Ehrenwörtliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig angefertigt und keine weiteren als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, den 15. Juni 2011

Unterschrift

Abstract

The Small-Size-League of the international RoboCup-competition is – besides the simulation league - the fastest throughout the contest. Thus the algorithms for the playing-behavior are more complex and diversified then these in the other leagues. To effectively develop this kind of algorithms, an unconditionally available test-system is inevitable. As the setup of the whole SSL-equipment is not always possible due to organizational, geographical or temporal reasons, this is a severe problem.

As a solution many teams implemented their own simulations of the RoboCup-match to fit their needs. This not only enables developers to enhance their algorithms independently from the rest of the team, but gives them immediately feedback to their changes and thus installs a “software-in-the-loop”-process.

This paper describes the development of such a simulator in the team “TIGERS Mannheim”, which not only suits the needs of this team, but is highly customizable in terms of used protocols, robot-models and physical parameters, to provide a common software-framework which can be used by other teams, too.

Table of Contents

List of Abbreviations

API ..... **A**pplication **P**rogramming **I**nterface

DFKI ..... **D**eutsches **F**orschungszentrum für **K**ünstliche **I**ntelligenz

DOF ..... **D**egree **O**f **F**reedom

JME ..... **jM**onkey**E**ngine

LWJGL ..... **L**ight **W**eight **J**ava **G**ame **L**ibrary

ODE ..... **O**pen **D**ynamics **E**ngine

SSL ..... **S**mall-**S**ize-**L**eague

Table of Figures

1 Introduction

This seminar paper has been realized in the context of the RoboCup-Team “TIGERS Mannheim”. The RoboCup is a competition for evolving research and education in the area of robotics all over the world. The vision of this contest is to develop a team of humanoid robots which is capable of beating the FIFA world-champion in a soccer game by the year 2050. The biggest event is the international championship once a year, where robots play against each other in several leagues.

“TIGERS Mannheim” is a new team of undergraduate students from Germany, who started to think about attending the RoboCup – and especially the Small-Size-League – at the beginning of 2009. The project was started from scratch but with a highly motivated team. From the very beginning the goal was to successfully participate in the RoboCup-championship 2011 in Istanbul, and to found a solid base for new generations of students to come.

In the RoboCup SSL-League two teams consisting of 5 robots compete against each other. The soccer-field is about 6x4m large and the robot’s size is limited to 180x150mm. In contrast to the other leagues these robots have no complex motion-model as nearly all teams use a four-wheeled chassis which is way easier to control then a legged robot, for example. This design-constraints makes them very lightweight, a single robot does usually not exceeds 2,5kg. In combination, these factors lead to very fast robots, and cause the game-speed to be dramatically higher than in the other hardware-leagues, even the “Middle Size League”. This sets the focus of the SSL-League clearly on the tactics and behavior of the robots.

Developing such complex and diversified behaviors like “playing soccer” soon leads to a major problem: An unconditionally available test-system is inevitable. Developers need immediate feedback on their changes to effectively design algorithms for this task. And as the setup of the whole SSL-equipment is not always possible due to several reasons there is a strong necessity for some kind of substitution of the hardware.

During the last years some teams solved this problem by developing their own simulation software to install a “software-in-the-loop” process. But most of them are not published, and the rest is not powerful enough, yet (**[Twent]**).

The objective of this seminar paper is to create a concept for a simulation-environment for the SSL-League. The focus of this environment is to evaluate and support the development of the central control-software and especially the AI of “TIGERS Mannheim”.

2 Simulation in the SSL-League

This section describes the system-setup of a RoboCup SSL-match and analyzes the various aspects of it which have to be emulated. Besides technical necessities also operability is concerned and finally formulated as requirements for the simulation.

2.1 The System of the SSL-League

The general setup of the system is defined in the league rules (**[SSL11]**). They describe not only the rules of the game, but also the technical limitations to ensure a fair game of robot-soccer.

A SSL-match consists of two teams with five robots each, an orange (gulf) ball and a soccer-like field, with the dimension of about 6x4meters. The rules say that these robots diameter may not extend 180mm, and their height not 150mm. All robots of one team are typically controlled by a central control unit (CCU) a next to the field.

