Cognitive Concepts in Autonomous Soccer Playing Robots

Martin Lauer

Institute of Measurement and Control, Karlsruhe Institute of Technology, Engler-Bunte-Ring 21, 76131 Karlsruhe, Germany

Roland Hafner, Sascha Lange, Martin Riedmiller

Institute of Computer Science, University of Freiburg, 79110 Freiburg, Germany

Abstract

Computational concepts of cognition, their implementation in complex autonomous systems, and their empirical evaluation are key techniques to understand and validate concepts of cognition and intelligence. In this paper we want to describe computational concepts of cognition that were successfully implemented in the domain of soccer playing robots and show the interactions between cognitive concepts, software engineering and real time application development. Beside a description of the general concepts we will focus on aspects of perception, behavior architecture, and reinforcement learning.

Key words: autonomous robots, reinforcement learning, behavior architecture, robot soccer

1. Introduction

The development of autonomous, intelligent systems has been in the focus of computer science, engineering, and cognitive science since many years. In a wide field of disciplines many techniques have been developed which can serve as building blocks for these kind of systems. Examples are logical systems for reasoning and representation, visual perception systems, machine learning, and cognitive architectures, among others. However, all of these techniques cover only small subsets of all abilities a fully autonomous system must be equipped with to be able to operate in a complex environment.

More than just assembling various modules from the before mentioned disciplines the development of autonomous, cognitive systems requires approaches that develop the system as a whole taking into account the many recurrent dependencies between individual modules. E. g. it is well known that approaches of learning and reasoning heavily depend on the representation of the environment which is provided by the perception process which again depends on the sensors and the physical setup of the system. As well, decisions made by a process of learning or reasoning cause changes of the system which might influence the perception process again. Hence, all modules of such a system including the physical setup interfere with each other in a complex manner. Among others, this argument is one of the key ideas in the debate of embodiment [1].

To foster the development of complex autonomous systems and to be able to compare different approaches the idea of creating soccer playing robots and to compare their performance in games has been brought up in the last decade [2]. The overall goal is to further develop techniques in robotics, artificial intelligence, and cognitive systems so that a team of autonomous humanoid robots is able to play successfully against the human soccer world champion team in the year 2050. Meanwhile, after more than 10 years of development, robot soccer has brought the ideas of an integrated development of cognitive techniques to a broad scientific community, fostered the development of techniques in perception, reasoning, and behavior generation for autonomous robots, and created many interesting insights into the nature of artificial cognitive systems.

In this paper we want to describe a selection of cognitive techniques that were developed by the robot soccer team *Brainstormers Tribots*, one of the most successful robot soccer teams, and embed them into the scientific context of cognitive systems. The paper will concentrate on the question of an adequate software and agent architecture, issues of perception and sensing, and reinforcement learning approaches to make the robots learn autonomously a reasonable behavior.

We will start with a survey of *RoboCup* and a brief introduction to our team *Brainstormers Tribots*. It is followed by a description of the software framework that we are using for our robots in section 3, the techniques of sensor fusion and world representation in section 4, our behavior framework in section 5, and reinforcement learning techniques for soccer robots in section 6.

2. Robot Soccer

2.1. RoboCup

The idea to make robots play soccer has been organized within an international scientific community named *RoboCup* which organizes yearly world championships and regional championships on several continents. To be able to compare the performance of different teams a set of leagues has been defined in which the robots differ in terms of autonomy, sensing, and robot hardware. By doing so, different aspects of robot soccer can be analyzed in parallel [3, 4]. These leagues include:

- the *soccer simulation league* in which the soccer robots are simulated agents in a soccer simulation environment. This league uses high-level abstraction of hardware, sensors, and actuators and focusses on high level multi agent strategies.

- the *small-size-league* which uses wheeled robots with a height of 15cm. The robots get their commands by wireless communication from an external computer that is connected to a fixed camera above the field which yields an eagle-eye-perspective of the situation. This way of sensing ensures that all agents share the same, complete knowledge of their environment.

- the *standard-platform-league* which uses fully autonomous, legged robots. All teams use the same robot hardware. The computational power of the robots and the cameras is very limited so that an important question in this league is to make algorithms work with bad sensory input and limited computational power.

- the *middle-size-league* which uses 80cm high, wheeled, fully autonomous robots with on-board computers and on-board sensors. The robots are not standardized like in the standard-platform-league so that the league combines software and hardware development.

- the *humanoid league* which investigates biped, fully autonomous robots. Since the control of biped robots is much harder than the control of wheeled robots the main focus is to achieve smooth, fast, and stable movements in bipedal running and kicking.

Besides the before mentioned leagues some more leagues have been introduced during recent years which focus on different applications than soccer playing, i.e. rescue scenarios, educational robots, and human-robot interaction. These applications are not further discussed here.

2.2. RoboCup Middle-Size-League

Since we want to focus on the middle size league throughout this paper we will describe the challenges of this league a little bit closer. The robots are equipped with different kind of sensors to perceive their environment and actuators to move themselves, dribble and kick the ball. Since the robots must be fully autonomous they are equipped with on-board computers for the interpretation of the sensor signals and the calculation of a reasonable behavior. To start and stop the robots remotely when the game is interrupted and continued they obtain start- and stop-signal via wireless communication from a referee computer. The robots of a team may also exchange information via wireless communication. However, remote control of the robots is strictly prohibited. A team consists of up to five robots.

Games are played between two teams on a soccer field of 18m length and 12m width (see fig. 1). The fields look like normal soccer fields. There are only a few adaptations to the needs of soccer robots, e.g. the ball is colored orange to simplify the process of visual perception. However, these adaptations are removed step by step.

The game is controlled by a human referee. The rules are adapted from human soccer and a halftime takes 15 minutes. There are fouls (e.g. pushing another robot), kick-off and free-kick procedures, corner kicks, goal kicks, and penalty kicks. Instead of performing a throw-in the ball is kicked in when it leaves the field on the sideline since the robots are not yet equipped with arms to throw the ball.

While the abilities of the soccer robots were very limited in the early days of robot soccer [3] important improvements concerning the perception of the robots, the dynamics of the games, and the cooperation between the robots could be achieved throughout recent years. The maximal driving speed has increased up to $3\frac{m}{s}$, the kicking strength up to $7\frac{m}{s}$ [5, 6]. The robots are able to dribble the ball [7] and to perform chip kicks. Most robots find a path to a target position even in a complex and dynamically changing geometric configuration [8, 9]. Cooperative strategies can be observed among many teams [10, 11, 12, 13, 14], pass playing becomes more and more usual. The team strategies are typically based on roles where one robot is designated

