B-Smart

(Bremen Small Multi-Agent Robot Team)

Team Description for RoboCup 2009

Tim Laue[1], Armin Burchardt[2], Sebastian Fritsch[2], Nils Göde[2], Kamil Huhn[2], Teodosiy Kirilov[2], Eyvaz Lyatif[2], Markus Miezal[2], Markus Modzelewski[2], Ulfert Nehmiz[2], Malte Schwarting[2], Andreas Seekircher[2], Ruben Stein[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-bsmarter@informatik.uni-bremen.de

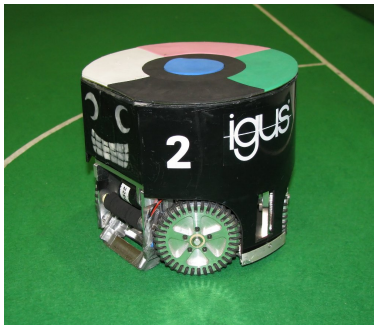
Abstract. This paper documents the current technical state of B-Smart's hard- and software as well as upcoming changes in the near future. For giving an overview, existing systems and their principles will be outlined briefly. Most texts concentrate on planned or already completed modernizations which have been carried out since spring 2008.

1 Introduction

B-Smart is a project for computer science students in the advanced study period at the Universität Bremen. The team is competing in the RoboCup Small Size League, a technically simplified but fast and tactically challenging version of robot football. After the RoboCup world championship in 2006, the project was relaunched for another two years as a sequel to several similarly named projects. Some of the team members decided after the projects official end (in fall 2008) to continue developing and maintaining the hard- and software to participate in the RoboCup 2009 in Graz. This would be the seventh participation in a row as B-Smart has taken part in RoboCup competitions since 2003.

After some years of continuous redesigning, we have not significantly changed the mechanical (cf. Sect. 2.1) as well as the electronic (cf. Sect. 2.2) design of our robots during the last season. Our current platform has been considered as being good enough for our purposes.

Nevertheless, our software has undergone several changes. The general architecture still consists of two main components: The *Vision*, which generates the world model from image data, and the *Agent*, which makes decisions for robot movement on the basis of this world model. Both applications consists of several sub-components of which the most important ones are described in Sect. 3 and Sect. 4.

a) 

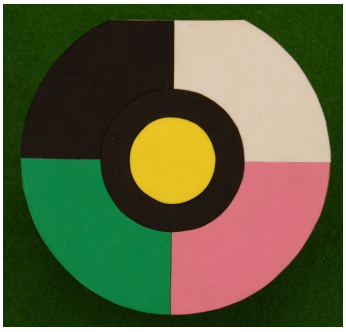
b) 

Fig. 1. a) Current B-Smart robot b) Cover of B-Smart robots

2 Hardware Architecture

2.1 Mechanical Design

The main chassis consists of two laser-cut aluminum plates, which are connected by four motor mounts. The robots have a diameter of $177mm$, their height is $144mm$. The driving system consists of four wheels each having 36 subwheels. The back 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 hard shoots and low power passes. There is also a chipkick 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.

A fully assembled robot is shown in Fig. 1a.

2.2 Electronic Design

The electronic design consists of four major components: The *Foxboard*, *Rabbitboard*, *Motorboard*, and *Kickerboard*. This modular design is an advantage when fast diagnostic of error-prone parts is needed. Furthermore, it offers the ability of replacing damaged components with spare ones, so it is more likely to have a broken robot repaired in a short period of time.

High-level control and communication is the main purpose of the *Foxboard*. It runs an embedded Linux (http://www.acmesystems.it), which has been customized to fulfill the higher latency requirements by the simultaneous use of the three communication interfaces. Furthermore it can be used as a programmer for updating the *Rabbitboard*'s firmware without disassembling the robot.