Both teams units receive their knowledge about the robots positions from two cameras installed several meters above the field. The data of these cameras is interpreted by a central program called “SSL-Vision”[1] and distributed via multi-cast over a local network connecting the vision-server and the two teams. This program has been standardized in 2009, and no other nor additional vision-software is allowed since then.[2]

Another part is the Referee-software, which is controlled by a human referee. It uses the same network to send messages describing the current state of the game to the control-units of both teams.

[1] The projects website: http://*code.google.com/p/**ssl-vision***/

[2] These cameras and the corresponding software “SSL-Vision” were standardized in 2009 to reduce the advantage single teams gained through their superior visual recognition-software and to adjust the focus of the league on the actual gameplay.

An overview of this setup is given in Figure 1:

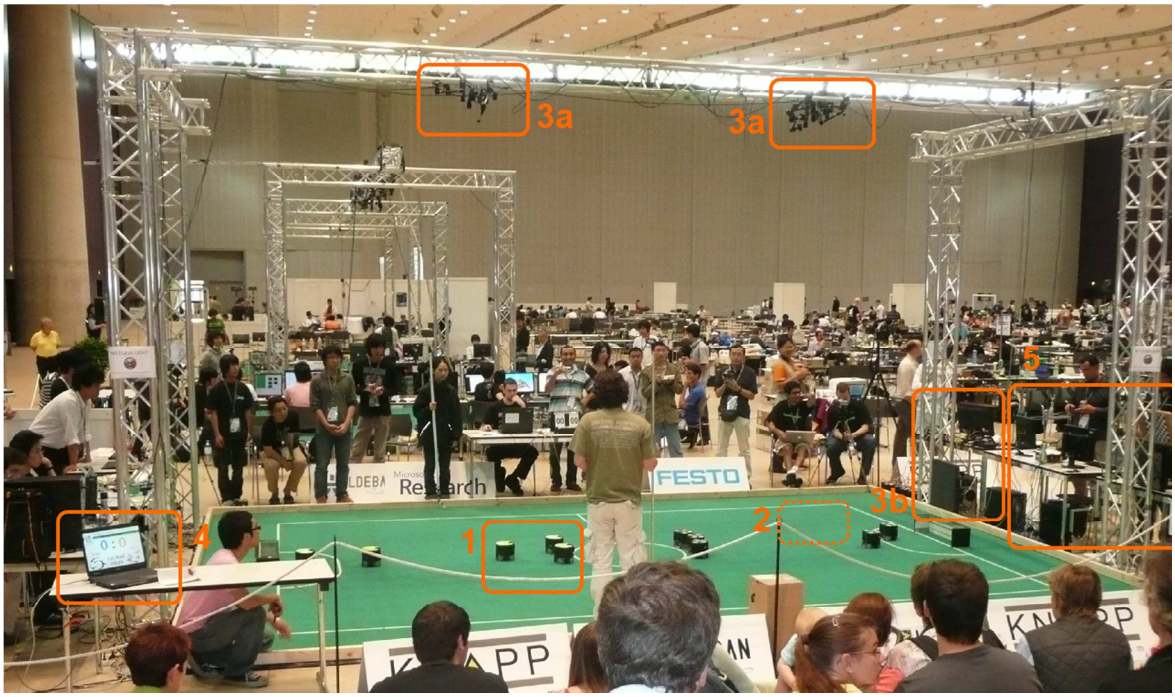

Figure 1: A typical SSL-match setup (from the RoboCup 2009 in Graz, Austria)

Summarized, the explained setup for one team consists of five main parts:

1. The five robots
2. The physical world around the robots (the field, the ball and the opponents)
3. The positioning system (the cameras and SSL-Vision)
4. The referee-software
5. The central control unit/software

The following diagram visualizes the abstract data flow:

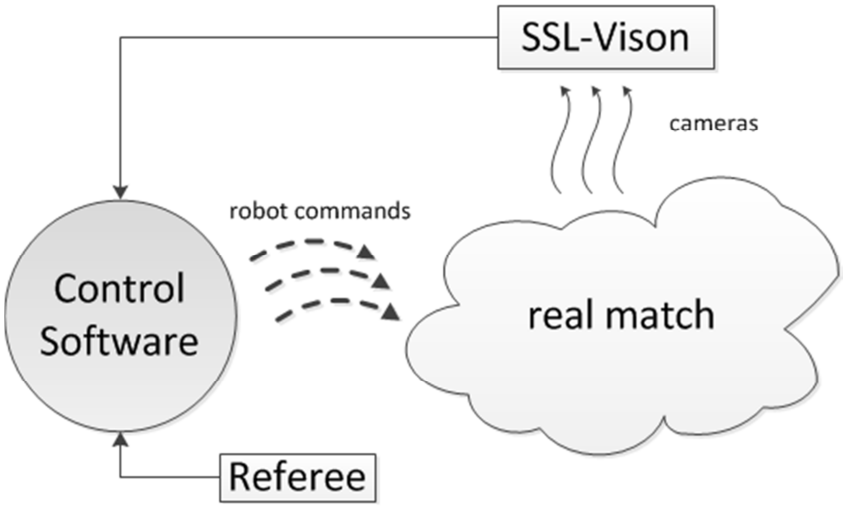

Figure 2: The general data flow of a SSL-match

For a single team the system design is pretty clear from the informatics point of view. Regarding the conditions stated above, the process can be simplified to the following three steps:

- Receive and interpret the data from “SSL-Vision” and the Referee-software
- Let the control unit decide which robot does what
- Send the resulting commands to the robots

On this level of design the system has lots of similarities to a simple control loop. For every point in time there is an “actual state”, represented by the robots on the field plus ball (game situation), the “target state”, which is the actual chosen tactic in the control unit, a “delta s”, by whom the actual state is manipulated by the control unit and a “E-state”, which stands for the disturbance in this system (through measurement errors, errors in the execution and the enemies).

2.2 The Basic Requirements

With the premises of the SSL-League and of the central control unit it is possible to derive basic requirements for the simulation. Besides these needs especially the CCU as test-object has some aspects that are worth implementing special testing possibilities.

First of all, the simulation has to behave absolutely transparent to the CCU. It has to provide the same interfaces as the real SSL environment does. Thus it has to provide the same information of the position of the objects (ball, robots) in the same format SSL-Vision uses. Furthermore, the connections to the robots have to use the same interface (see Figure 2).

Behind these interfaces the generated environment has to behave the same way the real hardware does. The ball, the field and the robots must be simulated physically at least to such an extent that it allows the application of the same schemes that are applied in the real match. To do so it has to take care of collisions between objects (e.g. ball-ground or ball-robot), the masses of the objects and the forces which are applied to them.

Besides the physical representation there is another important aspect of the simulation: the logical behavior. The simulated robots need to be augmented with

additional control logic which models mechanical and electronical procedures which cannot be simulated physically. Moreover, the behavior implemented on the robots microprocessors must be reproduced in the simulation.

To enable the same way of interaction with the simulation as with the real robots and field, the data generated in it should be visualized to the developer. This allows him to intuitively compare the “real” situation with the intended and to effectively develop his algorithms the same way as if he sat beside a real match. This visualization should also provide a possibility to interact with the objects in the simulation to move them around and reproduce certain situations.

2.3 Further Requirements for the Simulation Software

Besides this basic functionality there are a couple of other aspects that are desirable and very useful from the AI-developers point of view. As stated above (see 2.1) the control software not only consists of the AI but of a couple of modules covering various aspects of the general task.

One of these modules – the prediction module - filters the position data from the vision-system and validates it again. This is necessary because the vision-data is still very defective due to precision-errors. To be able to test this module, there has to be a possibility to modify the absolute positions from physical simulation with a similar error.

When finally the AI-part of the control software is developed other functionality is needed. To effectively test specific game situations, some kind of automated generation of this situation is very useful. This not only includes the positioning and movement of the opponent’s robots but also the generation of the referee-signals that have any impact on this situation.

