GERMANTEAM

RoboCup 2001

Hans-Dieter Burkhard
Uwe Düffert
Matthias Jüngel
Martin Lötzsch

Nils Koschmieder
Tim Laue
Thomas Röfer
Kai Spiess
Andreas Sztybryc

Ronnie Brunn
Max Risler
Oskar von Stryk

Institut für Informatik,
LFG Künstliche Intelligenz,
Humboldt-Universität
zu Berlin,
Rudower Chaussee 25,
12489 Berlin, Germany

Bremer Institut für
Sichere Systeme, TZI,
FB 3 Informatik,
Universität Bremen,
Postfach 330440,
28334 Bremen, Germany

Fachgebiet Simulation
und Systemoptimierung,
FB 20 Informatik,
Technische Universität Darmstadt,
Alexanderstr. 10,
64283 Darmstadt, Germany

Abstract

The GermanTeam is a joint project of several German universities in the Sony Legged Robot League. This report describes the software developed for the RoboCup 2001 in Seattle. It presents the software architecture of the system as well as the methods that were developed to tackle the problems of motion, image processing, object recognition, self-localization, and robot behavior. The approaches for both playing robot soccer and mastering the challenges are presented. In addition to the software actually running on the robots, this document will also give an overview of the tools the GermanTeam used to support the development process.

Contents

1 Introduction **5**
1.1 History . . . 5
1.2 Scientific Goals . . . 5
1.2.1 Humboldt-Universität zu Berlin . . . 5
1.2.2 Universität Bremen . . . 6
1.2.3 Technische Universität Darmstadt . . . 6
1.3 Contributing Team Members . . . 7
1.3.1 Humboldt-Universität zu Berlin . . . 7
1.3.2 Universität Bremen . . . 7
1.3.3 Technische Universität Darmstadt . . . 7

2 System Architecture **7**
2.1 System Dependent Processes . . . 8
2.1.1 HSensor . . . 8
2.1.2 HMotion . . . 9
2.2 System Independent Processes . . . 9
2.2.1 HIntegra . . . 9
2.2.2 HControl . . . 9
2.3 Service Processes . . . 9
2.3.1 HArbiter . . . 9
2.4 Debugging Support . . . 9
2.4.1 HTerm . . . 9
2.4.2 HLogFile . . . 10
2.5 DogControl . . . 10
2.5.1 Log Files . . . 11
2.5.2 Debug Keys . . . 11
2.5.3 Run System Independent Processes . . . 11
2.5.4 Dialog Interface . . . 11

3 Motion **11**
3.1 Motion Net . . . 12
3.2 Motion Description Language . . . 12
3.3 Walking . . . 13
3.3.1 Parameters . . . 14
3.3.2 Inverse Kinematics . . . 14
3.4 Odometry . . . 14
3.5 Adaptive Head Control . . . 14
3.5.1 Robot Motion Mode . . . 15
3.5.2 Ball Tracking Mode . . . 15
3.5.3 Landmarks Scanning Mode . . . 15
3.5.4 Automatic Mode . . . 15

4 Image Processing **15**
4.1 Image Segmentation . . . 15
4.2 Creating Color Tables . . . 16
4.2.1 YUV Tool . . . 16
4.2.2 HSI Tool . . . 16
4.3 Detection of Blobs . . . 17
4.4 Recognition of the Ball . . . 18
4.4.1 Blob Filtering . . . 18
4.4.2 Center Point and Diameter . . . 19
4.4.3 Intersection of Middle Perpendiculars . . . 19
4.4.4 Selecting Points . . . 20
4.5 Recognition of Team-Mates and Opponents . . . 20
4.5.1 Blob Clustering . . . 20
4.5.2 Calculating Angle and Distance . . . 21
4.5.3 Indications of Other Robots . . . 21
4.5.4 Calculating Positions of Other Robots . . . 21
4.6 Recognition of Flags and Goals . . . 21
4.6.1 Compensating for Head Rotations . . . 21
4.6.2 Spatial Constraints . . . 22
4.6.3 Recognition of Flags . . . 22
4.6.4 Recognition of Goals . . . 22

5 Self-Localization **23**
5.1 Localization Based on a Representation of the Environment . . . 23
5.1.1 Representation of the Environment . . . 23
5.1.2 Updating the Representation of the Environment . . . 23
5.1.3 Calculating the Position . . . 24
5.2 Monte-Carlo Localization . . . 24
5.2.1 Motion Model . . . 25
5.2.2 Observation Model . . . 25
5.2.3 Resampling . . . 27
5.2.4 Estimating the Pose of the Robot . . . 27
5.2.5 Results . . . 29

6 Behavior **29**
6.1 Architecture . . . 30
6.1.1 Last Year's Approach . . . 30
6.1.2 Simple State Machines . . . 30
6.1.3 Multi Layer State Machines . . . 31
6.1.4 Problems of the Architecture . . . 33
6.2 Debugging Tools for Behavior . . . 34
6.2.1 BeliefViewer . . . 34
6.2.2 BehaviorTester . . . 35

6.3 Implemented Behavior 35
6.3.1 Contest 35
6.3.2 Challenge 37

7 Conclusions **38**

8 Future Work **39**
8.1 Revising the Existing System 39
8.2 Self-Localization without Global Landmarks 39
8.3 Probabilistic Modeling 39
8.4 Communication 40
8.5 Cooperation 40
8.6 Formalization 40
8.7 Kinetic Modeling 40

1 Introduction

1.1 History

The GermanTeam is the successor of the Humboldt Heroes who already participated in the Sony Legged League competitions in 1999 and 2000. Because of the strong interest of other German universities, in March 2001, the GermanTeam was founded. It consists of students and researchers of five universities: Humboldt-Universität zu Berlin, Universität Bremen, Technische Universität Darmstadt, Universität Dortmund and Freie Universität Berlin. However, for the system presented in this document, the Humboldt Heroes only had reinforcements from Bremen and Darmstadt. The two other universities will actively participate with the beginning of the winter semester.

Based upon the results of the Humboldt Heroes, the software for RoboCup 2001 and the German Open 2001 was developed. After the RoboCup 2000 in Melbourne the team from the Humboldt-Universität decided to re-implement everything except their motion architecture, which kept nearly unchanged from the year before. In spring 2001 the cooperation with the Universität Bremen and the Technische Universität Darmstadt started. As the group from Darmstadt got its robots at the beginning of May and the team members from Bremen received their machines at the end of May, the GermanTeam had only two to three months to integrate the newcomers.[1] Therefore, much of the remaining time till the beginning of the RoboCup event had to be spent for learning and administrative purposes. The new members had to understand the existing software, the tools, and the functions of the robot systems. The original team members had to explain their system, had to organize workshops, and had to lead the development.

1.2 Scientific Goals

All the universities participating have certain special research interests, which they try to carry out in GermanTeam's software.

1.2.1 Humboldt-Universität zu Berlin

Due to their special interests in agent control and learning, the Humboldt University's main research interests are both robotic architectures for autonomous robots based on mental models and the development of complex behavior control architectures.

Even simple environments and tasks for autonomous robots require sophisticated robot architectures that satisfy real-time conditions. Such designs have to manage the sensor readings, the actuator control, the representation of internal and external objects, the planning system, and the needs of debugging and visualization—leading to a robust and extendable software model that can be applied to different robotic platforms. It is important to define useful representations such as interfaces for the data exchange between interchangeable modules. It is not avoidable to implement general and easy-to-use techniques for debugging and visualization. Continuing the last year's work the group will pay much attention on that topic.

[1] Without the robots and without a signed NDA, productive work was not possible earlier.

For robots with complex tasks in natural environments, it is necessary to equip them with behavior control architectures that allow planning based on fuzzy and incomplete information about the environment, cooperation without communication, and actions directed to different goals. These architectures have to integrate both long-term plans and short-term reactive behaviors. They have to protect themselves against the fanatic following of a plan selected once, and against frequent changes between their behaviors. Architectures should be easy to expand. The group dealt with that topic for some years and developed several approaches, as, e. g., the use of *Case Based Reasoning*. A current subject of the group's research are *multi layer state machines*. Carrying on that approach, next year, the group wants to formalize the implemented behaviors in a high level language as, e. g.., XML.

1.2.2 Universität Bremen

The main research interest of the group is the automatic diagnosis of the strategies of other agents, in particular, of the opposing team in RoboCup. A new challenge in the development of autonomous physical agents is to model the environment in an adequate way. In this context, modeling of other active entities is of crucial importance. It has to be examined, how actions of other mobile agents can be identified and classified. Their behavior patterns and tactics should be detected from generic actions and action sequences. From these patterns, future actions should be predicted, and thus it is possible to select adequate reactions to the activities of the opponents. Within this scenario, the other physical agents should not to be regarded individually. Rather it should be assumed that they form a self-organizing group with a common goal, which contradicts the agent's own target. In consequence, an action of the group of other agents is also a threat against the own plans, goals, and preferred actions, and must be considered accordingly. Acting under the consideration of the actions of others presupposes a high degree of adaptability and the capability to learn from previous situations. Thus these research areas should also be emphasized in the project.

The research project focuses on strategy detection of agents in general. However, the RoboCup is an ideal test-bed for the methods to be developed. The technology of strategy recognition is of large interest in two research areas: on the one hand, the quality to forecast the actions of physical agents can be increased, which plays an important role in the context of controlling autonomous robots, on the other hand, it can be employed to increase the robustness and security of electronic markets. This project idea is also a proposal for participation in the priority program "Cooperating teams of mobile robots in dynamic environments" funded by the Deutsche Forschungsgemeinschaft (German Research Foundation).

During the two months of preparation time for this year's competition, the group from Bremen was only able to build the foundation for these goals, i. e., a robust self-localization and the recognition of other robots were implemented.

1.2.3 Technische Universität Darmstadt

The primary goal of the team at Technische Universität Darmstadt in the roughly three months from obtaining the robots until the RoboCup in Seattle was getting started with the hard-/software

provided by Sony, the existing tools and software mainly from the team at Humboldt University, and setting-up a collaboration between distributed groups.

The long-term goals of the team in Darmstadt are conceptual and algorithmic contributions to all sub-problems involved for a successful autonomous team of soccer playing legged robots (perception, localization, locomotion, behavior control). Furthermore, the group in Darmstadt is developing tools for kinetic modeling of the robot dynamics taking into account masses and inertias of each robot link as well as motor, gear and controller models of each controlled joint. Based on these nonlinear dynamic models computational methods for dynamic off-line optimization and on-line stabilization and control of dynamic walking and running gaits are developed and applied. These methods will be applied to develop and implement new, fast, and stable locomotions for the Sony four-legged robots. We hope to present first results at RoboCup 2002.

1.3 Contributing Team Members

At the three universities, many people contributed to the GermanTeam:

1.3.1 Humboldt-Universität zu Berlin

Graduate Students. Uwe Düffert, Matthias Jüngel, Martin Lötzsch.

Scientific Staff. Hans-Dieter Burkhard.

1.3.2 Universität Bremen

Graduate Students. David Bernau, Stefan Hebeler, Denny Koene, Jürgen Köhler, Nils Koschmieder, Lang Lang, Tim Laue, Lucas Mbiwe, Kai Spiess, Andreas Sztybryc, Hong Xiang.

Scientific Staff. Bernd Krieg-Brückner, Thomas Röfer, Ubbo Visser.

1.3.3 Technische Universität Darmstadt

Graduate Students. Ronnie Brunn, Martin Kallnik, Michael Kunz, Nicolai Kuntze, Sebastian Petters, Max Risler, Michael Wolf.

Scientific Staff. Markus Glocker, Michael Hardt, Oskar von Stryk.