The *Rabbitboard* controls the speed of the four motors with a 200Hz PID control loop. Furthermore, it monitors all important hardware states, especially

the light-barrier which is used for accurate detection if the ball is ready to be shot. The Atmega128 suits these needs well and is part of the board. The *Motorboard* is used for driving and monitoring the four attached motors via H-Bridges (VNH2SP30).

The *Kickerboard* is directly controlled by the Atmega128 on the Rabbit-board. It charges one $2200\mu F$ capacitors to 200V. The chipkick and the normal kicking-mechanism are activated by two high-power mosfets. The force of the kick can therefore be varied from 0 to 8m/s by the microcontroller which opens the mosfets for a certain time.

3 Vision

As almost every team in SSL does, a global vision system is used. Pictures from two *AVT Marlin F046C* FireWire cameras are combined and pushed through a pipeline of processing modules in order to provide an internal world model to the other software components.

3.1 Image Processing

One of the features of our vision system is the flexible module for the computation of the camera perspective. Theoretically, it is possible to place cameras on lower angles than the standard 90 degrees, still being able to recognize all objects.

Our system also identifies regions of interest. The basic idea is to recognize areas in the camera picture where the the robots or the ball could be. The combination of three methods (information about previously detected blobs, difference of two pictures, and detection of contrast regions) provides a stable identification of regions where possible objects can be recognized later. This process reduces the amount of false blob detections and minimizes data needed for follow-up calculations.

We use a HSV based color segmentation approach for detecting robot blobs and the ball. The HSV color model provides a more stable color classification when used in environments with changing luminosity (e.g. dynamic highlights and shadows on the field). This is due to the fact that the hue component of the physical object color stays relatively constant when the light intensity falling on it changes. To perform color segmentation, a Fast Fourier Transformation of the picture is used to set thresholds in each color class and to build a look-up table used later for real-time color classification.

Our circular cover (cf. Fig. 1b) consists of four colored areas of the same size. The order of the colors (black, white, magenta, green) encodes the ID, whereas the black field is located left of the front to indicate the robot's orientation. Furthermore, to ensure a reliable detection of the marker's center, a black ring separates the four fields from the center blob.

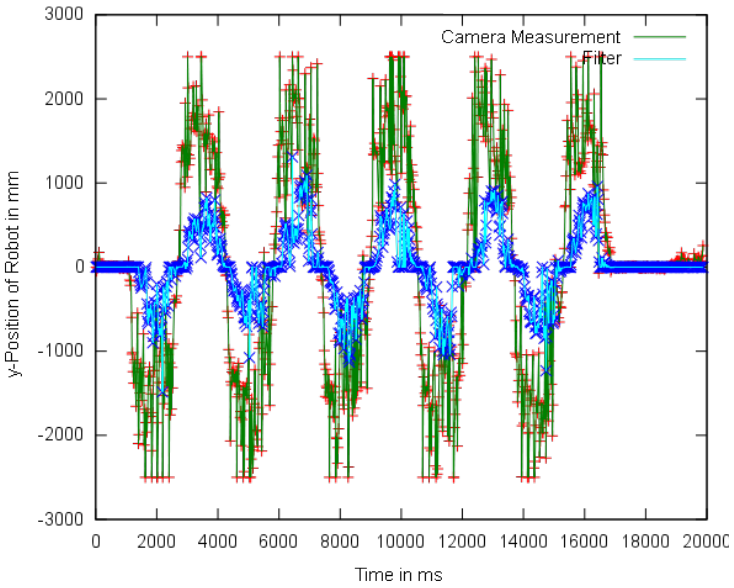


Fig. 2. Comparison of direct camera measurement and filtered value while performing sinus-shaped movement in y-direction

3.2 Smoothing of velocity measurements with particle filters

In previous years, the velocity of the ball and the robots was measured by plain differences of positions in a certain time. This is sufficient to gain a rough estimate but leads to problems when algorithms like path finding or behavior routines depend on this value. Reasons for this inaccuracy can be found in the noise of the compressed image data with small resolution and changing lighting conditions. Moreover, these effects are amplified by slight miscalculations or improper software configurations. Changes of only a few pixels in position can cause big noise on the velocity value.

