Rheinisch-Westfälische Technische Hochschule Aachen
Knowledge-Based Systems Group
Prof. G. Lakemeyer, Ph.D.

Simulation of the RoboCup Logistic League with Fawkes and Gazebo for Multi-Robot Coordination Evaluation

Bachelor-Thesis
by

Frederik Zwilling
Matrikelnummer: 304314
frederik.zwilling@rwth-aachen.de

November 30, 2013

First Supervisor:	Prof. Gerhard Lakemeyer, Ph.D.
Second Supervisor:	Prof. Dr. Matthias Jarke
Advisor:	Dipl. Inform. Tim Niemueller

Hiermit versichere ich, dass ich die Arbeit selbststandig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Place, Date

Signature (Frederik Zwilling)

Acknowledgements

Danke bla bitte blub.

Contents

List of Figures

Chapter 1

Introduction

Multi-robot systems are combinations of multiple robots that work together on specific tasks. Currently, they are mostly used in assembly lines, where many robot-arms simultaneously do repetitive tasks on the same objects. More autonomous examples of multi-robot systems can be found in the warehousing domain. Here, many robots bring demanded goods from the storage and store new ones [16]. The major advantages of multi-robot systems are high flexibility and the possibility to run different tasks in parallel. The amount of possible applications is large. Beside assembly and warehousing, they can also be used in rescue scenarios [21], soccer [20], planetary exploration [11], logistics and more. Developing a robotic system is challenging. Robots have to detect objects, localize themselves, reason about their surrounding and manipulate it. Developing a multi-robot system is even harder because the robots have to do all these tasks in coordination with other robots. Furthermore, the systems have to be robust against the failure of single robots and should be as efficient as possible. Because of these challenges, testing is an essential part of the development process. It is necessary to identify mistakes in the source code and to evaluate how the system performs in a complex environment. Often, tests reveal problems the developer did not expected. However, testing can be difficult and time consuming for several reasons. The robot and the environment have to be available and set up. A component to be tested can depend on other components that are still being developed. Some tests require cautious execution because the robot could harm itself or the environment and full system tests have to run a longer time. Testing a multi-robot system is even harder. On the one hand the testing effort scales with the number of robots. On the other hand the system is more complex and therefore requires more test runs for evaluation.
In this thesis, we tackle these problems by developing a multi-robot simulation environment. This environment has to be physically and visually realistic. The robot software runs like in the real world and gets simulated sensor data instead of real sensor data. When the robot software uses actuators, the actions are executed in the simulation. This brings many advantages and can speed up the testing process. With a simulation environment, the majority of tests needs neither available robots nor the environment, the setup can be automated and unfinished components, that other components depend on, can be

simulated as well. Furthermore, we want to evaluate the whole multi-robot system by measuring its performance in multiple runs. This enables us to easily compare different configurations and strategies in the simulation.
Such a simulation is useful in most robot developments. In this thesis we concentrate on the logistics domain and the mobile robot platform *Robotino*. We participate in the *Logistic League sponsored by Festo (LLSF)* with the *Carologistics* team. The Carologistics is a joint team consisting of the Knowledge-based Systems Group at RWTH Aachen University, the IMA/ZLW & IFU Institute Cluster at RWTH Aachen University and the Department for Electrical Engineering and Information Technology, Robotics Group at FH Aachen. LLSF is an industrially motivated competition within the RoboCup initiative. Three robots have to manage the material flow in a production area. The goal is to produce as many ordered products by feeding different machines in the production area with resources and intermediate products. To achieve a good performance, robust robot behavior and an efficient scheduling is necessary. Because of this, we have a special need for a multi-robot simulator, which can test the performance of single robots as well as the efficiency of the whole multi-robot system.
We use the *Fawkes* robot software framework to control the robots and we choose *Gazebo* as robot simulator. Important tasks of this thesis are the connection between Fawkes and Gazebo, the simulation of sensor data and the modeling of the LLSF environment and the actions of the robot in this environment. Because we want to use the multi-robot simulation to evaluate and improve our own LLSF system, we will also develop some concrete improvements and compare the performances of this system with different configurations. The majority of these improvements relate to the high level agent, which is responsible for the decisions of the robot and the coordination between the agents.
In chapter 2, we show the background of this thesis. This includes descriptions of the Robotino, the RoboCup and LLSF, Fawkes, Gazebo and other used tools. In chapter 3, we present related work about other simulations and agent strategies. Furthermore, we describe the current LLSF solution of the Carologistics team. The approach of this thesis is described in chapter 4. In chapter 5, we present the implementation. This includes a description of the developed simulation modules and the improvements on our LLSF system. The evaluation results of the changes of the agent and the simulation itself are shown in chapter 6. In chapter 7, we suggest future work and conclude.

Chapter 2

Background

This chapter presents the background of the thesis. On the one hand, we describe the RoboCup and LLSF in section 2.1 and the Robotino in section 2.2. On the other hand, we present the most important software for this thesis. In section 2.3 we present the two robot software frameworks ROS and Fawkes. Section 2.4 describes the robot simulator Gazebo, followed by additional software in section 2.5 which includes the data exchange format Protocol Buffers, the CLIPS Rules Engine and the document database MongoDB.

2.1 RoboCup

The *RoboCup*[1] is an international robotics competition founded in 1997 [22]. It provides standard problems as a platform for artificial intelligence and robotics research. Research teams from all over the world compete in different leagues to benchmark their robotic system. The RoboCup provides a research testbed, in which participating teams implement new approaches and make them robust against the challenges of the real world complexity. Furthermore, the competition leads to comparison and evaluation of different approaches. Every year, the competition takes place at the RoboCup world championship, which is hosted by changing nations. There are also offshoots, such as the RoboCup GermanOpen in Magdeburg.
Initially, RoboCup started as a robot soccer world cup. The goal "By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup." [38] shows how far the RoboCup is wants to push the development of artificial intelligence and robotics. Since then, RoboCup has grown and now features a variety of different leagues in different domains. In the following we give an overview of the RoboCup leagues in 2013:
Soccer Leagues: The majority of the RoboCup leagues are soccer leagues with different robot sizes and platforms. There is a *Standard Platform League (SPL)* where the teams have to use about 60 cm high humanoid NAO robots without any hardware modifications. This enables the teams to focus on vision, behavior and motion tasks. There is also a

[1] `http://www.robocup.org`

league for other humanoid robots. This league is divided into three sub-leagues with kid, teen and adult size robots. The *Middle Size League (MSL)* features up to five robots per team on a 18 m x 12 m field with each robot having a footprint smaller than 52cm x 52cm. In the *Small Size League (SSL)*, there are six robots, each with a diameter up to 18 cm, per team.

Soccer Simulation Leagues: There are also two soccer simulation leagues which focus on artificial intelligence and team strategy instead of developing robot hardware. There are a 2D simulation league and a 3D simulation league. In the 3D simulation league, teams consisting of eleven virtual NAO robots compete against each other in a realistic soccer environment. In chapter 3, we show both simulations in detail.

RoboCup@Home: The RoboCup@Home league is a competition about service robots. The robots have to assist humans in a domestic environment. There are a variety of tasks, such as following a human, serving drinks and handling emergency situations. Especially important in this league are the technical and open challenge. Here, the teams can present their own ideas and how the robot implements them.

Rescue League: The rescue league is a test-bed for urban search and rescue robots. The robots have to find victims in a setup course that simulates an urban area after an earthquake. The league has high demands on robot hardware and is divided into several fields, such as autonomous robots and remote-controlled robots.

Rescue Simulation Leagues: There are also two rescue simulation leagues called *Agent Competition* and *Virtual Robot Competition*. In the Agent Competition many heterogenous agents have to fight a large disaster in a city. The Virtual Robot Competition is about finding victims with a group of robots in a smaller area. We show both in chapter 3 in more detail.

RoboCup@Work: The RoboCup@Work league targets working tasks with the YouBot. The YouBot is a mobile robot equipped with a five degree of freedom arm. The goal is to perform work-related tasks, such as identifying and handling different objects, placing them in bins and transportation.

Logistic League: We describe this league in detail in the following subsection.

2.1.1 Logistic League sponsored by Festo

The Logistic League Sponsored by Festo (LLSF) is a competition within the RoboCup. LLSF aims to foster scientific work on autonomous solutions for logistics and to provide a test-bed for existing approaches [32]. The participants have to find new approaches and improve already existing ones to optimize material and information flow in logistics. LLSF takes place in a simplified production hall [35]. Figure 2.1 shows a part of the 5.6m x 5.6m hall with two Carologistics robots, three machines and some material-pucks. The main task of LLSF is to produce and deliver ordered products as efficiently as possible by feeding machines with resources and semi-finished products. The participants can use up

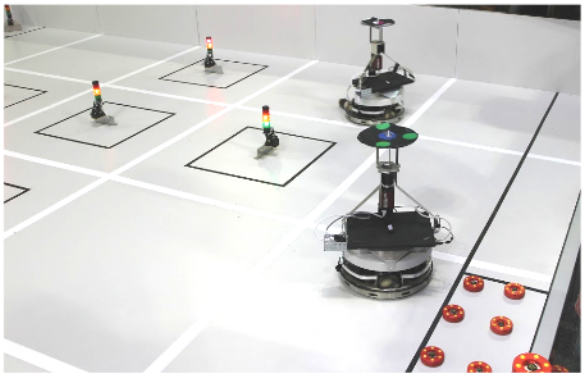


Figure 2.1: Part of the LLSF field

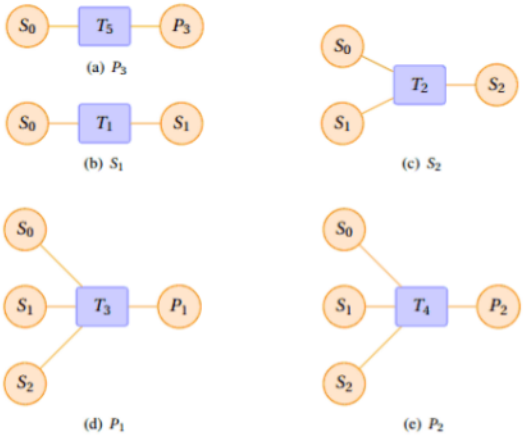


Figure 2.2: LLSF Production Chain [35]

to three Robotino robots by Festo [18]. We present the Robotino in the next section in detail. Orange pucks represent resources and products. They are equipped with an RFID-chip[2] which is needed to store the different product states of the pucks. The machines are represented by the RFID-readers with signal-lights on top. The signal-lights indicate the current status of a machine, such as "ready", "producing" and "out-of-order". The machines need resources and semi-finished products as input and turn the last input puck into a further precessed puck. All other input pucks are turned into consumed pucks which can be recycled to produce new resource pucks. There are different types of machines. The type defines the needed input and the produced output of a machine. Figure 2.2 shows the production chain. There are the raw-material pucks S_0, intermediate product pucks S_1 and S_2 and product pucks P_1, P_2 and P_3. The machine types are labeled with T_1 to T_5. Beside these regular machines, there are also recycling machines, which turn consumed pucks into raw-material pucks, and delivery machines which are used for delivering finished product pucks. Teams are awarded with points for delivering finished products, producing complex pucks and recycling. The game is divided into two phases. In the first phase, the *exploration phase*, the robots have three minutes to explore the production area to identify which machine is of which type. Teams also receive points for each correctly reported machine-type. The second phase is the actual *production phase* and lasts 15 minutes. The league also features an automated referee box (Refbox). It controls machines and communicates with participating robots during the game. The Refbox gives orders which products are to be produced, informs the robots about the game state and rewards points for achieved goals.

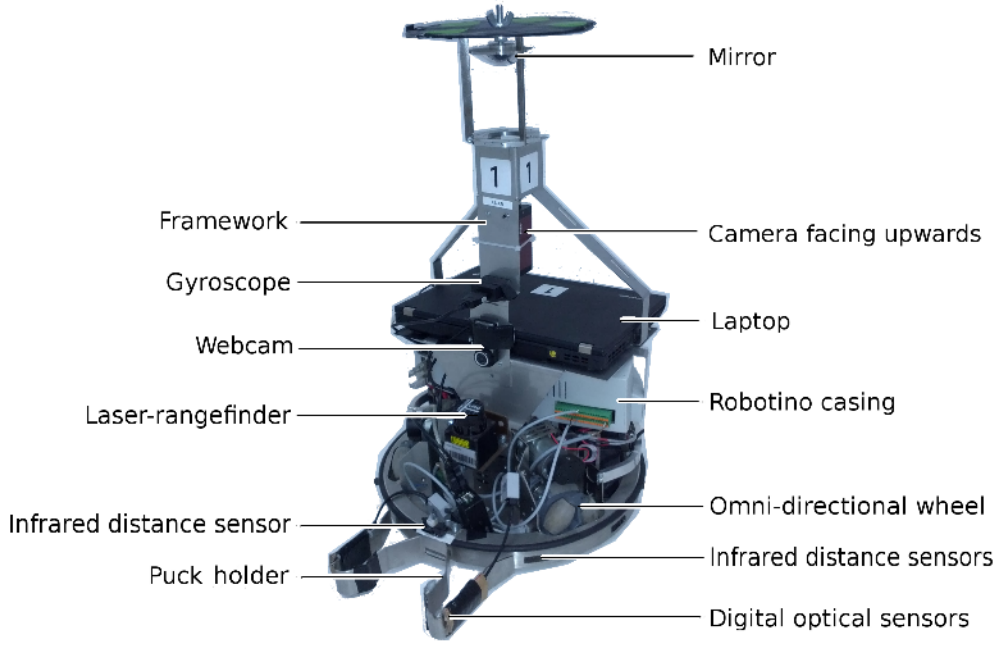


Figure 2.3: A Robotino extended by Carologistics

2.2 Robotino

The *Robotino* [18] is a mobile robot developed by Festo Didactic[3]. It is used for research and education. The shape of the Robotino is a cylinder with 37cm diameter and 21cm height. It has omni-directional wheels to be able to drive in any direction and turn at the same time. It is equipped with nine infrared distance measuring sensors ordered around the robot, a bumper to detect collisions and a webcam. A micro-controller and an embedded PC with 500 MHz and 256 MB RAM are used to control the robot [34]. The PC runs with a Linux Operating System. The Robotino is designed to be extendable to be used in different applications. Therefore additional sensors and actuators can be attached. At the front of the Robotino, there is a loading bay with some space for extensions. Figure 2.3 shows a Robotino extended by Carologistics. In addition to the default sensors shipped with the Robotino, we added a laser-rangefinder for localization and obstacle detection, a gyroscope and a framework which holds an omni-directional camera. The omni-directional camera consists of a camera facing upwards and a hyperbolic mirror above. The camera can see the whole area around the Robotino through the mirror. We adapted the omni-vision camera from former MSL-robots and now use it to detect LLSF pucks. The framework also holds a laptop connected to the Robotino, because we need more computational power than the Robotino provides for localization and path planning. We also attached a simple puck holder to the front. This puck holder is equipped with an infrared distance sensor to recognize if there is a puck in the holder and a digital optical sensor on each side of the puck holder. These are used to align the robot in front of LLSF

[2]Radio-frequency identification allows the wireless identification of objects with small chips.
[3]`http://www.festo-didactic.com`

machines. To observe the signal-lights on the machines, we use a webcam.