2 System Architecture

The software of the GermanTeam is based on a blackboard architecture. The system consists of several processes that run in parallel, and that communicate via a *shared memory* that contains all the data structures that must be shared between the components of the system. During the

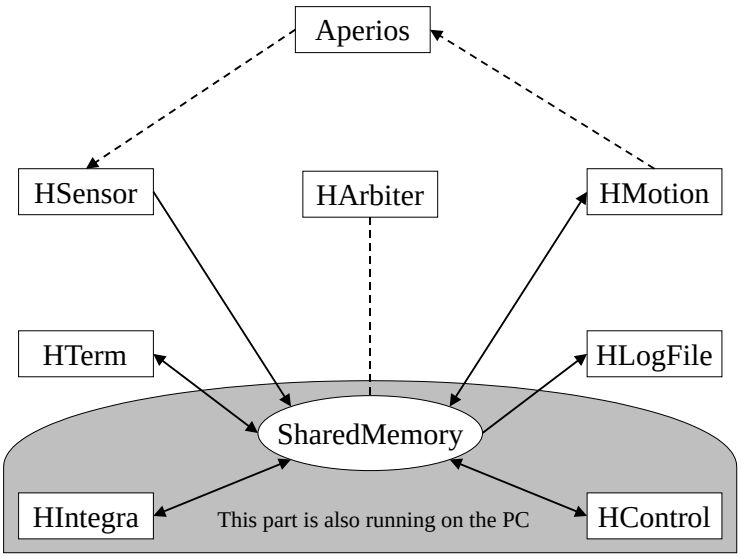

Figure 1: System architecture. Dotted arrows represent Aperios-queues. Solid arrows stand for shared memory accesses. The dotted line indicates that HArbiter is the creator and the owner of the shared memory.

development, these processes can be distributed between the robot and a PC, both linked by a serial cable. On both platforms, shared memory areas are used to store the information. As the serial link cable only provides a very small bandwidth, data is only exchanged between both shared memory areas on demand. On the robots, the processes run under Aperios, on the PC, the program *DogControl* that runs under Microsoft Windows builds the framework for debugging (cf. Sect. 2.5). DogControl contains nearly all the tools that support the development, and that are required for maintenance, e. g. to create color tables (cf. Sect. 4.2). Both the code running on the robots and the debugging tools that run under Microsoft Windows were developed using Microsoft Visual C++.

The system on the robot consists of several processes (cf. Fig. 1), i. e. Aperios-objects, that run in parallel. The processes fall in four categories: *system dependent processes*, *system independent processes*, *service processes*, and *debugging support*:

2.1 System Dependent Processes

2.1.1 HSensor

This process receives the sensor data from the robot and writes it into the shared memory. Only two kinds of data are obtained: images from the CCD camera and the states of all joints and other sensors of the robot. The camera image is stored together with the assumed pose of the robot's camera and the distance measurement of the PSD sensor.

2.1.2 HMotion

HMotion generates all motions and sends them to the robot. In addition, it also controls the LEDs in the robot's head. The functionality of HMotion is described in Chapter 3.

2.2 System Independent Processes

System independent processes can run both on the robots and on the PC as parts of DogControl. If they are running under DogControl, all normal methods of debugging such as setting breakpoints, etc. can be applied.

2.2.1 HIntegra

The task of this process is to establish a world model. It implements methods for image processing, head motion control, self-localization, and object recognition. The implementation of HIntegra is covered in the Chapters 4 and 5.

2.2.2 HControl

This class contains the behavior model. It controls which action has to be taken in which situation. For this, HControl employs the world model provided by HIntegra that contains information about the state of the robot and its environment. HControl decides which motions to perform and sends them to HMotion (via the shared memory). Chapter 6 will discuss the behavior architecture in detail.

2.3 Service Processes

2.3.1 HArbiter

HArbiter is the only service process in the system. Its task is to create the shared memory region, and to provide its address to the other processes. While the shared memory is created, several of its entries are filled by information loaded from files, e. g. the color table.

2.4 Debugging Support

Debugging support processes are only required during the development, they are unnecessary while playing soccer in the contest. Preprocessor symbols decide whether the system is built as a *debug version* or as a *release version*. In case of a release build, the debug processes are obsolete.

2.4.1 HTerm

This class implements the communication of the robots with DogControl on the PC. In the shared memory, there is a special queue that can be used by all processes to send information to DogControl. This queue is read by HTerm that performs the actual communication via the serial link.

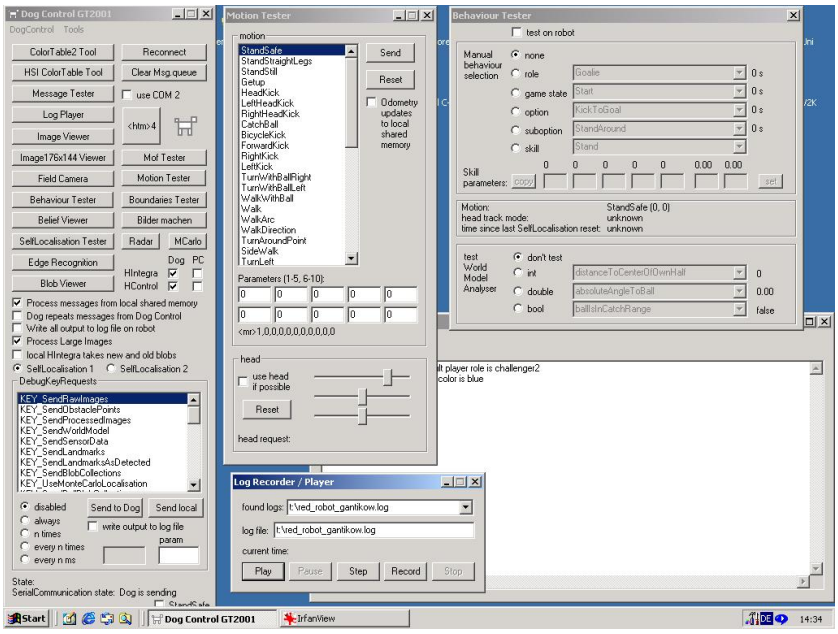

Figure 2: DogControl in action. The main window on the left side contains start buttons for several tools and control elements for debugging. The other windows are tools for motion and behavior testing, a terminal for debugging messages and our log player.

Any kind of information can be transferred by HTerm, but for some types of data, it will take quite long, because the bandwidth is limited to approximately 2 KB per second.

2.4.2 HLogFile

In the shared memory, there is a second queue that is read by HLogFile, and then the acquired data is stored in a log file on the Memory Stick. As the robot stops all operations when writing to the Memory Stick, HLogFile only writes data after a certain combination of buttons was pressed on the robot. The log files can be read by DogControl to, e. g., use images to create a color table. HLogFile is used for huge amounts of data, e. g. for sequences of images, because it costs too much time to transfer them via the serial link.

2.5 DogControl

DogControl integrates nearly all the tools the GermanTeam used to support the development[2]. It runs under Microsoft Windows, and it is written in C++. It provides the possibility to visualize nearly everything that is required, and it can remotely control the robot via the serial connection.

The most important features of DogControl are:

[2]The only other tool is the Motion Config Tool described in section 3.2.

2.5.1 Log Files

DogControl is able to record and play back log files. That way, the information necessary for a certain development step can be acquired from the robot, recorded, and then it can be replayed as many times as required, even without using the robot. In addition, the log files recorded by HLogFile can also be replayed.

2.5.2 Debug Keys

Debug keys are used to control the information sent via the serial link, and to influence the robot program at run-time. They are accessible from a list in the main dialog of DogControl. They cannot only be switched on or off, it can also be specified, how often or in which intervals they are active. For instance, a debug key can be configured in a way that the robot sends a world model every 500 ms, while a second debug key can state that ten blob collections (cf. Sect. 4.3) should be sent in sequence.

2.5.3 Run System Independent Processes

The system independent processes (cf. Sect. 2.1) of the robot control software can directly run as a part of DogControl. This is especially useful in conjunction with the replay of log files, as it allows to develop parts of the software without using a robot.

2.5.4 Dialog Interface

DogControl provides an interface that allows to easily add own dialogs. Many dialogs instantiate this interface for several purposes: creating color tables (cf. Sect. 4.2), viewing camera images, blob collections (cf. Sect. 4.3), and recognized objects (cf. Sect. 4.5) and landmarks (cf. Sect. 4.6), displaying the Monte-Carlo distribution (cf. Sect. 5.2), and testing motions (cf. Chapter 3) and behaviors (cf. Sect. 6.2), etc.

3 Motion

In GT2001, the object HMotion is the only object which has an outgoing communication queue to the robot. Therefore, it is responsible for generating all joint data for the legs and for the head to control the movements of the robot. It also produces the data sent to the LEDs which are used to display various information, e. g. behavior states.

HMotion processes and executes motion requests, head motion requests, and LED requests from the shared memory, which are generated by HControl and HIntegra. HMotion is triggered by a timer. To be able to react to new requests instantaneously, HMotion is the object with the smallest cycle time.

3.1 Motion Net

To control the motions of the robots, we implemented a motion net similar to Open-R's OMoNet. The motion net manages the transitions from the current motion to any other requested motion. This allows for combining all available motions, and it makes the development of new motions easy.

We use our own implementation to have full control over the execution of transitions, and to be able to adjust the motion net to our needs.

3.2 Motion Description Language

The specification for one single motion consists of the description of the desired action and the definition of a set of transitions to all other motions. This is simplified by using groups of motions e. g. it is possible to define that the transition from motion X to any other motion always goes via motion Y.

As most simple motions (like kicking or standing up) can be defined by sequences of joint data vectors we developed a special motion description language, in which all our motions are defined. Programs in this language consist of transition definitions, jump labels, and lines defining motor data. A typical data line looks like this:

```
~ ~ ~   ~ ~ ~   -350 -190 1750   -350 -190 1750   -1840 -40 2500   -1840 -40 2500   1 25
```

The first three values represent the three head joint angles, the next three values describe the mouth and the tail angles, followed by the twelve leg joint angles, three for each leg, all angles in milliradians. The last but one value decides whether specified joint angles will either be repeated or interpolated from the current joints angles to the given angles. The last value defines how often the values will be repeated or over how many frames the values will be interpolated, respectively. The tilde character in the first six columns means that no specific value is given (don't care). This has special importance for the head joint angles as it allows head motion requests to be executed.

The *Motion Config Tool* (cf. Fig. 3) was developed by the team from the Humboldt-Universität zu Berlin for Robocup 2000. The tool is a graphical front-end for specifying motions in the motion description language. It compiles them into C code with a Flex/Bison-based parser. The parser checks the motion set defined for consistency, i. e., it checks for missing transitions from one motion to another and for transitions to unknown motions. For more complex motions, which cannot be realized by motion description sequences but require special routines to generate motor data, it is possible to have function calls in the motion code to execute C code functions. This way, all walking type motions are integrated.

For interactively testing motions and their transitions, DogControl provides a dialog to request specific motions. The requested motion and the transition from the current one will be executed immediately. Furthermore, a second dialog is provided to transmit new motion descriptions to the robot and execute them without the need to recompile anything.

