every self-related information while disallowing to access any other information. There are no parameters expected.

CHAIN_VIEW   View model meant to be used for robot chain related algorithms, i.e. every robot can see k neighbor robots position. Besides this no more information is visible. When using this view type you have to specify the variable $k \in \mathbb{N}$ using the parameter variable `CHAIN_VIEW_NUM_ROBOTS`.

ONE_POINT_FORMATION_VIEW   View model meant to be used for one point formation algorithms, i.e. every robot can see every other robots position, velocity and acceleration only in a limited view radius r. The coordinate-system and id of each robot is not visible. When using this view type you have to specify the variable $r \in \mathbb{R}$ using the parameter variable `ONE_POINT_FORMATION_VIEW_RADIUS`.

B.1.6 Example of a main project file

A main project file may look like:

```
 1  #
 2  # Description about configuration.
 3  #
 4
 5    PROJECT_NAME="My Exciting Project"
 6    COMPASS_MODEL="NO_COMPASS"
 7    ROBOT_FILENAME="myrobots"
 8    OBSTACLE_FILENAME="myobstacle"
 9    STATISTICS_MODULE="0"
10    ASG="ASYNCHRONOUS"
11    ROBOT_CONTROL="ROBOT_TYPE_ROBOT_CONTROL"
12    MASTER_VIEW="GLOBAL_VIEW"
13    SLAVE_VIEW="ONE_POINT_FORMATION_VIEW"
14    SLAVE_ONE_POINT_FORMATION_VIEW_RADIUS="5.0"
15
16    MARKER_REQUEST_HANDLER_TYPE="STANDARD"
17    STANDARD_MARKER_REQUEST_HANDLER_DISCARD_PROB="0.5"
18    STANDARD_MARKER_REQUEST_HANDLER_SEED="1"
19
20    TYPE_CHANGE_REQUEST_HANDLER_TYPE="NONE"
21    # no additional variables needed
22
23    POSITION_REQUEST_HANDLER_TYPE="VECTOR"
24    VECTOR_POSITION_REQUEST_HANDLER_DISCARD_PROB="0.1"
25    VECTOR_POSITION_REQUEST_HANDLER_SEED="3"
26    VECTOR_POSITION_REQUEST_HANDLER_MODIFIER="(VECTOR_TRIMMER,1.5);(
          VECTOR_RANDOMIZER,5,2.5)"
27
28    VELOCITY_REQUEST_HANDLER_TYPE="VECTOR"
29    VECTOR_VELOCITY_REQUEST_HANDLER_DISCARD_PROB="0.1"
30    VECTOR_VELOCITY_REQUEST_HANDLER_SEED="3"
31    VECTOR_VELOCITY_REQUEST_HANDLER_MODIFIER="(VECTOR_TRIMMER,1.5);(
          VECTOR_RANDOMIZER,5,2.5)"
```

B.2 Robot file

The robotfile uses a csv-compatible format. Therefore the information for one robot has to be saved in exactly one line of the file. Each line contains the following data. The

order of this data is important!

- ID-number
- initial position (x, y, z)
- initial type (for instance master, slave,...)
- initial velocity (x, y, z)
- initial acceleration (x, y, z)
- initial status (maybe sleeping or ready; still has to be specified more precisely)
- initial marker information (still has to be specified)
- algorithm to use (shortcut for an algorithm; still needs to be specified)
- color (using this color a robot is marked for instance for a special treatment during the visualization; this color isn't used anywhere else)
- coordinate system axes (triple $x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3$; this field will be left empty, if axes are supposed to be generated uniformly at random)

The first line always is (column headers):

```
1    "ID","x-position","y-position","z-position","type","x-velocity","y-velocity","
     z-velocity","x-acceleration","y-acceleration","z-acceleration","status","
     marker-info","algorithm","color","x-axis-1","x-axis-2","x-axis-3","y-axis-1",
     "y-axis-2","y-axis-3","z-axis-1","z-axis-2","z-axis-3"
```

Each non-number is quoted.

B.2.1 Example of a robot file

```
1    "ID","x-position","y-position","z-position","type","x-velocity","y-velocity","
     z-velocity","x-acceleration","y-acceleration","z-acceleration","status","
     marker-info","algorithm","color","x-axis-1","x-axis-2","x-axis-3","y-axis-1","
     y-axis-2","y-axis-3","z-axis-1","z-axis-2","z-axis-3"
2    0,5.3,9.2,6.4,"master",1.5,2.5,3.5,1.5,2.5,3.5,"sleeping",0,"MASTER_ALGO"
     ,0,1,0,0,0,1,0,0,0,1
3    1,2.5,4.2,8.8,"slave",1.5,2.5,3.5,1.5,2.5,3.5,"ready",0,"SLAVE_ALGO"
     ,0,1,0,0,0,1,0,0,0,1
```

!   **TODO** type really "master"/"slave" or rather 0/1? (scenario_generator generates latter atm)

B.3 Obstacle file

Like the robot file the obstacle file uses a csv-compatible format. Therefore the information for one robot has to be saved in exactly one line of the file. Each line contains the following data. The order of this data is important!

- type (marker, sphere or box)

- position (x, y, z)

- marker information (still needs to be specified)

- $x/y/z$-lengths or radius (depending on type)

The first line always is (column headers):

```
1  "type","x-position","y-position","z-position","marker-info","size-info","",""
```

Each non-number is quoted.

B.3.1 Example of an obstacle file

```
1  "type","x-position","y-position","z-position","marker-info","size-info","",""
2  "box",2.0,3.0,4.0,0,1.0,2.0,3.0,
3  "sphere",3.4,5.2,5.1,0,5.0,"",""
4  "marker",3.5,1.4,5.1,0,"","",""
```

As you can already see in the example, if the type of an obstacle is sphere, then the last two values must be empty, i. e. ",". Analoguos, if the type is marker, the last three values must be empty, i. e. ",",".