3 Choice of Technology

After having defined the requirements for the simulation the next task is the identification of the proper technology to use to fulfill these needs. There are multiple robot simulation frameworks available and lots of other teams in the SSL-League engaged the same task. The following section introduces some of these frameworks and evaluates their applicability for the simulation of a SSL match.

3.1 Robot Simulation Frameworks

Robot simulation frameworks that are stable and have a large community of supporters would be a big advantage. These frameworks offer a lot of benefits like ready-to-use physical simulation of mobile robot platforms and sensors or other functionality.

Simbad 3D, for example, is a multi-robot simulator for educational purposes written in Java. It is based on Java3D[3] and the physical simulation and collision detection is relatively simple and limited to 2D. The simulation can be influenced by implementing different “robot controllers” which define the robots behavior. These robots may use the available sensors (range, contact, vision) to detect the environment around them.

Though Simbad 3D is generally useful in the given context, it has several major disadvantages. First, its physical simulation makes no demand on being physically correct. Second, the degree of freedom left to the developer is very restricted. The implementation of the robots internal logic and its influences on the environment (kicking, dribbling) would involve a lot of work-arounds or even reprogramming, for instance. Third, the base-technology Java3D is not the first choice for 3D visualization anymore as it is out of date and not under active development by now. **[Simbd]**

The following picture shows *Simbad 3D* in action.

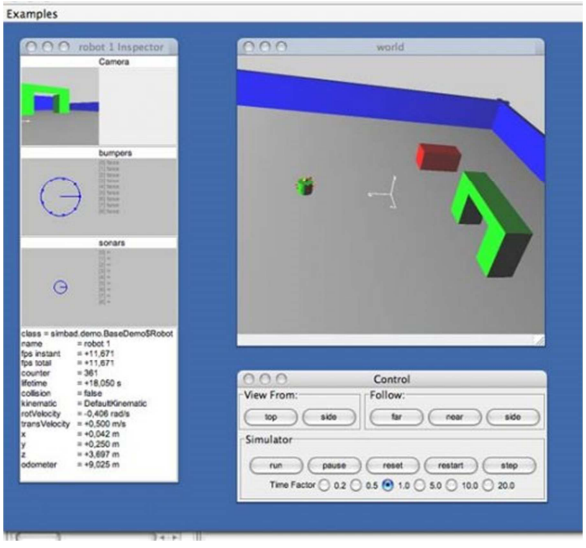

Figure 3: Simbad 3D displaying the simulated robot and generated output [Simbd]

[3] Java3D is a Java class library providing functionality to create and display 3D graphics. Its development was started in 1997 by Sun Microsystems.

Another program worth mentioning is *Gazebo*. It is part of the *Player*-project, which provides cross-platform robot device interfaces for the interaction with real robots via a TCP protocol. *Gazebo* has been developed to provide a simulation environment for *Player*-compatible robots. It provides the physically plausible 3D simulation of rigid bodies using the *Open Dynamics Engine* (ODE). Furthermore, lots of sensors can be used either directly in the robots control program or via the *Player*-interface. The following picture shows a robot with attached camera, whose input is visualized.

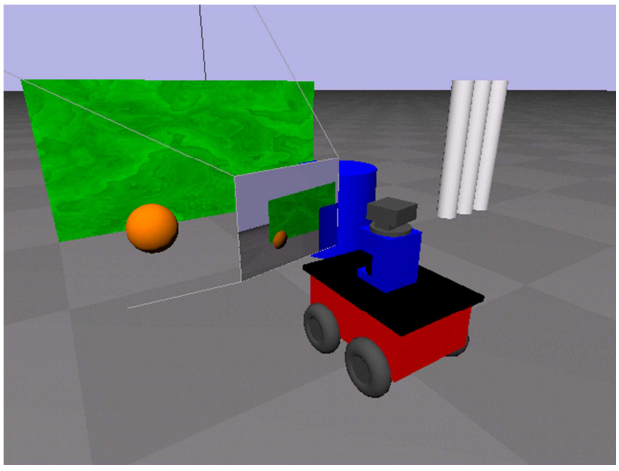

Figure 4: Gazebo simulating a robot with mounted camera [Gazeb]