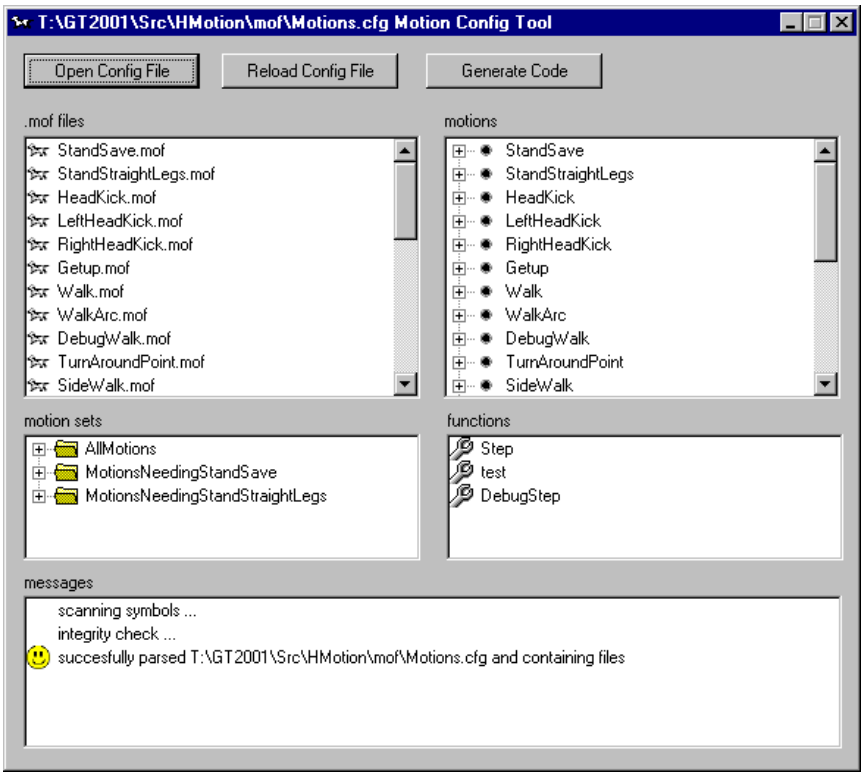

Figure 3: Motion Config Tool. The window in the upper left shows all motion files. The window beneath contains the motion sets in which some motions are summarized. The right side displays all motions, transitions, and functions. The result of the integrity check is shown in the message window at the bottom.

3.3 Walking

We implemented walking utilizing a trot gait. Two diagonally opposite legs perform the same movement, while the other two legs move with a half leg phase offset. The legs that are on ground move with a fixed velocity backward or forward in the desired walking direction while walking forward or backward respectively. When the legs are in the air, they move in a half circle back to their starting position relative to the robot body. The walking speed is determined by the distance each leg is moved in the walking direction. Turning is done by moving the four legs in different directions so that while they are on the ground, they describe a circle around the robot body. Turning and straight walking can be combined resulting in curved walks, e. g., walking sideways and turning results in circling around a point in front of (or behind) the robot.

The position of every leg relative to the robot body is defined by a small set of parameters at any given time. The angles needed to position the legs correctly are calculated with inverse kinematics.

3.3.1 Parameters

The fixed parameters that define the walking style are:

- Leg lift height
- Positions the four legs have in the middle of the ground phase.
- Time for one full gait cycle

To get the maximum performance we tested many combinations for these parameters. The goal was to get the fastest walk which still is reliably stable.

Furthermore there are variable parameters defining the current walk:

- The stride length defines the walking speed as the time for one step is the same for all speeds.
- Direction
- Turning speed

3.3.2 Inverse Kinematics

After the desired leg position is calculated, it is necessary to calculate the required leg joint angles to reach that position. First, the knee joint angle is calculated as it is determined by the distance from shoulder joint to the paw. Next, the angle of the shoulder joint that rotates the leg to the side is calculated from the side distance between the paw and the shoulder. Finally, the distance from the shoulder to the paw in robot body direction defines the other shoulder joint angle.

3.4 Odometry

To be able to use odometry for self-localization, HMotion is responsible for writing the current odometry values into shared memory. These values are the speed and the turning speed of the currently executed motion. Those values are taken from an odometry table which contains speeds for all motions with all possible parameter combinations. The odometry table is loaded on startup from a text file on the memory stick. This allows to quickly adjust the table to different conditions which is necessary since different underground causes different movement speeds.

3.5 Adaptive Head Control

Head motion requests are used to control the movements of the head. They contain head joint angles that are passed to the robot, and they are executed in parallel to other motions. The head of the robot can be rotated in three dimensions. As the camera is installed in the head, the main purpose of the head motion is to control the view of the robot. There are conflicting goals for the head control: on the one hand, the robot needs to search for and keep track of the ball. On the

other hand, it is required to look for the flags and the goals as a prerequisite for self-localization. In addition, the head is used as part of some motions, e. g. kicks. Therefore, there exist several modes that control the motion of the head:

3.5.1 Robot Motion Mode

This is an implicit mode, because whenever a robot motion is performed that requires controlling the head, its commands have priority over all other head control modes.

3.5.2 Ball Tracking Mode

In this mode, the head searches for the ball, and when it has detected it, it tries to follow the ball. After it has lost the ball again, it will first search towards the direction of the last ball motion. Only if this was not successful, it will again perform an iterative search motion.

3.5.3 Landmarks Scanning Mode

When the head control module is in this mode, it will scan for landmarks, i. e. flags and goals.

3.5.4 Automatic Mode

In this mode, the head control module will automatically switch between the *Ball Tracking Mode* and the *Landmarks Scanning Mode*. Only if the self-localization module reports that the calculation of the current position is uncertain, the camera will scan for landmarks (cf. Chapter 5) . Otherwise, it will track the ball. This mode is used in most situations.

4 Image Processing

4.1 Image Segmentation

The robot's most important sensor is a CCD camera. The GermanTeam used 88×72 pixel images up to the Robocup German Open 2001 in Paderborn, and we switched to 176×144 pixel images afterwards. Besides more accurate distance and angle measurements, we gained better color values since there is no longer an interpolation which causes the mixing of original colors and the loss of contrast. HIntegra starts processing the YUV-images by performing a color segmentation. The color of every pixel is assigned to one of eight color classes, e. g. the orange of a ball, the yellow of a goal and the landmarks, or the green of the field and the landmarks. This assignment is performed by using a lookup table, the so-called color table. The color table assigns each color from the $256 \times 256 \times 256$ YUV color space to a color class by using the YUV-intensities as indices into the lookup table. As such a table would require 16 MB of memory, only the six most significant bits of each color channel are used, reducing the size of the resulting $64 \times 64 \times 64$ table to 512 KB, with each cell referring to a $4 \times 4 \times 4$ sub-cube of the original color space.

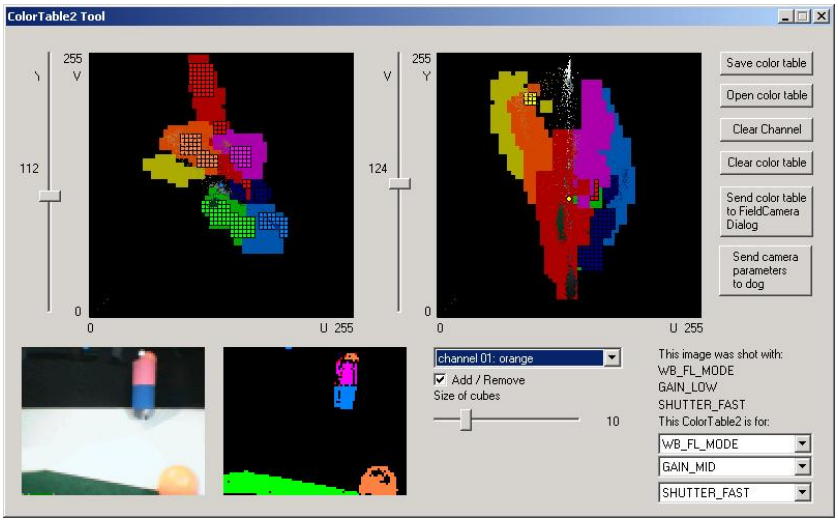

Figure 4: Creating color tables in YUV color space

While segmenting an image, it is also run-length encoded to simplify the construction of the blobs, and to compress the data. The resulting RLE-image contains a set of runs for each line. Every run is represented as a color class and the left and right boundaries of a sequence of pixels with the same color class.

4.2 Creating Color Tables

The color tables are constructed with DogControl on the PC using images we recorded with the robots. The GermanTeam developed two different tools to help the user to build color tables:

4.2.1 YUV Tool

The first tool allows to manipulate the YUV color table directly by point-and-click operations on an image. Around a selected pixel, a cube of variable size in YUV space is assigned to the same color class. As the user has to build the color table cube by cube, he or she will need plenty of time and a large number of different pictures. However, the resulting color table is very accurate with respect to the ellipsoids the color classes occupy in the color space.

4.2.2 HSI Tool

The other tool uses the HSI color space to create the color table. It allows the user to specify the minimum and maximum values for hue, saturation, and intensity for each color class using sliders and gives an instant feedback by real-time segmentation of four images at the same time. These images are directly taken from the shared memory, and they are held as long as needed. It is also possible to update one image simultaneously with the one in the shared memory. The tool saves the color table both in YUV and in HSI format. The HSI approach leads to good results

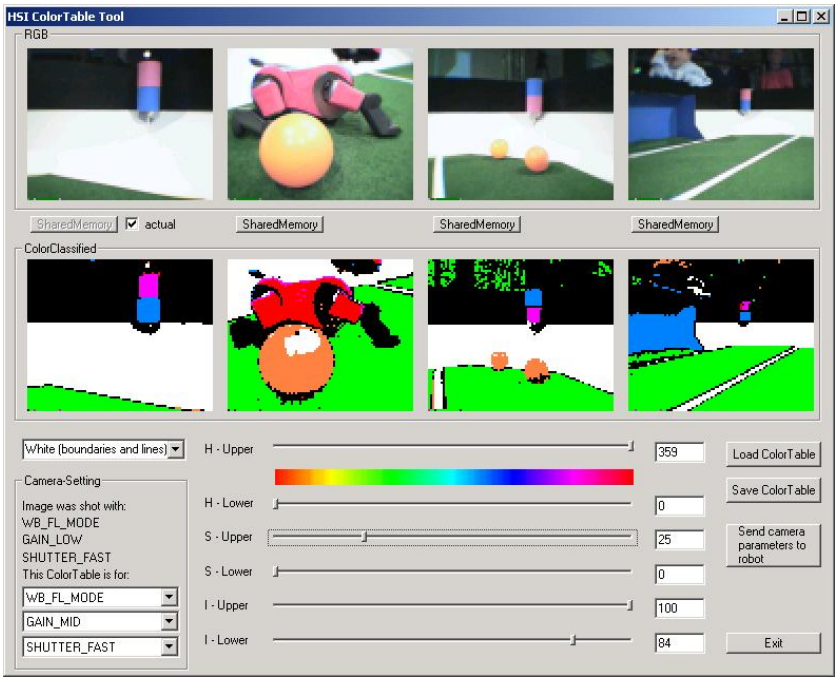

Figure 5: Creating color tables in HSI color space

very quickly by defining big sectors in the color space, and it is more tolerant against changing light conditions.

Both tools can be used in combination: first, the HSI tool is used to quickly generate a color table, and then the YUV can be employed for fine tuning, if required. In the future, the GermanTeam will follow both approaches while trying to speed up the process of building color tables with the goal of developing an automatic tool enabling the robot to adapt the color class to changing light conditions without the need for user interaction.

4.3 Detection of Blobs

The result of the color segmentation of an image is a RLE image, in which each run consists of a color class (orange, yellow, green ...) and its left and right borders. In a further step, these regions are clustered to so-called *blobs*, resulting in a new representation of the image, the *blob collection*.

A blob is a data structure representing a region of connected runs of the same color class. For each region, eight characteristic points N, NE, E, SE, S, SW, W, and NW are stored. The points are named after the points of the compass. Each of the eight points represents the maximum expansion of the blob in a certain direction, either vertically, horizontally, or diagonally. The point N is one of the topmost points in that region, the point E is one of the right most points in that region, and so on.