2.3 Robot Software Frameworks

2.3.1 Robot Operating System

The *Robot Operating System (ROS)*[4] is an Open Source framework designed to operate on many robot platforms and to provide a standardized integration framework [37]. It provides a collection of useful Open Source libraries and tools for the development process of robot-software. Currently, it is widely used and has a large community. Each component integrated in ROS runs an own process and is called *node*. ROS features peer-to-peer communication between nodes. Nodes can register a publisher or subscriber to a specified *topic*. If a publisher of a topic sends a message, every subscriber of same topic receives the message. We will describe the message protocol Google Protobuf used by ROS later. The communication is managed by a process called *Roscore*. Roscore acts as a server. All nodes have to connect to the server in order to operate.
We use ROS in LLSF for two reasons. First, we use ROS for motion planning. The ROS package *Movebase* provides a global path planning and local motion planning with collision avoidance. We provide a global navigation graph for the LLSF field, the position and orientation of the robot, laser data for obstacle detection and motion goals by publishing messages to specified topics. With this data, Movebase computes a motor command. We subscribe to the motor command topic and execute received commands. Second, we use the *Rviz* package for visualization. On the one hand we visualize incoming laser data and the localization of the robot in the LLSF field with all position hypotheses of our Adaptive Monte Carlo Localization (amcl). explain?. On the other hand we visualize the global and local motion plan. Moreover, we use Rviz to send localization hints to amcl.

2.3.2 Fawkes

Fawkes[5] is an Open Source robot software framework developed primarily at the Knowledge-based Systems Group[6] (KBSG) at RWTH Aachen University [33]. It is designed to run on multiple platforms and domains. It is written for Unix systems and follows a component-based software design [10]. It provides an infrastructure to load and unload binary components (implemented as *plugins*) at run-time. Fawkes features a *blackboard* as communication structure between plugins. The blackboard lists structured entries called *interfaces*. Plugins can read and write interfaces with a specified id. There can be many readers but only one writer for an interface. Furthermore, Fawkes organizes plugin activity in threads to make use of multi-core architectures. Because of this design, Fawkes is flexible and

[4] `http://www.ros.org`
[5] `http://www.fawkesrobotics.org`
[6] `http://www.kbsg.rwth-aachen.de/`

dynamic. The interchangeability of the plugins, which is caused by well defined interfaces, makes hardware abstraction and reuse easy. This is also an important advantage for the development of the simulation because the simulation can easily exchange lower-level robot control and sense plugins. The robot control plugins on higher levels can operate as usual because the interface is the same. The features of Fawkes are provided as *aspects*. These aspects are based on aspect-oriented programming [19] and give access to a particular feature. Threads that want to use a feature can inherit from the corresponding aspect. For example there are aspects for accessing the blackboard, logging and timing [31]. In this thesis, we provide communication with the simulation by using such an aspect. Two other important features for this thesis are the centralized clock and the configuration. The centralized clock provides the time for all plugins. Especially important for this thesis is the possibility to give Fawkes an alternative time-source. So, it is easy to exchange the system time with the simulation time for all plugins without having to modify them. Also important for this thesis is the way Fawkes provides configurations. The configuration-files for all plugins are described in the same format. A super-ordinate configuration file chooses which configuration files have to be loaded. There is the possibility to start Fawkes with a specified configuration by setting a super-ordinate configuration file. Furthermore, there is the possibility to override configuration values.
In comparison to ROS, the advantage of Fawkes is the tighter integration of the used components. The main loop of Fawkes controls the execution order of all component threads. This makes it possible to match certain time criteria for the components and to provide a *sense-think-act* cycle.
We use Fawkes as our main robot software framework. Nearly all software components for our robot are integrated as plugins into Fawkes. Therefore, in this thesis, we have to provide the sensor data from the simulation and get commands for the actuators in Fawkes. bbsync? behavior engine?

2.4 Gazebo

Gazebo[7] is an Open Source robot simulator [25]. It provides a platform to simulate all kinds of robots with sensors and actuators and their environment in a three dimensional world. It runs on UNIX and is written in C++. Originally, Gazebo was designed for large outdoor environments, but it is well suited for indoor environments, too. The development was started as part of the Player/Stage project [14] and was an alternative to the two dimensional simulator stage. Gazebo became independent later. Now, it has s strong relation to ROS and is even available as a package in ROS. Gazebo is an frequently used simulator. It is, for example, the simulator chosen for the recently held Virtual Robotics Challenge founded by DARPA. We present the use of Gazebo in this challenge in the related work.

[7] `http://gazebosim.org`

Gazebo uses proven Open Source engines for graphical presentation and physics. *Ogre*[8] is the graphic engine used in Gazebo. It allows rendering graphically realistic environments with reflections, shadows and detailed textures. This is important for the simulation of camera sensors because often reflections of light sources and inconstant lighting can cause problems. Gazebo uses the *Open Dynamics Engine (ODE)* [9] and *Bullet* [10] to simulate physical interactions between objects. It features an abstraction layer for physics and can therefore use both physic engines interchangeably. Bullet features lower computation cost and more complex shapes than ODE, whereas ODE has a larger documentation and background in robot simulators. Furthermore, the abstraction layer for the physics engine allows the access to detailed parameters of the engines.
A simulation in Gazebo consists of different elements: *Models* describe physical objects in the simulation, such as a table or a robot. The model is built out of *joints* and *links* which consist of a number of geometries. These geometries can either be visible objects or collision objects used for physics. Joints connect two links and define the possible motion of the links in relation to each other. For example, joints are used to define axes of wheels. Furthermore, a model can contain sensors which are represented by a set of properties to define the sensor. A *world* describes the whole simulation environment in Gazebo. It consists of models and light sources. It also has a range of properties for the physics simulation, rendering and the general simulation. Models and Worlds can be defined in an XML-like format called *Simulation Description Format (SDF)* which is based on the *Unified Robot Description Format (URDF)* of ROS but is optimized for the use in a simulation. Beside the physical description of objects in the simulation there are also *plugins* in Gazebo to modify the simulation at run-time. These plugins belong to a model, a sensor or a world. They inherit from the Gazebo API and use it to model the behavior of different objects. For example, there are plugins for robot models, which apply motion to the corresponding model if the robot receives a movement order, and plugins for the world, which can spawn new models.
Out of the box, Gazebo includes a variety of generalized sensors and models of common robots and other objects such as tables, cups and even buildings. For example, there are robot-models for the PR2 by Willow Garage, Atlas by Boston Dynamics and the Kuka YouBot. In the Gazebo framework, there is basic support for a set of general sensors. This includes cameras, contact-sensors, GPS, inertia measurement units, laser sensors and sonar sensors. The support of all these common sensors and models allows easy development of a specific simulation environment. Not included models, sensors and plugins can be added and fit well in the existing context.
The reason why Gazebo was chosen as the most suitable simulator for this thesis is shown in the related work.

[8] `http://www.ogre3d.org`
[9] `http://www.ode.org`
[10] `http://bulletphysics.org`

2.5 Additional Software

2.5.1 Protocol Buffers

Protocol Buffers (Protobuf)[11] is a data interchange format developed by Google. Protobuf serializes data in a pre-defined structure for efficient data transportation and storage. It is flexible and language-independent. The user can define a data structure. Protobuf then uses this data structure to generate code in a specified programming language. The code is able to efficiently serialize and deserialize data. The generation of code in the different languages makes it easy to include Protobuf in an implementation. The data structures are kept simple and can be built hierarchically. In the following, we use the term type to refer to the data structure behind a Protobuf message. Because of its efficiency and simplicity, Protobuf is widely used. Especially ROS, Gazebo, the LLSF Refbox and our agent use Protobuf to transport messages. However, Protobuf also has the downside that a received sequence can only be deserialized if the type of the message is known. Because Protobuf has the additional feature that the data structures are down-compatible against extension, this downside can be compensated by manually adding the message type as a field to the message. In this thesis, we use Protobuf for the communication between Fawkes and Gazebo.

2.5.2 CLIPS Rules Engine

The *CLIPS* rules engine[12] is an expert system developed by NASA[13] with rule-based, object-oriented and procedural programming paradigms [12, 44]. It basically consists of a *fact base*, a *knowledge base* and an *inference engine*. The fact base acts as a kind of working memory [34]. *Facts* in the fact base represent pieces of information. The values of a fact can be in an array or structured defined before. The knowledge base consists of *rules* and *functions*. Rules represent heuristic knowledge and are the basic part of the expert system. They are composed of an *antecedent* and a *consequent*. The antecedent contains an ordered set of conditions. If all conditions of the antecedent can be matched by facts in the fact base, the rule is applicable. The consequent contains a set of actions which are executed if a rule is applied. For example, these actions can assert new facts to the fact base or delete existing ones. Functions represent procedural knowledge. The inference engine realizes a forward-chaining mechanism. It matches the fact base to the antecedent of the rules to determine which rules are applicable. Then, it decides which applicable rule to execute by a priority defined in the rules. After executing the rule and probable changes to the fact base, the whole method is repeated until no more rules are applicable.
The special feature of CLIPS is its integration into programming languages, such as C and Java. On the one hand, it is possible to run CLIPS and to dynamically assert facts

[11] `https://developers.google.com/protocol-buffers/`
[12] `http://clipsrules.sourceforge.net`
[13] National Aeronautics and Space Administration (NASA) of the USA

from a program. On the other hand, functions in CLIPS can call functions of the language CLIPS is embedded in. This integration makes CLIPS a good choice for the control of an embedded system. We use CLIPS for our high level control agent. Here, we encode the world belief and which actions to take in which situations.

2.5.3 MongoDB

MongoDB[14] is an Open Source, document-oriented database [9, 24]. The basic elements in MongoDB are documents. Documents are grouped in so called collections which are contained in a database. Documents contain a set of key-value pairs, can be nested and are schema-free. This means that there are no constraints to the keys in a document. This provides a flexible and simple use. MongoDB uses indexing to increase querying speed. cut off MongoDB section or better description?
In this thesis, we use MongoDB to store statistics about simulated LLSF games. MongoDB also gives us the possibility to log sensor data and the belief and decisions of the agents [24].

[14] `http://www.mongodb.org/`

Chapter 3

Related Work

In this chapter, we present related work of this thesis. In section 3.1, we describe alternatives to Gazebo and why we choose Gazebo. In section 3.2, we give an overview of other popular simulations, especially a simulation that also uses Gazebo and other multi-robot simulations. Section 3.3 describes our current multi-robot strategy as well as other attractive strategies our simulation should be able to evaluate. In the following, we will use the terms *multi-robot* and *multi-agent* interchangeably because the common term in literature is multi-agent and we want to use multi-robot. A multi-agent system can consist of fully abstract agents, but we are dealing with multi-robot system and therefore robots which interact physically with their environment.

3.1 Other Simulators

3.1.1 Stage

Stage[1] is an Open Source multi-robot simulator for a two dimensional environment. It is a part of the *Player/Stage* project [14]. *Player* is a robot control server, which provides a simple connection between a robot control program, sensors and actuators of a robot. Stage uses this connection and replaces the sensors and actuators by simulated ones. Stage was a popular simulator in the last years and is able to simulate a large amount of robots. For the simulation, Stage uses simple and computationally cheap models, that provide still enough fidelity for most applications. Furthermore, the computational effort is linear in the number of robots. Because of that, Stage can simulate a large amount of robots [41]. This makes Stage attractive for the simulation of swarms and multi-robot systems with many robots.

[1] `http://playerstage.sourceforge.net/index.php?src=stage`

3.1.2 Webots

Webots[2] is a commercial 3D simulator [29]. It is platform independent and features a whole development environment for robot software including an editor and an API explain term? for inter-robot communication. Webots supports many programming languages and several standard robots and sensors out of the box. It is also able to simulate multiple robots and provides a ROS and Matlab introduce Matlab? interface. Webots was used in a RoboCup Soccer simulation league [30].

3.1.3 USARSim

USARSim[3] is a 3D simulator developed for urban search and rescue scenarios [7]. It was evolved to be able to simulate other domains too. Therefore, the abbreviation USARSim now stands for "Unified System for Automation and Robotics Simulation" instead of the previous "Urban Search and Rescue Simulation" [6]. It is platform independent and free of charge for research and education. The advantages of USARSim are the graphical realism of the Unreal engine[4] and the physical realism by the PhysX engine[5]. USARSim also provides an interface to ROS [26] and is able to simulate multiple robots.
USARSim is used as a basis for the RoboCup Rescue Simulation League we also present in section 3.2.2.

3.1.4 SimSpark

SimSpark[6] is an Open Source 3D simulator [5]. It was developed by the RoboCup community for the RoboCup 3D Soccer Simulation and has been used there since 2004. Therefore SimSpark is able to simulate multiple robots and specialized on the NAO as soccer robot. There are several improvements and new features of SimSpark [40,46], such as visualization of agent intentions, realistic servo motors for the NAO and better support of heterogeneous robot teams.

3.1.5 Robotino Sim Professional

Robotino Sim Professional[7] is a 3D simulator for the Robotino developed by its manufacturer Festo. It is a commercial software and only usable on Microsoft Windows. Furthermore, it is limited to the Robotino and the default sensors the Robotino is shipped with.

---

[2] http://www.cyberbotics.com
[3] http://sourceforge.net/projects/usarsim
[4] http://www.unrealengine.com/udk
[5] https://developer.nvidia.com/physx
[6] http://simspark.sourceforge.net
[7] http://www.festo-didactic.com/int-en/learning-systems/software-e-learning/robotino-sim-view/robotino-sim-professional.htm

3.1.6 Choice for Gazebo

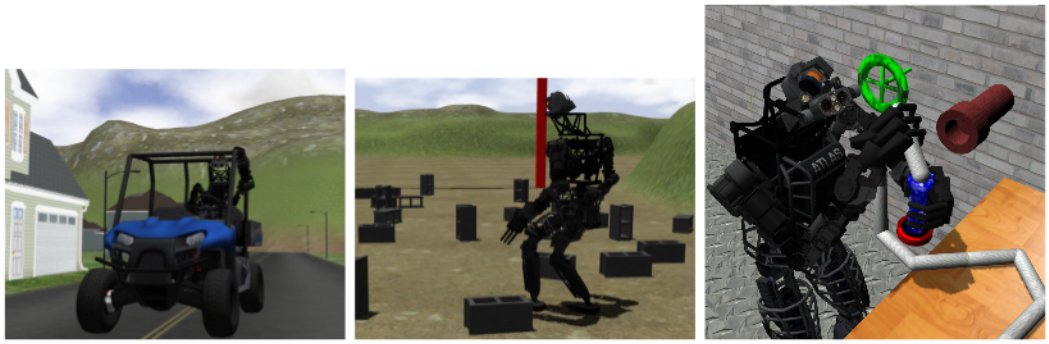
We already presented Gazebo and some of its advantages in section 2.4. Here, we explain why we have chosen Gazebo instead of an alternative simulator we described above.
The stage simulator is well suited for multi-robot systems and comparatively fast and easy to use. However, the restriction to two dimensions is a problem. We want to simulate vision components to be able to test our system as a whole and under more realistic conditions. We also want to be extendable for future changes of LLSF or the use of the simulator in an other domain, such as the RoboCup@Home league.
Webots is a proven simulator with many features, but has the same problems as Robotino Sim Professional. On the one hand, both simulators are commercial and on the other hand, both are no Open Source software. Therefore, it can be difficult to expand the simulation how we need it.
SimSpark is not so well suited as Gazebo because it and the community behind are mostly specialized on the RoboCup Soccer Simulation. Gazebo is used in large variety of domains and has a bigger community. Gazebo is well founded by the Open Source Robotics Foundation[8] and Willow Garage[9] and there are also many plans to develop Gazebo further[10]. Therefore, Gazebo is likely to become even more important in the future.
An other important argument for Gazebo is that it was used at KBSG before. It was used as a simulator for MSL [4] and for scene reconstruction for fault analysis [24]. We will present both applications later in this chapter.
There are only few disadvantages of Gazebo. A disadvantage is that Gazebo can not handle a large amount of robots. The complexity of the simulations limits the simulation speed when using multiple robots. The number of robots to simulate at a reasonable speed is in the order of ten [25]. This can become a problem if the simulation runs on a slow computer or there are more robots to simulate at the same time. However, these disadvantages have a small impact on this thesis because we use only three robots in LLSF.