To reduce the above mentioned effects, a compensation algorithm was implemented. The idea was to feed one filter for each object to track with continuous position data. Based on those measurements and precise time stamps on each picture, the algorithm is able to estimate the position of the moving object. Of central importance for this algorithm are the multiple instances of the object which are tracked over time. Each one is initialized with the measured position and added random gaussian noise to simulate the uncertainty of the camera measurement. After this, a velocity vector is determined by pure random noise. Now the second measurement is awaited and every virtual state gets moved as if his position and velocity were true for the passed amount of time. The resulting state is now compared with the real measured one and rated accordingly.

Some cycles later, the virtual states by this converge towards the real state and smooth its changes as one outlier is not able to invalidate the whole set of virtual states. Nevertheless, this convergence also tends to result in problems with high dynamic environments where spontaneous changes in the estimated value, velocity in this case, are likely to happen. Unfortunately, exactly this is what happens if a small size robot just stops in a few milliseconds due to strong motors. To solve this, approaches such as particle injection are utilised. Thereby, in every step some particles are generated from direct camera measurement or hypothetical engine stops. This grants the filter the ability to react faster on sudden changes while estimating not too many states and keep good performance [1].

The analysis of the implemented filter revealed that there is a high potential for smoothing the ugly measurements used until now (cf. Fig. 2). However there are certain drawbacks which might get solved by further refinement of the implementation. This involves e.g. a delayed estimation of up to two frames produced by the converging principle of this method. It might be possible to overcome this by adding additional measurements of motor data and artificially increase the amount of passed time during the estimation of new states. Furthermore, the filter needs many parameters e.g. for noise values or certain thresholds. Resulting from this, the overall filter performance may vary heavily.

4 Agent

The Agent provides the artificial intelligence part of our software system. It receives an abstract representation of the world (world model) and calculates motion vectors for each robot of the own team. In order to aid debugging, the application is divided into three basic module groups: *pre-behavior, behavior and post-behavior* (Fig. 3).

The *pre-behavior* modules execute calculations based on the world model and valid for every agent.They may also modify the world model e.g in order to handle removed or defected robots and avoid their inclusion in the strategy calculations.

The *behavior* modules calculate the desired move for each robot, based on the data provided by the world model and the pre-behavior modules. The used approach is based on *XABSL*, which is described shortly in the next section. A new addition are subsidiary modules which try to rate the strategy and improve the decisions taken (see sect. 4.4, 4.5).

The third group, the *post-behavior* modules, processes the results of the behavior execution and converts them to robot commands. The processing is done per robot and includes control, path-planning and obstacle avoidance.

The last step is to queue the robot commands for sending through the *communication module*