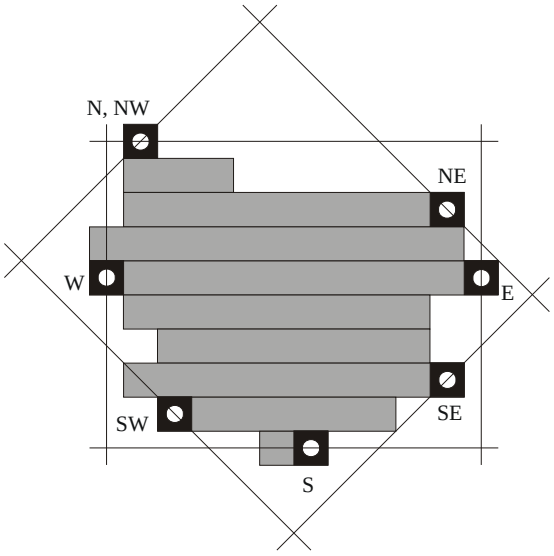

Figure 6: The characteristic points of a blob.

We implemented an algorithm based on a state machine that directly operates on the run data structure and finds the boundary of each eight-connected region. This algorithm jumps from run to run, visiting all the runs lying on the boundary of the region. Such a state machine was described by Quek (2000). While jumping from run to run the eight points N, NE, etc are updated. So after the last jump, the topmost, top rightmost, rightmost, etc point of the region are known. Figure 6 shows a connected region of runs with its characteristic points.

All the blobs found in a RLE image are stored in a blob collection. The detection of the ball, the flags, the goals, and the other robots operates on that collection of blobs. In addition to the blobs determined, the collection also stores information about the robot when the underlying image was shot: the resolution of the original image, the angles of view of the camera, the values of the neck joints, and the value of the distance sensor.

4.4 Recognition of the Ball

The ball is the most significant object in the soccer game. Therefore, it is important to recognize it as often and as precise as possible. It is also essential to exclude wrong ball detections resulting from misreadings in the color image.

4.4.1 Blob Filtering

It is assumed that the ball is either perceived on the carpet, or that it is so close that the carpet cannot be seen anymore. Therefore, the GermanTeam recognizes the ball as the largest orange blob that either extends more than half of the image width or height, or that is neighbored by a green blob.

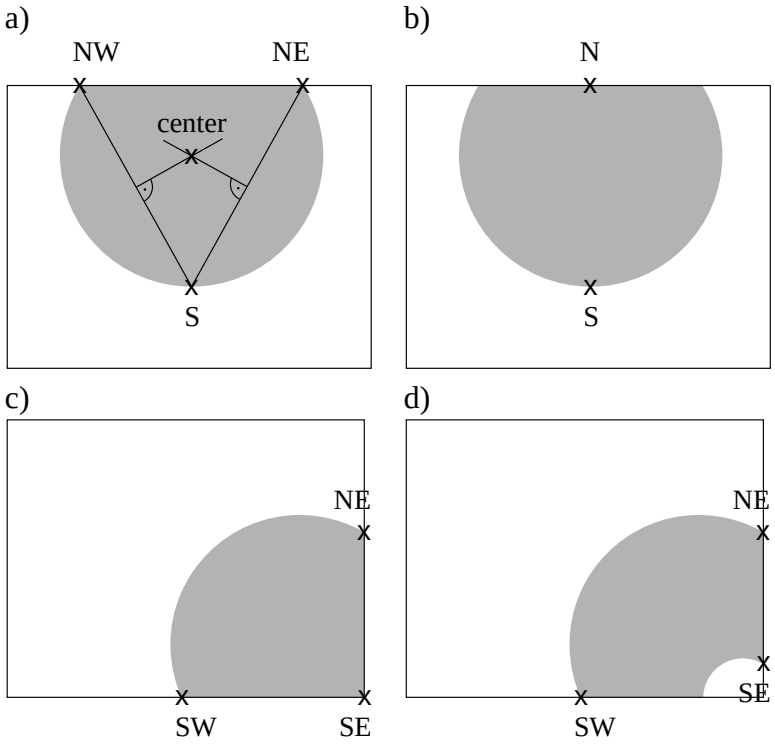

Figure 7: Calculating the center point. a) Intersecting the middle perpendiculars. b) *S* can be used for the calculation of the center, *N* cannot. c) *NE* and *SW* lie on the border of the ball, *SE* does not. d) A spot light in the corner. Although not in a corner, *SE* is not located on the border of the ball.

4.4.2 Center Point and Diameter

To determine the position of the ball, two features must be extracted from the blob collection: the ball's diameter and its center point. The diameter in the image can be related to the diameter of the ball in reality to determine its distance from the camera, taking both the image resolution and the opening angle of the camera into account. Together with the center point, the ball's relative position according to the camera can be established.

Although the calculation of both diameter and center seems to be straightforward, a little bit more effort has to be spent to determine the location of the ball even in situations when it cannot be seen completely, i. e. only a part of the ball is shown in the camera image. In such cases, the width and the height of a blob cannot be used to determine its diameter, because it is partially hidden, and the center of such a blob is not the center of the ball.

4.4.3 Intersection of Middle Perpendiculars

Therefore, the GermanTeam uses the intersection point of two middle perpendiculars to determine the center point of the ball (cf. Fig. 7a). In comparison to the usage of width and height, the advantage of this approach is that three arbitrary points on the border of the ball are enough

to calculate its center. As has been described in section 4.3, each blob consists of eight points. The major question for the approach presented here is, which three among these eight points are used to determine the ball's center. If the ball is completely contained in the image, i. e. no part sticks out to the sides, all eight points lie on the border of the ball. However, if the image does not contain the whole ball, some of the points lie on the image border rather than on the border of the ball. Therefore, they cannot be used to determine the ball's center point.

4.4.4 Selecting Points

When deciding which points can be used, it must be distinguished between points representing horizontal or vertical extrema and points that stand for diagonal extrema. While horizontal and vertical maxima cannot be used if they are located on "their" border of the image, e. g., if the point *N* is at the upper edge of the image (cf. Fig. 7b), diagonal extrema have only to be excluded if they lie in an image corner (cf. Fig. 7c).

From the points that satisfy these conditions, the two are selected that are furthest away from each other. Then, a third point is chosen that is farthest away from the two points already selected, and that ensures that at least one of the three points chosen is not located at any image border, which reduces the possibility of calculating a wrong ball position from an orange blob with a white spot light in an image corner (cf. Fig. 7d).

After the three points were selected, the center point can be determined by intersecting the middle perpendiculars of lines connecting these points (cf. Fig. 7a). The largest distance of the center to one of the points chosen specifies the radius of the ball. Using this information, the direction and distance of the ball relative to the camera are known, and after also employing the pose of the camera on the robot, the position of the ball on the field in the robot's system of coordinates can be calculated.

4.5 Recognition of Team-Mates and Opponents

The robot's world model consists of its own global pose, i. e. its Cartesian position and its orientation, and the poses of the other robots. The self-localization module performs the calculation of the own pose, while the obstacle module determines the locations of the other robots.

4.5.1 Blob Clustering

The result of processing an image showing one or more other robots is a blob collection containing a lot of red or blue blobs. These blobs are grouped, so that each group of blobs represents one robot. All blobs that either overlap horizontally or have a maximum horizontal distance of, e. g., 10 pixels are assumed to belong to the same robot.

4.5.2 Calculating Angle and Distance

For each group of blobs the angle to the corresponding robot is calculated from the angle of the center of the blobs in the image and from the pose of the robot's head. The distance to another robot is derived from the height of the largest blob in the group.

4.5.3 Indications of Other Robots

A group of blobs is said to be an *indication* of a robot. Such an indication leads to the assumption that there might be a robot at the indicated position. The position is determined from the own pose and the calculated angle and distance to the indication of the robot. All these assumed positions of robots are stored in a list for two seconds.

4.5.4 Calculating Positions of Other Robots

In the list of possible robot positions of the last two seconds, the positions with the biggest "dense" are determined by finding maxima in the distributions in x and y direction. Those maxima are matched with the representation of the other robots in the current world model leading to an updated world model.

4.6 Recognition of Flags and Goals

The recognition of the flags and the goals is required to perform the self-localization of the robots on the field. Each flag consists of a unique combination of colors, whereas each goal has only one color. As they are located on the green carpet, they can also be detected as an arrangement of two colors. This is important, because the blob collection may contain areas with random colors, resulting from objects above the field, or from mixtures between colors of different objects on the field. Grouping two colored areas together provides the possibility to use spatial constraints to filter out most misreadings.

4.6.1 Compensating for Head Rotations

As the head of the robot can be turned in three dimensions, the flags and goals may be rotated in the image. Therefore, the first step of the object recognition is to use the measured joint angles of the head to correct the rotations. As all the objects required for localization are of rectangular shape, the blob collection is transformed into a list of blobs described as vertical and horizontal angular ranges. For each edge of these rectangular ranges, it is stored, whether it was located at the border of the unrotated, original camera image. This information will be used later, because edges resulting from the image border cannot be used for localization (cf. Chapter 5). Note that the transformation is only applied to rotational components. It does not compensate for the camera's translational offset relative to the robot's body, i. e., after the transformation, all angular bearings are relative to the robot's body axes, but they still lead off from the position of the camera.

4.6.2 Spatial Constraints

After the transformation, the borders of the rectangular blobs are oriented in parallel to the horizontal and vertical axes. Therefore, vertical and horizontal relations can be handled separately. The GermanTeam uses the thirteen relations of the Allen (1983) calculus to describe the spatial constraints. To encode, e.g., that blob B is immediately above blob A, it can be written:

$$A_y \text{ meets } B_y \tag{1}$$

The subscript y denotes that the relation is applied to the vertical ranges of A and B. However, the relations of the calculus are to precise to be used directly, e. g., *meets* means that A exactly ends where B begins, and the image processing method employed cannot deliver this precision. Therefore, one range is enlarged a little bit, so that "above" can be encoded as

$$\text{enlarged}(A_y) \text{ overlaps } B_y. \tag{2}$$

In addition, "above" requires both blobs to overlap in horizontal direction. In Allen's calculus, this can be written as

$$\neg(A_x < B_x \vee A_x \text{ meets } B_x \vee A_x \text{ metBy } B_x \vee A_x > B_x). \tag{3}$$

Combining both expressions, the final definitions of "above" and its complement "below" are

$$\begin{aligned} A \text{ above } B &= \text{enlarged}(A_y) \text{ overlaps } B_y \wedge \\ &\quad \neg(A_x < B_x \vee A_x \text{ meets } B_x \vee A_x \text{ metBy } B_x \vee A_x > B_x) && (4) \\ A \text{ below } B &= B \text{ above } A. && (5) \end{aligned}$$

4.6.3 Recognition of Flags

To find the flags, the algorithm runs through all pink blobs, sorted by size in descending order, and tries to find an appropriate partner from the yellow, green, and sky blue blobs. Again, larger blobs are found first. A good partner is a blob that is either above or below the pink blob, and that has nearly the same size. For each pink blob, the search stops when the first appropriate counterpart was found.

4.6.4 Recognition of Goals

The recognition of the goals is a little bit more complicate, because other robots can partially hide them, and therefore split them into more than one blob. Therefore, the approach for detecting a goal is to search for the largest blob with the goal's color that is not a part of a flag, then to find the largest green area below that blob,[3] and finally to accumulate all blobs with the color of the

[3]The carpet shall be located below 0° in transformed angular coordinates, because the camera is always above the carpet. This assumption is used as a further criterion for green blobs.

goal above that green area. If the goal has at least a certain minimum area, i. e. the size of a goal seen from the other side of the field, it is assumed to be detected.

5 Self-Localization

The GermanTeam tried two different approaches for self-localization. First, we implemented a localization based on a representation of the environment. During the competition in Seattle we added a second approach—the Monte-Carlo localization. The self-localization is performed each time a new image is processed. It is based on the blob collection and the current odometry data, i. e. the motion vector of the robot since the last self-localization. Both self-localization approaches update global position of the robot in the world model.