3.2 Simulations

3.2.1 Virtual Robotics Challenge

The *Virtual Robotics Challenge (VRC)* is a competition by DARPA[11]. The goal of the competition is to solve challenging tasks with a humanoid robot in a simulation. VRC is the first part of the *DARPA Robotics Challenge (DRC)*[12]. DRC aims to spur the development of robots that can operate in a disaster scenario if the situation is too dangerous for humans. An example for such a scenario is the disaster in the Fukushima nuclear power plant after the tsunami in March 2011 [42]. The developed robots should be able to operate

[8] `http://osrfoundation.org`
[9] `http://www.willowgarage.com`
[10] `http://gazebosim.org/wiki/Roadmap`
[11] DARPA is the Defense Advanced Research Projects Agency of the USA.
[12] `http://www.theroboticschallenge.org`



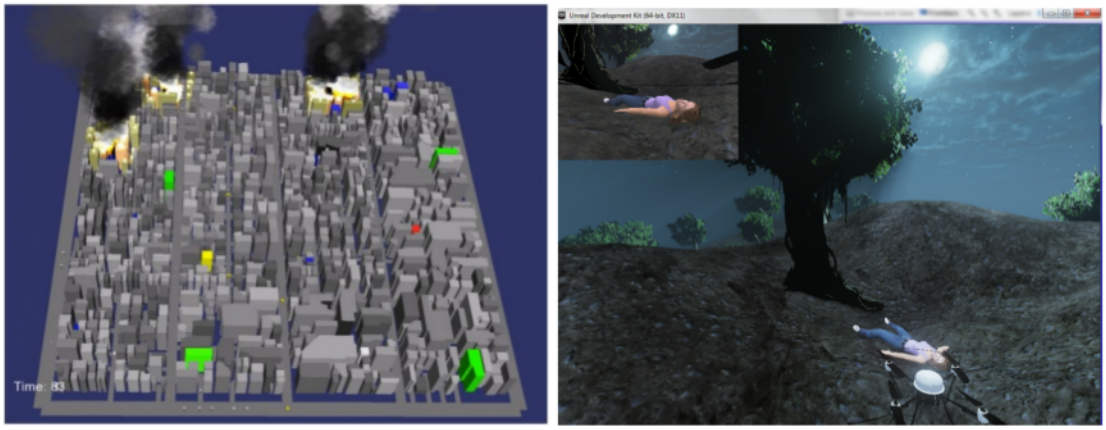
(a) Driving a car (b) Crossing rough terrain (c) Connecting a hose

Figure 3.1: Tasks of the Virtual Robotics Challenge [2]

in environments made for humans, even if the environment is damaged, and to use tools made for humans, such as screwdrivers and cars. During the competition, the robots act partly autonomously and are supervised by human instructors. For the challenge, DARPA provides six humanoid robots. The Robots are called *ATLAS* and are developed by Boston Dynamics. The six best teams of VRC receive a ATLAS robot for free. Therefore, ATLAS is the only robot simulated in VRC. However, the participating teams can also build their own robots. VRC consists of three different tasks [13] showed in Figure 3.1. In the first task, the robot has to walk to a car, enter it and drive along a street. The chassis of the car consists of a framework to simplify the entry. On the road, there are obstacles the robot has to avoid. The second tasks tests the walking capabilities of the robot. The robot has to cross slippery and irregular terrain to test balancing and a terrain with obstacles to test perception and footstep planning. The last task is about manipulation. The robot has to connect a hose to a pipe by plugin it in and screwing it down. Afterwards, it has to open a valve. All tasks have randomized parameters, such as the position of obstacles and goals.
VRC is related to this thesis because it also uses the Gazebo simulator [1]. Therefore, it shows what Gazebo is capable of and how important a simulation is for the development and research of cutting-edge technology. Although the technology fostered by DRC is important and useful without doubt, it seems questionable what DARPA as a military agency will use this technology for.

3.2.2 RoboCup Simulation Leagues

In the RoboCup competition, there are different simulation leagues. On the one hand, there are the 2D and 3D soccer simulation leagues and on the other hand there are two rescue simulation leagues. In all of those leagues, multi-robot coordination is an important field. This is not surprising because it is much simpler to benchmark multi-robot strategies in a simulation. There is no need to operate real hardware and difficult low level control and sense problems, that are not in the focus of the competition, can be simplified or left out.



(a) Agent Competition [23] (b) Virtual Robot Competition [3]

Figure 3.2: Rescue Simulation Leagues

The *Rescue Simulation League (RSL)* aims to benchmark intelligent software agents and robots in a disaster scenario [3]. RSL is separated into two leagues with focus on different scales. Figure 3.2 shows snapshots of both simulation. The *Agent Competition* is about a large scale disaster in a city with multiple robot teams. The *Virtual Robot Competition* is about finding victims in a burning house or limited outdoor area with a team of eight robots and one human operator. The operator can give high level commands, such as the area where to look for victims, and is needed to verify the observations of the robots. So, the robots should find and approach victims autonomously and the operator has to confirm the find. The task of the agents include victim-detection, autonomous generation of maps, navigation and multi-agent coordination. In the Agent Competition, there is no realistic robot control and perception necessary. The focus is on the coordination of many heterogeneous agents and taking high level decisions. There are teams of fire brigade, police and ambulance, each with up to 30 simulated agents. Each agent acts autonomously and can communicate with other agents. The tasks of the agents include exploration, scheduling and planning in cooperation with other agents and building of agent teams (e.g. to fight fires more effectively).

The soccer simulations of the RoboCup are similarly split in two leagues. In the *2D Soccer Simulation League*, 22 agents in two teams play soccer in a two dimensional environment. A single server called Soccer Server simulates and controls the game with its physics, the actions of the agents and the perception of the agents. Each agent is an autonomous program and communicates with the soccer server to receive noisy sensor data and send action commands to influence the simulation [39]. The main challenges of the 2D Soccer Simulation League are planning the actions of each agent when playing in offense and reacting on the opponent and predicting his movement when playing in defense. The *3D Soccer Simulation League* features a more realistic simulation. This league fully simulates nine NAO robots per team. Therefore, the low level control and perception are an important part of the challenge [27]. This is an important opportunity for the teams

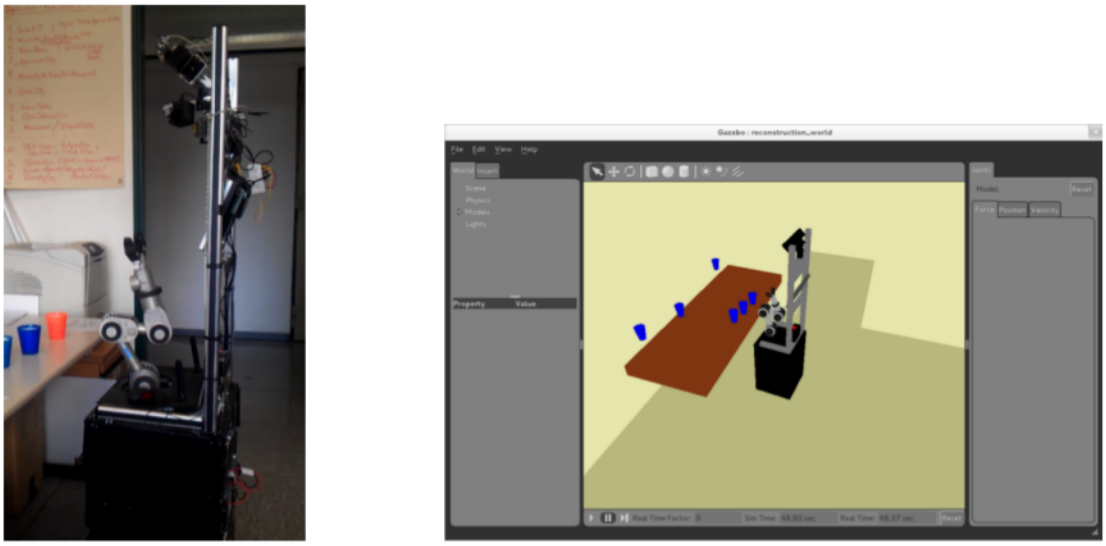
(a) 2D Soccer Simulation League [43] (b) 3D Soccer Simulation League [27]

Figure 3.3: RoboCup Soccer Simulation Leagues

participating in SSL to test their approaches first in the simulation because SSL also uses the NAO as robot platform. The communication between agents and the Soccer Server in this league is similar to the 2D league. The rules of the league are inspired by the FIFA football rules but have some reasonable extensions [36]. For example a goal directly form the kick off is not allowed and if too many robots are in a small area around the ball, the position of some robots is reset.
All simulations we have looked at in this section are similar to the simulation we developed for LLSF. The simulations are important for benchmarking multi-agent systems because the system can be tested with low effort in comparison with real multi-robot systems and some details, such as low level control and perception, can be left out. Furthermore, the simulations have in common that the evaluation of the whole system can be done by looking at the results and progress of the simulated games. Especially by 2D Soccer Simulation teams, this is used by to do reinforcement learning [17, 28]. Similar as it is possible to improve a real multi-robot system in SPL by improving the performance of the multi-robot system in the 3D Soccer Simulation League [45], we also want to improve our real system by using a simulation.

3.2.3 Scene Reconstruction for Fault Analysis

Some work was already done with Gazebo and Fawkes. Bastian Klingen developed a scene reconstruction for fault analysis in his diploma thesis [24]. The scene reconstruction was primarily made for a mobile robot with a Microsoft? Kinect and a laser range sensors for perception and a gripper arm for manipulation. The task of the robot was to grep a colored cup on a table. The scenario and the reconstruction are shown in Figure 3.4. Because this gripping task involves many components and often fails if a single component fails, a tool was needed to analyze unsuccessful attempts afterwards. This tool reconstructs the scene with the position of the robot, its sensor data and world belief. Therefore the movements, sensor data and belief of the robot have to be logged while performing a grep. The logging was also done with MongoDB because of the needed speed. Gazebo was



(a) Grasping a cup in the real scene (b) Reconstruction in the Gazebo simulator

Figure 3.4: Scene Reconstruction for Fault Analysis [24]

used to visualize the perception and belief of the robot to reconstruct the scene. Fawkes was used to control the robot, to log the needed data and to control Gazebo and read the log while reconstructing the scene. Together with the scene reconstruction, Klingen provided a tool to analyze faults to find components that causes the faults. Therefore, he constructed a dependency tree of the components and ordered it in a data-information-knowledge hierarchy.

To establish the communication between Fawkes and Gazebo, Klingen developed a Fawkes plugin that provides the communication objects of the Gazebo API as an aspect. In this thesis, we use this plugin and expend it according to our needs. Similar but not so sophisticated as in the thesis by Klingen, we want to provide a possibility to reconstruct faults to find the causes. We do not record the perception of the robots in the simulation, but we record the movement of all simulated objects and the high level decisions of the agents which can be used to derive the belief of the robot. The motion of the objects in the simulation can be recorded by Gazebo and the agent decisions by logging the results of our rule-based production system CLIPS.

3.2.4 Simulation Environment for the Middle Size League

An other work with Gazebo and Fawkes developed a simulation environment for MSL [4]. This simulation environment became necessary because the field size and the amount of robots per team was increased in MSL. Therefore, testing became more difficult. The work used an early version of Gazebo and Player as interface between Fawkes and Gazebo. On the one hand, it showed that Gazebo is capable of simulating complex robot components that cause problems in other simulators. This was shown for omni-directional wheels and an omni-directional camera. Both devices are non-trivial to simulate and often used

in MSL. On the other hand, the work proposed the idea of *multi-level abstraction*. If a robot software contains separate components for steps that are based on each other, it is useful to simulate not only the hardware level, the lowest abstraction level, but also higher abstraction levels. For example if the robot software includes a component that reads an image from a webcam and a component that computes the state of a traffic light from this image, the simulation can provide the simulated image as well as the information about the traffic light. A simulation with multi-level abstraction is able to switch between different abstraction levels. This can be useful to test components on a higher level independent of low level components which can introduce an error or are not finished. Multi-level abstraction has the additional advantage that the components of a higher abstraction level often can be simulated with less computational effort. This makes it easier to run the simulation on a slow computer. However it is not always reasonable to introduce multiple abstraction layers, because some components, such as a collision avoidance, can not be simulated easily. In this thesis, we also use multi-layer abstraction to be able to test high level components more individually.

3.3 Multi-Agent Strategies

In this section, we present our current high level agent approach, which we want to evaluate and improve during the thesis. We also present some advanced and attractive multi-agent strategies. On the one hand, our simulation should be able to simulate and evaluate these approaches as well and we might want to use one of these approaches in the future because our current one has some limitations.

3.3.1 Incremental Task-level Reasoning

Our current implementation of the high-level agent uses the CLIPS rules engine. Our implementation is based on the idea of incremental task-level reasoning [34]. This means that the agent takes the next best possible action whenever the robot is idle. Simplified, every action is represented by a CLIPS rule. The action is possible if the antecedent of the rule can be matched by the fact base. The consequent of the rule includes the execution of the action itself and changes to the fact base to remember the current situation. If new perception data is available for the agent or the execution of an action finished or failed, this is asserted as a fact and can therefore cause in the application of some rules to update the belief of the agent in the fact base or to execute the next best action. If more than one action is possible, the action with the highest priority is executed. simplify sentence? The incremental reasoning approach with CLIPS has several advantages. Actions and handling perception can easily be represented by rules which work with the current belief of the agent. The approach is computationally inexpensive, because no costly planning is needed, and it is easy to cope with incomplete knowledge because the agent just takes the next best action and updates the belief if new information is available. 2012 necessary? In

the LLSF competition of 2012, incomplete knowledge was a part of the problem because the robots had to find out the types of the machines by trying how they react on different input pucks. The refbox, which was introduced in 2013, now provides the information about the machine types. However, dealing with incomplete knowledge is still important, especially if the agent recognizes that it made a mistake and the world belief is partially wrong.

Until the RoboCup 2013 we have been using only two robots. For coordination of these two robots, we use a role based approach with additional resource locking. Every agent has a specific role, which determines for which tasks the agent is responsible for and which machines it may use. We have one role for an agent which only produces P_3 pucks and delivers them. This agent continuously provides easy points because P_3 is the simplest product, which needs only one resource puck (see Figure 2.2). In the following, we call this role the P_3-*role*. A second role is responsible for producing more complex products, the P_1 and P_2 pucks. This role can provide a large amount of points, but is more risky because of the higher amount of intermediate steps. We call this role the P_1P_2-*role*. Whether the P_1P_2-role produces P_1 or P_2 pucks is determined by the distance between the needed machines to the machine for P_3 products. This spacing is necessary because the different robots drive independently and crossing paths can cost much time. Most of the objects in LLSF, such as machines and the delivery zone, can only be used by one robot at a time. Therefore, we implemented a master-slave resource locking mechanism for the agents. If an agent decided to drive to a position, he requests the lock of this position and waits till he receives the lock. Then, he drives there and frees the lock when he leaves the position. The roles are also used for multi-robot coordination in the exploration phase. The roles determine which machine an agent investigates first. After the first machine, the agents continue the investigation of the other machines in the same cycle till all machine types are identified. The agents can pass each other on the cycle.

There already is a less sophisticated simulation, which is able to test if the high-level agent basically works. This simulation is implemented within the CLIPS-agent and provides information about the execution of actions and perception results in a trivial way. A few seconds after every action is called from the agent, the action is reported as successful. The perception results match exactly the ground truth published by the refbox. This is useful to test the behavior of the agent in the best-case scenario. Though, this simulation is only useful to test agent decisions and can therefore not indicate many important real world problems. Furthermore the simulated situation is not graphically visualized and therefore difficult to evaluate. Testing and evaluate multi-robot coordination is also not possible in this simulation because every agent uses an own simulation and the different time-spans for different actions can not be taken into account.

3.3.2 Marked Based Strategies

more task allocation strategies with e.g. dynamic role assignment?