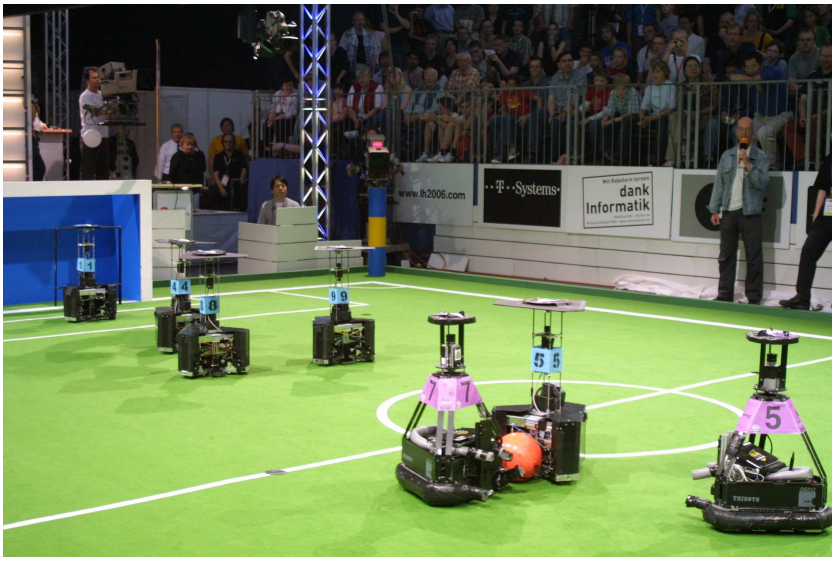

Figure 1: Picture from the final of RoboCup 2006 in the middle size league. During the last two years the color coding of the corner poles and the goals has been removed and the field size has been extended.

as goal keeper, one or two as attackers and the remaining ones as defenders. Some teams are using dynamic exchange of roles depending on the situation.

The teams compare the performance of their robots in games and in so-called technical challenges, in which the robots have to complete a predefined benchmark, e.g. showing cooperative game playing (2006), playing on a soccer field without colored landmarks (2007), or playing with an arbitrarily colored ball (2008). Furthermore, the teams are presenting their best scientific and technical improvements of the last year to the other teams. The best further development is rewarded with a special technical challenge award.

Since every team is allowed to build its own robots, the mechanical setup of the robots varies from team to team. However, like in an evolutionary process some solutions have been shown to be superior to others so that variants of these approaches have been realized by almost all teams.

For instance, holonomic drives [15] have replaced differential drives which have been used before[1]. Holonomic drives consist of three or more mecanum wheels, each of which can be driven individually by an electrical motor. By

[1]holonomic drives are also called *omnidirectional* drives

combining at least three wheels arbitrary movements can be realized, i.e. the robot is not restricted to forward and backward movements but can also move into any other direction and turn at the same time independently.

Similar to holonomic drives omnidirectional cameras are used by many teams [16]. Figure 6 shows an image of the soccer field taken by an omnidirectional camera. Omnidirectional cameras are built out of a digital camera with standard lens and a mirror of curved shape which is mounted above the camera. Hence, the robot obtains a 360 degree view with only a single camera. Although the advantages of this technique are evident it comes at the cost of low image resolution for far areas and a strong distortion of the image.

At present, the recognition of objects in the camera images is predominantly based on color information [17, 18]. However, many teams are working on approaches using shape and gray level edges [19, 20] since the color coding of objects like the ball will disappear during the next years. The information extracted from the images is used in sensor fusion and tracking approaches to obtain a consistent geometric model of the environment [21, 22]. The robots are able to determine their own pose on the field [23, 24, 25] and estimate the ball movement [26, 27, 28].

Besides cameras some teams are using additional sensors like lidar [29] or optical odometry [30]. Interestingly, although lidar presently plays a very important role in the domain of autonomous robots it has almost disappeared in robotic soccer during the last years so that the sensing of the robots has become more similar to human sensing. Reasons can be found in the fact that the dynamics of the games has increased so that the sampling rate of typical laser scanners is too low and that the game has become three dimensional, i.e. the ball leaves the ground during chip kicks. Hence, a single row laser scanner cannot detect the ball in critical situations.

Furthermore, the soccer robots are equipped with kicking devices. Typically, these devices are electrically or pneumatically driven levers which can kick the ball with a speed of up to 20 km/h and a maximal height of 3m above ground [5]. But not only the strength of kicking is important but also the ability to control the ball movement accurately. E.g. to play a pass it is inappropriate to execute a chip kick but it is better to play a flat kick with moderate strength and to control kick strength and kick direction accurately.

For improved ball handling most of the robots are also equipped with "dribbling fingers", flexible foam fingers which keep in contact with the ball throughout dribbling so that the ball does not roll away so easily. Although

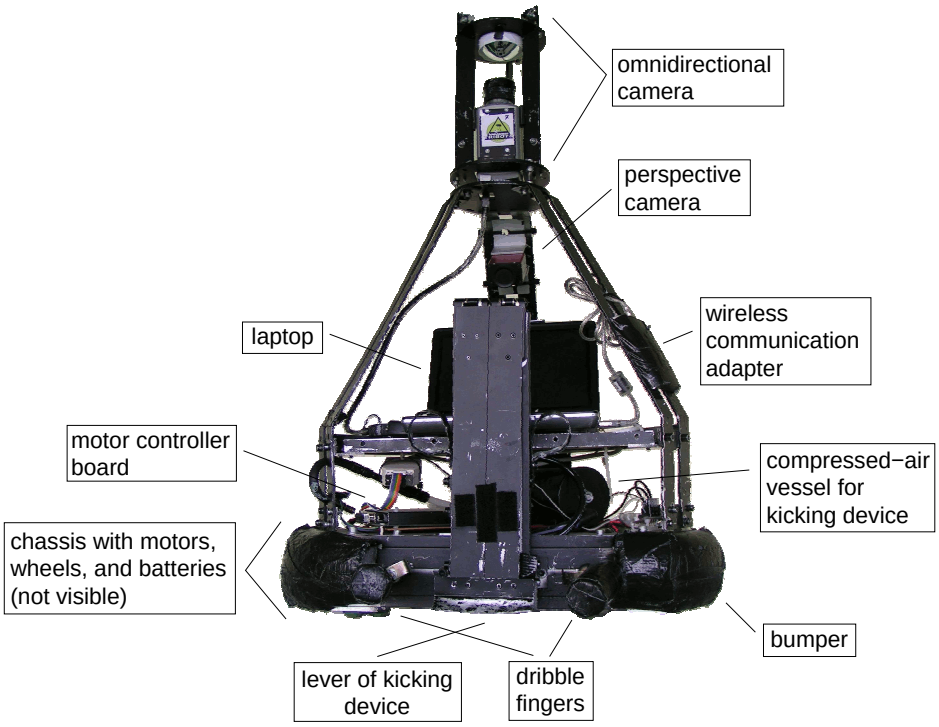

Figure 2: A robot of the team Brainstormers Tribots without paneling exhibits the main components of a soccer robot. The image shows the state of the year 2007.

these devices are passive and only put a small force on the ball they have been shown to improve ball handling considerably. Some teams developed active dribbling devices that are able to control the ball even if the robot is moving in backward direction [31].

While the before mentioned robot equipment is similar in almost all teams, the robots of different teams differ in the overall configuration of these devices. As example, the configuration of the team Brainstormers Tribots is shown in figure 2. Since the physical properties of a robot limit the cognitive abilities in scene understanding and behavior execution the co-development of both, the mechanical setup of the robots and the algorithms and concepts of cognition is one of the key issues in robot soccer and is one of the major differences to classical forms of research.

2.3. Team Brainstormers Tribots

One of the teams participating in the RoboCup middle size league is the team *Brainstormers Tribots* which has been initiated in 2002 in our research group. The team is composed out of master, bachelor, and Ph.D. students and the number of members varies between eight and fifteen. It is one of the most successful RoboCup teams of recent years becoming world champion in 2006 and 2007 and winning the technical challenge in 2006, 2007, and 2008.