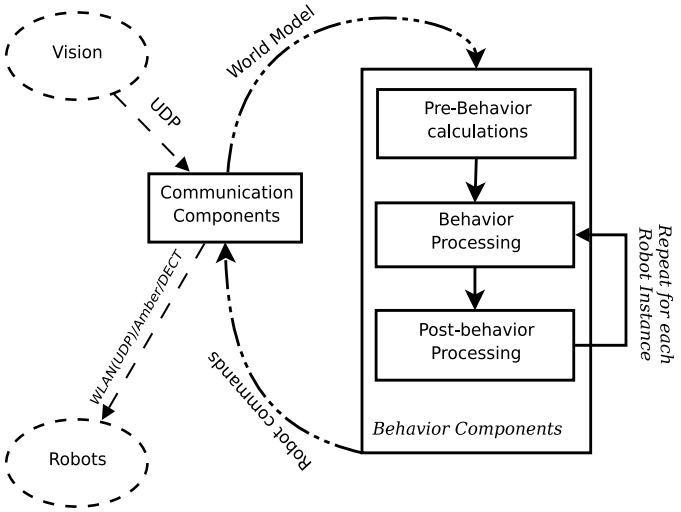


Fig. 3. Main loop structure of the agent software

4.1 Behavior Control and XABSL

The *Extensible Agent Behavior Specification Language (XABSL)* [2] is an XML-based behavior description language which can be used to describe behaviors of autonomous agents. It simplifies the process of specifying complex behaviors and supports the design of both very reactive and long term oriented behaviors. It uses hierarchies of behavior modules that contain state machines for decision making. *XABSL* has been successfully applied in the Four-Legged League by the *GermanTeam* [3].

Behaviors are specified in a C-like programming language and transformed to an intermediate code which is interpreted by the *XABSL* engine. This shortens the system's compile time, allows more effective behavior development, and becomes very handy when changes have to be applied in the halftime of a running game.

After traversion of the XABSL trees a destination position is available to each agent.

4.2 Path-Planning

The path-planning system is based on the "Real-Time Randomized Path Planning for Robot Navigation" approach by Bruce and Veloso [4]. This approach has been used several years by the team CMDragons. The system is a combination of path-planning with RRT (Rapidly-Exploring Random Trees) and a reactive system for collision avoidance called DSS (Dynamic Safety Search, [5]).

The path-planning system has to find a free path from a given start position to a goal position. In every cycle the RRT algorithm builds a tree in the state

space beginning with a node at the start position. The tree is extended by adding new nodes towards different target positions. Only with a small probability the tree is extended directly towards the goal position. Mostly the target position is a randomly chosen position on the field. Once a free path has been found the nodes on this path are saved as waypoints (see Execution Extended RRT [4]). These waypoints can be chosen in the next cycle as target positions. Because of the short cycle time, most parts of a previously found path can be reused to increase the replanning efficiency. This algorithm does not search for the optimal solution. Due to the high dynamic environment its preferable to find some free path in a short time, than searching a long time to find the best path.

The state space used for the RRT consists only of two dimensional field positions. Robot dynamics like speed or acceleration constraints are disregarded. This would lead to a much larger state space and the pathplanning would not be fast enough. This means that its not always possible to follow a path given by the RRT with maximum speed. Nevertheless the desired acceleration is set to the maximum acceleration in the direction given by the path. After these accelerations are calculated for all robots the second part of the path-planning system is used to avoid collisions.

The desired accelerations are checked by the Dynamic Safety Search described in [5]. For every robot a trajectory is calculated. In the first segment of the trajectory the given acceleration is executed for one cycle, after this the second segment executes the maximum deceleration to stop the robot. The DSS checks these trajectories for collisions and changes unsafe accelerations. If no safe acceleration can be found, it is at least possible to stop the robot without collisions.

4.3 Positioning

Every robot chooses its positioning behavior depending on the assigned role. Whereas striker and defender act similarly when in possession of the ball, they have different tasks when they are not. The striker without ball has a prioritized list of actions to choose from. If the ball is not already moving towards the robot, it tries to get to a free point preferring a position that has a free line between itself and the goal / the ball. To find the best position on the field, a grid of positions (cf. Fig. 4) of which every single one is rated according to a set of parameters is used. As it turned out to be very expensive to recalculate all positions in every cycle, a two stage system was implemented: First, a wider grid is calculated from which a well rated area of positions is taken, then the positions in the chosen area are rated more precisely. If no position matching the criteria can be found, the striker positions diagonally behind the robot possessing the ball. This leads to a dramatically decreased number of turnovers in duels.