The simulation should be able to test and compare complex multi-robot coordination strategies. An example of such a strategy, which is attractive for our needs and used for orientation in the design process of the thesis, is *Murdoch* [15]. Murdoch is an auction-based task allocation strategy. It solves a special instance of the *Optimal Assignment Problem*. The Optimal Assignment Problem is to assign n weighted jobs to m workers with non-negative skill ratings $h_{i,j}$ for each worker-job combination. Each worker is capable of doing one job at a time and each job can only be done by one worker. The task is to find the assignment which optimizes the performance, taking into account the weights of the jobs and the skill ratings of the workers to the jobs. The Murdoch method was especially developed for multi-robot scenarios. Therefore, it was designed to be able to handle robot failure and communication message loss. If a robot discovers a new task, it becomes an auctioneer and publishes a broadcast message to inform other robots about the task. All robots that are capable of doing the task calculate a metric of how well they probably will perform on the task. The metric is sent as a bid to the auctioneer. Finally, the auctioneer decides after a predefined time interval which robot gets the task and informs this robot. After the allocation, the auctioneer observes the progress of the task and starts a new auction if the robot assigned in the first place fails. This method is to some degree decentralized because the different auctions and metric calculation are not executed by a single robot. However, this approach is not optimal for a set of tasks because every auction only considers a single task. [15] showed that the method performs well in different multi-agent tasks, such as cooperative manipulation.
Murdoch allows a dynamic and situation dependent task allocation and is especially well suited for heterogeneous robots with different abilities. It is also a useful method to allow reallocation of tasks. An agent which is executing a tasks and has already completed a part could change his task if a new task is more important. The agent could simply include the penalty, that it has to cancel a task, and the advantage, that it has from completing the previous task, in his bid for the new task. An example in our domain is a robot which is carrying an intermediate product S_1 for the production of a new P_1. If a S_1 is needed at an other machine with higher importance, the robot can change the task. The cancel of the old task is no problem because the agent can just start a new auction for the old task. Disadvantages of Murdoch are the high demand of communication and the difficulty to detect failures.

Chapter 4

Approach

In this chapter, we describe the approach of this thesis by presenting the main ideas to reach our goals for the simulation and high-level agent in section 4.1. In section 4.2, we present the architectural approach.

4.1 Goals and Approaches

Realistic simulation: An important goal of the thesis is a high realism of the simulation. A more realistic simulation allows better testing because the simulation is more similar to the reality. Therefore, more real problems can be simulated. The realism of the simulation has many different aspects. The first group of aspects are about the realism of the sensor data. There are some sensors that are easy to simulate, such as distance sensors, bumpers and gyroscopes. Others, especially cameras, are more difficult to simulate realistically. The choice of Gazebo with ODE and Ogre for the visual appearance already allows us to simulate a wide range of sensors realistically. An other aspect of the simulation-realism is noise in the sensor data. For all metric sensors, we simulate this as Gaussian noise with the calculated value as mean and a configurable variance. The second group of aspects are about the realism of the physical environment. Objects have to behave in the simulation and the reality in a similar way. This is especially important for robot movement and manipulation tasks. Here, Gazebo with ODE also provides high realism out of the box. Nevertheless, it is important to find good physical parameters for the simulated objects. Examples of those parameters are the mass and coefficients of friction. An other important aspect of simulation-realism is that the simulation should not introduce new problems which do not occur in reality. Such a problem, which occurred in the development of the thesis, is that the simulation ran slower than the system time although the robot software used the system time.

Different levels of simulation: Sometimes, it is also useful to test with a more abstract and less realistic simulation. This allows testing the optimal case without sensor noise or without using low level components, such as localization, if those produce new errors or are not finished yet. To be able to simulate on different levels, we use the multi-level

abstraction approach [4] presented in the chapter 3. We use multi-level abstraction for sensing and for actuators. How we use multiple abstraction levels is shown in the next section about the architecture.

Compatibility with original robot software: We want to achieve that the robot software runs in the simulation in the same way as in reality. This is important because otherwise it would result in additional effort to keep the robot software running in the simulation and this effort can cause more difference between simulation and reality. In the thesis, the simulation components should provide the same interfaces as the real components that are simulated. In Fawkes, this is easy to achieve because the interfaces between components are well defined and components using an interface do not recognize whether the interface is provided by the simulation or a real component.

Efficient testing: We want to make the use of the simulation as efficient as possible. To achieve this, we provide scripts which automatically start the full simulation with all needed programs. Without these scripts, it would take much time. For a full simulation of an LLSF game, it is necessary to start Gazebo, the Refbox, a controlling Fawkes instance and, for the Robotinos, three times Fawkes, `roscore` and `move_base`. Each program-instance has to be started with correct parameters. Another approach we use to increase the efficiency is the feature to load different configurations and setups. For example, the scripts are able to load the simulation with different numbers of Robotinos, abstraction levels and simulation environments. Loading different configurations for the robot software is a feature of Fawkes and useful for us. We also integrated this in the scripts. We also provide the possibility to draw objects into the simulation. This makes it possible to visualize belief and intention of a robot to identify mistakes or ways to improve the system [40].

Multi-robot system evaluation: An important part of this thesis is the evaluation of multi-robot systems. Here, the features we already mentioned in the previous paragraph are especially useful because the testing effort (e.g. to setup the simulation or analyzing the coordination between robots) often scales with the number of robots. To evaluate the performance of the multi-robot system faster without having to observe the simulation all the time, we provide the possibility to run multiple simulation runs, in our case runs of LLSF games, automatically. With this feature, the simulation can run, for example, 20 times over night. To analyze executed simulation runs, we keep statistics of each run. For our LLSF domain, we keep the amount of achieved points with a detailed list of all actions that provided points with the time. These statistics can easily be extended by the amount of collisions between robots, waiting time for resource locks or other useful measurements. Sometimes, it can happen that single simulation runs have a very different result. To find the reason for this result, we keep log files of each run and also record the simulation itself. With this record, the simulation run can be reconstructed. The record contains the position and movements of all objects in the simulation. Here, it is also useful to draw additional information, such as the localization of the robots, in the simulation to be able to analyze faults without having to look in the log files. Automated simulation

runs also allow the comparison of different configurations of the multi-robot system. The automation script can be started with a number of configurations to test. This can be useful for finding the best performing role combination or parameters such as thresholds.

Expendability and flexibility: An essential criterion of a good simulation is the expendability to adapt to future changes. On the one hand, there will be changes to the Carologistics Robotino and to the LLSF. These changes have to be easy to implement in the simulation, otherwise the simulation would not remain useful. On the other hand, it is attractive to use the simulation also in other domains, such as RoboCup@Home. To achieve the expendability of the simulation, we provide a framework for simulated devices in Gazebo and Fawkes. Furthermore, we focus on developing small modules that are exchangeable and reusable. In Gazebo, new modules can easily be added to robot and world plugins. In Fawkes, it is easy to implement a new plugin which provides the wanted features because the plugin can use the provided Gazebo aspect. The simulation should also be flexible to adapt to small changes. Therefore, we provide several configuration files for the simulation environment, for the simulated objects and robots and for plugins in Gazebo and Fawkes.

Agent improvements: It is easy to see that the current approach we showed in section 3.3 has limitations which obstruct a significantly better performance of our multi-robot system. The static role approach does not allow a tighter cooperation between the agents or that the agent behavior can take the situation into account. For example, if there is not enough time to produce an other P_1 or P_2 puck or there are no more P_1 or P_2 pucks ordered, the P_1P_2 role would still start start another production. Therefore, we want to introduce a dynamic role change to change in specified situations to another role, such as producing P_3. Furthermore, we want to introduce recycling because it can lead to simple points and provides an alternative possibility to get new S_0 pucks. A third way to improve the performance of our system is to make use of three robots instead of just two.

4.2 Architecture

4.2.1 Simulation

The basic structure of the simulation is shown in Figure 4.1. On the right side, there is the Fawkes robot software framework we want to run in the simulation. On the left side, there is the simulator Gazebo. Roughly, the simulation is composed of a simulation environment, which handles the physical and visual simulation of the environment with all included objects, and plugins associated to the world and all models with some kind of behavior. There are plugin instances for the sensors and actuators of each robot as well as plugins that control the world. At this point, Fawkes can be seen as a composition of plugins, which are used on the real robot system, simulation plugins, which communicate with the simulation and exchange original sensor and actuator plugins, and the blackboard. The

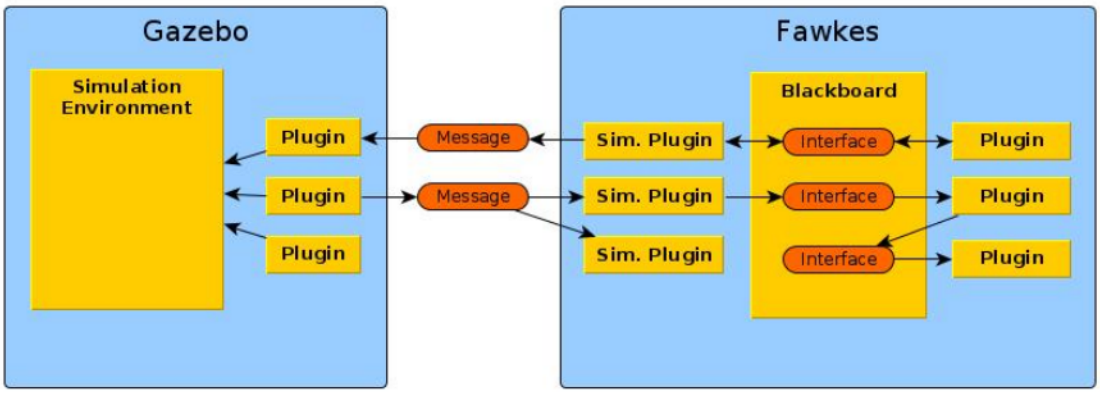


Figure 4.1: Architecture of the simulation

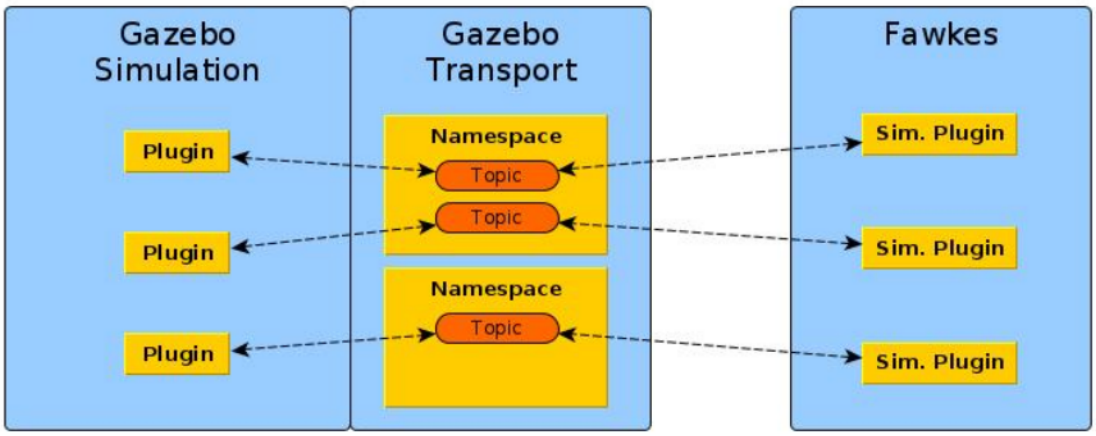
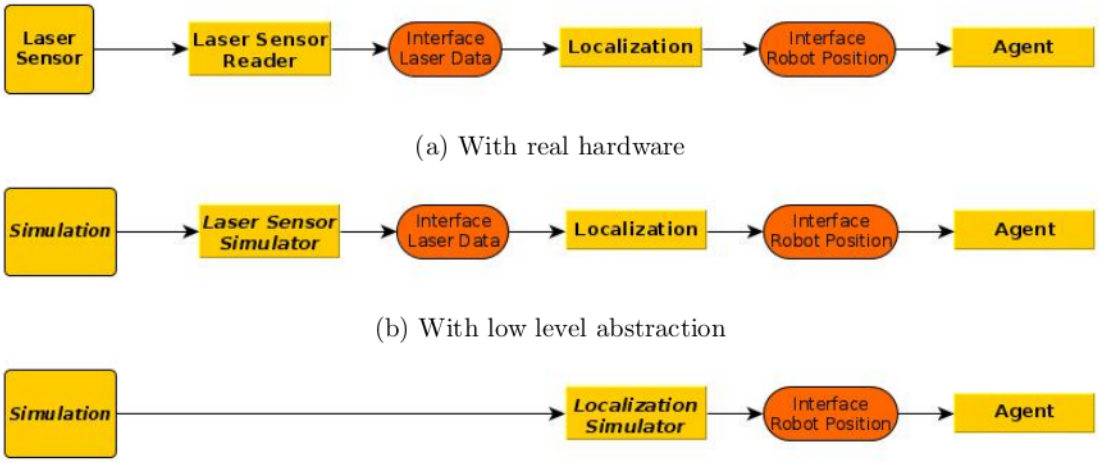


Figure 4.2: Communication between Fawkes and Gazebo

blackboard manages several interfaces which are used for communication between Fawkes plugins. The Fawkes simulation-plugins send actuator messages to the corresponding Gazebo plugins and receive messages from sensor-plugins in Gazebo. This architecture has the advantage that it is flexible and extendable. To add a new sensor or actuator to the simulation, it is sufficient to create or modify a Gazebo plugin and a Fawkes simulation-plugin to provide the corresponding interface. Furthermore, the architecture is flexible enough to allow multi-level abstraction as we see later in this section. It is also easy to change plugins on either side. It is even possible to use the Gazebo simulation with an other robot operating system than Fawkes. The other system just has to send and receive the same messages as our simulation-plugins.

4.2.2 Communication Fawkes-Gazebo

The communication between Fawkes and the Gazebo simulation has a specific structure to organize the distinction between multiple robots and different sensors and actuators. Figure 4.2 shows this structure. Fawkes simulation-plugins and Gazebo plugins communi-



(a) With real hardware

(b) With low level abstraction

(c) With high level abstraction

Figure 4.3: Example for multi-level abstraction (The arrows indicate data flow.)

cate over a message-transport system. In our case, this system is embedded in the Gazebo Transport API. In the message-transport system, there are different namespaces which act as communication channels. There is one namespace for each simulated robot and for the simulated world. The topics in a namespace represent what the messages that are send on this topic are about. Each plugin can listen to or send on the topics in a namespace. For example, there are topics about laser data and motor commands. This architecture is extendable because it is easy to add new namespaces or topics. It is flexible, because the communication through a topic in a namespace is a many-to-many relationship, and it is comfortable because instances of plugins in different Fawkes instances can listen to the same topic and get data from different simulated robots because the plugins listen do different channels.

4.2.3 Multi-level Abstraction

Figure 4.3 shows how multi-level abstraction fits in our simulation architecture. We choose a simple example about the localization with laser data. In Figure 4.3a, there is the system as it is used in the real application. There is the laser sensor hardware, a plugin which reads out the sensor and publishes the laser data in an interface, a localization plugin which uses the laser data interface to determine a robot position and to provide the position in an other interface. Then, the robot agent uses the position information from the interface. Figure 4.3b shows the system in the simulation with low level abstraction. The hardware is replaced by the simulation and the laser sensor reader by a simulation-plugin which gets the laser data from the simulation. This simulation-plugin provides the same interface as the original plugin. An other possibility with high level abstraction is shown in Figure 4.3c. Here, the localization plugin is replaced by a simulation-plugin which simply looks up the position of the robot in the simulation and writes it in the interface. The agent can now be

simulated with ground-truth position data and without using of the original localization plugin. An example for multi-level abstraction for actuators could be the movement of a robot. It is possible to simulate the motors with motor commands from an interface or the motion of the robot by applying the motion command in an interface directly to the simulated robot.