In parallel to developing the robot hardware depicted in figure 2 the team was creating the control software containing software modules for visual perception, geometric and dynamic representation of the environment, robot behavior, inter-robot communication, hardware drivers, and monitoring. The growth of the control software from 40,000 lines of code to 150,000 lines of code within six years reflects the increasing complexity of the robot behavior, the requirement of better perception and representation of the environment, and the desire to increase the autonomy of the robots incorporating self-surveillance and self-calibration approaches [32].

Located at a machine learning research group the major scientific focus of the team was on developing autonomous behavior based on learning approaches, especially reinforcement learning [33]. Furthermore, the domain of fusing sensory information and to build a consistent internal representation of the robot became a large field of scientific work. In the subsequent sections of this article we will show some major results from these areas of reserach.

3. Software Architecture

3.1. Software Framework of a Single Robot

While we introduced the hardware setup of a typical soccer robot in the last section we will focus on the software, now, and introduce the framework used by the team *Brainstormers Tribots*. As mentioned before the control software runs on an on-board computer, i.e. a laptop, and has to consider input and output from several sensors and actuators. Figure 3 shows the information flow between these devices. While the on-board computer obtains the camera images directly from the cameras, the motors and the kicking device are actuated by a motor controller board which communicates with the on-board computer periodically. The motor controller board implements PID controllers for motor control and adapts motor voltages autonomously.

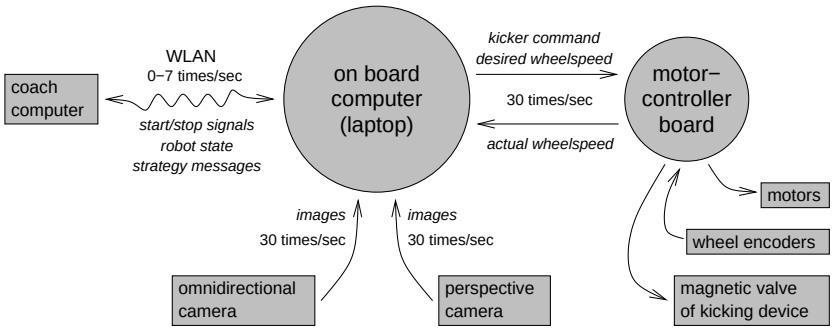

Figure 3: Illustration of the devices incorporated in a soccer robot. While the motor controller board implements the actuation of the motors and the kicking device the cognitive processes of perception and behavior generation are executed on the on-board computer. The latencies as well as the frequencies of communication differ from device to device.

Therefore, from the perspective of the on-board computer it can be interpreted as a complex actuator with a certain degree of autonomy that manages motor control upon desired wheel velocities.

The on-board computer can communicate via wireless communication with an autonomous coach computer. It allows to transmit information about the strategy of the team, the present situation on the field, and intentions of teammates.

A software framework that is able to deal with such a heterogeneous structure of devices, latencies, and communication frequencies and that can be used to generate complex behavior must meet the requirements of three domains:

- it must carfully design the asynchronous structure of the devices and model their individual latencies, exhibit a clear control flow so that the temporal dependencies are well-defined, and must meet real-time constraints, i.e. the computation time of each software component must be limited. In our case, the software was designed to evaluate 30 camera frames per second and to communicate with the motor controller every 33 milliseconds.

- it must be tractable due to the ideas of software engineering. Since the code size easily exceeds a hundred thousand lines of code and since it is developed in teamwork over several years it needs strict modularity with reliable interfaces, easy interchangeability of modules, and real-time monitoring, logging, and debugging techniques

- it must fulfil the demands of a cognitive architecture in the sense that it allows the integration of different techniques of cognitive modeling like learning, planning, reasoning, perception, etc. In particular, to apply reinforcement learning it was important to model the environment as a discrete time Markov decision process (MDP) [33].

To support all these requirements we developed a modular software structure with five software components. Each component solves one subtask which is specified by its input-output-relationship (see fig. 4).

- the **vision component** grabs images from the cameras, extracts objects from the images and provides geometric information about the objects in robocentric coordinates. It encapsulates all tasks related to the camera, its geometric properties, and image interpretation.

- the **behavior component** generates reasonable driving commands for the robot. It encapsulates the robot behavior and the soccer strategy. It is described in detail in section 5.

- the **controller and drivers component** communicates with the motor controller and encapsulates the kinematic model of the robot. It provides information about the desired driving speed and the actual speed measured by the wheel encoders.

- the **world model** which collects preprocessed sensory input from the other components (e.g. object positions from the vision component, actual robot velocities from the controller and drivers component), integrates and fuses it to obtain a consistent view of the environment. The information is represented on a geometrical and dynamical level, i.e. in terms of positions on the field and velocities. We avoid symbolic respresentation at this point since we believe that an adequate symbolic representation is closely related to the robot behavior and different strategies might require different symbolic representation.

- the **control and communication component** which manages the control flow, calls the update methods of the other components, and is responsible for the communication with the coach computer. It implements a perception-action-loop that is executed 30 times per second.

The interfaces of all components are designed according to the *design-by-contract*-principle [34], i.e. the syntactical interfaces are supplemented by

preconditions on the information available for the component, postconditions on the information provided by the component, general obligations, and constraints on the maximal computation time. Although we are not using automatic assertion checking in our software these design principles helped to implement code that preserves temporal dependencies, that can be executed in real time, and that allows to combine software moduls developed by different programers in different years. Alternative implementations for each component can be plugged into the software framework easily without changing the existing code. The specification of the components is strong enough to define the information exchange between them, however, it is weak enough to allow the implementation of different cognitive approaches and techniques.

While the control flow is managed by the control-and-communication component, the data flow is managed by the world model. In contrast to a database [35], the world model processes its input by fusing all information to a consistent description of the present situation and providing information on a higher level of abstraction to the other software components. It is further described in section 4.2.

3.2. Communication and Coach Computer

The rules of robot soccer allow the robots to communicate with each other and with an external "coach computer" via wireless LAN. The coach computer must work autonomously without manual interference. It might be used for monitoring and team coordination. However, the communication bandwidth is not very large, the delays in communication vary between 50ms and more than 1s, and the reliability of communication is low, i.e. communication breaks down occasionally.

To enable the robots to communicate with each other the software framework contains a communication interface that allows to exchange typed and untyped information. The typed information exchange is used to transmit messages independent of the robot strategy, e.g. the ball position, the robot position, the state of the robot, etc. For some pieces of information like the robot state or start/stop signals the information exchange is safe, i.e. messages are copied by the receiver, while for dynamically changing information like the ball position no copying is used.

