UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fachgruppe Algorithmen und Komplexität
Projektgruppe 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 RobotSwarmSimulatorto 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 RobotSwarmSimulatorou 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 RobotSwarmSimulatorinary.
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 RobotSwarmSimulatorwith Parameters

TODO describe the package structure

The RobotSwarmSimulatorexists as executable for Linux, MacOS and Windows in the corresponding directories. Each execution of the RobotSwarmSimulatorneeds specific parameters that are mandatory. By running the RobotSwarmSimulatoryou have to specify at least, which kind of execution do you want. By adding the parameter `--generate` the generation mode is started to create new input-files. On the other side by adding the parameter `--project-file=<your input>` the common simulation mode is started. Also a run with special parameters like `--help` or `--about` is possible. Thus a typical run look like:

```
./RobotSwarmSimulator --help
./RobotSwarmSimulator --project-file <path_to_testdata>/testfile_2
./RobotSwarmSimulator --project-file <path_to_testdata>/testfile_2
                      --history-length 10
./RobotSwarmSimulator --generate --distr-pos=17.0
```

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  General options:
2    --help              shows this help message
3    --version           shows version of RobotSwarmSimulator
4    --about             tells you who developed this awesome piece of software
5
6  Generator options:
7    --generate                  switch to generator mode
8    --seed arg (=1)             seed for random number generator
9    --robots arg (=100)         number of robots
10   --algorithm arg (=NONE)     name of algorithm or lua-file
11   --worldfile arg (=newrandom) world-file for output
12   --robotfile arg (=newrandom) robot-file for output
13   --obstaclefile arg (=newrandom) obstacle-file for output
14   --distr-pos arg (=0)        distribute position in cube [0;distr-pos]^3
15   --distr-vel arg (=0)        distribute velocity in cube [0;distr-vel]^3
16   --distr-acc arg (=0)        distribute acceleration in cube [0;distr-acc]^3
17   --distr-coord               distribute robot coordsystems uniformly
18
19 Simulation options:
20   --project-file arg      Project file to load
21   --history-length arg (=25) history length
```

3.1 General options

--help Lists all possible options.

--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`, but you really should set this value. 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.

--distr-pos=arg Distributes the position of robots uniformly at random in box of size `arg` × `arg` × `arg`. If not set, all robots are at position zero.

--distr-vel=arg Distributes the velocity of robots uniformly at random by vectors in box of size `arg` × `arg` × `arg`. If not set, all velocities are zero.

--distr-acc=arg Distributes the acceleration of robots uniformly at random by vectors in box of size `arg` × `arg` × `arg`. If not set, all velocities are zero.

3.3 General options

--project-file Specifies the project file use. Do not add extension `.swarm`. Mandatory for simulation.

--history-length=arg Sets the history length i.e. the length of the ringbuffer to store world-objects until they get visualized. Standard is 25. 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`,`A`,`S`,`D` 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

```
1 function main()
2     robots = get_visible_robots();
3     center = get_position(get_own_identifier());
4     for i = 1, #robots do
5         center = center + get_position(robots[i]);
6     end
7     center = center / (#robots+1);
8     add_position_request(center);
9 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

Display with GNUPlot To display statistical information about a finished simulation run the correct GNUPlot configuration file needs to be opened with GNUPlot. On Unix systems (with installed gnuplot) the interactive GNUPlot shell can be used ((`> load ■gnuplotfile.plt■`). More commands can be displayed using `> help`. Also it is possible to directly display the file with `$ gnuplot -persist ■gnuplotfile.plt■`. The additional option `-persist` causes the window to stay open after the file has been opened. For more informaion on gnuplot read the manual page `$ man gnuplot`.

Display configuration A file `output_20091224_184129_ALL.plt` might look like this:

# 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	

Each line contains extremly valuable statistical data for a certain time. The start of a new column is marked by one (or more) whitespaces. comment lines are started with a '#' sign.

The file `gnuplot_20091224_184129_ALL.plt` also contains the formatting of the extremly valuable statistical data. The formatting can be changed if so desired.

```
# 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
```

The lines have the following meaning:

- `set title '`*`Titel`*`'` generates a title within the upper part of the graphical display.

- `set xrange [`*min*`:`*max*`]` limits the visible area (horizontal) to the area between *min* ans *max*, `set xrange []` scales the extremly valuable statistical data based on the datapoints
- `set grid` displays a beautiful grid.
- `set pointsize` *multiplikator* scales the size of the points (if there are any) according to *multiplikator*.
- `xlabel` *'Label'* labels the x-axis with useful information.

These choices only concern the graphical display. The configuration of the displayed data is done later.