Chapter 5

Implementation

In this chapter, we describe how we implemented the simulation. Section 5.1 presents the various components we developed by describing what their task is, what we noticed during the implementation and why we implemented it this way. module dependencies extra? In section 5.2, we describe the dependencies and interaction between the modules we developed and in section 5.3, we present the improvements of the multi-agent system we implemented during the thesis.

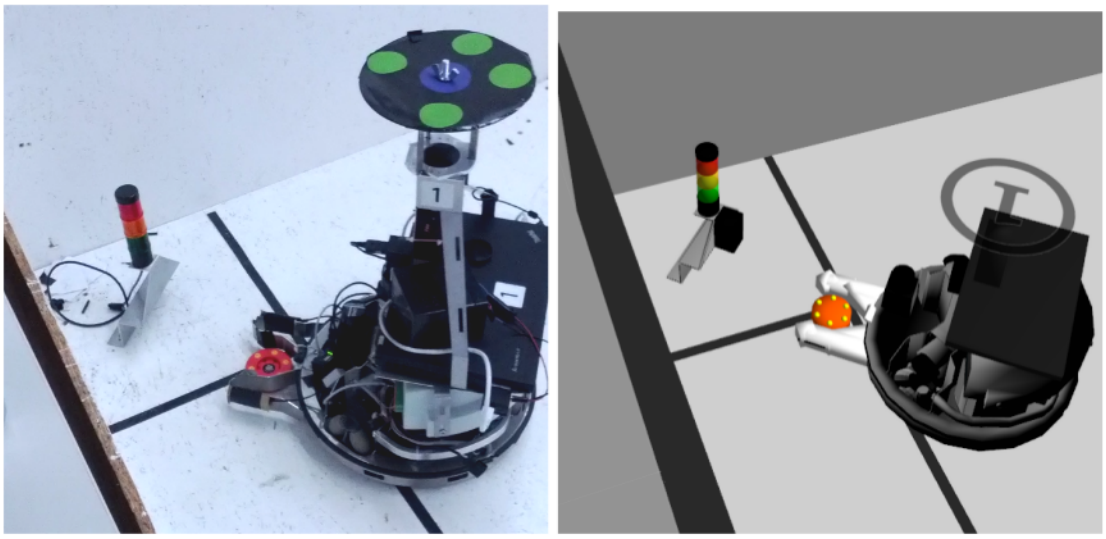
5.1 Components

5.1.1 Gazebo Models

In order to simulate the LLSF environment, we need to model all objects appearing in this domain. The models developed in this thesis are available at `https://github.com/zwilling/llsf-models.git`. Here, we present our simulation models and why we modeled them this way. Figure 5.1 shows a comparison between a real LLSF scene and the same scene in the simulation. In this comparison, the most important objects of the simulation can be seen. In the following, we describe each model in detail:

LLSF Field: The model of the LLSF field has a rather simple structure. It consists of a ground plane and four side walls. For the visual appearance and possible future changes, the field has a visual representation with lines and colored areas just as the real field. The machines are not realized as a component of the field and are attached to the field in the description of the simulation world.

Machine: The model of the machines matches their real counterpart structurally. Though, it is challenging to represent the lamps consisting of colored Plexiglas and a LED abbreviation? inside. We decided to use simple colored cylinders in the simulation. If the lamp is turned off, we use a dark and slightly transparent color and, if the light is turned on, we use a a bright color. This looks reasonable in the simulation and the images from the simulation are sufficient for our vision plugin which determines light states of the machines. The vision plugin measures the brightness at the position where it expects the machine-lamp to be and it even was not necessary to change brightness thresholds. In Figure 5.1a,



(a) A Robotino delivers a puck to a recycling machine.

(b) The same situation in the Gazebo simulation

Figure 5.1: Comparison of the real scene and the simulated scene in Gazebo

the black RFID box at the front of the machine is missing because we currently do not have the RFID readers.

Puck: The visual appearance of the puck model can be seen in Figure 5.1b. Physically, they are represented by a single cylinder. The difficult part was to find good friction parameters for the surface of the cylinder. On the one hand, it should be easy to slide the puck across the floor. On the other hand, the puck should stay inside the gripper of the Robotino when the Robotino turns. If the friction parameters are too small, the pucks move outside the gripper during turning because of the centrifugal force. mention other solution?

Robotino: The model of the Robotino is the most complex model because it holds different sensors, the casing and the puck gripper. It is also the most important one because it represents the robot we want to simulate. The major visual difference between real world and simulation is caused by the missing framework on top of the Robotino and the visual appearance of the puck gripper. Both is not important because we do net detect other Robotinos with a camera. We detect them with the laser sensor. Therefore and because of the manipulation with the gripper, the physical representation is more important. picture of collision elements? The physical model of the Robotino is composed of the gripper and two cylinders, one on the ground to represent the basic circle of the Robotino and one on laser and machine height. The cylinder on laser height is smaller and shifted back so that it does not block the laser sensor and it fits better to the real shape. The gripper has in the simulation a similar shape as the real one. We needed to assign higher friction parameters to the inside of the gripper than to the outside because in reality the puck slides into the gripper if the front side of the gripper and stays in the gripper while turning. Because we were not able to simulate this with a single set of friction parameters, we modeled an

additional geometry for the inside of the gripper with higher friction parameters. Originally, we intended to add wheels to the physical design for the final version. However, the omni-directional wheels caused an abnormal physical behavior in the simulation. The wheels irregularly bounced on the ground because of gravity and collision. Because we could not solve this problem in an arguable time and there is no important advantage, we decided to stay at the simple model without wheels. We only loose the possibility to physically simulate odometry with a realistic error. Therefore we introduced an artificially odometry error in the Gazebo plugin for the Robotino. **Simulation World:** The world file simply combines the developed models to an LLSF environment. Our world consists of the LLSF field, 16 machines with configurable orientation, 20 pucks and three Robotinos.

5.1.2 Gazebo Plugins

In order to control the simulation environment and models created before, we need Gazebo plugins. Each plugin belongs to an object in the simulation, such as a robot, a sensor or the simulation world. We developed two main plugins, one for a Robotinos and one for the simulation world. Both plugins are composed of smaller reusable modules, which can easily and dynamically be turned on and off. possible refactoring into smaller plugins in future work

World Plugin Our Gazebo plugin for the simulation world is called *llsf_world*. It manages the LLSF field, collects simulation data which is needed by several plugin instances and sends general simulation information to Fawkes. An important component of the llsf_world plugin is the simulation data table. It contains general data about the simulation and can be used by all plugins. In our case, the table contains machine locations, machine light-signals and puck locations. This simple data table allows decoupling information gathering modules, storage and modules which use this information. This is useful to make the simulation flexible and expendable for future changes. mention race conditions no problem? We implemented a module which receives messages about puck states and light states of the machines and writes it into the data table. Another module uses this data in the data table to apply the light-signals to the visuals representation of the machines. Furthermore, there is one plugin, which writes the position of pucks into the data table. The RFID readers are simulated by another module which uses the puck and machine positions. The results are stored in the data table and then sent to the Refbox. We also implemented a module to remove delivered pucks from the field. In reality, the referee does this task. There is also a module which sends the simulation time to Fawkes for synchronization.

Robotino Plugin The plugin for the Robotino robot mainly consists of sensor and actuator modules. We have implemented two actuator modules. One simulates the motor. It receives motor commands from Fawkes and applies the motion to the Robotino model. Because setting the velocity in Gazebo only means to apply a corresponding impulse to

the model, friction of the model with the ground leads to a resulting velocity which is smaller than the original intended velocity. Therefore, we increase the velocity set to the model so that the resulting velocity is similar to the intended one. The other actuator module operates on a higher abstraction level. It keeps pucks inside of the puck holder if the Robotino is turning. This module is optional but useful because of the difficulty to simulate the puck in the holder physically correct. The modules we implemented for the sensors also can be separated into low-level and high-level abstraction. On the low abstraction level, we developed modules for a gyroscope, a laser range finder, a webcam, a infrared distance sensor to detect pucks in the gripper and digital optical sensors on the sides of the gripper. Most of these modules use sensors defined in the Robotino model. On the high abstraction level, we developed modules for ground truth localization, machine light-signal detection and puck detection. The ground truth localization can be used in Fawkes in exchange of the amcl results and as a basis to simulate motor odometry. The machine light-signal and puck detection both use the simulation data table so that the multiple Robotino plugin instances do not have gather the information separately.

5.1.3 Fawkes Plugins

The Fawkes plugins developed in this thesis provide access to the simulation in Fawkes. In order to identify plugins for the Gazebo simulation quickly, we named them with the prefix *Gazsim*. We divided the plugins into two groups. The first group of plugins generally cover the access to the Gazebo simulation, sensors and the Robotino robot without limitation to a specific domain or task. All these plugins can be found in the simulation branch of the Fawkes repository[1]. The second group of plugins are needed only for the simulation of the LLSF. They can be found in our LLSF repository[2] which includes the general Fawkes repository. The access to the LLSF repository currently is limited because it contains new code we want to take advantage of in the LLSF competition.

Common Gazebo and Robotino Plugins

Gazebo: The plugin called *Gazebo* provides general access to the Gazebo Transport API which is needed to communicate with the simulator. Therefore, it provides a Fawkes aspect also called *Gazebo*. After connecting to the Gazebo simulator, the aspect gives access to two communication nodes. A node provides communication via a namespace in the Gazebo transport API. One node is responsible for the communication with the simulated robot Fawkes should control and has to be initialized with the name of the robot in the simulation. The other node is responsible for the communication with the simulation world and therefore used for robot independent information. It is also possible to spawn new models and visuals through this node. This is useful for controlling the simulation and visualizing tasks, for example to show robot intention. We extended the original Gazebo

[1] http://git.fawkesrobotics.org/fawkes.git
[2] http://git.fawkesrobotics.org/fawkes-robotino.git

plugin implemented by Klingen in [24] to be able to simulate multiple robots with different Fawkes instances.

Gazsim-Robotino: The *Gazsim-Robotino* plugin exchanges the Robotino plugin which connects Fawkes with the hardware of the Robotino robot. Therefore, our plugin provides the same interfaces as the original plugin. The original plugin provides motor and sensor features, each with an own interface. Because both are combined in the original plugin, we also combine these two features in the simulation plugin. However, we developed separate modules for these two features to improve re-usability. The module responsible for the motor sends the intended motor movement to Gazebo and receives the position of the robot in the simulation to compute the odometry. Because sending many Protobuf messages has been found to be computationally costly, we only send messages if he intended movement has changed. Here, we also introduced an error to the odometry which occurs more strongly when the speed changes quickly. As in reality, the odometry also changes if the robot drives against an obstacle and does not change it's position. The module responsible for the sensors of the Robotino receives information from the simulated gyroscope and distance sensors of the Robotino.

Gazsim-Laser: The *Gazsim-Laser* plugin exchanges the laser plugin and provides data from a simulated laser range sensor. The plugin converts received data into the in Fawkes used format and writes it into the laser-interface.

Gazsim-Localization: The plugin called *Gazsim-Localization* receives the position of the robot in the simulation and publishes this information in the corresponding interface. Normally, this information is provided by plugins, such as amcl. Therefore, this plugin operates on a higher level of abstraction. It uses ground truth information to allow testing high level components without error in the localization.

Gazsim-Webcam: The *Gazsim-Webcam* plugin receives a camera image from the gazebo simulation. It converts the received image from RGBexplain abbrv?, which is used in Gazebo, into YUVexplain abbrv?, which is used in Fawkes, and writes the image to a shared memory buffer. Unlike in other simulation plugins, there is no interface because vision plugins can directly load a camera through Fawkes tools. However, it is possible to load a shared memory image in the same way as a real camera. Vision plugin just have to be configured to use an other camera-identification string. We identify the shared memory images with the robot name as prefix because multiple Fawkes instances use the same set of shared memory images.

Gazsim-Timesource: The *Gazsim-Timesource* plugin provides the simulation time in Fawkes. This is important to avoid timing problems that occur if the computer is not able to run the simulation at real-time. Furthermore, it allows to run the simulation faster than real time to speed up the testing process. The plugin replaces the default time-source in Fawkes. To provide a precise time and to avoid bad performance because of many synchronization messages, we decided to estimate the simulation time between two received time-messages. We estimate the time with the current simulation speed and the real time passed since the last time synchronization message. To guarantee a monotonous time, we

do not set the time in Fawkes back if the simulation has slowed down between two time synchronization messages. This can create a time-difference and is more acceptable than a worse performance with many messages and a frozen time between two messages.

Gazsim-Comm: The *Gazsim-Comm* plugin simulates the communication between multiple Fawkes instances without using Gazebo. It runs on a separate Fawkes instance which does not control a simulated robot and acts similar to a hub. It uses a set of peer-to-peer connections to different program instances and forwards broadcast messages from one peer to all other peers. This solves the problem that usually the different Fawkes instances communicate on the same ports which is not directly possible if the instances run on the same computer. We preferred this approach over others because we also want to simulate communication over a poor wireless network. This is an important problem we want to simulate because during the RoboCup competition the wireless network usually is over-crowded. The package loss is simulated by the plugin by setting the likelihood for messages to be dropped before forwarding.

Gazsim-Vis-Localization: The *Gazsim-Vis-Localization* plugin visualizes the localization of a robot in the simulation. It spawns a label which identifies the robot with an arrow which indicates the orientation of the robot above the position where the robot believes to be. The visualization is only visible from above so that the robot can not see the label in the simulation. This visualization is useful during testing because a wrong localization is one of the most common reasons for misbehaving. Without this visualization in the simulation, it would be time-consuming to check the localization of multiple robots because it would be necessary to look into a separate Rviz for each robot. Furthermore, the visualization can be used to indicate the localization of the robots in a reconstruction of an automated simulation run.

LLSF specific Plugins

Gazsim-Light-Front: The plugin *Gazsim-Light-Front* exchanges the vision plugin *Light-Front* which determines the light-signals of machines by using a webcam. Light-Front uses the laser-cluster detection to know where to look for the machine light in the camera picture. Gazsim-Light-Front also uses laser-cluster results and the positions of the machines to determine on which machine the Robotino is looking at. If there is a machine near the position returned by the laser-cluster, the plugin determines the light-signals by using the last message received from Gazebo. The plugin also writes the visibility history, which indicates how long the vision result is stable, and the ready flag, which indicates if the plugin beliefs it has correctly determined the light state. This plugin also operates on a higher level of abstraction and provides ground truth information about the light-signals. This is useful for testing the agent because a wrongly detected light can have a crucial impact on the performance of the system.

Gazsim-Puck-Detection: The plugin *Gazsim-Puck-Detection* replaces the plugin *Omnivision-*

Pucks which finds pucks by using the omni-directional camera. Gazsim-Puck-Detection receives the positions of the robot and the pucks from Gazebo and writes the position of the pucks relative to the robot into the corresponding interface. A puck is only recognized if it is in a certain area around the robot. The maximal distance is similar to the real range in which Omnivision-Pucks can detect pucks. Here, we also have a plugin with high level abstraction. Omnivision-Pucks sometimes has problems with two pucks lying next to each other because it can interpret them as one puck. This can cause problems when trying to grab a puck and therefore providing ground truth data about the puck positions can be useful for testing.

Gazsim-LLSFrbcomm: The plugin *Gazebo-LLSFrbcomm* connects the LLSF Refbox with the Gazebo simulation. It connects to the Refbox as a client because only clients receive information about the machine light-signals. On the one hand, it sends received information about the machine light-signals to Gazebo via Protobuf messages. On the other hand, it receives messages from Gazebo about pucks placed or removed under RFID readers of machines and sends this information to the Refbox. We did not directly implement the connection to the Refbox in the simulation because Fawkes already provides a simple way for the connection and we do not want to duplicate the code used there. This plugin also sends the simulation time to the Refbox. In this context, we modified the Refbox to use the simulation time messages similar as the Gazsim-Timesource plugin. Similar as Gazsim-Comm, the plugin can run in a robot independent Fawkes instance.

