B-Smart

(Bremen Small Multi-Agent Robot Team)

Team Description for RoboCup 2010

Tim Laue[1], Sebastian Fritsch[2], Kamil Huhn[2], Arne Humann[2], Michael Mester[2], Jonas Peter[2], Bastian Reich[2], Max Trocha[2]

[1] Deutsches Forschungszentrum für Künstliche Intelligenz GmbH,
Sichere Kognitive Systeme, Enrique-Schmidt-Str. 5, 28359 Bremen, Germany
[2] Fachbereich 3 - Mathematik / Informatik
Universität Bremen, Postfach 330440, 28334 Bremen, Germany
grp-bsmart@informatik.uni-bremen.de

Abstract. This is the Team Description Paper of the RoboCup team B-Smart for the Small Size Robot League competition at RoboCup 2010. We will give an overview of the current development status of our system's hard- and software and we will outline our plans for the near future.

1 Introduction

B-Smart (Bremen Small Multi-Agent Robot Team) is a team of students from the University of Bremen which has regularly participated in the Small Size Robot League at RoboCup competitions since 2003. During the past few months the team members have been replaced completely by new members, because the former team members graduated or will graduate in the near future. Therefore, the team size was reduced from thirteen to six team members which makes developing new features a lot harder but not impossible.

The hardware of our robots has not been changed for about three years now (cf. Sect. 2), because we think that it taps almost its full potential. A better hardware performance could only be achieved by a complete redesign of the robots which is not feasible for us at the moment.

As a consequence, we currently concentrate on improving our artificial intelligence. A general overview of the software system is given in Sect. 3. A first great step in making behavior engineering more efficient and thus our robot team hopefully behaving more intelligent is the development of a new behavior description language called MABSL [1]. It has been designed especially for multi-robot environments, a detailed description can be found in Sect. 4. A second focus in order to increase the team performance is the optimization of behavior parameters via machine learning as described in Sect. 5.

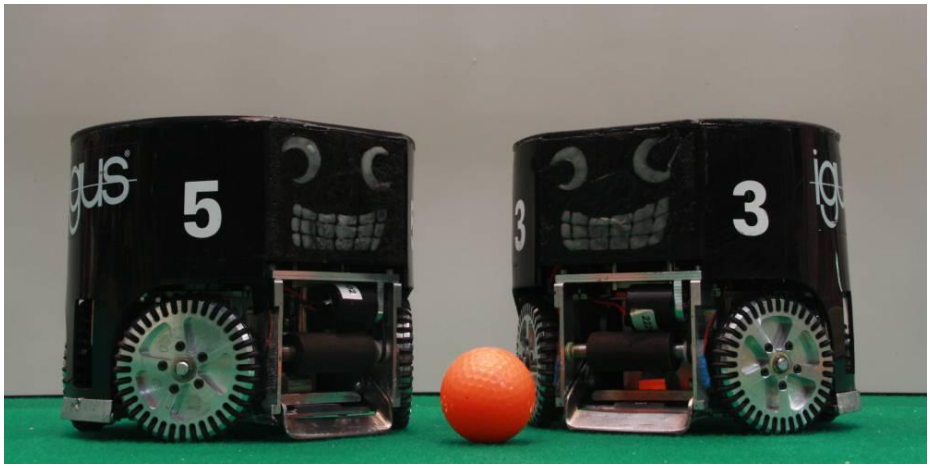

Fig. 1. Current B-Smart robots.

2 Hardware Overview

As mentioned before, our robot's hardware has not changed since about three years. For the sake of completeness, this section briefly summarizes all relevant facts.

2.1 Mechanical design

The main chassis of our robots consists of two laser-cut aluminum plates, which are connected by four motor mounts. On top of the motor area, there is a low tower which holds several circuit boards (cf. Sect. 2.2). On top of the whole inner parts we put a cylindric, black cover made of plastics. The robots have a height of 149 mm and their diameter is 175 mm.

The driving system consists of four wheels each having 36 subwheels. The rear wheels have an angle of 45° to the roll axis and the front wheels 53°. Each wheel is actuated by a *Faulhaber* 2342S006CR DC-Motor, connected with gear wheels in a ratio of $12:1$.

The robot has a low shot kicking mechanism for straight shoots and low power passes. There is also a chip kick for high passes. Above the kicking mechanism, a dribbler is mounted, which is driven by a *Faulhaber* 2224U006SR DC-Motor and connected to the axis of the dribbler by an o-ring. To comply with SSL rules, the dribbling system and the chassis conceal only 18 percent of the ball. Two fully assembled robots are shown in Fig. 1.