5.1 Localization Based on a Representation of the Environment

If a robot has to play soccer, he should know where he is located on the field. The size and the appearance of the field do not change during the game and are well known. So what the robot can do is to orientate itself using characteristic marks in the environment, e. g. the flags and the corners of the goals. We decided to equip the robot with a *representation of the environment* for two reasons: A single image usually carries not enough information to derive the exact position of the robot. Sometimes no characteristic marks are seen for a longer time, e. g., while tracking the ball. When a human soccer player closes his eyes, he still has a representation of the environment. If he walks some steps with closed eyes, this representation is updated. As the soccer player has this representation, he still knows where he is located on the field. Therefore, there are three problems to be solved for the robot: what is a good representation of the environment, how to obtain such a representation, and how to calculate the position using the representation.

5.1.1 Representation of the Environment

The robot has a representation of each landmark. Landmarks are the six flags and the two corners at the ground line of each goal. A representation of a landmark consists of both the distance and the angle to that landmark. For both values a validity is stored, indicating how reliable the value is. The representations of the single landmarks are not connected. The values for distance and angle of one landmark do not depend on the values of a different landmark. So the whole representation of the environment might be distorted. But this distortion ensures that wrong values for single landmarks do not spoil the entire representation of the environment.

5.1.2 Updating the Representation of the Environment

To obtain a satisfactory representation of the environment the robot uses *vision* and *odometry*. Using odometry means that the robot knows the effect of its movements on its position. If the robot walks forward, the representation of each landmark is shifted backward, if the robot turns left the representation of each landmark is turned right, etc. The validities of the representations

decrease when moving, because the precision of the knowledge about the effect of the motion on the position is not accurate enough.

Each time a new image is processed, the representation of each landmark recognized in the image is updated according to the distance and the angle calculated for that landmark. The distance to a landmark is calculated from its size in the image. The angle to the landmark is determined from the position of the landmark in the image and the angles of the neck joints. As the information about a landmark obtained by vision is quite reliable, the validities of the distance and the angle are set to a higher value when a landmark is seen.

5.1.3 Calculating the Position

The position of the robot is calculated based on the representation of the environment. That calculation only uses the angles and the distances (with their validities) of the representations of the landmarks. As the representation of the whole environment might be distorted, the landmarks causing such a distortion are said to be invalid. The angles to three of the landmarks lead to a possible position of the robot. For each triple of landmarks this position is calculated and stored in a list. The validity of each position is set to the product of the validities of the angles of the three landmarks. All positions deviating too much from the average position are removed from the list. The position of the robot is calculated as the average of the positions in the list weighted by their validity.

5.2 Monte-Carlo Localization

In its third game and during the challenge, the GermanTeam used a Markov-localization method employing the so-called Monte-Carlo approach (Fox *et al.*, 1999). It is a probabilistic approach, in which the current location of the robot is modeled as the density of a set of particles (cf. Fig. 10a). Each particle can be seen as the hypothesis of the robot being located at this position. Therefore, such particles mainly consist of a robot pose, i. e. a vector representing the robot's x/y-coordinates in millimeters and its rotation θ in radians:

$$pose = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{6}$$

A Markov-localization method requires both an observation model and a motion model. The observation model describes the probability for taking certain measurements at certain locations. The motion model expresses the probability for certain actions to move the robot to certain relative positions.

The localization approach works as follows: first, all particles are moved according to the motion model of the previous action of the robot. Then, the probabilities for all particles are determined on the basis of the observation model for the current sensor readings, i. e. bearings on landmarks calculated from the actual camera image. Based on these probabilities, the so-called *resampling* is performed, i. e. moving some of the particles to the location of the sample with the highest probability. Afterwards, the average of the probability distribution is determined,

representing the best estimation of the current robot pose. Finally, the process repeats from the beginning.

5.2.1 Motion Model

The motion model uses values $\Delta_{odometry}$ from the odometry table (cf. Sect. 3.4) to represent the effects of the actions on the robot's pose. In addition, a random error Δ_{error} is assumed, according to the following definition:

$$\Delta_{error} = \begin{pmatrix} 0.1d \times \text{random}(-1 \dots 1) \\ 0.02d \times \text{random}(-1 \dots 1) \\ (0.002d + 0.2\alpha) \times \text{random}(-1 \dots 1) \end{pmatrix} \tag{7}$$

In equation (7), d is the length of the vector from the odometry table, i. e. the distance the robot walked, α is the angle the robot turned.

For each sample, the new pose is determined as

$$pose_{new} = pose_{old} + \Delta_{odometry} + \Delta_{error} \tag{8}$$

Note that the operation $+$ involves coordinate transformations based on the rotational components of the poses.

5.2.2 Observation Model

The observation model relates real sensor measurements to measurements as they would be taken if the robot were be at a certain location. Instead of using the distances and directions to the landmarks in the environment, i. e. the flags and the goals, this localization approach only uses the directions to the vertical edges of the landmarks. The advantage of using the edges for orientation is that one can still use the visible edge of a landmark that is partially hidden by the image border. Therefore, more points of reference can be used per image, which can potentially improve the self-localization.

As the utilized sensor data are bearings on the edges of flags and goals, these have to be related to the assumed bearings from hypothetical positions. To determine the expected bearings, the camera position has to be determined for each particle first, because the real measurements are not taken from the robot's body position, but from the location of the camera. Note that this is only required for the translational components of the camera pose, because the rotational components were already normalized during the recognition of the flags and the goals (cf. Sect. 4.6). From these hypothetical camera locations, the bearings on the edges are calculated. It must be distinguished between the edges of flags and the edges of goals:

Flags. The calculation of the bearing on the center of a flag is straightforward. However, to determine the angle to the left or right edge, the bearing on the center $bearing_{flag}$, the distance between the assumed camera position and the center of the flag $distance_{flag}$, and the radius of the flag r_{flag} are required (cf. Fig. 8):

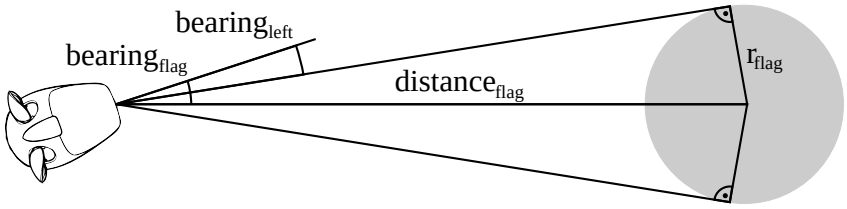

Figure 8: Calculating the angle to an edge of a flag.

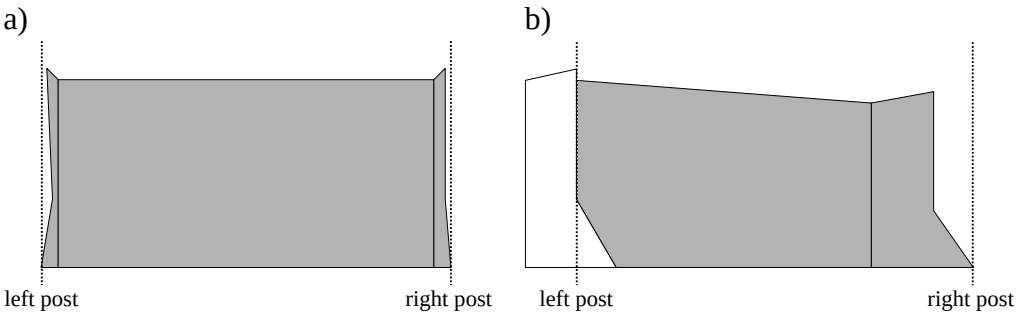

Figure 9: The goal posts as edges of a color blob. a) Frontal view. b) Side view.

$$bearing_{left/right} = bearing_{flag} \pm \arcsin(r_{flag}/distance_{flag}) \tag{9}$$

Goals. The front posts of the goals are used as points of reference. As the goals are colored on the inside, but white on the outside, the left and right edges of a color blob representing a goal even correlate to the posts if the goal is seen from the outside, with one exception: as the goal posts consist of a perpendicular part and a diagonal part, the diagonal corners are the most extreme points when looking from the inside, while the perpendicular section forms the edge of the side of the goal that is seen from the outside (cf. Fig. 9). As the (hypothetical) location of the robot is known when calculating the bearings on the goal posts, this distinction can always be made, resulting in the correct determination of the expected angles.

Probabilities. The observation model only takes into account the bearings on the edges that are actually seen, i. e., it is ignored if the robot has *not* seen a certain edge that it should have seen according to its hypothetical position and the camera pose. Therefore, the probabilities of the particles are only calculated from the similarities of the measured angles to the expected angles. Each similarity s is determined from the measured angle $\omega_{measured}$ and the expected angle $\omega_{expected}$ for a certain pose by applying a sigmoid function to the difference of both angles:

$$s(\omega_{measured}, \omega_{expected}) = \begin{cases} \mathrm{e}^{-100d^2} & \text{if } d < 1 \\ \mathrm{e}^{-100(2-d)^2} & \text{otherwise} \end{cases} \quad \text{where } d = \frac{\left|\omega_{measured} - \omega_{expected}\right|}{\pi} \tag{10}$$

The probability p of a certain particle is the product of these similarities:

$$p = \prod_{\omega_{measured}} s(\omega_{measured}, \omega_{expected}) \tag{11}$$

5.2.3 Resampling

In the resampling step, the samples are moved according to their probabilities. There is a trade-off between quickly reacting to unmodeled movements, e. g., when the referee displaces the robot, and stability against misreadings, resulting either from image processing problems or from the bad synchronization between receiving an image and the corresponding joint angles of the head. Therefore, resampling must be performed carefully. One possibility would be to move only a few samples, but this would require a large number of particles to always have a sufficiently large population of samples at the current position of the robot. The better solution is to limit the change of the probability of each sample to a certain maximum. Thus misreadings will not immediately affect the probability distribution. Instead, several readings are required to lower the probability, resulting in a higher stability of the distribution. However, if the position of the robot was changed externally, the measurements will constantly be inconsistent with the current distribution of the samples, and therefore the probabilities will fall rapidly, and resampling will take place.

The filtered probability p' is calculated as

$$p'_{new} = \begin{cases} p'_{old} + 0.1 & \text{if } p > p'_{old} + 0.1 \\ p'_{old} - 0.1 & \text{if } p < p'_{old} - 0.1 \\ p & \text{otherwise.} \end{cases} \tag{12}$$

Resampling is done in two steps: first, the 20% of the samples with the smallest probabilities are moved to the position of the particle with the highest probability, massing the samples at the most probable location. Then, the particles are moved locally according to their probability, the more probable a sample is, the less it is moved. This can be seen as a search for the best position.

The samples are moved according to the following equation:

$$pose_{new} = pose_{old} + \begin{pmatrix} 100(1-p') \times \text{random}(-1 \ldots 1) \\ 100(1-p') \times \text{random}(-1 \ldots 1) \\ 0.5(1-p') \times \text{random}(-1 \ldots 1) \end{pmatrix} \tag{13}$$

5.2.4 Estimating the Pose of the Robot

The pose of the robot is calculated from the sample distribution in two steps: first, the largest cluster is determined, and then the current pose is calculated as the average of all samples belonging to that cluster.

Finding the Largest Cluster. To calculate the largest cluster, all samples are assigned to a grid that discretizes the x-, y-, and θ-space into $10 \times 10 \times 10$ cells. Then, it is searched for the $2 \times 2 \times 2$ sub-cube that contains the maximum number of samples.