Gazsim-LLSF-Control: The plugin *Gazsim-LLSF-Control* works in combination with the automation scripts. It waits until the Refbox announces that the game is over and then stops the simulation run. The automation script recognizes that the simulation stopped and can start a new run. This plugin also can be started in a robot independent Fawkes instance.

Gazsim-LLSF-Statistics: The plugin *Gazsim-LLSF-Statistics* keeps statistics about the result of automated simulation runs. For example, it keeps track of the amount of points scored in the exploration and production phase, the types of pucks being produced and the used configuration. When the game is over, the plugin writes these statistics to MongoDB. The statistics collected in a simulation run are inserted as a single document in the collection which includes all simulation runs of the schedule initiated with the automation script.

5.1.4 Automation Scripts

There are two major reasons why we decided to develop scripts to automate starting the simulation. On the one hand, a full LLSF simulation with three simulated robots requires a large amount of running programs. It is necessary to start Gazebo, the Refbox, a Fawkes instance for robot independent plugins and for each robot a separate Roscore, Movebase and Fawkes instance. Starting all programs with the required parameters is time consuming and can be automated. On the other hand, we need an automated start-up if

we want to run multiple simulation runs. This is useful to get an overview of the system performance and to compare different strategies and configurations. Therefore, it should be possible to let the simulation run 20 times with different configurations over night.
We implemented the automation scripts as simple bash scripts because they are powerful and easy to use and extend. There are two main automation scripts called *gazsim.bash* and *gazsim-schedule.bash*. The script gazsim.bash can start or stop a simulation and has several options. It is possible to start the simulation with a specific configuration and amount of robots. The simulation can also be started in a headless mode[3]. It is also possible to determine which level of abstraction should be used and if the simulation starts with ROS, recording and statistics. With the script gazsim-schedule.bash, multiple simulation runs can be scheduled. It is possible to define a set of configurations, which should be compared, the amount of simulation runs for each configuration and the level of abstraction.

5.2 Module Dependencies

already mentioned most dependencies, maybe a figure with all modules? would become huge

5.3 Agent Improvements

The main task we want to use our simulator for is the improvement of our multi-robot system. Therefore, improving our agent in this thesis is doubly important because we need it anyway and it is necessary to evaluate how useful our simulator is for the development of a multi-robot system. In the following, we present the three major improvements we made during this thesis. These improvements are not independent from each other because we combine them to improve the system even further. In addition, we also made several minor improvements. We mention these during this section and the evaluation chapter, when we present some of the problems found with the simulation. Spill the small beans fast or do not mention them?

5.3.1 Using Three Agents

Before the thesis, we used only two of three robots in the competition. Because we want to maximize our performance in LLSF, we have to use all three robots. In this thesis, we extend our current role based approach to work with three robots. We introduce new roles which can be easily allocated to the three robots.
First, we introduced two new roles by splitting the P1P2-role which produces P1 or P2 depending on what lets the P3 agent more space. Now we have a P1-role and a P2-role, which produce only P1 or respectively P2. Both roles use a disjoint set of machines to

[3] In the headless mode, Gazebo runs without a graphical user interface. This can save computation time. Although there is no visualization for the user, all visual sensors are still simulated.

avoid conflicts while producing intermediate products. To coordinate the use of resources shared by multiple robots, such as the delivery machines, we use the existing locking mechanism which scales well with an additional robot. Therefore, we can also assign multiple robots to the P3-role because there is no context information needed to produce a P3. Actually, assigning multiple robots to the P3-role seems to be a good idea because P3 is often highly demanded and producing P3 provides simple and quick points. There also are adjustments we had to make to use three robots in the exploration phase. The three agents have to manage which robot explores which machine. Our current approach defines a circular path passing all machines. Each agents start at a specified machine and explores all unknown machines along this path and can pass already explored machines and machines that are blocked by other agents. To use this approach with three agents, we separated the start machine from the role in the production phase and introduced a new set of starting machines.

5.3.2 Recycling

Another opportunity we did to use before the thesis is recycling. Recycling of consumed pucks can provide quick points because each recycled puck gives five points and takes only around two seconds to produce at a recycling machine. When using three robots it is also useful to reduce over-crowding of the insertion area where robots usually get new resource pucks from. The implementation of the recycling took two basic steps. First, we implemented necessary skills of the robot to grep a consumed puck lying next to a machine and to use a recycling machine. Here, the simulation already revealed a problem we did not know about before. We have to use a modified method to use recycling machines because the two digital optical sensors, which we use to check if the robot stands directly before the machine, does not work in the corner of the field. The sensor triggers if there is an obstacle in a certain range in front of the sensor and if the robot stands near a recycling machine, the sensor already triggers because the wall of the field is in this range. Because we can decrease this range in the simulation, it is possible to test and continue the rest of the development necessary for recycling. This is not so easily possible in reality. The second part of the implementation was including recycling in the agent. Therefore, the agent has to keep in mind where consumed pucks are and to know how to recycle them. We also implemented the possibility to configure when to do the recycling. Currently, the agent can recycle whenever it is possible and it needs a new S0 or when the production of a P1 or P2 is finished. We decided to introduce no recycling role because it would be less effective than doing recycling as an alternative way to get a new S0. The reason is that an agent with a recycling-only role would drop the new resource puck which could be used for a production step.

5.3.3 Dynamic Role Change

Our current static role based approach has the downside that the role allocation can not be adapted to the current situation. In some situations, there are roles which are significantly less useful than other roles. For example, assigning a P1-role to a robot makes no sense if there are no more P1 pucks ordered. Therefore, the robot should better produce P2 or P3 pucks. To be able to adapt the role allocation to the current situation, we introduced a dynamic role change. It is based on a set of rules, which define under which circumstances a role is changed into an other role. This is similar to the approach of [8] where the dynamic role change is formalized by an automaton with the roles as states and the changing rules as transitions. The most important role changes we implemented is from the P1P2-role, P1-role and P2-role to the P3-role and depends on the time left for the production phase. The production of a complex product takes a long time with our current approach. In five test runs we made with a single robot with the P1P2-role, the production without the delivery took in average about 10:30 minutes. Therefore, it is not reasonable to start a second production after the first one or, in general, if there is not enough time left. Producing P3 with multiple robots is more useful because the capacity of the T5 machine is better used. mention change after failed production?

Chapter 6

Evaluation

In this chapter, we evaluate the results of the thesis. The evaluation is separated into two parts. In the first part, we evaluate our simulation. In the second part, we evaluate of the multi-robot system and our improvements with the simulation.

6.1 Simulation

The evaluation of the simulation includes how good the simulation is in terms of realism and problems it can simulate, the computational effort with the resulting simulation speed, the use and advantages of the simulation and the limitations we encountered.

6.1.1 Realism

In this section, we show how realistic our simulation is, which problems we can simulate and which problems we can not simulate. Because there is no general measurement method to determine the realism of a robot simulator, we divide the problem and investigate all building block of the simulation. Afterwards, we also give a qualitative review how the simulation as a whole differs from reality.
Visually, our simulation can represent the basic structure of the LLSF environment. This already allows us to perform our vision tasks, which include brightness detection and color finding, in the simulation. However, it is difficult so simulate some problems that appear in reality. Especially reflections and ambient light can cause false positive vision results we can not simulate. Physically, Gazebo is able to simulate a realistic interaction between objects, but it is difficult to find appropriate friction parameters for objects, what can result in weird physical behavior. Furthermore, we have to compose the physical representation of an object by simple geometries because complex geometries are computationally costly to simulate. This also can cause a difference between simulation and reality. We are able to simulate movement of the Robotino, collisions with objects and other robots and pushing pucks well. However, we simulate the movement of the Robotino and carrying pucks in a gripper on a higher abstraction level because of already mentioned problems with omni-directional wheels and pucks moving out of the gripper during turning. The

distance sensors of the simulation produce realistic data because the data is easy to compute and we add Gaussian noise to the data. The only problem with the added noise in the current version is that the variance of the error added to a computed distance is constant whereas in reality the error depends on the distance. Therefore, in the simulation, the noise in the measured distance is higher as in reality for near objects. This seems to be no problem. We have not recognized a difference between the localization with amcl in the simulation and in reality. The gyroscope sensor also is easy to simulate and we have not recognized a difference to real sensor data. The communication between robots and the Refbox can also cause problems in reality. We simulate package loss which is the major cause for the problem. We do not simulate communication delay because this problem is not so important in the LLSF environment. In the simulation, the impact of communication problems on the multi-robot system is similar to what we have observed during the RoboCup 2013.

The simulation speed also can have an impact on the realism of the simulation. We minimized this impact by synchronizing the time of Fawkes and the Refbox with the simulation time. Because of the estimation method we use there to decrease the amount of sent Protobuf messages, a small time difference remains. We measured this difference 5 times. append measurements? Every time, the difference was less than 25 seconds for an 18 minutes game. Therefore, the impact on the performance is small. Slower update rates of sensors and movement commands are useful to increase the speed of the simulation but can also cause differences between simulation and reality. We use update rates which are compromises between realism and computational speed. The update rate of the laser range finder is $5Hz$, what is the half frequency of the real sensor, and there is no impact on the localization recognizable. The update rate of the webcam is $2Hz$ and therefore much slower than in reality. This increases the delay of vision results and has no important impact on the performance of a robot because the plugin light_front detects light states with a delay of about one second.

We compare the performance of our system at the RoboCup 2013 with the performance of the system with the same configuration in the simulation in section 6.2. The raw amount of points achieved in simulation and reality can not be used to evaluate the realism of the simulator because of different conditions. In the real competition, teams can take misbehaving robots out of the game and can restart a single robot once. We do not use this possibility in automated simulation runs. Furthermore, less vision failures which can have a large impact on the performance occur in the simulation. However, LLSF games in reality and our simulation look very similar because the robots perform the same actions and the problems that cause the robots to loose time or behave wrongly are the same or similar. For example the Movebase movement, which looks nervous and does much recovery behavior when facing obstacles, is the same in the simulation and the problem of interpreting a green light as a finished production, although the robot did not correctly place a puck under the machine, happens in the same way.

6.1.2 Computational Performance

The computational performance is important for a good simulation because the simulation speed drops significantly with a poor computational performance. The *simulation speed* is the time interval that can be simulated per second. For example, if a simulation needs 10 seconds to simulate what a robot does in 5 seconds, the simulation speed would be 0.5. In the following, the simulation speed also is called *real-time factor*. The simulation speed is important for testing because it determines the time needed for testing. There are possibilities to increase the simulation speed and to reduce the computational cost of the simulation. It is also possible to run the simulation faster than real-time. Gazebo provides three parameters for adjusting the simulation speed and detail. First, it is possible to determine a *target real-time factor* which limits how fast the simulation may run. It is only an upper limit and may not be reached if the computer is too slow. Second, there is the *maximal step-size* which determines the maximal time interval between two times a component of the simulation can do updates. Increasing the maximal step-size allows the simulation to run faster because the simulation simulates a larger time with one step. However, this can cause the simulation to be less accurate because simulation components have a higher response time. The default value of the maximal step-size is 0.001. Third, there is the possibility to adjust the *real-time update-rate* which determines the update calls of the physical simulation per real-time second. By decreasing this parameter, the simulation becomes computationally less costly and the physical simulation of objects in the simulation becomes less detailed. This has an especially high impact on the simulation of collisions because collisions are detected later. The default value of the real-time update-rate is 1000
We could increase the simulation speed significantly by tuning these three parameters. However, we experienced an impact on the quality of the simulation. With a real-time update-rate below 600, we recognized unrealistic simulation of collisions between pucks and between robots. With a maximal step-size above 0.002 and a resulting real time-factor above 1.5, we recognized that the Robotino more often drives against obstacles. We think this is due to the higher distance the Robotino can travel between two update iterations of the Movebase. We think a maximal step size of 0.0015, a real-time-factor of 1 and a real-time update-rate of 750 are a reasonable compromise between quality and speed of the simulation. With these parameters we achieved the CPU-usage and real-time factor shown in Figure 6.1.

The Figure shows that computational power needed for the programs which control the robots also is another important bottleneck. This is caused by the fact, that we operate multiple robot control programs on the same machine. The impact on the simulation speed is especially high if there are obstacles on the path of a robot and Movebase costly re-plans the path repeatedly. can not be seen in Figure Therefore, the computational cost of the simulation highly depends on the number of simulated robots. We reduced this problem by distributing the simulation and the robot control programs over multiple

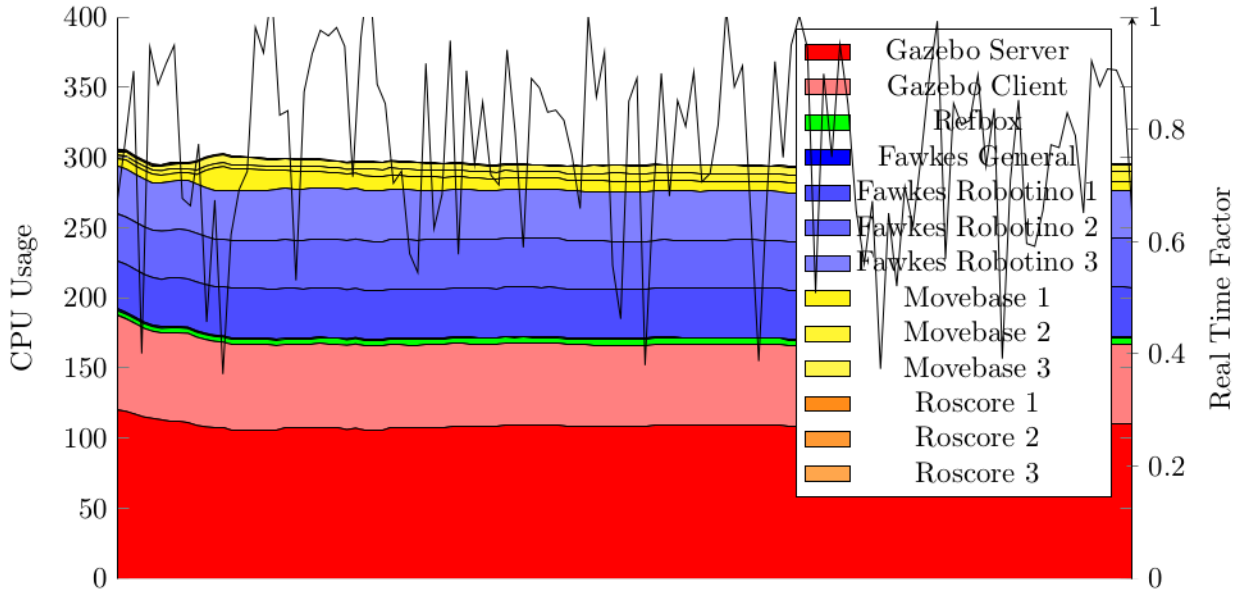


Figure 6.1: CPU Usage and Real Time Factor over a whole LLSF game legend

computers. This is especially easy to do with Movebase and helps us to avoid the major simulation speed drops when robots have navigation problems. The memory usage of the simulation is no real problem. During the whole simulation run shown in Figure 6.1 all parts of the simulation used together less than 800 megabyte memory.