Gazebo is a well-established framework with lots of features. But for the purpose of simulating a SSL-match it is not very suitable. Although it is useful for simulating not only one but a few robots it emphasizes the input of sensor data and its analyses and not the highly dynamic interaction between 10 robots. **[Gazeb]**

Besides these robot-simulation frameworks there are others especially designed to suit the needs of the RoboCup-tournament. *SimSpark* is one of them: It has originally been developed for the “3D Soccer Simulation League” of the RoboCup. Today it has grown to a very sophisticated, generic and highly configurable multi-agent simulation system (*Spark*) which can be used for a wide range of purposes. *SimSpark* provides a solid basis for 3D simulations of multiple autonomous robots with physically articulated bodies and has successfully used in the past RoboCup competitions. The following picture shows a running soccer simulation:

Figure 5: SimSpark simulating a game of soccer with 9 vs 9 Nao agents [SimSp]

Though *SimSpark* has many advantages it has *not* been chosen for the simulation of SSL-matches. This has several reasons. The first was the complexity, the drawback of abstraction in general: The effort to implement the features needed for the SSL-League is hard to estimate, especially for someone not involved in the project. Second, Spark, the simulation-engine itself, is very generic but provides absolutely no support for simulating robot soccer. This has been specifically implemented for the "3D Soccer Simulation League", but offers no direct advantage for any other soccer-simulation. And third, the framework did huge developments in the past, and has not been stable enough when first considered as base-technology. **[SimSp]**

Another simulation which has been developed in the context of the RoboCup is *SimRobot*. Is has been developed by the RoboCup-team *B-Human* in collaboration with the DFKI (Deutsches Forschungszentrum für Künstliche Intelligenz) since 1994 for various research-projects. It is able to simulate various models which are defined in a XML-languange. Its source-code is freely available as the team has decided to release it. **[BHuC10]** Though this would enable it to simulate a SSL-match, its focus is clearly set on the simulation of camera-images and sensor data for more autonomous leagues. **[BHu10]** But the problems with *SimRobot* are similar to the ones with *SimSpark*. It provides a lot of functionality but only a small piece of it can be directly applied to an SSL-simulation. Furthermore, the application had performance-issues with ten SSL-models when initially considered as alternative.

The last simulation framework presented here is the most promising one considering the ideas behind it: *Tewnta* – or "F-180"– is especially designed for the RoboCup

SSL-League. It is meant to build the basis for further work on a generic SSL-simulator to be used by the whole league. The following picture shows a running test-match.

Figure 6: Tewnta F-180 simulator displaying a test-match [Twent]

But the F-180 simulator has several major disadvantages. It does not provide any physical simulation of the objects, but uses some simple rules for the balls and robots movement. Furthermore the simulation happens only in the 2D plane, which is not enough as an important goal-chance in the SSL-league is to lob the ball over the enemy defense. **[Twent]**

The small selection of robot simulation frameworks presented above shows that there are lots of simulation tools each of them emphasizing a certain aspect of simulating mobile robots. Some of them are well established, stable and provide some of the needed functionality out-of-the-box. But many in the SSL-league teams – like *Skuba* **[Sku10]**, *ER-Force* **[ErF10]** or *CMDragons* **[CMD10]** - decided to write their own simulation software for a good reason: None of these frameworks provide special advantages in fulfilling the requirements for the SSL-league.

3.2 Simulation/Game-Engines

In contrast to ready-to-use robot frameworks game/simulation engines only provide the core functionality a simulation application would need, like visualization of primitives or collision detection and physical behavior. But in opposite to the former these offer a lot more freedom: They not just offer the possibility of configuration /modification – which every framework does in one way or another – but are designed to be used for whatever the developer feels like.

