RoboLog Koblenz 3D – Team Description for 3D Development Competition 2005

Heni Ben Amor, Joschka Boedecker, Anita Maas, Jan Murray, Oliver Obst,
Achim Rettinger, Christoph Ringelstein, and Markus Rollmann

Universität Koblenz-Landau, AI Research Group, D-56070 Koblenz
{amor,jboedeck,amaas,murray,fruit,achim,cringel,rollmark}@uni-koblenz.de

Abstract. This is a description of the work by members of the team RoboLog3D for the 3D Development Contest at RoboCup 2005 in Osaka. Our main contributions are a powerful scene description language, new robot models, more FIFA rule implementations, and an offline trainer that facilitates experiments and data collection in the new simulator environment. Many of the implementations use the *Zeitgeist* framework that comes with the 3D simulator, or use its flexible plug-in mechanism.

1 Introduction

The RoboCup 2005 3D Development Competition is an attempt to speed up the development of the current simulation environment by encouraging members of the community to present implementations of improvements, and add missing functionality to it. In this description, we give an overview of our contributions for this competition. Parts of it are already implemented at the time of this writing, others are still under development.

As was outlined in the rules for this competition, various topics can be thought of that would improve on or add to the functionality of the current simulator. Our team worked on several of them in parallel, using the flexible *Zeitgeist* [2, 3] plug-in system of the 3D server in the implementation process.

2 Scene Description Language

In [4], Obst and Rollmann presented a powerful Scene Description Language called RubySceneGraph. It maps the scene graph structure of a scene for the simulator to the nesting of Lisp-like expressions. The importer for the scene description files relies on the Zeitgeist class factory services to create an object of the requested type. A node expression can be parameterized with function calls in order to access properties of a scene node in the scene graph. A function call expressed as S-expression is realized using the Ruby script function exported from the corresponding C++ class. Further details can be found in [1].

3 New Robot Models

Using the scene description language described above, we implemented two different kinds of robot models for the simulator. The first one is a small car with two front wheels with active motors and a back wheel in the rear. It has a vision perceptor and a kick effector to affect the ball. This robot model is included in the current 3D server implementation. It is illustrated in figure 1

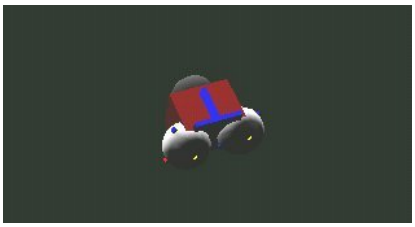

Fig. 1. The small car robot model as included in the current simulator

Furthermore, we are working on implementing a model for a HOAP-2 robot from Fujitsu. This is an attempt to prepare the grounds for future development towards humanoid simulation in the 3D Simulation League.

4 Rule Implementation

We will implement the FIFA offside rule for the simulation. If time permits, we will also try to implement a special goalie agent which is allowed to catch the ball during the match.

5 Offline Trainer

We developed a special monitor client that can be used to send commands to the server which affect the properties of the ball or the agents on the field as described in [1]. It is possible to affect the position and velocity of these objects, but also internal player parameters like the battery state. The trainer can be used in addition to a regular monitor. It does not visualize the current simulation, but it receives all the information describing the scene at every simulation step. Two modes are available for the trainer, namely interactive and automated mode.

In interactive mode, commands can be sent to the server from the command line using S-Expressions. As an example, if the position of agent number 1 of the team playing on the left side is to be changed the point $(0.0, 0.0, 0.0)$, the corresponding S-Expression would be:

```
(agent (team L)(unum 1)(pos 0.0 0.0 0.0))
```

In automated mode, the program is provided with a file of *scenarios*, i.e., initial configurations for player and/or ball positions and velocities. Furthermore, it will use *evaluation functions*, to determine the termination condition for a scenario. The trainer will reset the current scenario once the termination condition is met and record the results for further evaluation. Every scenario is repeated a specified number of times before the next scenario is used. This is done for all the scenarios in the given file. Using the automated mode, the trainer can be used to easily setup experiments and collect data for evaluation purposes.

6 Monitor GUI

A further improvement we plan to make to the server is to implement a monitor with additional functionality. First, it would be desirable to have access to the scene graph of the simulation. This would, for instance, facilitate the construction of new robot models for the simulation. Second, the trainer program described above could be intregrated into the monitor and could also benefit from the easier usability provided by the graphical user interface.

References

1. Robocup soccer server 3d manual. available at http://www.sf.net/projects/sserver/, 2004.
2. Marco Kögler. Simulation and Visualization of Agents in 3d Environments. Technical report, Universität Koblenz-Landau, 2003.
3. Marco Kögler and Oliver Obst. Simulation league: The next generation. In Daniel Polani, Andrea Bonarini, Brett Browning, and Kazuo Yoshida, editors, *Proceedings of the RoboCup Symposium 2003*, July 2003. An LNAI version of the proceedings will appear later.
4. Oliver Obst and Markus Rollmann. SPARK – A Generic Simulator for Physical Multiagent Simulations. In Gabriela Lindemann, Jörg Denzinger, Ingo J. Timm, and Rainer Unland, editors, *Multiagent System Technologies – Proceedings of the MATES 2004*, volume 3187 of *Lecture Notes in Artificial Intelligence*, pages 243–257. Springer, September 2004.