6.1.3 Use for Multi-Robot System Development

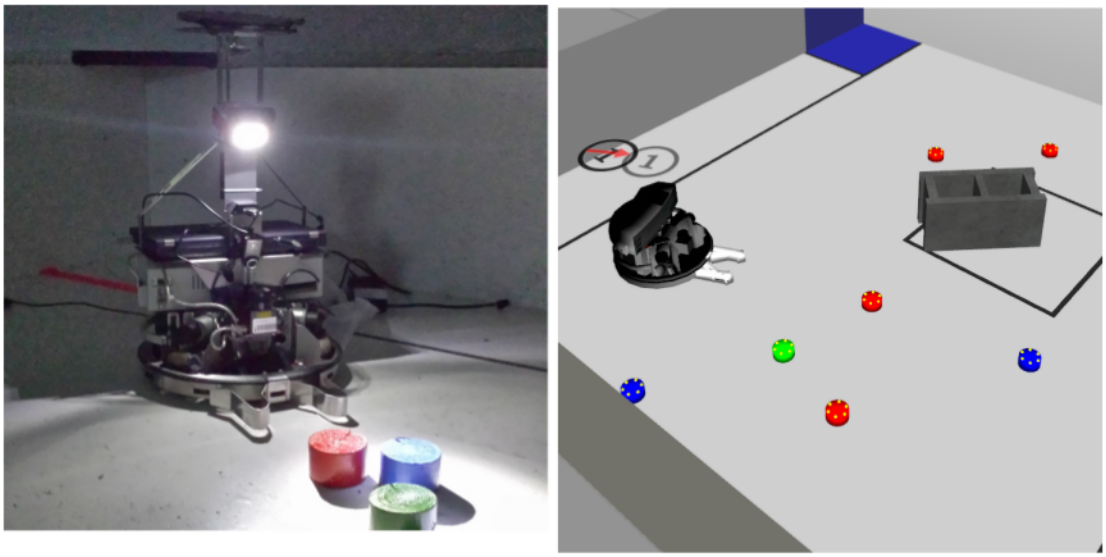
In this section, we argue how useful our simulation is for the development of a multi-robot system. Because we already mentioned many advantages and disadvantages of design and implementation decisions in detail, we focus here mainly on the practical experience of improving our multi-robot system with the simulator. The probably most important practical advantage of the simulator is the possibility to test the robot software without needing real robots and the environment. This allows testing although we currently do not have a working LLSF field. Furthermore, it allows better feedback in the development process because it is possible to instantly test changes after implementing. Therefore, the quality of the resulting software can be higher than without the simulator. Another beneficial advantage is the fast setup of a simulation test. There is no need to start the robot or to synchronize source code. The scripts can start a full LLSF simulation as well as a prepared setup to test robot skills. This allows fast testing of different components. What also has bee found very useful for evaluation of our multi-robot system is the script for automated simulation runs. It simplifies evaluating and comparing different strategies and configurations because of the recorded statistics and the possibility to look afterwards at runs in detail. This allows finding causes of poor performances which happen irregularly.
more advantages
The multi-level abstraction approach has turned out to be not so important for the agent

changes and evaluation of our multi-robot system in this thesis. The reason is that the components we can simulate on multiple levels of abstraction have a similar reliability in the simulation. For example, the detection of machine lights by the vision plugin is very reliable in the simulation, because there are no reflections in the simulation and the region, where the vision plugin looks for the machine light in the image, is at the right location. However, this indicates that a wrong placement of this region in reality might not be caused by the vision plugin but by inaccurate values for the position and orientation of the camera on the robot. Although now multi-level abstraction is not so important, it can become indispensable in the future, for example when multiple components are in development at the same time.
How useful the simulation is for the development can be seen at the amount of real problems we found by testing in the simulation. For example, we discovered that currently we can not pick up pucks that lie besides the gripper because the Robotino firstly tires to turn to the puck and so pushes the puck to an other position. We also discovered that we can not correctly align in front of a recycling machine because the digital optical distance sensors can not distinguish between the plate of the machine and the wall near to the recycling machines. We could reproduce both problems in reality. We also found and fixed a couple of bugs, such as a wrong calculation of the angle towards a target orientation in the navigator.
Beside these points the simulation is very useful for the evaluation of our multi-robot system. The experiences and discoveries we made in the evaluation is shown in section 6.2.
problem test specific situation

6.1.4 Hackathon

We also used our simulation in the Hackathon in November 2013. The Hackathon was an event organized by Bonding[1] and Carologistics. In one night, the participants got an insight into developing skills (behavior engine in background) with Fawkes and LUA in a search and rescue scenario. The task was to find colored pucks with the Robotino at night and bring the pucks to a safe area. Figure 6.2a shows a Robotino doing this task. In addition to the equipment in LLSF, the Robotino holds a flashlight and a webcam looking downwards to detect pucks. The simulation was an essential part of the Hackathon for multiple reasons. There were about 40 participants, which worked in teams of two or three people, and only 4 available Robotinos and fields. Therefore, the simulation was necessary to provide the possibility of frequently testing and testing was important because the participants had no experience with LUA, the Robotino and our behavior engine. The scripts allowed easy and fast starting of the simulation and all needed components to operate the simulated Robotino. So, the participants did not need to know how to start Gazebo, ROS and Fawkes with additional plugins. Furthermore, the simulation provided a safe environment where no hardware can be damaged. The successful use of the simulation

[1] Boning is a student association which organizes events to provide insights into the working life and contacts to companies looking for employees (`http://www.bonding.de`).



(a) The Robotino is looking for colored pucks.

(b) Same scene in the simulation

Figure 6.2: Bonding Hackathon 2013: The Robotino has to find pucks with a webcam and bring them in a safe area.

at the Hackathon also shows the capabilities of the simulation. Many teams primarily tested their code in the simulation and added only slight changes after testing on the real robot. This shows that the simulation is very similar to the real application. Feedback from some participants showed that the simulator is easy to use and runs with a high simulation speed close to real time even on a slow notebook. We were able to increase the simulation speed because the Hackathon task did not require so much physical accuracy as the LLSF task. However, the GUI of the simulator reacted more slowly than on faster computers. We think this was due to missing graphic drivers.
The use of the simulation at the Hackathon also showed that the simulation is extendable and easy to modify. The new approach to detect pucks with different colors works well with the webcam module developed for machine light detection and modifying the scripts and models for the Robotino and the LLSF field with colored pucks and obstacles was easy and fast. We also deactivated the modules and plugins we did not need in the Hackathon to save performance.
The Hackathon was successful, most teams were able to develop a working skill which saved multiple pucks in the final competition on the real robot although they worked the first time with LUA, the Robotino and the behavior engine. Some teams performed very well and saved nearly all pucks.

6.1.5 Limitations

In this subsection, we describe the limitations of the simulation. First of all, a simulation can not substitute testing with a real robot completely because the reality is too complex to simulate it exactly. It is especially difficult to simulate realistic physics and graphics.

Because of that, we were not able to appropriately simulate omni-directional wheels with realistic odometry errors and reflections on machine signal-lights which cause problems for machine light recognition. Furthermore, the available performance on current computers limits the number of simulated robots and sensor update rate and precision at a reasonable simulation speed and physical accuracy. Unfortunately, some limitations of the simulation of low level sensors and actuators also restrict the possibility to test components on a higher level with dependencies on these low level components. For example we can not the reaction of the navigation on a realistic odometry error. We only can introduce an artificial odometry error. A limitation of our automated simulation run is the human interaction in the multi-robot system. In LLSF, teams can remove misbehaving robots from the game and restart a robot a single time. This can only be done in the automated tests if a human is present all the time. As we show in this thesis, we were able to develop a useful and sufficiently realistic simulation inside all these limits.

6.2 Multi-Robot Strategies

In this section, we evaluate our multi-robot system and the improvements we made with the simulation. This shows what the simulation can evaluate and where we have to improve our system in the future. We compare the improved system to the version at the RoboCup 2013 in Eindhoven. As mentioned in section 3.3, we used two Robotinos, one with the P_3-role and one with the P_1P_2-role. Unfortunately, the real performance in Eindhoven is not suitable for comparison with simulation runs because of the human interaction with misbehaving robots in the real game and less wrong perception results in the simulation. Furthermore, we also applied small improvements to the system that cause the system to perform better than in Eindhoven. Therefore, we rather use simulation runs with the same configuration as in Eindhoven as a basis for comparison. Figure 6.3 shows the performances with two robots. We simulated 10 to 15 runs for each configuration. All experiments in the simulation have in common that there are more outliers with poor performance because we can not restart misbehaving robots in the simulation. This also scatters the performance of single simulation runs because the amount of lost points caused by a misbehaving robot depends on the time when the misbehaving starts. Some of the most common causes for bad performances are shown in Figure 6.4. Figures 6.4a and 6.4b show situations in which the Robotino gets stuck during low level movement. Often these situations block the robots the rest of the game because there is no timeout to detect that the robot is stuck or the recovery method is not able to solve the situation. These situations have a large impact on the performance of the system. The situations shown in Figure 6.4c and 6.4d happen during navigation. Often, such situations get solved, but it can take much time. These situations also can block the whole multi-robot system if one of the involved robots locks a position other robots need too.
Because these problems limit the possibilities of the system and therefore the impact

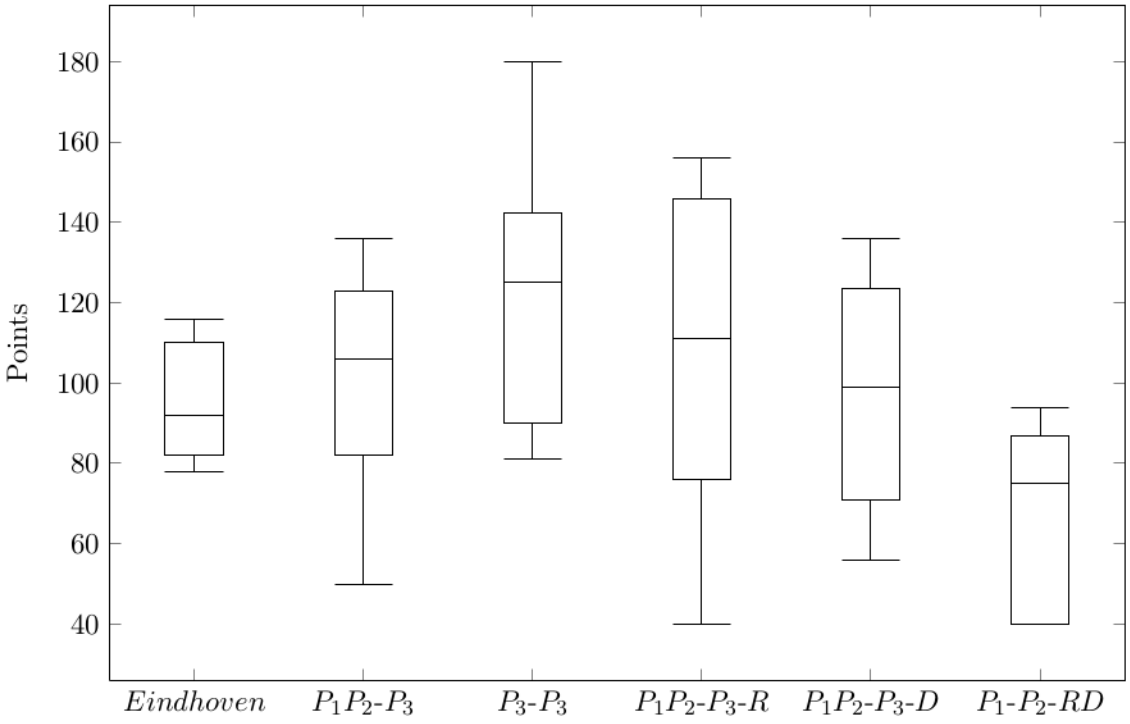


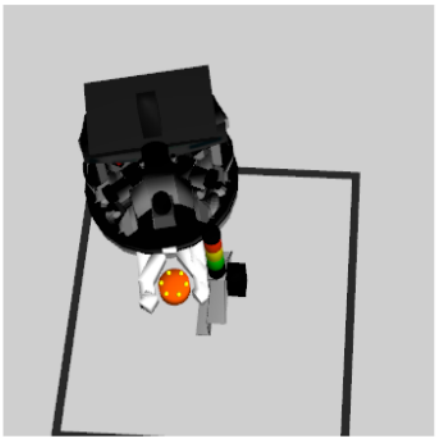
Figure 6.3: The box-plots show the amount of points reached in multiple LLSF simulation runs with two robots. *Eindhoven* shows the real performance at the RoboCup 2013, all other results were simulated. The labeling marks used roles and improvements (R: recycling, D: dynamic role change). Each configuration was tested 10 to 15 times.

of agent improvements, good runs of different configurations are more meaningful when comparing different agents.

As we can see in Figure6.3, two P_3 agents perform better than one P_1P_2 and one P_3 agent. This is due to the fact that P_3 provides more points per production step and is less likely to fail because of the lower amount of steps needed to finish a P_3 puck. We did not use two P_3 agents at the RoboCup in Eindhoven although this was already possible. We simply were not able to test it because all testing time was needed for components on a lower level. This again shows how useful the simulation is for evaluation.

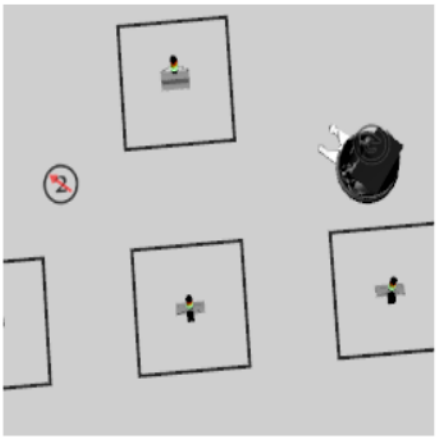
6.2.1 Dynamic Role Change

We expected that the P_1P_2-P_3-D configuration performs 10 to 20 points better per run than the P_1P_2-P_3 configuration because after finishing the first complex product the P_1P_2 agent becomes a second P_3 agent. Surprisingly, the performance of P_1P_2-P_3-D is not better than the performance of P_1P_2-P_3. The reconstructions of the simulated P_1P_2-P_3-D runs show that a role change happens late and only rarely because it takes a large part of the game-time to finish the complex product and when the P_1P_2 agents gets stuck, it happens often in a way shown above and therefore the role change does not happen. The effects of the dynamic role change on the P_1-P_2-RD configuration is similar. The statistics about the produced and delivered pucks show that the change worked in 30% of the games

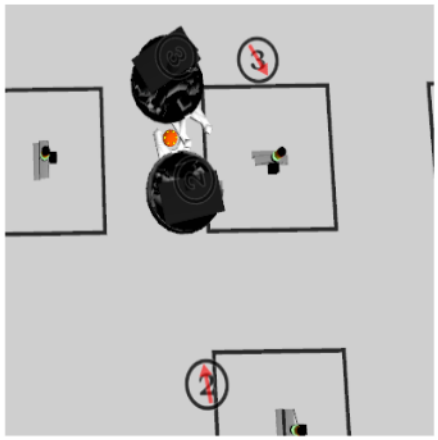


(a) The Robotino wants to turn left, but can not reach the target position because of the machine.

(b) Stuck Robotino after approaching the machine from an inappropriate angle



(c) The Robotino is wrongly localized and can not reach the destination.



(d) Robotinos having troubles passing each other.