There are several well-established game development frameworks, many of them stable and with a great community. In the following there is a small extract of open-source frameworks, without the claim to be complete:

- OGRE (C++, APIs for: python, Java, C#)
- Irrlicht (C++)
- Havok (C++)

Each of them either has a physics-engine directly integrated or has good support for the integration of such an engine, which was one of the major criteria for the choice as base technology. Another criterion was the programming language: As Java is the most common language in our team - all our software is written in it - and to keep our pool of technology homogenous the simulation should also be developed with Java.

These criteria lead to the choice of the "*jMonkeyEngine*" (JME). The jMonkeyEngine-project was founded in 2003 by Mark Powell with the aim to develop a complete 3D engine in Java. From the beginning performant realtime-simulation and fast adaption for game development was emphasized. After a lot of positive feedback and the help of a fast-growing community currently version 3.0 is available, managed by a team of six core-developers. jME is released under the BSD license and is open-source. It provides a single, consistent API to the underlying frameworks either "Lightweight Java Game Library" (lwjgl) or "Java OpenGL" (jogl) for visualization and several other technologies for 3D-sound and physical simulation. **[JME]**

The jME version 2.1 chosen for the development for the SSL simulation is backed by a fully integrated Java-wrapper for the ODE. This physics engine has originally been developed by Russell Smith in 2000. It supports collision-detection for and the physical simulation of rigid bodies with friction. These features make it attractive for a wide variety of applications – educational and entertaining alike. **[ODE]**

4 Design of the Simulator

With the requirements and having a technology chosen the actual realization of the SSL simulation can be planned. This section describes how the different ideas and technologies should work together.

4.1 System

The basis forms the jME simulation engine. It provides the visualization and the simulation-loop with physics simulation. This engine has to be controlled by some instance which starts and stops the simulation. On startup, the models of the robots have to be generated, loaded and initialized. These robots have to maintain some kind of communication-endpoint, to be reachable for the central control unit. During the simulation the position data of robots and ball are converted to the SSL-Vision format and sent to a specific multicast-group on the network. The following figure gives an overview of the system:

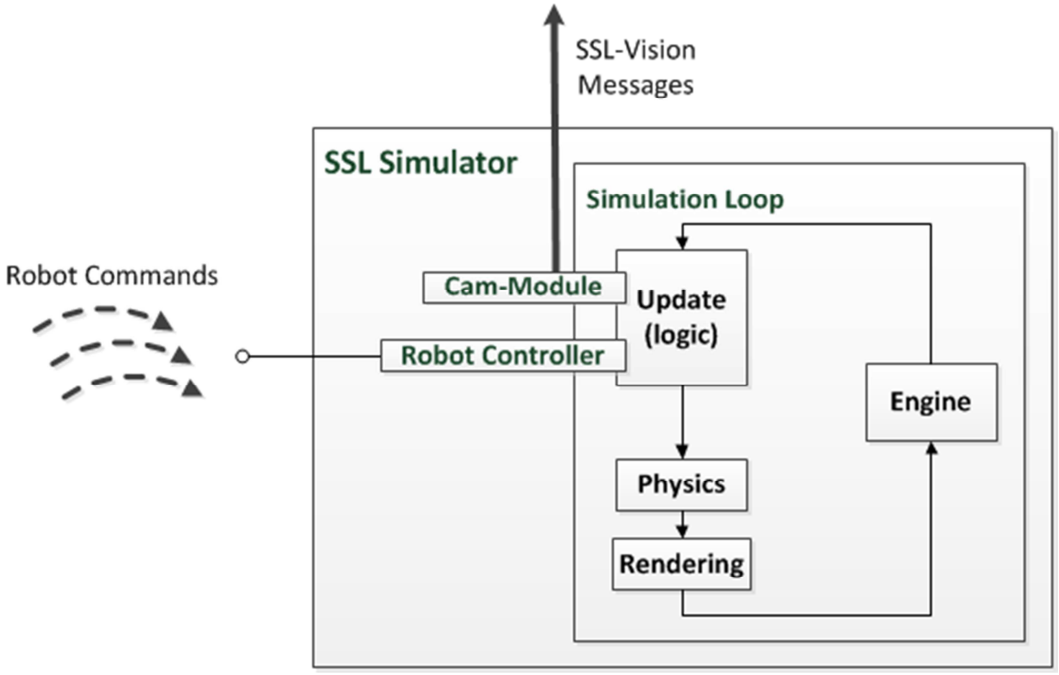

Figure 7: The general dataflow in the SSL simulator

On the left the incoming robot commands from the CCU are received and interpreted. During the update of the simulation loop these commands are applied to the robot models which change the state of the physical simulation. After the update the new physical state is calculated by ODE: collisions are detected, appropriate forces are applied and the resulting movements performed.