- `plot` *'Dateiname.plt'* starts the display process. Data is read from the input file. The parameter `using 1:2` uses the first column as x-axis and the second column as y-axis. `title 'avg_speed'` adds a title for the function defined by this. The addition `with linespoints` causes the data points to be displayed with lines between them. `linespoints` can be replaced by:
  - `lines` connects each data point with a line
  - `dots`displays datapoints as dots (for many datapoints)
  - `points` displays datapoints as points
  - `linespoints` displays a combination of *points* and *lines*
  - `impulses` displays a vertical line for each data point

Result

More about GNUPlot

[1] Homepage: `http://www.gnuplot.info/`
[2] Introductory course: `http://userpage.fu-berlin.de/~voelker/gnuplotkurs/gnuplotkurs.ht`

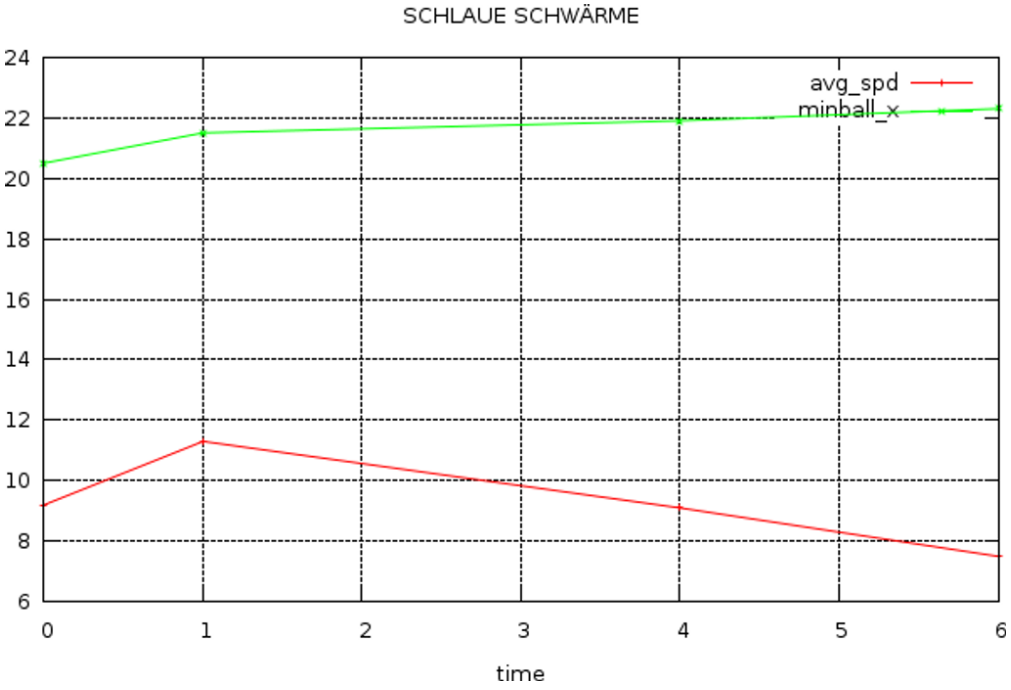

Figure 1: Example output from GNUplot

B Input-file Specifications

There are exactly four kinds of input files for the RobotSwarmSimulator This includes the project specification files and also the Luascript-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. Luafile 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}, {INACTALL}, {MASTERS}, {ACTMASTERS}, {INACTMASTERS}, {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.

```
(VECTOR_MODIFIER_1);(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:

```
#
# Description about configuration.
#

  PROJECT_NAME="My Exciting Project"
  COMPASS_MODEL="NO_COMPASS"
  ROBOT_FILENAME="myrobots"
  OBSTACLE_FILENAME="myobstacle"
  STATISTICS_MODULE="0"
  ASG="ASYNCHRONOUS"
  ROBOT_CONTROL="ROBOT_TYPE_ROBOT_CONTROL"
  MASTER_VIEW="GLOBAL_VIEW"
  SLAVE_VIEW="ONE_POINT_FORMATION_VIEW"
  SLAVE_ONE_POINT_FORMATION_VIEW_RADIUS="5.0"

  MARKER_REQUEST_HANDLER_TYPE="STANDARD"
  STANDARD_MARKER_REQUEST_HANDLER_DISCARD_PROB="0.5"
  STANDARD_MARKER_REQUEST_HANDLER_SEED="1"

  TYPE_CHANGE_REQUEST_HANDLER_TYPE="NONE"
  # no additional variables needed

  POSITION_REQUEST_HANDLER_TYPE="VECTOR"
  VECTOR_POSITION_REQUEST_HANDLER_DISCARD_PROB="0.1"
  VECTOR_POSITION_REQUEST_HANDLER_SEED="3"
  VECTOR_POSITION_REQUEST_HANDLER_MODIFIER="(VECTOR_TRIMMER,1.5);(
      VECTOR_RANDOMIZER,5,2.5)"

  VELOCITY_REQUEST_HANDLER_TYPE="VECTOR"
  VECTOR_VELOCITY_REQUEST_HANDLER_DISCARD_PROB="0.1"
  VECTOR_VELOCITY_REQUEST_HANDLER_SEED="3"
  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):

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

Each non-number is quoted.

B.3.1 Example of an obstacle file

```
"type","x-position","y-position","z-position","marker-info","size-info","",""
"box",2.0,3.0,4.0,0,1.0,2.0,3.0,
"sphere",3.4,5.2,5.1,0,5.0,"",""
"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. ",",".