The untyped information exchange is used for messages which are related to the strategy. Like in a pipes-and-filters approach messages in terms of strings can be exchanged between the robots and the coach computer. The

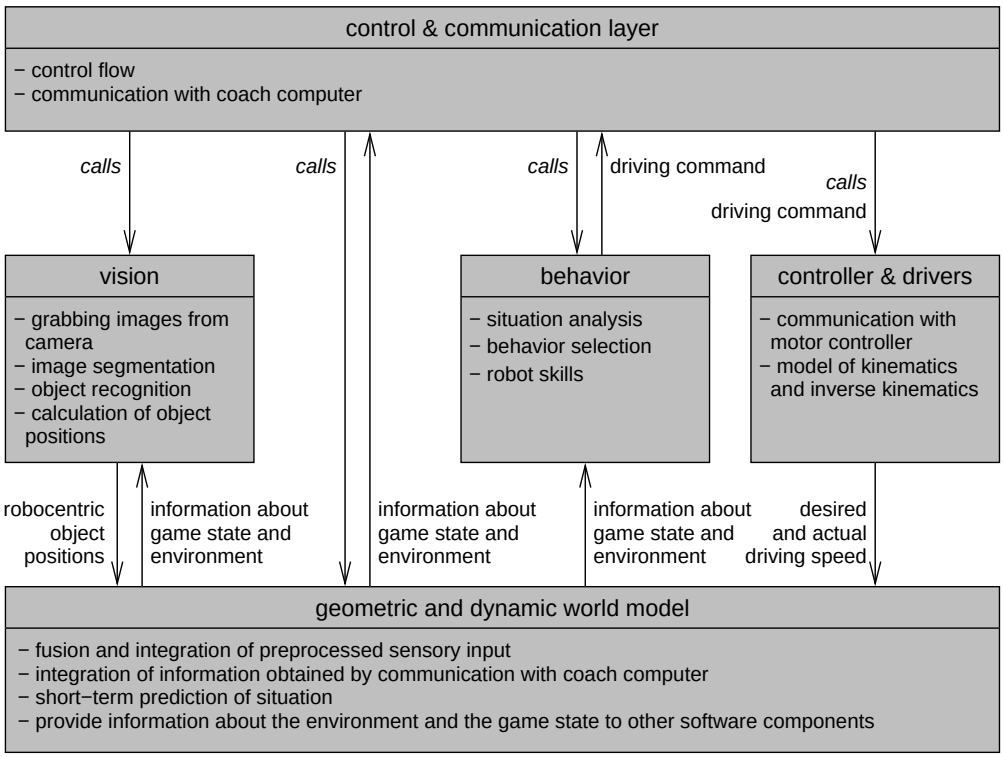

Figure 4: Structure of the main software components. Each of the components is equipped with a small, abstract interface which allows to replace alternative implementations easily.

interpretation of the messages depends on the strategy applied. Hence, the communication framework does not restrict communication to a predefined set of messages.

We use a star-like network topology where the coach computer is the central node of the network and robots are only communicating with the coach computer. The coach computer might interpret the messages it received from the robots or it might forward them to the other robots. The robots and the coach computer are exchanging messages between five and seven times per second.

4. Perception and Information Fusion

4.1. Visual Perception

One important capability of an autonomous robot is to sense its environment. In the domain of robot soccer optical sensors have been turned out to be the most powerful sensor type. As described in section 2 omnidirectional sensors are used by almost all teams including our team. Figure 6 shows an image of a soccer environment taken by such a sensor. The robot must be able to recognize relevant objects like the ball, the field markings, and the other robots. Since image interpretation must be done in real time the computational complexity of the algorithms is extremely limited.

To simplify the process of image interpretation some of the relevant objects are color coded: the ball is orange, the field is green, the field markings are white, and the robots are mainly black. Hence we could use low level procedures like thresholding in the HSI-color space to detect regions of interest for object recognition. To be able to play under the conditions of mixed sunlight/artificial light with varying brightness we additionally use an adaptive auto exposure and auto white balance procedure.

Additional heuristics are used to eliminate misclassified objects. These heuristics consider the shape of the colored areas, their size, and their relative positions. However, a small percentage of erroneously detected objects remains. The advantage of this kind of image interpretation is that it can be executed very quickly. On a 1 GHz subnotebook the analysis of one image with 640×480 pixels takes less than 10 milliseconds.

Finally, the positions of all objects found are transformed into robocentric coordinates projecting them onto the ground plane. Thereto, we use a geometric model of the catadioptric camera which is calibrated once before the games.

While the interpretation of the omnidirectional camera images is done in a similar way by almost all robot soccer teams we extended the optical system of our robots by a second camera to obtain a stereo system. In contrast to approaches which use two omnidirectional cameras of the same type in a stacked setup [36, 37, 38] we combined the omnidirectional camera with a perspective camera that observes the area in front of the robot. Figure 5 illustrates our setup [39]. The advantage of this setup is that we can combine the 360° view of the omnidirectional camera with the long and narrow field of view of the perspective camera. Similar to peripheral vision in the perception of humans the omnidirectional camera offers information of a large area with small image resolution while the perspective camera provides higher resolution images with smaller aperture angle similar to foveal vision. Additionally, the robot can calculate the distance to objects in the overlapping part of the fields of view of both cameras.

The three-dimensional position of objects can be obtained analyzing the epipolar geometry of the camera system. In contrast to standard stereo camera systems we have to consider the strong distortion of the omnidirectional camera and the different resolution of both images in the overlapping part of their fields of view. Figure 6 shows a pair of images of the stereocamera system. Due to the special mounting of the two cameras the epipolar lines in the omnidirectional camera are radial lines while the epipolar lines in the perspective camera are lines which intersect in a point below the image.

For the task of soccer playing full three-dimensional perception is not necessary since we can assume that objects like robots, field markings, and the goals are standing on the floor. Only the ball might leave the ground so that we can restrict computationally intensive stereo matching approaches to this object category. Thereto, the ball is detected in both images of a stereo image pair using color segmentation as described above. After determining the center of the segmented areas we apply depth calculation based on a trigonometric model of the camera projections. Taken together the time needed to analyze the perspective camera image and to calculate the depth estimates, the stereoscopic camera requires approximately additional 7 milliseconds per image pair so that we can analyze up to 30 frame pairs per second.

4.2. Information Fusion

Due to the limitations of the optical sensor, its restricted field of view, and the imperfection of the object recognition approaches, the information

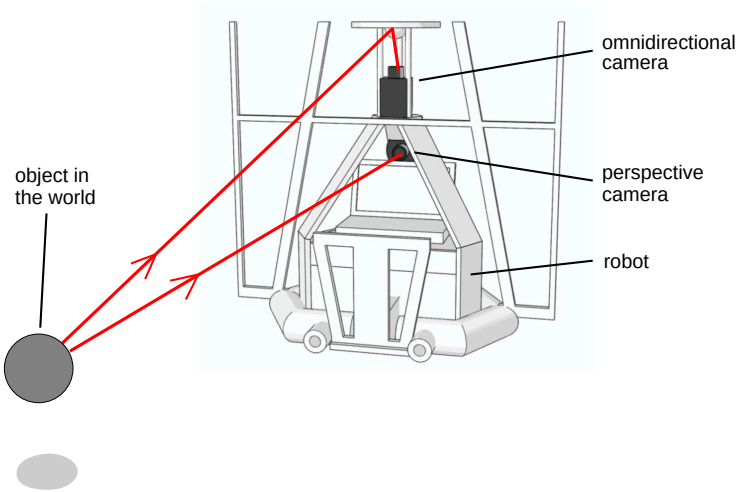

Figure 5: Sketch of the stereo camera system combining an omnidirectional camera (on top) and a perspective camera (below). The red lines show the optical path.