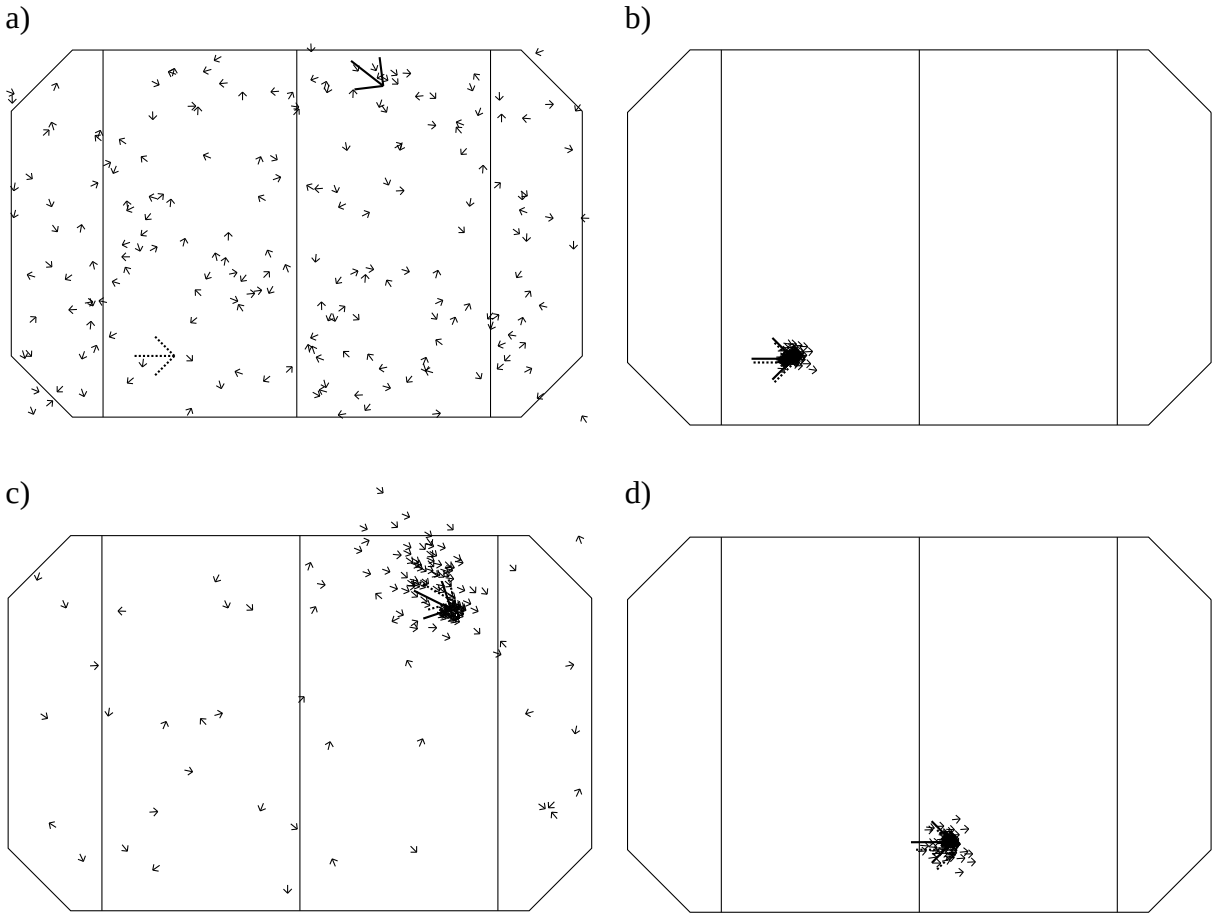

Figure 10: Distribution of samples during Monte-Carlo localization. The large dotted arrow marks the real position of the robot, the solid arrow marks the estimated location. a) Initial distribution. b) After a complete head scan (back and forth). c) A complete head scan after manually putting the robot away from (b). d) Starting from (a), after a complete head scan while walking.

Calculating the Average. All samples belonging to that sub-cube are used to estimate the current pose of the robot. Whereas the mean x- and y-components can directly be determined, averaging the angles is not straightforward, because of their circularity. Instead, the mean angle θ_{robot} is calculated as:

$$\theta_{robot} = \text{atan2}(\sum_i \sin\theta_i, \sum_i \cos\theta_i) \tag{14}$$

Certainty. The position determined is assumed to be *certain* if the largest cluster contains at least 90% of the samples, and if the minimum p_i' of all particles in that cluster is larger than 0.8. If the robot pose is certain, there is no need to scan for landmarks (cf. Sect. 3.5). Otherwise, the acquisition of new observations has to be forced, i. e. the robot head has to search for flags or goal posts.

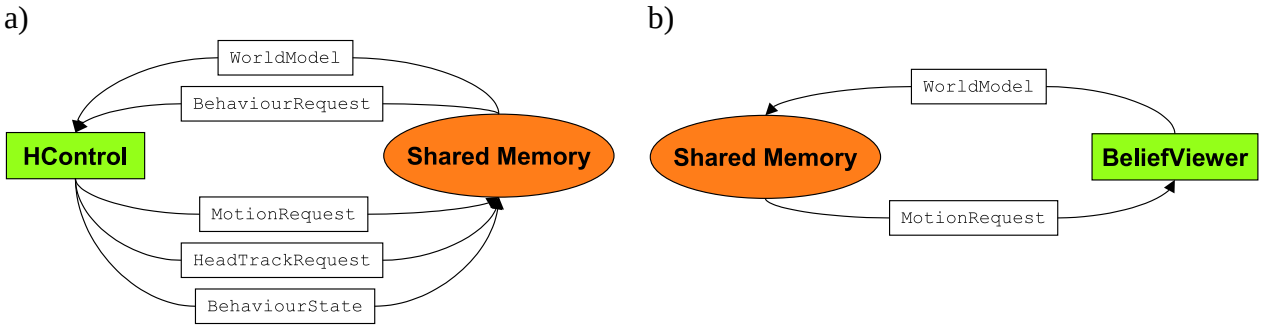

Figure 11: a) Interfaces of HControl. b) Interfaces of the *BeliefViewer* dialog in DogControl.

5.2.5 Results

Figure 10 depicts some examples for the performance of the approach using 200 samples. The experiments shown were conducted in the robotics simulator SimRobot (Röfer, 1998). To improve the realism of the simulation, the measurements of the angles of the head joints have a random error of up to $\pm 5°$. The simulated motions deviate by a maximum of $(\pm 5\%, \pm 1\%, \pm 20\%)$ from the odometry offsets used. Nevertheless, the approach is able to quickly determine a good estimation of the current location, and is able to re-localize after the robot was moved manually. It was precise enough to reach the forth place in the localization challenge during RoboCup 2001.

6 Behavior

The Aperios object HControl controls the behavior of the robot. It receives the *world model* from Hintegra (cf. Fig. 11a). A world model contains the robots knowledge about the world: its own position and direction of view, the distance and direction both to the ball and to the other players, messages given with the switches on the head and the back of the robot, and internal states of the robot, e. g., whether it has fallen down, or whether it was picked up. To reduce the complexity of the system, HControl has no access to internal information of other modules. So the world model can be seen as the knowledge HControl is allowed to know.

The main output of HControl are motion requests sent to HMotion. These requests consist of identifiers and parameters to select the current motion of the robot, e. g. a gait or a kick. In addition, head motion requests are sent to HIntegra. HIntegra automatically controls the movements of the head depending on the current state of the self-localization module (cf. Sect. 3.5), but in some situations it is necessary to force HIntegra to perform a specific head motion. Then, HControl has to select one of the head motion modes described in section 3.5.

HControl also receives behavior requests from the *BehaviorTester* dialog in DogControl, and will provide behavior states for debugging purposes. Thus during the development of behaviors, the robot can remotely be controlled from DogControl. This allows to develop large parts of the behaviors without using a physical robot, but by employing a simulation instead. The simulator is integrated into DogControl as the *BeliefViewer* dialog (cf. Fig. 15). This dialog processes the motion requests stored in the shared memory and employs the data provided from the odometry

table (cf. Sect. 3.4) to simulate the movements of the robot (cf. Fig. 11b). Using the computer mouse, the simulation allows to arbitrarily place the robot on the field. The simulation of behaviors is not as realistic as testing on the field, but it simplified the development a lot.

6.1 Architecture

In contrast to our last year's solution, in 2001 the behavior architecture followed a very different approach:

6.1.1 Last Year's Approach

In the RoboCup in Melbourne, the Humboldt Heroes used the "Option Model" approach. There was a set of possible behaviors (options) that were able to generate "plans", i. e. sequences of motion commands. Depending on the environment, these options could calculate an expected utility. On top of this, there was a deliberator that queries all the options for their expected utility, and that selected the option that provided the maximum. To prevent options from frequently changing, i. e. to switch between different plans very often, they could request to be continued, no matter what happened in the environment.

The biggest problem of the approach was to guess the utilities. It was not easy to ensure that the right option was selected in the right situation. In addition, the system was not very handy for adding new options, because it always had to be checked, whether new utilities fitted into the hierarchy of the other utilities. Finally, we had to implement workarounds in the deliberator that selected some options independently from their utilities.

6.1.2 Simple State Machines

This year, we tried a new approach: for the German Open, a Europe-open RoboCup competition that took place in Paderborn in June, we decided to use a combination of state machines with decision trees. There were a variety of states that could generate motion requests with a simple decision tree. States were executed until they selected another state to be switched to. The decision on which behavior is active is not done by a deliberator, but by the selected behavior itself. The underlying assumption of this approach is that the different behaviors themselves know best whether it is good to continue their execution, and if not, which other behavior is a useful successor.

In Fig. 12 two examples are shown. The left figure depicts three states of a simplified goal keeper (in the real implementation, there are more states). In each state, a decision tree generates motion commands depending on the environment, and depending on the period of time this state is already active. After each iteration, a state has to return which state to process next. In the example, the robot will remain in the state "play ball" until it either cannot see the ball anymore, or it is too far away from the goal. Under both conditions, "return to goal" will be the following state.

A severe problem occurs if the state machine illustrated in Fig. 12b is integrated into the same code: both state machines share the state "play ball", but both have different conditions

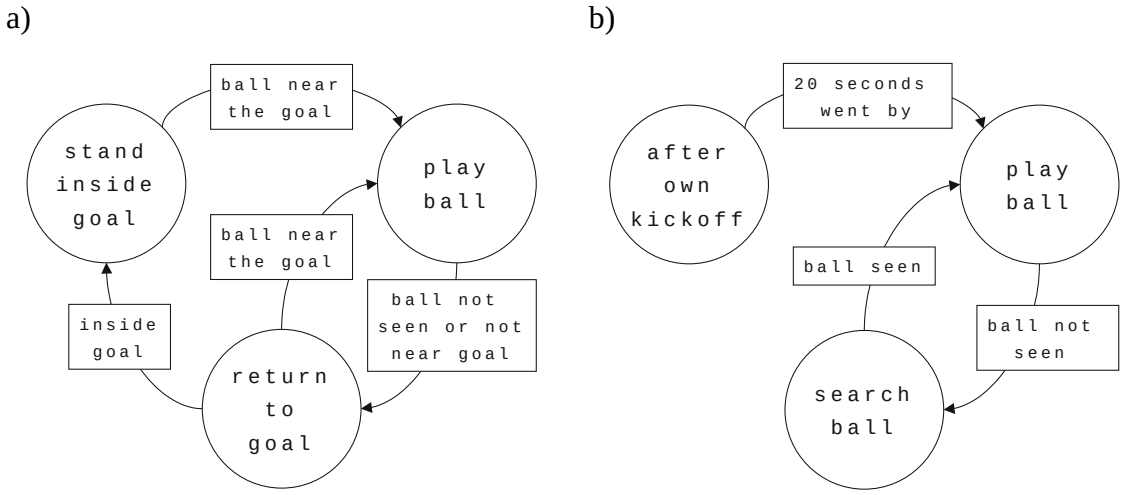

Figure 12: Simplified examples of state machines used during the German Open 2001. a) States for a goalie. b) States for a field player.

for leaving the state. If the code “play ball” should not be duplicated, i. e. only one instance is used, a decision has to be made, which transition to use, depending on an external parameter, the role of the player. The examples in Fig. 12 are still considerably simple. In more complex systems, the problem of transitions being dependent on external parameters appears very often in that architecture.

