UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fachgruppe Algorithmen und Komplexität
Projektgruppe Schlaue Schwärme

February 25, 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 **4**

2 Starting Simulator **4**
2.1 General options 5

3 Using the 3d-Interface **5**

A Input-file Specifications **5**
A.1 Main projectfile 6
A.1.1 Variables 6
A.1.2 Input specification for Request Handler 8
A.1.3 Vector Modifiers 9
A.1.4 Example of a main project file 10
A.2 Robot file 11
A.2.1 Example of a robot file 11
A.3 Obstacle file 12
A.3.1 Example of an obstacle file 12

1 Introduction

This document shall help you how to use the RobotSwarmSimulator For this purpose here are listed all input-parameters, project-file specifications and GUI-inputs.

2 Starting Simulator

The RobotSwarmSimulatorexists as executable for Linux, MacOS and Windows. By starting from command-line you need to specify the mandatory project configuration file (`*.swarm` or otherwise the `--generate` option an the according parameters. Thus a typical execution of the RobotSwarmSimulatorlooks like this:

```
./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
```

! **TODO** all parameters from help should be documented here in full length

All of the parameters listed in Listing 1 can be used. Further information for this parameters can bin found in the following sections.

Listing 1: RSS Helpline

```
General options:
  --help              shows this help message
  --version           shows version of RobotSwarmSimulator
  --about             tells you who developed this awesome piece of software

Generator options:
  --generate                   switch to generator mode
  --seed arg (=1)              seed for random number generator
  --robots arg (=100)          number of robots
  --algorithm arg (=NONE)      name of algorithm or lua-file
  --worldfile arg (=newrandom) world-file for output
  --robotfile arg (=newrandom) robot-file for output
  --obstaclefile arg (=newrandom) obstacle-file for output
  --distr-pos arg (=0)         distribute velocity in cube [0;distr-pos]^3
  --distr-vel arg (=0)         distribute velocity in cube [0;distr-vel]^3
  --distr-acc arg (=0)         distribute velocity in cube [0;distr-acc]^3
  --distr-coord                distribute robot coordsystems uniformly

Simulation options:
  --project-file arg      Project file to load
  --history-length arg (=25) history length
```

2.1 General options

--help Lists all possible options

--version Current version of RobotSwarmSimulator[--about] The developer team

3 Using the 3d-Interface

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

Space Start/Stop

q 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 W for up, S for down

Arrow-Keys left, right, before, behind

mouse-movement mouse for spinning

+, - increase/ decrease simulation-speed by constant

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

c Change camera

A 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"’.

A.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_]`$^+$

A.1.1 Variables

The main project file contains the variables defined in table 1:

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
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	-
MARKER_REQUEST_HANDLER	see section ??	Marker Request Handler to use	-
MARKER_CHANGE_REQUEST_HANDLER	see section ??	Marker Change Request Handler to use	-
TYPE_CHANGE_REQUEST_HANDLER	see section ??	Type Change Request Handler to use.	-
POSITION_REQUEST_HANDLER	see section ??	Position Request Handler to use	-
VELOCITY_REQUEST_HANDLER	see section ??	Velocity Request Handler to use	-
ACCELERATION_REQUEST_HANDLER	see section ??	Acceleration Request Handler to use	-
ROBOT_CONTROL	see section ??	RobotControl to use	-

Table 1: Variables in the main project file

Additional restrictions and information:

- The order of the variables in the main project file isn't important.
- If a variable doesn't appear in the main projectfile, then its default value will be used (if there exists a default value, otherwise an exception will be thrown while loading the main projectfile).

A.1.2 Input specification for Request Handler

< .mine For each kind of request handler you want to use, insert the appropriate variable in the main project file. (For each kind of request for which the type of request handler is not specified none will be used) There are the following request handler kinds:
`MARKER_REQUEST_HANDLER_TYPE, TYPE_CHANGE_REQUEST_HANDLER_TYPE, POSITION_REQUEST_HANDLE`
`VELOCITY_REQUEST_HANDLER_TYPE, ACCELERATION_REQUEST_HANDLER_TYPE`

The possible values of the types are at the moment:

`MARKER_REQUEST_HANDLER_TYPE` $\in$ {`STANDARD, NONE`}
`TYPE_CHANGE_REQUEST_HANDLER_TYPE` $\in$ {`STANDARD, NONE`}
`POSITION_REQUEST_HANDLER_TYPE` $\in$ {`VECTOR, NONE`}
`VELOCITY_REQUEST_HANDLER_TYPE` $\in$ {`VECTOR, NONE`}
`ACCELERATION_REQUEST_HANDLER_TYPE` $\in$ {`VECTOR, NONE`}

Depending on the chosen type more variables have to be specified.