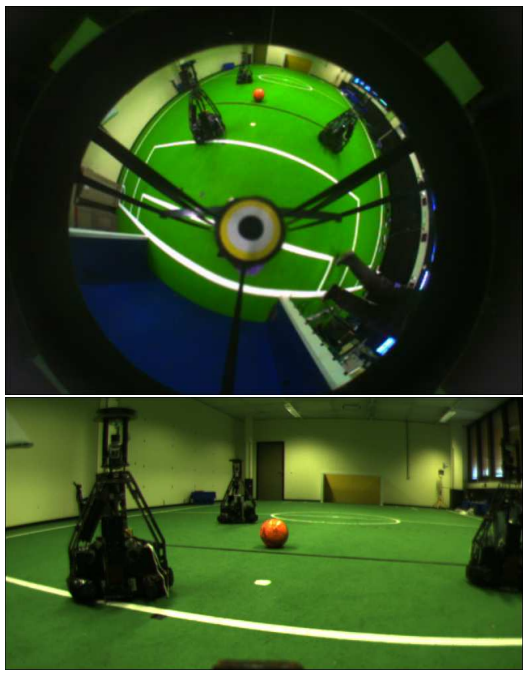

Figure 6: Pair of images from the omnidirectional camera (above) and the perspective camera (below).

obtained from the video sensor is incomplete, noisy, and sometimes even erroneous. It only shows a spotlight of the present situation without taking into account the dynamics of the scene. Hence, it does not deliver any information about the movements and the interactions of the objects. Moreover, some objects might not be recognized due to occlusions or due to the limited field of view. Therefore, a behavior which relies solely on the information extracted from the present camera image necessarily would be quite simplistic. Precise interaction with moving objects, e.g. the ball, would be impossible.

To overcome these limitations we developed algorithms which fuse, integrate, and filter the information obtained from the camera images and which build a consistent geometric and dynamic model of the situation on the field. We name this model the *world model* of our robots. Note, that our world model acts on a geometric and dynamic level, not on a symbolic one. The interpretation of geometric configurations is left for the behavior component of our software. By using a geometric and dynamic world model we can overcome the before mentioned limitations and predict the development in future, at least for a short-term period.

The process of information fusion and extraction of dynamic information consists out of several subtasks including:

- estimation of the pose of the ego robot (self-localization)
- estimation of the angular and linear velocity of the ego robot
- estimation of the ball position and its velocity
- estimation of the position of the other robots and their movement
- recognition of stuck situations in which the ego robot is blocked by another robot

4.2.1. Self-Localization

Two sensors are available for the robot to solve the task of self-localization: the odometry, i.e. the movement of the wheels measured by wheel-encoders, and the camera images. None of both sensors provides the full information. On the one hand, the wheel-encoders are very noisy due to slippage and kinetic effects so that dead reckoning easily loses track of the actual position, on the other hand, the images of the camera often do not provide enough information to determine the position uniquely. Therefore, we combine both ideas for our self-localization approach.

The key idea of visual self-localization is to use landmarks, significant objects that can be recognized easily, and to measure the distance and the angles at which the landmarks are seen in the camera image. However, the soccer field does not provide an adequate number of landmarks. Therefore, we are using the white field markings instead. Although, white lines can be extracted easily in the camera images, they are not uniquely identifiable, the robot does not know which line segment it actually sees.

We modeled the task of self-localization as an error minimization task by defining an error function which takes its minimum at the most likely robot pose. Given a set of positions $\vec{s}_i$ in robocentric coordinates at which white points in the image have been found, the problem is to solve:

$$\underset{\vec{p},\phi}{minimize}\ E(\vec{p},\phi) := \sum_{i=1}^{n} \rho(d(\vec{p} + \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \vec{s}_i)) \tag{1}$$

where $\vec{p}$ denotes the position of the robot and ϕ its orientation. $\rho(\cdot)$ is a robust error measure, e.g. the error measure of an M-estimator [40], $d(\cdot)$ is the function that measures the distance of a point $\vec{x}$ on the soccer field to the closest field marking. The distance function d uses a geometric model of the soccer field. Hence, $E(\vec{p},\phi)$ measures the error between the model of white markings and the white points seen in the camera image assuming robot position $\vec{p}$ and orientation ϕ. In case of $\vec{p}$ being the actual position and ϕ the actual pose the white points found in the image match the model very well so that the error E is small. Figure 7 illustrates this case.

Using a nonlinear minimization approach and starting from an initial guess which is not too far away from the actual pose we can find the minimum of E efficiently. However, in some situations the structure of the line points that are recognized in the camera image is poor, i.e. not enough line points are found or the line points all belong to parallel lines. In these cases the minimum of E is not distinctive and we cannot determine the pose uniquely. To overcome these problems we developed an approach that estimates the reliability of the pose estimate exploiting the local curvature of the error term E around its minimum. If the curvature is large the minimum is distinct and the estimate is reliable.

Taking together the pose estimate obtained from the white lines in the camera image and the way covered since the previous camera image measured by the wheel encoders we can complete the self-localization approach using a Kalman filter [41]. The initial pose used to find the minimum of E is

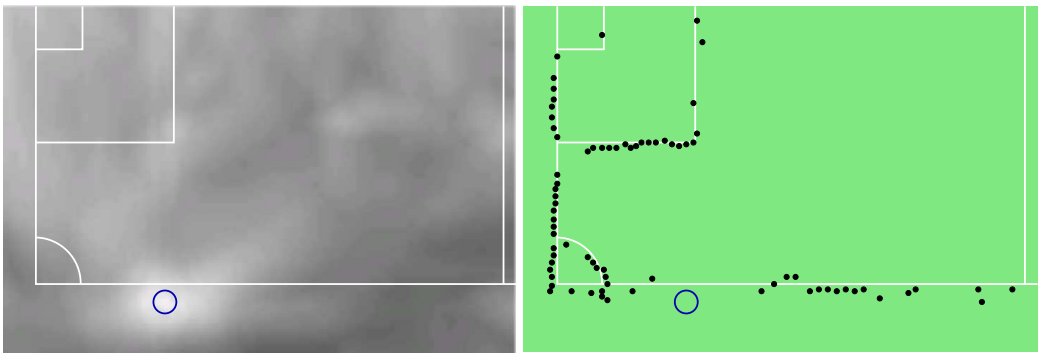

Figure 7: Illustration of the self-localization approach. Based on white field markings recognized in the camera image the robot pose is estimated. The figure on the left shows a greylevel plot of the error term E which is minimized. For each possible robot position the grey level indicates the value of E assuming an optimal robot orientation. Bright colors refer to small values of E, dark colors to large values. The optimal position, i.e. the minimum of E is indicated by the circle. For this optimal position estimate, the figure on the right shows a projection of the white points recognized in the camera image (black spots) onto a map of the soccer field. The black spots are very close to the field markings of the model.

taken from the predicted pose of the Kalman filter while the optimal pose found by minimizing E is used for the innovation step. The details of our self-localization approach are described in [42].