4.2 Visualization and Interaction

The main application consists of a GUI which allows the user to configure and control the simulation. After starting a SSL-match between two teams the field and all physical models are visualized to the developer. Moreover, these models are

moveable like in reality to test specific situations, thus the simulation provides the same functionality like the real hardware does. The resulting GUI is shown in this figure:

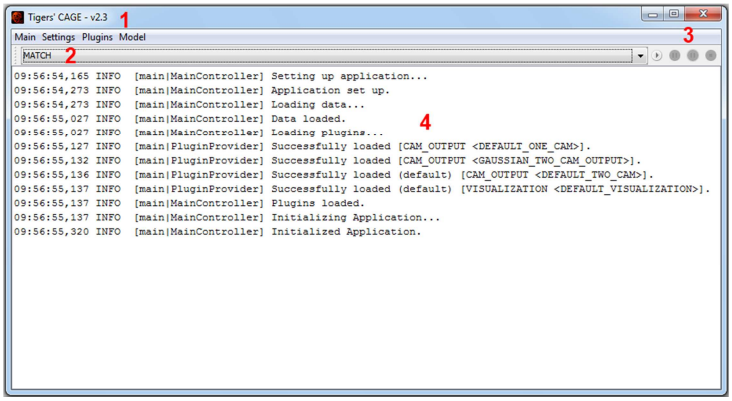

Figure 8: The simulators main GUI

It consists of three parts: The application menu, which lets the user configure and modify the simulation (1), the control-bar which defines the type of simulation and provides the functionality to start, pause and stop it, and a log-view (4).

The visualization happens in a separate (OpenGL) window:

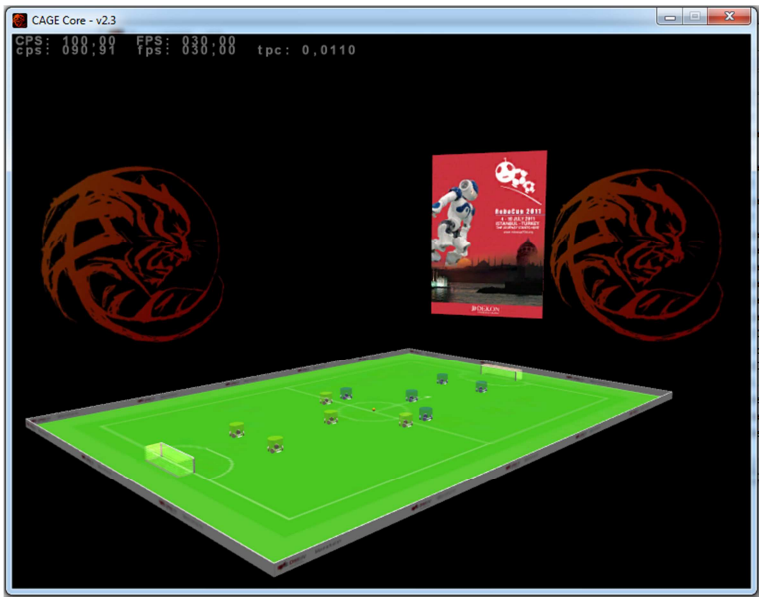

Figure 9: A running SSL simulation

Figure 9 shows a running SSL simulation. It contains 5 robots for each team, the ball and the field with the two goals. Through this window the physical models can be moved per drag-and-drop.