For `MARKER_REQUEST_HANDLER_TYPE=STANDARD`

- `STANDARD_MARKER_REQUEST_HANDLER_SEED` $\in \mathbb{N}$
- `STANDARD_MARKER_REQUEST_HANDLER_DISCARD_PROB` $\in [0, 1]$

For `TYPE_CHANGE_REQUEST_HANDLER_TYPE=STANDARD`

- `STANDARD_TYPE_CHANGE_REQUEST_HANDLER_SEED` $\in \mathbb{N}$
- `STANDARD_TYPE_CHANGE_REQUEST_HANDLER_DISCARD_PROB` $\in [0, 1]$

For `POSITION_REQUEST_HANDLER_TYPE=VECTOR`

- `VECTOR_POSITION_REQUEST_HANDLER_SEED` $\in \mathbb{N}$
- `VECTOR_POSITION_REQUEST_HANDLER_DISCARD_PROB` $\in [0, 1]$

- VECTOR_POSITION_REQUEST_HANDLER_MODIFIER : Liste von Vector Modifiern (siehe A.1.3)

For VELOCITY_REQUEST_HANDLER_TYPE=VECTOR

- VECTOR_VELOCITY_REQUEST_HANDLER_SEED $\in \mathbb{N}$
- VECTOR_VELOCITY_REQUEST_HANDLER_DISCARD_PROB $\in [0,1]$
- VECTOR_VELOCITY_REQUEST_HANDLER_MODIFIER : List of vector modifiers (see A.1.3)

For POSITION_REQUEST_HANDLER_TYPE=VECTOR

- VECTOR_ACCELERATION_REQUEST_HANDLER_SEED $\in \mathbb{N}$
- VECTOR_ACCELERATION_REQUEST_HANDLER_DISCARD_PROB $\in [0,1]$
- VECTOR_ACCELERATION_REQUEST_HANDLER_MODIFIER : list of vector modifiers (see A.1.3)

For a specification of VECOTR_MODIFIERS see section A.1.3.
=======

A.1.3 Request Handler without Vector Modifier

The value of a Request Handler without Vector Modifier is a tuple of the form:

(TYPE,DISCARD_PROB,SEED)

with
TYPE $\in$ {STANDARD,NONE}
DISCARD_PROB $\in [0,1]$
SEED $\in \mathbb{N}$

A.1.4 Vector Modifiers

VECTOR_MODIFIERS is a (not necessarily nonempty) list, i. e.

VECTOR_MODIFIERS=VECTOR_MODIFIER_1;VECTOR_MODIFIER_2,...

The order of the elements of this list is important.
If there shall be used no Vector Modifier 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 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. VECTOR_MODIFIER_TYPE $\in$ {VECTOR_DIFFERENCE_TRIMMER,VECTOR_TRIMMER,VECTOR_RANDOMIZER}

If VECTOR_MODIFIER_TYPE=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).

If VECTOR_MODIFIER_TYPE=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).

If VECTOR_MODIFIER_TYPE=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).

A.1.5 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 excepts certain class specific parameters.

UniformRobotControl This class assigns each robot the same view type. E.g. each robot should have a global view to the world. The concrete view type needs to be defined using a `VIEW` variable. The possible values of `VIEW` are definied below (see **??**).

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 **??**). E.g. you may specify `MASTER_VIEW = GLOBAL_VIEW` to set the view for master robots to global view. Note that exactly one view type should be defined for each robot type.

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`

A.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"
```

```
10  ASG="ASYNCHRONOUS"
11  VIEW="0"
12
13  MARKER_REQUEST_HANDLER_TYPE="STANDARD"
14  STANDARD_MARKER_REQUEST_HANDLER_DISCARD_PROB="0.5"
15  STANDARD_MARKER_REQUEST_HANDLER_SEED="1"
16
17  TYPE_CHANGE_REQUEST_HANDLER_TYPE="NONE"
18  # no additional variables needed
19
20  POSITION_REQUEST_HANDLER_TYPE="VECTOR"
21  VECTOR_POSITION_REQUEST_HANDLER_DISCARD_PROB="0.1"
22  VECTOR_POSITION_REQUEST_HANDLER_SEED="3"
23  VECTOR_POSITION_REQUEST_HANDLER_MODIFIER="(VECTOR_TRIMMER,1.5);(
        VECTOR_RANDOMIZER,5,2.5)"
24
25  VELOCITY_REQUEST_HANDLER_TYPE="VECTOR"
26  VECTOR_VELOCITY_REQUEST_HANDLER_DISCARD_PROB="0.1"
27  VECTOR_VELOCITY_REQUEST_HANDLER_SEED="3"
28  VECTOR_VELOCITY_REQUEST_HANDLER_MODIFIER="(VECTOR_TRIMMER,1.5);(
        VECTOR_RANDOMIZER,5,2.5)"
```

A.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.

A.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) !

A.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.

A.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. ",",".