4.4 Coach

For every offensive standard situation, more than one tactical behavior is prepared, which the robots are able to perform. The *Coach* is trying to get the best

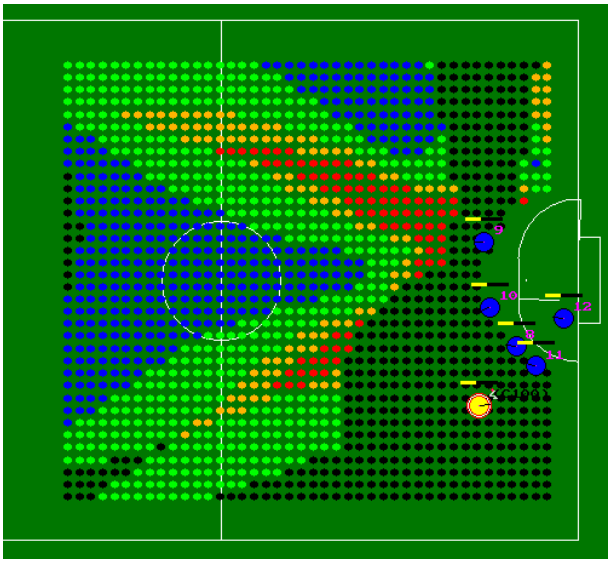


Fig. 4. Visualization of the yellow team's position rating grid, showing good positions in red.

behavior suitable against the current opponent team. It rates the previously executed behaviors and chooses the best rated. The results are improving, the more situations the *Coach* evaluates.

4.5 Coordinator

To optimize the decisions made by *XABSL*, the Coordinator system can overwrite and/or redistribute the calculated output symbols, if it detects a possible improvement resulting from the advantage of knowing which decisions have been made for every single robot. The Coordinator also replaces some parts of the *XABSL* behavior for some situations that benefit from a centrally managed control.

4.6 Communication

The system uses UDP multicast sockets for communication between wired system components, e.g. the Vision and the Agent application. This allows to start programs on different computers without changing parameters and to work in small groups in the same physical net without influencing each other. The world model and the robot commands which are communicated, can be captured and analyzed.

Regarding the wireless communication between the Agent and the robots, the connection quality of a particular medium depends on the conditions on the

venue, so our wireless communication is able to use more than one transport medium.

A modular communication interface has been installed so different modules can be added and used simultaneously. Currently we can use WLAN IEEE 802.11b, Amber Wireless (868 MHz radio) and DECT for communication with the robots. The same packets are sent via every media and a sequence number in the packet header ensures that old packets will be dropped. If one medium fails, the robot will instantly use another medium.

Feedback containing information about battery charge and other status information that may affect behavior and other components is sent from the robots. This allows a dynamical adjustment of formation (e. g. a robot without working kick should not be striker) during the game and can be used to calibrate PID values.

B-Smart team members

Sebastian Fritsch, Sven Hinz, Kamil Huhn, Teodosiy Kirilov, *Tim Laue*, Eyvaz Lyatif, Marc Michael, Markus Miezal, Markus Modzelewski, Alexander Martens, Ulfert Nehmiz, *Thomas Röfer*, Malte Schwarting, Andreas Seekircher

References

1. Röfer, T., Laue, T., Thomas, D.: Particle-filter-based self-localization using landmarks and directed lines. In: In RoboCup 2005: Robot Soccer World Cup IX, Lecture Notes in Artificial Intelligence, Springer (2005)
2. Lötzsch, M., Risler, M., Jüngel, M.: XABSL - A Pragmatic Approach to Behavior Engineering. In: Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS), Beijing, China (2006) 5124–5129
3. Röfer, T., Brunn, R., Czarnetzki, S., Dassler, M., Hebbel, M., Jüngel, M., Kerkhof, T., Nistico, W., Oberlies, T., Rohde, C., Spranger, M., Zarges, C.: GermanTeam 2005. In: RoboCup 2005: Robot Soccer World Cup IX. Lecture Notes in Artificial Intelligence (2005)
4. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: Proceedings of IROS-2002, Computer Science Department (2002)
5. Bruce, J.: Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments. PhD thesis, Carnegie Mellon University (2006)