The self-localization approach is able to track the pose of a robot very reliably and accurately once an initial pose is given. The error in position determined in experiments was, on average, less than 20 cm on a field of 12 m length and 8 m width. Moreover, analyzing the value of E it is also possible to distinguish situations in which the robot knows where it is (low value) from those situations in which it lost track of its position (high value). We use this information for monitoring. It is also possible to relocalize the robot autonomously in the latter situations using global search instead of local search to minimize E. We provide an outline of such a relocalization procedure in [42] and we were able to demonstrate that it works successfully in practice at the RoboCup 2007 technical challenge.

The self-localization approach based on error minimization works efficiently. On a 1 GHz subnotebook we need less than 10 milliseconds computation time for one update. A particle filtering approach [43, 44] either needs a multiple of time or yields estimates with lower accuracy [25].

4.2.2. Ball Movement Estimation

Another important prerequisite for soccer playing is to know the position and movement of the ball. The cameras of the robot are only providing snapshots of the present situation without velocity information. Furthermore, depending on the position of the ball relative to the robot it is visible in the image of both cameras (omnidirectional and perspective one) so that we can determine its three-dimensional position, or it is only visible in one of the camera images (cf. figure 6).

To overcome this problem we developed an approach to estimate the position and movement of the ball [27, 39]. It estimates the ball movement projected onto the ground plane (x-y-plane) and the movement in the vertical direction (z-direction) and can also deal with mixed monocular and binocular observations.

The basic idea of motion estimation in the ground plane is to assume a linear ball movement, i.e. the ball movement can be described by a linear model of the form:

$$\vec{b}(t) = \vec{b}_0 + \vec{v} \cdot t \tag{2}$$

where $\vec{b}(t)$ refers to the ball position in the ground plane at point in time t, $\vec{b}_0$ is an initial ball position, and $\vec{v}$ the ball velocity. Obtaining a sequence of observations of the ball we can use linear regression to estimate the model parameters $\vec{b}_0$ and $\vec{v}$. In case of very noisy observations ridge regression [45] improves the results slightly.

To take into account that the ball velocity might change we use a temporal windowing technique, i.e. we only consider the latest observations for estimating the ball movement. Moreover, we integrated a consistency check that observes whether new observations fit to the present motion model. If not, we assume that the ball has been kicked or has collided with another object. In such a case we reduce the number of observations that are used for the regression analysis to the latest two and forget the older ones so that the adaptation of the velocity estimate to the new ball movement is speeded up.

Due to the special setup of the stereo camera system described in section 4.1 we do not always obtain binocular observations of the ball but sometimes only monocular. Hence, we have to deal with a mixed set of monocular and binocular observations. We modeled this scenario as an incomplete data estimation problem in which the distance of monocular observations from the observer is missing. We derived a maximum likelihood estimator for this

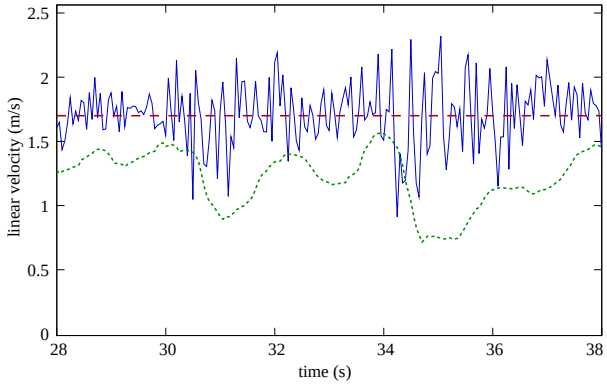

Figure 8: Example of the linear velocity driven by a soccer robot. While the dashed line shows the desired velocity which is constant over the whole time, the solid line shows the actual velocity measured by the wheel encoder. Interestingly, the velocity estimates obtained from the self-localization positions (dotted line) remain significantly below. The difference can be explained by slippage.

situation which is able to take into account both, monocular and binocular observations. It is based on an ECM algorithm [46] and alternately estimates the parameters of the motion model and the missing distances. Compared to an estimator which ignores monocular observations it yields significantly better results.

4.2.3. Egomotion and Other Robots

Furthermore, the robot must be able to determine the position and velocity of the other robots on the field and its own velocity. For both tasks we use similar algorithms as for the estimation of the ball velocity. Other robots are assumed to perform a movement with constant velocity so that we can use the same approach as for the estimation of the ball velocity. The only difference is that we cannot identify the other robots unambiguously so that we have to solve a data association problem first.

The estimation of the egomotion is also done using a regression approach. The position estimates obtained from self-localization are used as observations. Although the wheel encoders already provide information on the velocity of the ego robot an independent estimate is necessary since the odometry suffers from slippage. Figure 8 illustrates this phenomenon.

We are using a motion model with constant linear and angular velocity to estimate the egomotion of the robot, i.e. we assume that the robot moves on

a circular arc of unknown radius. Although this motion model is non-linear we can use a hierarchical approach to estimate all parameters with linear regression. Thereto, we estimate the angular velocity first. Once we know the angular velocity we can simplify the remaining estimation task and solve it analytically. The whole approach is described in [23].

Not only is the odometry-independent estimate helpful to create a reasonable behavior of the robot but also can it be used for self-surveillance of the robot. It sometimes happens that the robot wants to drive into a certain direction although an obstacle blocks its way. Thus, the robot gets stuck. While the wheels are turning the robot does not move. Although it will never be possible to avoid these situations completely, they are undesirable since they prevent the robot from executing its intentions and since the motors of the robot might get damaged. Comparing the odometry with the egomotion estimates it is possible to recognize stuck situations [23]. If the difference between both becomes too large over a longer period of time (e.g. 500 milliseconds), the robot is assumed to be stuck. To avoid false alarms we further exclude time intervals of acceleration from this analysis since the egomotion estimates are not as accurate as necessary during these intervals.

4.3. Latencies

On the basis of the geometric and dynamic world model the robots must determine their behavior. However, all information that is created within the information fusion process refers to the point in time at which the latest camera image has been made. In contrast, all decisions which are made by the robot will take effects at a certain point in the future. Since the environment is changing dynamically, the situation at this point in future will be different from the situation shown in the latest camera image. The time interval between the last camera image and the point until decisions take effect is between 150 and 200 milliseconds for our robots, depending on the actions that are taken by the robot. Assuming a camera frame rate of 30 frames per second this means that the effects of an action do not become visible until the fifth-next camera image. Meanwhile, a robot driving with $2\frac{m}{s}$ has covered a distance of at least 30 cm which is more than the diameter of the ball.

It is well known from the domain of control theory that systems with latencies are difficult to control and tend to overshoot. From the point of view of stochastic modeling they do not meet the Markov-condition which states that the system behavior in future only depends on the present state of the

system but not on past states or past actions. This property is very important for reinforcement learning approaches (see section 6) and it can simplify behavior generation a lot. Therefore, we tried to extend the geometric world model in such a way that it violates the Markov condition as little as possible.

The key idea is to use the model of the environment to predict the future configuration on the soccer field [47]. By doing this, the decisions of the behavior generating process are made upon the predicted state of the soccer environment for the point in time at which the decisions take their first effects. To predict the future pose of the robot we consider the latest desired wheel velocities and use an approximate motion model of the robot. The future ball position and the position of the other robots are predicted assuming constant linear velocities and taking into account interactions between the ego robot, the ball, and other robots.