Figure 6.4: Failures and problems causing bad performance

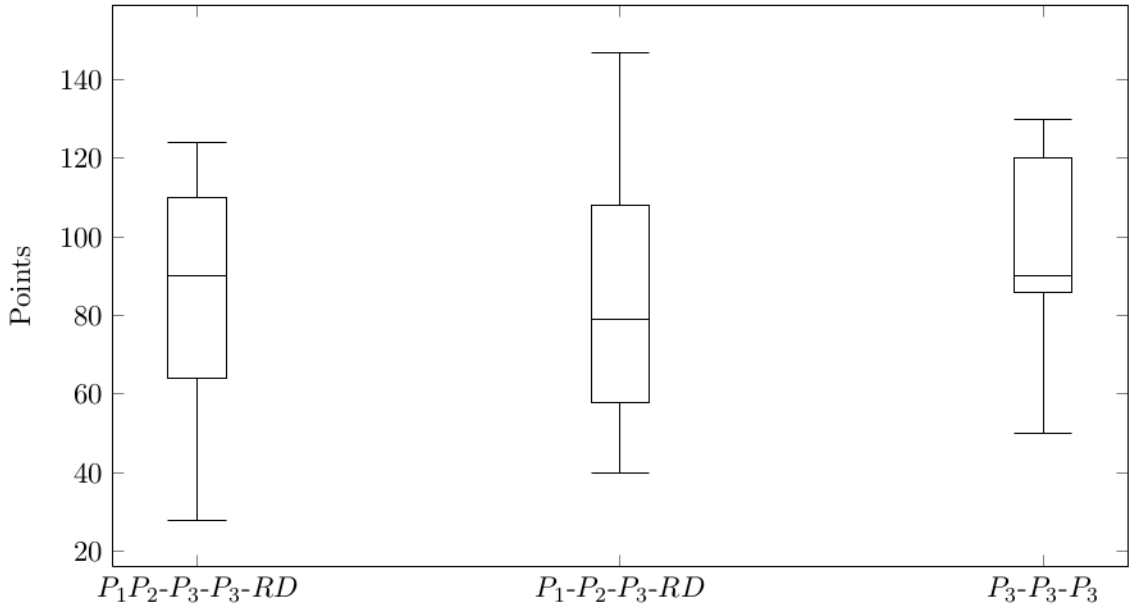


Figure 6.5: The box-plots show the amount of points reached in multiple LLSF simulation runs with three robots. The labeling and test population is equivalent to Figure 6.3.

and provided 10 to 30 additional points.
Overall the dynamic role change has potential that is limited by the slow production speed and situations in which the robot gets completely stuck.

6.2.2 Recycling

The P_1P_2-P_3-R configuration mostly performs better than the P_1P_2-P_3 configuration. The statistics show that the P_1P_2 agent scores 5 to 10 additional points via recycling. The first 5 recycling points are scored early in the game after the S_2 production and further recycling points depend on the finished production of a complex product. The reconstructions of some simulation runs, especially with three robots, show that the recycling helps to avoid the bottleneck at the insertion area where robot get new S_0 pucks. However, Figure 6.3 shows that there are also a couple of games with a worse performance than P_1P_2-P_3. The reason is that the recycling step is more likely to fail than just getting a new S_0 puck. There are cases in which the Robotino gets stuck at a machine when trying to grep a consumed puck and in which the Robotino has problems with leaving the recycle machine. In no simulated game the recycling was necessary because of S_0 pucks running out of stock. This might change in the future if we can produce more and faster.

6.2.3 Role Configurations for Three Agents

To find a good role assignment for three robots and figure out the problems we need to tackle in the future, we compare several configurations here. Figure 6.5 shows the evaluation results with the configurations P_1P_2-P_3-P_3-RD, P_1-P_2-P_3-RD and P_3-P_3-P_3. Surprisingly, no configurations with three robots performs better than a similar configura-

tion with two robots. The reasons can easily be seen in some reconstructions. First, using three robots lead to significantly more navigation conflicts, such as in Figure 6.4d, than using two robots. This often also effects the third robot when it has to pass the other two robots or it needs a resources that is locked by another robot. Second, the bottleneck of the current approach, which is the single point for getting new pucks, affects three robots more than two. Therefore, robots often have to wait until for other robots. Third, the failure of single robots remains a big problem. On the one hand, there are two robots left if one fails, but on the other hand a failing robot that locked an important resource still blocks the whole system.

6.2.4 Different Abstraction Levels

bad network influence eval

Chapter 7

Summary and Future Work

In this chapter, we show possible future work on the simulation and our multi-robot system for LLSF in section 7.1. Afterwards, we conclude the thesis in section 7.2.

7.1 Future Work

In the near future, the simulation and our multi-robot system solution for LLSF need to be changed because LLSF is going to be changed. At the RoboCup 2014, it is likely that two teams play at the same time on the same field with double size. The machines with RFID-readers might also get replaced by small assembly machines. This also demands hardware changes on the Robotino, such as adding a new gripper. We probably also add different sensors, such as a stereoscopic camera to the Robotino to get a more reliable puck and light-signal detection. The simulation and the agent have to be adapted to these changes.
The simulation can also be extended to work in other domains with other robots, such as domestic service robots. The motor and basic sensors we implemented can be reused for this. To improve the expendability of the simulation even further it is also useful to refactor the Gazebo plugins. The existing modules for sensors and actuators can be separated into plugins and configuration values can be put in separate configuration files which are read at the start of the plugins. This would allow reconfiguration and building of new simulated robots without recompiling.
The simulation developed in this thesis can be the foundation of a variety of future projects. For example, the the visualization possibilities could be used to visualize agent belief and intention, what would simplify the development of more advanced agent strategies, or the automatic simulation runs can be used to implement some kind of machine learning or parameter finding. The simulation is an important tool for future development for the LLSF. In the near future, we have to solve the problems we discovered in the section 6.2. This includes navigation-coordination with multiple robots to avoid that two robots block each others path, improving the locking approach to be more robust against failure, increasing reliability of low level skills and recovery from situations shown in Figure 6.4.

Furthermore, we have to develop large agent improvements, such as a more flexible and optimized task allocation strategy and producing without waiting at the machine until the production is finished to do other tasks in the meantime.

7.2 Summary

In this thesis, we have developed a multi-robot simulation for the LLSF. The simulation was designed to allow efficient testing of multi-robot coordination and and other important parts of a multi-robot system. We achieved this by implementing a realistic LLSF simulation in Gazebo and connecting this simulation to Fawkes. In Fawkes, simulation plugins exchange real sensor and actuator plugins and provide the same interfaces so that the components we want to test can operate in the same way as when running on a real robot. To increase efficiency during testing and evaluating, we implemented start-up scripts and automated test runs with statistics and reconstructions of simulated runs. The simulation also was designed and implemented in a way so that future changes on the LLSF and the Carologistics Robotino are easy to implement in the simulation. This also allows using the simulation in other domains and with other robots because the small plugins and modules we implemented can be reused. For better testing possibilities, we included multi-level abstraction which allows us to test high level components with simulated sensor data or ground truth information.
To improve our multi-agent system and test the evaluation capabilities of the simulation, we implemented a dynamic role change to adapt the task allocation to the situation and recycling to score more points more continuously during the production of complex products. Furthermore, we extended our previous role based approach to work with three instead of two robots in the exploration and production phase.
We evaluated how how realistic the simulation is. There are limitations of the simulation realism mainly caused by missing visual phenomenon, such as realistic reflections, and defining realistic physical properties, such as friction parameters. Within these limitations, we can realistically simulate the LLSF environment and sensors and actuators of the Robotino, what is more than sufficient for the purpose of evaluating multi-robot systems.
In the evaluation, we also analyzed several configurations with different role assignments and agent improvements and compared them with the performance of the previous system. We found that two P_3 agents perform significantly better than other configurations because producing P_3 pucks continuously provides many points is has the least likelihood to fail. Recycling improved the performance of the system because it provides points early in the production process of a complex product, but also added more steps to the production process what increased the likelihood to fail at some point during the production. The dynamic role change provided a smaller advantage than we expected because role change happens only rarely and late. The performance of all configurations was limited by the several problems we identified in the evaluation. These problems mainly include navi-

gation coordination, reliability of low level movement and failure handling. We noticed that these problems have to be solved before improvements on the high level agent can use their full potential. These problems especially limit the performance of configurations with three robots as we showed in the evaluation.
In the previous section about future work, we described that the simulation is extendable for future changes and can be the foundation of several future projects. We showed that the simulation is extendable and useful in the testing process during the Bonding Hackathon. The simulation is also an important tool for future developments on our solution for the LLSF. The simulation has already been successfully used in some developments beside agent improvements. Two of these developments are the use of an alternative navigation and collision avoidance plugin on the Robotino and the vision detection of colored pucks for the Hackathon.

Bibliography

[1] Evan Ackerman. DARPA Awards Simulation Software Contract to Open Source Robotics Foundation. `http://spectrum.ieee.org/automaton/robotics/robotics-software/darpa-robotics-challenge-simulation-software-open-source-robotics-foundation`, 2012.

[2] Defense Advanced Research Projects Agency. DARPA Robotics Challenge. `http://www.theroboticschallenge.org/`, retrieved Nov 4rd 2013.

[3] H Levent Akin, Nobuhiro Ito, Adam Jacoff, Alexander Kleiner, Johannes Pellenz, and Arnoud Visser. RoboCup Rescue Robot and Simulation Leagues. *AI Magazine*, 2013.

[4] Daniel Beck, Alexander Ferrein, and Gerhard Lakemeyer. A Simulation Environment for Middle-Size Robots with Multi-level Abstraction. In *RoboCup 2007: Robot Soccer World Cup XI*, pages 136–147. Springer, 2008.

[5] Joschka Boedecker and Minoru Asada. Simspark–concepts and application in the robocup 3d soccer simulation league. *Proceedings of the SIMPAR*, 2008.

[6] Stefano Carpin, Michael Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. USARSim: a robot simulator for research and education. In *2007 IEEE International Conference on Robotics and Automation*, pages 1400–1405. IEEE, 2007.

[7] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. USARSim: A robot Simulator for Research and Education. In *2007 IEEE International Conference on Robotics and Automation*, pages 1400–1405. IEEE, 2007.

[8] Luiz Chaimowicz, Mario FM Campos, and Vijay Kumar. Dynamic Role Assignment for Cooperative Robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, volume 1, pages 293–298. IEEE, 2002.

[9] Kristina Chodorow. *MongoDB: the definitive guide*. O'Reilly, 2013.

[10] Mark Collins-Cope. Component based development and advanced OO design. Technical report, Technical report, Ratio Group Ltd, 2001.

[11] Florian Cordes, Thomas M Roehr, Frank Kirchner, and DFKI Robotics Innovation Center Bremen. RIMRES: A Modular Reconfigurable Heterogeneous Multi-Robot Exploration System. In *Proceedings of the 11th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS'12)*, 2012.

[12] Chris Culbert, G Riley, and B Donnell. Clips reference manual. *A. I. Section, LBJ Space Center*, 1989.

[13] DARPA. Virtual Robotics Challenge Rules. `http://www.theroboticschallenge.org/local/documents/VRC%20Rules%20Release%202%20DISTAR%20Case%2021064.pdf`, 2013.

[14] Brian Gerkey, Richard T Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the 11th International conference on Advanced Robotics*, volume 1, pages 317–323, 2003.

[15] Brian Paul Gerkey. *On Multi-Robot Task Allocation.* PhD thesis, University of Southern California, 2003.

[16] Erico Guizzo. Three Engineers, Hundreds of Robots, One Warehouse. *Spectrum, IEEE*, 45(7):26–34, 2008.

[17] Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone. Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In *RoboCup 2006: Robot Soccer World Cup X*, pages 72–85. Springer, 2007.

[18] Ulrich Karras. Robotino–An Open Learning Mobile Robot System for Robocup. Festo Didactic, Denkendorf, Germany, 2009.

[19] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. *Aspect-oriented programming.* Springer, 1997.

[20] J.-H. Kim, H.-S. Shim, H.-S. Kim, M.-J. Jung, I.-H. Choi, and J.-O. Kim. A cooperative multi-agent system and its real time application to robot soccer. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, volume 1, pages 638–643 vol.1, 1997.

[21] H. Kitano. RoboCup Rescue: a grand challenge for multi-agent systems. In *Multi-Agent Systems, 2000. Proceedings. Fourth International Conference on*, pages 5–12, 2000.

[22] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The Robot World Cup Initiative. In *Proceedings of the First International Conference on Autonomous agents*, pages 340–347. ACM, 1997.

[23] Kleiner, Alexander and Göbelbecker, Moritz. Rescue3D: Making rescue simulation attractive to the public. 2004.

[24] Bastian Klingen. Scene Reconstruction for post mortem Fault Analysis using the Gazebo Simulator from an on-line recording Database on a mobile Robot. Master's thesis, RWTH Aachen, 2013.

[25] Nathan Koenig and Andrew Howard. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2149–2154. IEEE, 2004.

[26] Zeid Kootbally, Stephen Balakirsky, and Arnoud visser. Enabling codesharing in Rescue Simulation with USARSim/ROS. In *RoboCup Symposium*, 2013.

[27] Patrick MacAlpine, Samuel Barrett, Daniel Urieli, Victor Vu, and Peter Stone. Design and Optimization of an Omnidirectional Humanoid Walk: A Winning Approach at the RoboCup 2011 3D Simulation Competition. In *AAAI*, 2012.

[28] Artur Merke and Martin Riedmiller. Karlsruhe brainstormers-A reinforcement learning approach to robotic soccer. In *RoboCup 2001: Robot Soccer World Cup V*, pages 435–440. Springer, 2002.

[29] Olivier Michel. Webots: Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems*, 1(1):39–42, 2004.

[30] Olivier Michel, Yvan Bourquin, and Jean-Christophe Baillie. Robotstadium: Online humanoid robot soccer simulation competition. In *RoboCup 2008: Robot Soccer World Cup XII*, pages 580–590. Springer, 2009.

[31] Tim Niemueller. Developing A Behavior Engine for the Fawkes Robot-Control Software and its Adaptation to the Humanoid Platform Nao. Master's thesis, RWTH Aachen University, Knowledge-Based Systems Group, April 2009.

[32] Tim Niemueller, Daniel Ewert, Sebastian Reuter, Alexander Ferrein, Sabina Jeschke, and Gerhard Lakemeyer. RoboCup Logistics League Sponsored by Festo: A Competitive Factory Automation Testbed. In *RoboCup Symposium*, 2013.

[33] Tim Niemueller, Alexander Ferrein, Daniel Beck, and Gerhard Lakemeyer. Design Principles of the Component-Based Robot Software Framework Fawkes. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 300–311. Springer, 2010.

[34] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. Incremental Task-level Reasoning in a Competitive Factory Automation Scenario. In *Proc. of AAAI Spring Symposium*, 2013.

[35] Technical Committee of the RoboCup Logistic League. Rules of the Logistic League sponsored by Festo. `http://www.robocup-logistics.org/rules`, 2013.

[36] Organizing Committee of the RoboCup 3D Soccer Simulation League. RoboCup Soccer Simulation League 3D Competition Rules and Setup. `http://homepages.herts.ac.uk/~sv08aav/RCSoccerSim3DRules2013.1.pdf`, 2013.

[37] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, 2009.

[38] RoboCup. RoboCup Objective. `http://www.robocup.org/about-robocup/objective/`, 2013.

[39] RoboCup. Soccer Simulation League. `http://wiki.robocup.org/wiki/Soccer_Simulation_League`, retrieved Nov 4rd 2013.

[40] Justin Stoecker and Ubbo Visser. Visualizing and Debugging Complex Multi-Agent Soccer Scenes in Real Time. In *RoboCup Symposium*, 2013.

[41] Richard Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2-4):189–208, 2008.

[42] Wikipedia. Fukushima Daiichi nuclear disaster. `http://en.wikipedia.org/wiki/Fukushima_Daiichi_nuclear_disaster`, retrieved Nov 3rd 2013.

[43] Wikipedia. RoboCup 2D Soccer Simulation League. `http://en.wikipedia.org/wiki/RoboCup_2D_Soccer_Simulation_League`, retrieved Nov 4rd 2013.

[44] Robert M Wygant. CLIPS – A Powerful Development and Delivery Expert System Tool. *Computers & industrial Engineering*, 17(1):546–549, 1989.

[45] Yuan Xu, Heinrich Mellmann, and Hans-Dieter Burkhard. An approach to close the gap between simulation and real robots. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 533–544. Springer, 2010.

[46] Yuan Xu and Hedayat Vatankhah. SimSpark: An Open Source Robot Simulator Developed by the RoboCup Community. In *RoboCup Symposium*, 2013.