2.2 Electronic design

The electronic design consists of four major components: The *Foxboard*, the *Rabbitboard*, the *Motorboard* and the *Kickerboard*. It is constructed in a modular

manner and therefore offers the benefit to detect and handle a damage more quickly.

High-level control and communication is the main purpose of the Foxboard (produced by *http://www.acmesystems.it*). It runs an embedded Linux which has been configured to fulfill the higher latency requirements by providing the simultaneous use of three different communication interfaces (WLAN, Amber Wireless, DECT). Furthermore, the Foxboard can be used as a programmer for updating the Rabbitboard's firmware over Ethernet without the need of an external device.

The Rabbitboard controls the speed of the four motors through a 200 Hz PID control loop and it monitors all important hardware states, especially the light-barrier which is used for accurate detection of the ball in front of the kicking mechanism. The board also controls the dribbler. The core of the board is an Atmega128 μ-Controller which connects four 32-Bit quadrature counter and produces the PWM signals, that are used by the Motorboard for driving four attached motors over H-Bridges *(VNH2SP30)*. These H-Bridges also provide a digital diagnostic feedback signal about the connected motors.

Finally, the Kickerboard controls the chipkick as well as the normal kicking mechanism. It charges one 2200 μF capacitor to 200 V. The two mechanisms are activated by high-power MOSFETs. The force of the kick can be varied from 0 to 8 m/s by the microcontroller which opens the MOSFETs for a designated time.

3 Software Overview

The major component of our software is the so-called *agent*. It controls all robots of our team and sends them motion vectors and other commands. The agent also offers the graphical front-end for robot and behavior management. The program is written in C++ and is divided into three basic module groups: *pre-behavior*, *behavior*, and *post-behavior*.

The *pre-behavior* modules perform calculations based on the percepts received from the official SSL-Vision [2]. This involves e. g. creating the world model to handle removed robots, smoothing out noise from the received data, and predicting future robot positions and speeds. In addition to percepts from the vision, the pre-behavior modules also receive and propagate control information from the referee box to the behavior.

The *behavior* modules calculate the behavior for each robot. This year, we started to specify the behavior using MABSL which allows calculating commands for several robots instead of processing each robot only in its own context (see Sect. 4). This way we can make decisions more efficiently.

The *post-behavior* modules process the results of the behavior modules and make changes where appropriate. This involves collision avoidance, path planning, smoothing the path, and calculating the motion vector. For these tasks, a combination of Rapidly-Exploring Random Trees (RRT) and a reactive system for collision avoidance is used, as presented by [3]. After all calculations are done,

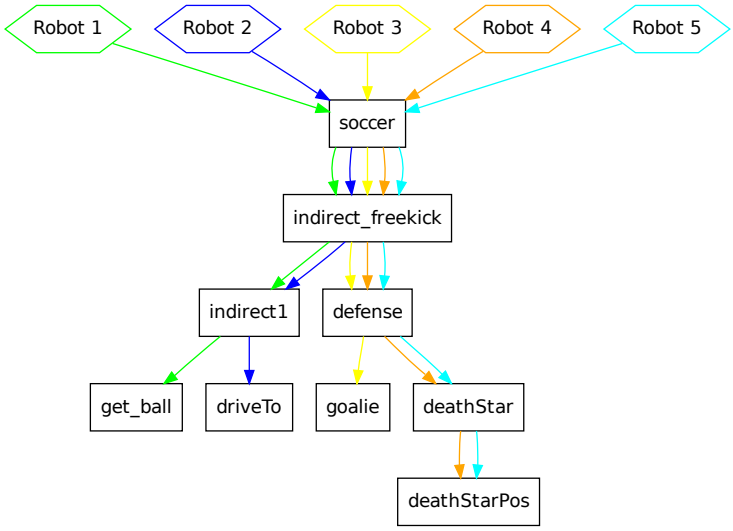

Fig. 2. Output of the MABSL decision tree just before the execution of an indirect free kick.

the commands are queued and sent to the robots through the communication module.

4 The Multi Actor Behavior Specification Language

We have used the *Extensible Agent Behavior Specification Language (XABSL)* [4] for several years now, but this year we switched over to a new behavior specification language called *MABSL (Multi Actor Behavior Specification Language)* which has been developed by [1]. The previous multiagent approach using XABSL, which treated all robots as autonomous instances, complicated a good team play of our robots because a single robot never knew the decision which another robot will reach.

Like XABSL, MABSL is also a system of hierarchical state machines. But every state machine can handle several robots. The behavior tree is specified like a Petri net: The state machines in MABSL are *places* and the edges between two machines are the *transitions* of the Petri net. Due to the fact that a state machine in MABSL can handle several robots, the robots can be splitted up and delivered to different state machines.