5. Behavior Framework and Strategy

5.1. Behavior Framework

5.1.1. Requirements

The generation of complex robot behavior is one of the core tasks in developing autonomous soccer robots. Various scientific approaches on this topic have been proposed in recent years. On the one hand, cognitive architectures like belief-desire-intention (BDI) [48], production rule systems [49], and others have been proposed to generate emergent behavior. On the other hand, engineering approaches from robotics and control theory have been discussed to implement robot behavior in an iterative process like Brooks' subsumption architecture [50], motor schemas [51], and hierarchical state machines [52]. Due to the advantages of modular software frameworks behavior-based approaches [53] have attracted an increasing interest throughout recent years.

For the development of soccer playing robots we could rely on a well-established theory of soccer playing. Hence, the main focus of our approach was not on creating emergent behavior but on transferring existing soccer strategies to robots. Due to the highly dynamic, continuously changing environment it was essential to achieve a very lively robot behavior with low response times that reacts immediately on changes. Further important issues have been the creation of a modular software framework for behavior generation which can be refined incrementally, clear interfaces, and a transparent decision making process.

5.1.2. Basic Idea

We follow a behavior based approach that allows us to combine purely reactive behaviors with deliberative building blocks and learned submodules. Thereto we were inspired by ideas from both worlds, the behavior focused, highly reactive subsumption architecture, and cognitive architectures using explicit shared world models, planning, and deliberation.

The result is a combination of a BDI-like control loop with slim modules as building blocks that calculate both, their applicability and desired actions. As in the subsumption approach, the complex behavior of the whole system is made up by the interaction of a number of relatively simple behavior modules each of which implementing only a small sub-strategy. The behavior modules are organized in a hierarchical tree structure that allows abstraction and task decomposition.

These principles can be compared to human soccer playing. A soccer player is able to perform a set of basic skills like dribbling the ball, running to a target position, kicking, etc. which can be interpreted in our framework as the building blocks of the strategy. They can be combined to form more complex moves like executing a wing attack or a complex dribbling move. In the next level of abstraction a number of moves can be combined to a complex attack strategy or a complex defense strategy. On the highest level of abstraction complex sub-strategies are combined to the overall soccer strategy. Additional, intermediate levels of abstraction are possible.

The task decomposition which is implemented using a hierarchy of behaviors can be compared to the assignments a coach would give his players. The coach describes the desired moves and actions depending on distinctive situations. These situations can be characterized using boolean conditions. Once a move is started it will be persued until either a) the prerequisites to continue this move are violated (e.g. the ball is lost during dribbling) or b) the move has reached its goal (e.g. a dribbling has reached its target position) or c) another move of higher priority can be executed (e.g. a promising shot on goal).

Behavior modules[2] can be interpreted as partial strategies which map situations onto driving commands. For soccer robots driving commands consist out of desired linear and angular velocities and an attribute to control the kicking device. To avoid limitations on purely reactive strategies behaviors

[2]or, for short, *behaviors*

must be able to store information about past situations and plans in their internal memory and allow for randomized policies [54]. Therefore, behaviors can be modeled as partial, randomized finite state machines which take as input situations of the environment and yield as output driving commands. Their transition function is total and allows to update their internal memory while their output function is partial and yields reasonable driving commands only for those situations which are in the domain of the behavior.

Given a set of behaviors as building blocks we can build a strategy upon them applying a generic decision scheme inspired by the BDI interpreter outlined in [55]. The decision scheme is shown in algorithm 1. Due to the similarity to the BDI interpreter we use the nomenclature of the BDI architecture from here on naming all available behaviors the *options* of the decision process and naming the option selected for execution the *intention*. As mentioned before, each option can be interpreted as a possible plan while the intention is the plan that the robot actually executes.

Algorithm 1 Generic decision scheme inspired by the BDI interpreter

loop
  determine the present situation s
  filter applicable options $A = \{option\ o | s \in domain(o)\}$
  select one option i from A and make it the intention (arbitration)
  execute i
end loop

Note that the third step in algorithm 1 which selects the intention from all applicable options is not yet specified. Various arbitration strategies are possible like selecting options randomly or selecting the highest ranked option in a priority list. Interestingly, the latter arbitration strategy can be interpreted as the decision making process implemented by the subsumption architecture. However, from the point of view of practicability not a single arbitration strategy will be adequate for all cases. Therefore, we provided four generic arbitration schemes which cover typical cases of arbitration. These are (a) highest priority first arbitration, (b) finish plan arbitration, (c) random arbitration, and (d) sequence arbitration. A common property of these arbitration schemes is that the arbitration strategy is not situation-specific, i.e. given the set of applicable options A it selects its intention without considering the present situation s. By doing so, we obtain a strictly modular behavior concept in which the arbitrators do not have to know anything

about the semantics of their options nor about the prerequisites necessary for applying them but all information about what a behavior does and when it can be applied, is completely encapsulated inside the behavior. The four arbitration schemes are outlined in section 5.1.4.

Combining a set of behaviors using an arbitrator generates a robot strategy that can be interpreted itself as a behavior. Those composite behaviors can contribute to other arbitration processes as options again. Hence, we can build a hierarchical tree structure of behaviors with the leaves of the tree being the simple building blocks of the strategy and the inner nodes being composite behaviors implementing arbitration strategies. In contrast to many other architectures that use a fixed number of layers in the behavior generating process (e.g. a strategy layer, a path planning layer, and an obstacle avoidance layer) the proposed method allows hierarchies of variable depth and branching factor. It fosters the ideas of task decomposition (composite behaviors), incremental refinement (adding options to a composite behavior or expanding a leave node to a subtree), and abstraction on different levels. An example of a behavior tree is shown in figure 9.

5.1.3. Implementation

The ideas presented in the previous section have been implemented as an object-oriented software design pattern [56] for behavior creation. An abstract interface models the general structure of a behavior as finite state machine. It is composed out of:

- a state transition function, potentially randomized
- an output function to generate driving commands
- an invocation condition that specifies in which situations a behavior might be applied
- a commitment condition that specifies in which situations an active[3] behavior can continue its work
- a method *gainControl* that is called when a behavior becomes active
- a method *looseControl* that is called when a behavior becomes inactive

[3]a behavior is said to be active if its output function is called to determine the present driving command (i.e. if it is the present intention of an arbitrator)

Note that the domain of a behavior is represented by the invocation condition and the commitment condition. In the sense of design-by-contract [34] the user of a behavior must guarantee:

- to call the transition function periodically independent of whether the behavior is active or not, and
- to call the *gainControl* method immediately before the behavior becomes active, and
- to call the *looseControl* method immediately after the behavior became inactive

Vice versa, the behavior guarantees that its output function returns a reasonable driving command whenever

- its invocation condition is true and it has been inactive before, or
- its commitment condition is true and it has been active before

To meet these constraints is essential to guarantee consistency of the behavior generation process.

In contrast to other approaches we do not use trigger or termination conditions since a trigger condition can be expressed with the help of the invocation condition while the termination condition can be simulated by means of the commitment condition.

Since the behavior interface is chosen as small as possible it does not make any assumption on the technique which is used to implement the behaviors, so that it can subsume reactive behaviors, learned behaviors, as well as behaviors which are based internally on a deliberation process. Deriving from the abstract behavior interface we can implement concrete behaviors like Patrol, ApproachBall, DribbleAlongTouchLine, etc. (cf. section 5.1.5).