A second issue was that almost identical parts of the source code occurred in the decision trees of different states. Both in the state “play ball” and in the state “return to goal state”, code for “walking somewhere” is required, once to the ball and once to the own goal.

A third problem appeared always when new states were introduced, because the transitions from many other states had to be changed.

6.1.3 Multi Layer State Machines

From the experiences made with that architecture during the German Open, we developed an advanced architecture that was used during the competition in Seattle. The problem of the influence of external parameters was solved by introducing layers into the architecture. For the example given in Fig. 12, Fig. 13a shows a possible solution that employs two layers: on the upper layer, there are two states that contain internal state machines, which have the same transitions and conditions as the state machines shown in Fig. 12. They also remain in the same state until a condition for another state is satisfied. On the lower layer, *skills* as “play ball” are represented. They are shared by all states in the upper layer. When a certain state in the upper layer is active, the corresponding skill in the lower layer is executed. By having two independent state machines in the upper layer, the problem of defining transitions depending on the context (here: the role of the robot) does not exist, i. e. it is solved by the second layer.

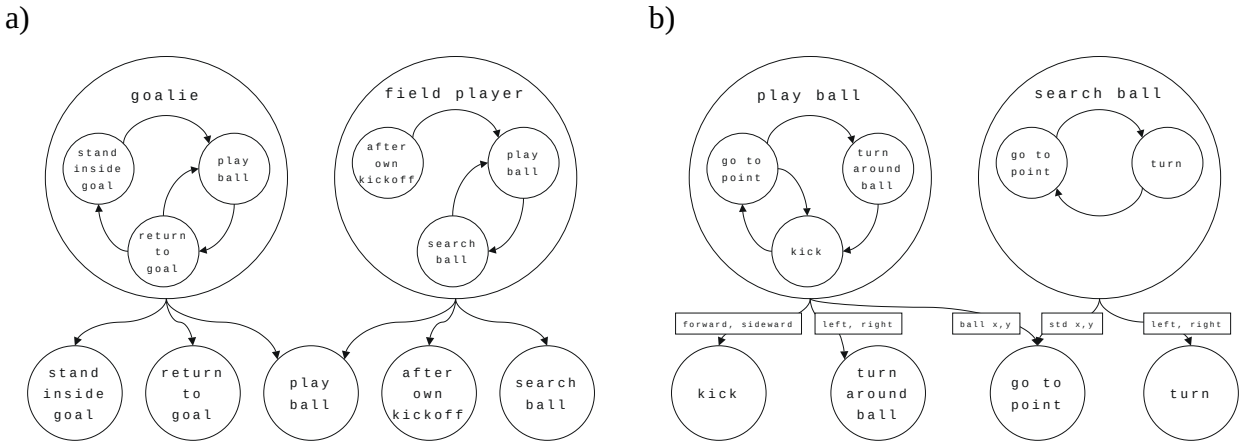

Figure 13: a) A two layer state machine approach for the example shown in Fig. 12. b) Parameterized skills.

The problem of reusing the same source code (e.g. code for “walking somewhere”) in different states was solved by introducing parameterized skills, as shown in Fig. 13b. The figure looks similar to Fig. 13a: on the higher layer are states with internal state machines, but the lower layer is different. It takes parameters from the higher layer and has no internal state machines.

Together, in the both graphs depicted in Fig. 13, there were three layers. And that might be enough for the examples discussed above. To be able to make both long-term and short-term decisions, the different behavior states were distributed over five layers (cf. Fig 14):

Roles define the general behavior of a robot during the whole game, e.g. “player1” or “goalie”, but also “defensiveGoalie” or “offensiveGoalie”.

Game-States are general states of the robot such as “waiting for a kickoff”, “waiting for a pass”, “searching the ball”, “playing the ball”, etc.

Options are long-term intentions, plans such as “passing the ball”, “kicking the ball to the goal”, “return to goal”, etc.

Sub-Options are short-term intentions, plans such as “kicking the ball forward to the goal”, “position in the left center of the own goal”, etc.

Skills are basal capabilities, implemented in decision trees. They select gaits or kicks, and they are purely reactive, without internal states. They are parameterized from the sub-options layer.

All the circles in figure 14 represent states with internal state machines that have corresponding internal states to the states on the next lower layer. The execution of the whole architecture starts at the first layer down to the fifth. In each layer, only one state is executed at the same time. The active state will determine the following state for the next lower layer. Only the skills initiate real motions of the robot.

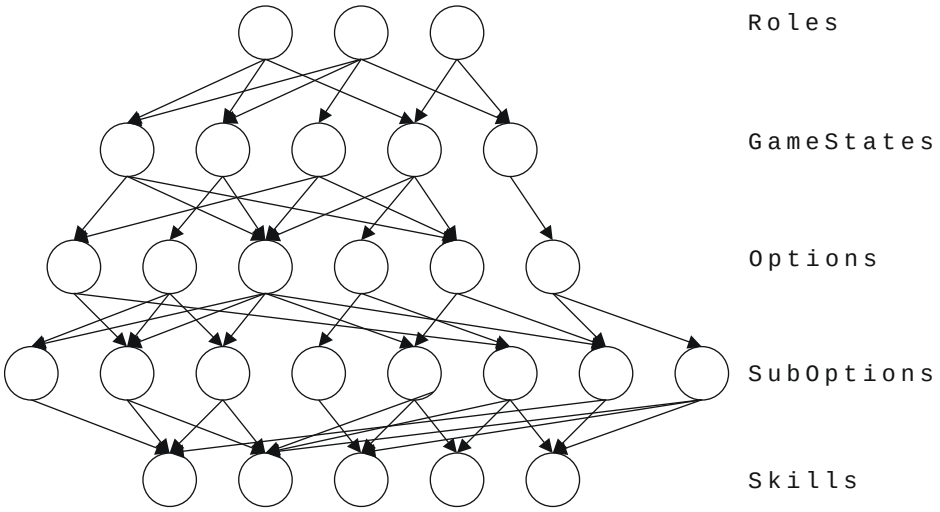

Figure 14: A five layer state machine approach as used for the RoboCup in Seattle with possible execution paths.

We tried to make every behavior on every layer independent from any context. That means, that they all have no prerequisites to be satisfied. A sub-option that shall kick the ball somewhere does that even if the ball is two meters away. In such a case, it may not be useful to execute that sub-option, but if the state is reached by mistake, the robot nevertheless will do something useful. The advantage of that method is that it is very easy to add or exchange behaviors, because they are independent from each other (within a layer).

6.1.4 Problems of the Architecture

Because we did not have enough time before the competition, we could not fix two big problems of the architecture:

1. The states in the internal state machines were unequivocal corresponding to existing states on the lower layer. That resulted in having only transitions and conditions between that states. If, for instance, it should be distinguish between a “turn left” and a “turn right” skill on the sub-option layer, we could not use a parameterized skill “turn”, but had to introduce a skill “turnLeft” and a skill “turnRight”, because only states corresponding to existing skills were allowed in the internal state machines for sub-options. As a result of that issue, a large number of states was required to implement the challenge 2.

   A solution of that problem would be to allow more states in the internal state machines than there are on the lower layer, having more than one state that selects the same state on a lower layer.

2. Because of the fixed five layer design a variety of states had to be introduced that do nothing but selecting a single state on a lower level. For example, the game-state “Wait-

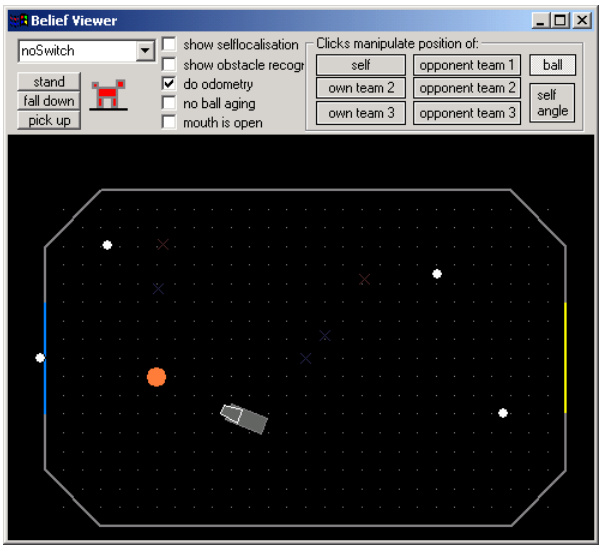

Figure 15: The BeliefViewer dialog.

ForOwnKickoff", does nothing but executing the "StandAround" option that again selects the "StandAround" sub-option that furthermore selects the "Stand" skill.

A solution for that issue would be to allow shortcuts.

6.2 Debugging Tools for Behavior

The hybrid software architecture of the GermanTeam 2001 allows to develop and test much of the behavior architecture as well as individual behaviors inside the DogControl environment. Two tools were decisive for that:

6.2.1 BeliefViewer

The dialog (Fig. 15) was originally designed to display world models exclusively, but it was enhanced to allow for direct interaction. To test the behaviors in specific situations the world model can be modified with the mouse. One can change the positions of other robots, the ball, and the player itself. The resulting world model is stored in the shared memory, where it is read from the HControl object running on the PC (cf. Fig. 11).

In addition, the movement of the robot caused through the behaviors could be simulated by the dialog. By reading the motion requests from the shared memory, where they were stored by HControl, and by employing the odometry table (cf. Sect. 3.4) to simulate motions in the world model, an idealized simulation of the robot is realized. Although the tool was used very often, it cannot replace a testing on the field.

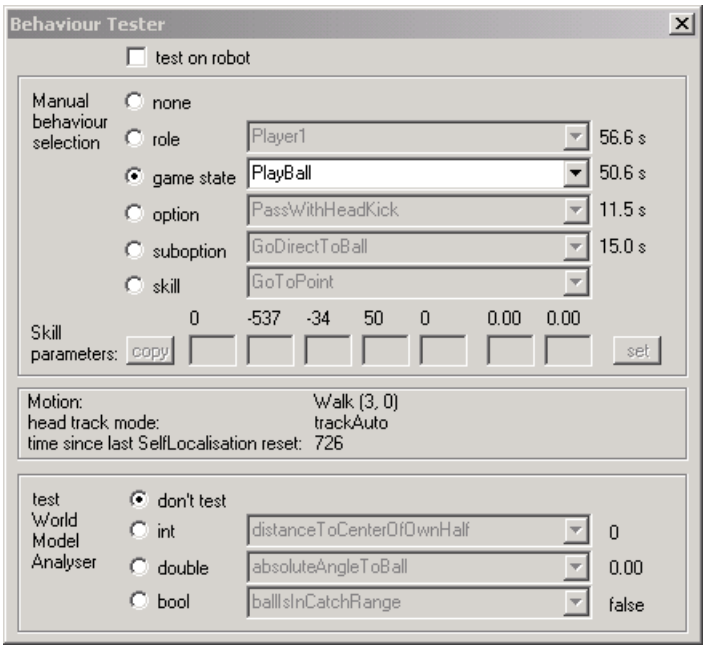

Figure 16: The BehaviorTester dialog.

6.2.2 BehaviorTester

To test and debug such a complex behavior architecture, the BehaviorTester dialog (Fig. 16) was developed. It can display the currently selected states on each layer, the duration, how long the states have already been executed, the motion request that resulted from the last execution of the architecture and much more. To test a specific behavior, one can select behaviors on any layer manually, the execution begins then from that state downward. The dialog sends behavior requests to the local shared memory or the shared memory on the robot. These behavior requests are taken from HControl and considered during the execution. So thanks to employed software architecture, debugging of behavior on the robot and on the PC is the same.

6.3 Implemented Behavior

In contrast to the RoboCup in Melbourne, we used the same program for both the soccer competition and the three the challenges in Seattle. Thereby, we could use a lot of the lower parts of our behavior architecture both for the games and for the challenge.