4.3 Interfaces and Connectivity

As demanded above (cf. 2.2) the central control unit of the TIGERS can transparently connect to the simulated robots. The communication happens via UDP and uses exactly the same protocol the real TIGER-robots understand. This protocol is parsed and interpreted by the simulator and applied to the logical and physical model (cf. 4).

The absolute state of the robots and the ball in the simulation is observed by a central instance. These data is converted to the SSL-Vision-format and sent to the same multicast-network SSL-Vision uses. Thus the generation of vision-data is transparent to the CCU, too.

5 Conclusion

In this seminar a concept for a simulation for the RoboCup SSL-League is developed. First, the SSL-environment is analyzed and the requirements for a possible simulation are gathered. Second, the way the central control unit of each team interacts with this environment is determined and further requirements are derived from this interaction. Third, available simulation technology is evaluated in respect of the collected requirements. Finally, a solution based on the chosen technology is described.

The resulting software allows the stable simulation of a RoboCup SSL-match. The jME-framework provides an integration of physical models with a 3D OpenGL-representation, which enables intuitive comprehension and interaction with the simulation. Seen from the CCUs point of view, this software is technically absolutely transparent compared to a real SSL-environment.

But there are still some requirements that are not satisfied, like an automation feature or the implementations of error-models for the vision-data (cf. 2.3). Moreover, the simulation still has performance problems when simulating a 5 vs. 5 match, and further features are always desirable. These issues should be addressed in further work.

Bibliography

[Twent] N.N.: *twenta - Robocup Small Size League (SSL) F180 Simulator*. URL: http://code.google.com/p/tewnta/ - last accessed: 15.05.2011.

[SSL11] RoboCup League: *Laws of the RoboCup Small Size League 2011*. URL: http://small-size.informatik.uni-bremen.de/_media/rules:ssl-rules-2011.pdf - last accessed: 11.06.2011.

[Sku10] Wasuntapichaikul, Piyamate et. al.: *Skuba 2010 Extended Team Description*. RoboCup Extended Team Description Paper (ETDP), online, URL: http://small-size.informatik.uni-bremen.de/tdp/etdp2011/SKUBA_ETDP_2011.pdf - last accessed: 11.06.2011.

[ErF10] Blank, Peter et .al.: *ER-Force: Team Description Paper for RoboCup 2010*. RoboCup Extended Team Description Paper (ETDP), online, URL: http://small-size.informatik.uni-bremen.de/tdp/etdp2011/ERFORCE_ETDP_2011.pdf - last accessed: 11.06.2011.

[CMD10] Zickler, Stefan et. al.: *CMDragons 2010 Team Description*. RoboCup Extended Team Description Paper (ETDP), online, URL: http://small-size.informatik.uni-bremen.de/tdp/etdp2010/cmdragons.pdf - last accessed: 11.06.2011.

[Simbd] N.N.: *Simbad 3d Robot Simulator*. URL: http://simbad.sourceforge.net/ - last accessed: 15.05.2011.

[Gazeb] N.N.: *Gazebo – 3D multiple robot simulator with dynamics*, URL: http://playerstage.sourceforge.net/index.php?src=gazebo - last accessed: 15.05.2011.

[SimSp] N.N.: *SimSpark*, URL: http://simspark.sourceforge.net/ - last accessed: 15.05.2011.

[BHu10] Röfer, T. et. al.: *B-Human Team Description for RoboCup 2010*. In J. Ruiz-del-Solar, A. Matsumoto, P. Plöger, E. Chown, L. Lee Yee

(Hrsg.), *RoboCup 2010: Robot Soccer World Cup XIII Preproceedings*. RoboCup Federation, 2010.

[BHuC10] Röfer, T. et. al.: *B-Human Team Report and Code Release 2010*. Online, URL: http://www.b-human.de /file_download/33/bhuman10 _coderelease.pdf - last accessed: 15.05.2011.

[JME] N.N.: *jMonkeyEngine*, URL: http://jmonkeyengine.com/ - last accessed: 16.05.2011.

[ODE] Smith, Russell: *Open Dynamics Engine (ODE)*, URL: http://www.ode.org/ - last accessed: 16.05.2011.