Figure 2 shows the output of a decision tree just before an indirect free kick will be executed. At first, all of our five robots are handled by the state machine *soccer* which decides to deliver all the robots to the *indirect_freekick* behavior.

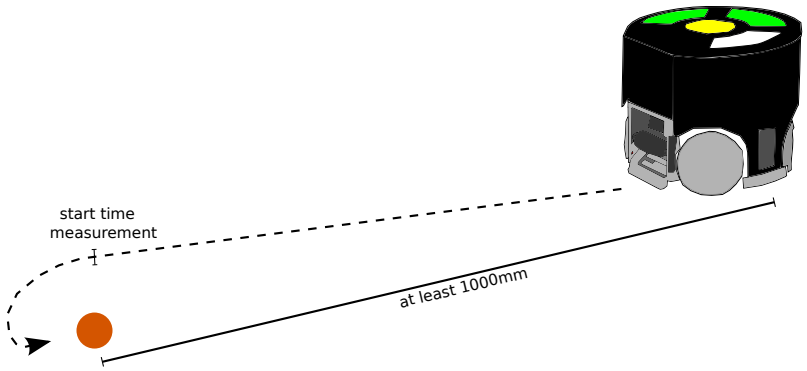

Fig. 3. Optimization of the *rotate around ball* behavior: First, it has to be assured that there is at least 1000mm distance between robot and ball. The robot drives to the position next to the ball and then starts to rotate around the ball. In this state, the time measurements starts and stops after kicking the ball towards the starting position. Combined with the angle between ball position, starting point, and the new position of the ball, the time measurement represents the rating of the trial for the PSO algorithm.

There, the robots are splitted up into two groups. The first group will execute the free kick and the other group is responsible for the defense.

5 Learning Parameters for Robot Behavior Using Particle Swarm Optimization

Using learning methods with real robots is one of the main topics for RoboCup [5]. In the Small Size League, it is fundamental to have fast and robust robot behaviors, since the game play is dynamic and the speed very high. Our concept is to improve parameters for existing behaviors using various learning methods. One of these methods is the Particle Swarm Optimization (PSO) established by Kennedy and Eberhard in [6]. This approach is based on the principles of evolutionary algorithms, however it needs less evaluations to find good positions in previously defined search space. The main idea is to imitate swarm behavior of flocks or herds. Inspired by procedures in the nature, PSO simulates the exchange of information about good positions via communication with other individuals within the swarm.

Our implementation [7] of the PSO algorithm is based on [8]. One example behavior to optimize is the rotation around a ball. Figure 3 shows the procedure.

The goal is to find a set of parameters that reaches the smallest fitness value during optimization. To avoid unreliable fitness values caused by stochastic noise, each set of parameters becomes evaluated three times. To achieve a robust final result, the trial with the worst value represents the fitness of the particle.

The optimization process, which involved 30 iterations and 20 particles, took about five hours. Compared to the set of parameters used so far, the new one is more robust. The old set had a failure rate of nearly 17%. A failure means, that the robot was not able to get control over the ball and just pushed it away instead of kicking it. The new set causes no failures and is about 420ms faster.

References

1. Fritsch, S.: Zentrale Verhaltenssteuerung für kooperierende Robotergruppen. Diploma thesis under examination. University of Bremen (estimated 2010)
2. Zickler, S., Laue, T., Birbach, O., Wongphati, M., Veloso, M.: SSL-vision: The shared vision system for the RoboCup Small Size League. In Baltes, J., Lagoudakis, M.G., Naruse, T., Shiry, S., eds.: RoboCup 2009: Robot Soccer World Cup XIII. Volume 5949 of Lecture Notes in Artificial Intelligence., Springer (2010)
3. Bruce, J.: Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments. PhD thesis, Carnegie Mellon University (2006)
4. Lötzsch, M., Risler, M., Jüngel, M.: XABSL - A Pragmatic Approach to Behavior Engineering. In: Proceedings of 2006 IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS), Beijing, China (2006) 5124–5129
5. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: Robocup: The robot world cup initiative. In: AGENTS '97: Proceedings of the first international conference on Autonomous agents, New York, NY, USA, ACM Press (1997) 340–347
6. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on. Volume 4. (1995) 1942–1948
7. Huhn, K.: Lernmethoden zur Parameteroptimierung von Roboterverhalten. Diploma thesis under examination. University of Bremen (2010)
8. Clerc, M.: Particle swarm optimization. ISTE, London (2006)