6.3.1 Contest

For the soccer contest we played with three different types (roles) of players: a goalie, a striker (“player1”) and a defender (“player2”). As we noticed during the German Open that it is important to play differently against different teams, we defined numerous roles to realize several

strategies. So we had a “player1” role, an “offensivePlayer1” role, and a “defensivePlayer1” role that differ in strategic decisions on the role layer in the architecture.

Goalie. Much of the time we spent on the goalie. It stays inside its goal, until the ball comes nearer than a certain distance. Then it walks to the ball and tries to clear it. If the ball is far enough away (from the own goal, not from the goalie) it returns backward to its own goal. When the ball is seen, but it is too far away, the robot walks sideward inside its own goal in the direction where the ball is seen on the field.

Field Players. During the competitions in the last years, all robots ran to the ball and obstructed each other very often. To avoid that this year, we tried to distribute the robots on the field, and to let the both field players pass the ball to each other. Therefore our striker (“player1”) stayed always in the opponent half of the field, even if it saw the ball, waiting for a pass by the “player2”. We thought that it would not be useful if three robots try to clear the ball out of the own half. The defender (“player2”) went to the own half if it did not see the ball, otherwise it tried to help the striker.

Kicks. As we were not able to let the robots dribble the ball, we always tried to kick the ball. We had different kind of kicks. At first, our head kick, in which the robot throws itself with the whole body on the ball, which is very effective, but leads to attrition of the hardware. The head kick was only used, when the robot detected opponent robots within a 50 cm radius around the ball. If there were no opponents near the ball, we tried one of four other kicks. They have in common that they require the ball to be grabbed (between the forelegs and the head) before. If the ball was successfully grabbed (which unfortunately did not happen very often), the robot could kick the ball sideward right and left, forward and even backward (using a bicycle kick).

Results. The three major reasons for not reaching the quarter final were:

- Although the motions of our robots were quite fast, it always took our robots longer to reach the ball than our opponents, because the transitions between the motions were too slow, and they were very unstable, which sometimes even lead the robots to falling down.
- During the first two games, we used the self-localization approach described in section 5.1 that unfortunately did not work at all. The estimated pose of the robot was always jumping, which confused the state machines used to control the robot. Therefore, we lost both matches. In the third game, we employed the Monte-Carlo localization method. We won the match against the team from the University of Washington, because against this newcomer, we had at least equal localization capabilities, but superior motions and behaviors.
- During the round robin, we competed against two very powerful teams: the world champion from the University of New South Wales and the team from Osaka that achieved the best result against the world champion during the whole competition.

6.3.2 Challenge

The challenge consisted of three tasks: the goalie challenge, the localization challenge, and the collaboration challenge. The challenges were tackled by implementing further roles for the robots, employing the same behavior architecture as during the contest: the “challenger1”, the “challenger2”, the “challenge3Player1”, and the “challenge3Player2”.

Goalie. The first task in the goalie challenge was to go from the fixed starting position to the goal as quickly as possible. We tested different approaches to reach the goal, and we decided that walking sideways to the goal is the fastest possibility as minimal turning is required.

After reaching the correct position in front of the goal, the goalie changed to an extremely wide stance. This stance was developed only for this challenge as it is wider than the normal stance used in the games. While waiting for the ball, no head motion is performed. Instead, the camera is fixed on the center point of the field to see the ball as soon as it is shot on the field. After the incoming ball has reached a specific distance, the trajectory of the ball is interpolated to decide whether it is necessary to move to one side to defend the goal. If it was decided that it is necessary to move the robot, it quickly falls to one side and makes one side step. This also required the development of new motions only used for this challenge.

The last part of the challenge was to clear the ball out of the penalty area after the successful defense. We did not write new code for this part but used the code of our standard goalie.

In the competition, we succeeded in defending the goal, but unfortunately we did not clear the penalty area before the time limit.

Localization. In the second challenge, the robot had to reach five predefined positions on the field as fast as possible, but in less than three minutes. As the number of positions reached had a higher importance in the assessment, we decided to focus on precision rather than on speed.

The employed Monte-Carlo approach provided a quite accurate position. It was more stable than the self-localization method used in the first two games (cf. Chapter 5). We improved the accuracy by slowing down the motions of the head, and by scanning only for the landmarks.

After starting the robot by pressing the head button, a short pause was implemented to give the localization engine enough time to calculate the start position. Then, the traveling-salesman problem for $1 + 5$ nodes was solved, and the shortest path for visiting all five locations was computed, starting at the current position. Our implementation was able to read the five positions from a file on the Memory Stick as described by Sony in the original version of the challenge.

We used an omni-directional walk to be able to reach the positions without the need of on-spot-turns. Therefore, the robot only needed to turn to see enough landmarks when it was getting too close to the border while looking at the outside of the field. The robot walked as fast as possible to the next target until it assumed to be within a certain range around the goal position. Then, it paused and stood until the position remained correct for a given period of time before it nodded the head to indicate that it assumed to have reached the correct position. Afterwards, it headed off for the next position. If the computed position left the range around the target location before the time threshold, e. g., because of a correction by the self-localization method due to new sensor readings, the robot corrects its position using a slower gait. To ensure that the robot

will not jump around a single position forever, the number of tries was limited to three, and after reaching that limit, the signal was given anyway.

At the RoboCup challenge in Seattle, the GermanTeam reached the fourth place by covering four of five targets.

Collaboration. In the third challenge, two robots had to perform a one-two that should be finish by kicking a goal. As the GermanTeam is currently not able to dribble the ball, we developed a gait, in which the robot carries the ball. The first challenger grabbed the ball, walked forward with the ball until it reached the center of the own half. Then, it kicked the ball over the center line. Afterwards, it walked to the left center of the opponent half, and stayed there, until the ball was returned. Unfortunately, during the first pass the ball hit an obstacle. In such a case, the robot should try a second pass, but actually that did not work well.

The second challenger first went to the right center of its own half. There, it waited until the ball passed the center line. Then, it should grab the ball, and passed it back to the left opponent corner of the field, hoping that the pass will arrive far enough in the opponent half. But as the pass by the first challenger failed, and the second try did not arrive far enough behind the center line, the second challenger had problems to grab the ball, because it was not allowed to grab a ball that was not located in its own half.

However, although our approach failed in the third competition, there were only a few teams that performed slightly better. Therefore, we still reached one of the better results.

Results. The GermanTeam reached the sixth place in the first challenge, the fourth place in the second, and also the fourth place in the third challenge. As a result, the we reached a respectable fourth place in the whole challenge.

7 Conclusions

Although the GermanTeam lost two games in the round robin, the members of the team are satisfied with what they have reached. It took only two months to introduce the new members of the GermanTeam, which is a good start-position for the development of the GermanTeam 2002. A powerful system and debugging architecture, useful algorithms for object recognition and self-localization, and a promising behavior architecture are a good base for the next year's project.

There is little experience with the cooperative work in the GermanTeam (about 4 months from the very beginning until Seattle), but it appeared as a challenge on its own. Due to bigger geographical distances, the team could meet only a few times. All the other communication had to be done over a mailing list, a news group and a to-do list on an internal website. The source code was shared within a CVS repository accessible from the internet.

As many other student projects, the team had a few problems with responsibilities, communication, and sometimes the quality of the implemented solutions, but despite that problems, the cooperation developed into success.

From our experience we would propose to build further united teams in the Sony Legged Robot League (even in RoboCup at all). It combines the different competences of the participating institutions. Besides this allows more universities to participate. Moreover, such regional groups can be the base for regional championships.

8 Future Work

With its new team members from different universities, the GermanTeam has now enough manpower to tackle several problems in robot soccer.

8.1 Revising the Existing System

Although the GermanTeam 2001 was equipped with all features required to play soccer, for the lack of time, most of them were neither carefully tested, nor fine-tuned. Therefore, all modules will be conscientiously revised and tested, before they will be integrated into GermanTeam 2002.

8.2 Self-Localization without Global Landmarks

During the last meeting of the Sony Legged League in Seattle, it was discussed to remove the global landmarks from the border of the soccer field. Although it seems that the new rules will not follow this idea, localization without global and unique points of reference is an interesting topic. It is not clear whether the GermanTeam 2002 will include such a capability, because it seems to be a waste of resources to develop such an approach while the rules do not require it. However, the Monte-Carlo localization approach described in section 5.2 only needs a function that judges the similarity between the current sensor readings and the readings that would have been taken if the robot were at a certain location. If such a similarity function is found that, on the one hand, is fast, and on the other hand, only relies on, e. g., the recognition of the carpet and the teammates, the current Monte-Carlo approach will be able the determine the robot pose without the color marks.

8.3 Probabilistic Modeling

As the use of the Markov-localization was quite successful in the GermanTeam 2001, probabilistic approaches will also be introduced to model the location of the ball and of the opponents. In contrast to the field of self-localization, in which the sensor resetting approach by Lenser and Veloso (2000) can only be used if an estimation of a global pose can directly be derived from the sensor readings, the sensor resetting method seems to be a promising approach for the probabilistic modeling of the locations of the opponents and the ball in a robot-centric system of coordinates. In addition, work that was done on tracking people (Schulz *et al.*, 2001) can also be integrated in such an approach.

8.4 Communication

Wireless communication will be allowed in RoboCup 2002. As the shared memory approach is not compatible to the implementation of wireless transmissions in Aperios, the communication architecture has to be redesigned. A simple protocol that is based on Aperios queues, and that completely encapsulates the operating system functions has already been implemented. It will be the foundation for the architecture of the GermanTeam 2002.

8.5 Cooperation

The introduction of wireless communication is also a challenge for the behavior control of the robots. Obviously, as every robot can send its location to his teammates, a visual recognition of the other team members is no longer required[4]. However, it is an interesting question, which information besides metrical coordinates shall be exchanged between the robots, e. g., it is possible to generate a shared world model, integrating the sensor information acquired by all members of the team. Such a model can be the basis for recognizing the strategy of the opponents.

8.6 Formalization

The behavior architecture employed by the GermanTeam 2001 was very complex. As this complexity is necessary to play robot soccer, the approach will also be used in 2002, but the system will be formalized, e. g. on the basis of XML, to make the approach more handy.

8.7 Kinetic Modeling

Tools will be developed for kinetic modeling of the robot dynamics taking into account masses and inertias of each robot link as well as motor, gear, and controller models of each controlled joint. Based on these nonlinear dynamic models, computational methods for dynamic off-line optimization and on-line stabilization and control of dynamic walking and running gaits are developed and applied. These methods will be applied to develop and implement new, fast, and stable locomotions for the Sony four-legged robots.

Acknowledgements

The GermanTeam thanks the Sony Corporation for their professional support (their aid for our hardware problems and the help during the introduction of the new robots), the Deutsche Forschungsgemeinschaft (DFG) for funding parts of that project and the organizers of the RoboCup in Seattle for the travel grants. The new members of the team, the Universität Bremen and the Technische Universität Darmstadt, thank the team from the Humboldt-Universität zu Berlin and the Sony Corporation for the chance to participate in the league.

[4] As long as such a recognition is not part of a cooperative self-localization approach.

References

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, **26**, 832–843.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte Carlo localization: Efficient position estimation for mobile robots. In *Proc. of the National Conference on Artificial Intelligence*.

Lenser, S. and Veloso, M. (2000). Sensor resetting localization for poorly modeled mobile robots. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*.

Quek, F. K. H. (2000). An algorithm for the rapid computation of boundaries of run length encoded regions. *Pattern Recognition Journal*, **33**, 1637–1649.

Röfer, T. (1998). Strategies for using a simulation in the development of the Bremen Autonomous Wheelchair. In R. Zobel and D. Moeller, editors, *Simulation-Past, Present and Future*, pages 460–464. Society for Computer Simulation International.

Schulz, D., Burgard, W., Fox, D., and Cremers, A. (2001). Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*.