Generic arbitrators are also derived from the behavior interface. Common to all arbitrators is that they contain a list of options[4] and an index variable that indicates the present intention. They implement the arbitration strategy in their transition function while the output function just calls the output function of the present intention. Moreover, they are equipped

[4] i.e. a list of objects derived from the behavior interface

with generic invocation and commitment conditions to guarantee the consistency constraints mentioned above. "Generic" means that the invocation and commitment conditions of the arbitrators only depend on the invocation and commitment conditions of their options.

5.1.4. Generic Arbitrators

Four generic arbitrators have been implemented which will be described in this section. To be able to describe all arbitration schemes in a logical form we introduce the following notation. O denotes the set of all options of an arbitrator, i_t its intention at time t, *invoc* and *commit* its invocation and commitment condition, A_t the set of applicable options at time t, and s_t the present situation. $invoc_o$ and $commit_o$ refer to the invocation and commitment conditions of an option o.

Highest priority first arbitration. The highest priority first arbitration is an arbitration scheme that organizes its options in a priority list of fixed order. Among all applicable options it selects the one with the highest priority and makes it its intention. Denoting with $\succ$ the order on the options induced by the priority list, the selection procedure and the generic conditions can be expressed by:

$$A_t = \{o \in O | invoc_o(s_t)\} \cup \{i_{t-1} | commit_{i_{t-1}}(s_t)\} \tag{3}$$

$$i_t \in A_t \text{ with } i_t \succ o \text{ for all } o \in A_t \setminus \{i_t\} \tag{4}$$

$$invoc(s_t) = \bigvee_{o \in O} invoc_o(s_t) \tag{5}$$

$$commit(s_t) = \bigvee_{o \in O} invoc_o(s_t) \vee commit_{i_{t-1}}(s_t) \tag{6}$$

Note that the highest priority first arbitrator will interrupt the present intention when the commitment condition of an option with higher priority becomes true. Hence, it implements a highly reactive arbitration scheme similar to the subsumption architecture.

Finish plan arbitration. Similar to the highest priority first arbitration the finish plan arbitration scheme uses a priority list to organize its options. In contrast to the before mentioned arbitrator it does not allow interruptions of its present intention by options of higher priority. Hence, it guarantees that an intention is executed until its own commitment condition becomes

false. While the invocation and commitment conditions as well as the set A_t are calculated in the same way as described for the highest priority first arbitration (3), (5), (6) the selection procedure can be described as:

$$i_t \in A_t \text{ with } \begin{cases} i_t = i_{t-1} & \text{if } \mathit{commit}_{i_{t-1}}(s_t) \\ i_t \succ o \text{ for all } o \in A_t \setminus \{i_t\} & \text{otherwise} \end{cases} \tag{7}$$

If no previous intention i_{t-1} exists the first case in (7) drops. The finish plan arbitration allows to model decisions which cannot be altered afterwards even if another option seems to be more promising than the one chosen. A typical example is the behavior of an attacker executing a penalty kick. Once he has decided to kick to the left or right corner of the goal he should not alter his plan although the situation might change slightly. Note, that the finish plan arbitration might be interrupted itself if it is used as an option for another arbitrator. In this case the present intention of the finish plan arbitrator is interrupted even though its commitment condition is true. This case resembles the situation in a soccer game when the referee interrupts the execution of a penalty kick.

Random arbitration. Similar to the finish plan arbitration the random arbitration makes decisions that cannot be altered afterwards. However, it does not organize its options in a priority list but assigns probabilities $p_o > 0$ to them. Among all applicable options it selects one randomly. The arbitration scheme can be expressed as follows while the invocation and commitment conditions as well as the procedure to determine A_t do not differ from (3), (5), and (6):

$$i_t \in A_t \text{ with } \begin{cases} i_t = i_{t-1} & \text{if } \mathit{commit}_{i_{t-1}}(s_t) \\ i_t \text{ is chosen randomly with} & \\ \quad \text{probability proportional to } p_{i_t} & \text{otherwise} \end{cases} \tag{8}$$

The random arbitration scheme allows to randomize the robot strategy which is an important technique in multi agent domain to obtain an optimal strategy [54]. Note that random selection only considers options with invocation condition true so that the arbitration strategy can only guarantee that the actual selection probabilities are proportional to the pre-defined probabilities p_o.

Sequence arbitration. While the previously described arbitrators select one among several possible reactions on a certain situation, the sequence arbitrator implements a sequential execution order of its options. It allows to create moves built out of several stages of processing while every stage is implemented as a behavior. As example, during a wing attack in soccer a soccer player dribbles the ball (first stage), then turns towards the middle of the field (second stage) and finally passes the ball to a teammate (third stage). Implementing the three stages as individual behaviors we can use the sequence arbitrator to generate a temporal sequence of them.

The sequence arbitrator operates as follows. It keeps its present intention until its commitment condition becomes false. If the invocation condition of the next option is true at the same time it becomes the new intention of the arbitrator. Otherwise the sequence arbitrator stops executing the sequence of options by altering its own commitment condition to false. The execution of the sequence always starts with the first option. If we denote with i_t the present intention, with $i_t + 1$ the option subsequent to the present intention, and with o_1 the first option we can describe the arbitration scheme as:

$$i_t = \begin{cases} i_{t-1} & \text{if } \mathit{commit}_{i_{t-1}}(s_t) \\ i_{t-1} + 1 & \text{otherwise} \end{cases} \tag{9}$$

$$\mathit{invoc}(s_t) = \mathit{invoc}_{o_1}(s_t) \tag{10}$$

$$\mathit{commit}(s_t) = \mathit{commit}_{i_{t-1}}(s_t) \vee \mathit{invoc}_{i_{t-1}+1}(s_t) \tag{11}$$

Selection rule (9) specifies that the point in time of switching from one option to the next one is triggered by the commitment condition of the present intention. It is motivated by the idea that a behavior should be able to finish its task before the subsequent behavior becomes active. However, in some cases it is more convenient to let the subsequent option determine the change-over point, i.e. as soon as the invocation condition of the subsequent option becomes true the execution of the previous option is interrupted and the subsequent option becomes the new intention. To be able to combine both methods in the same sequence we allow to specify for each option whether it might be interrupted or not. Denoting with $\mathit{interruptible}(o)$ whether option o might be interrupted we can reformulate (9):

$$i_t = \begin{cases} i_{t-1} & \text{if } \mathit{commit}_{i_{t-1}}(s_t) \wedge \neg(\mathit{interruptible}(i_{t-1}) \wedge \mathit{invoc}_{i_{t-1}+1}(s_t)) \\ i_{t-1} + 1 & \text{otherwise} \end{cases} \tag{12}$$

Another possibility to increase the power of the sequence arbitrator is to allow individual options to be skipped if their invocation condition is not true on time and the execution of the sequence would break, otherwise. Although the semantics of skippable options is quite simple, the combination of skippable and interruptable options might cause unexpected effects and is hard to analyze. A formal description of the resulting selection scheme and generic conditions is possible, however, it goes beyond the scope of this text.

5.1.5. Example

At this point let us illustrate the behavior framework with an example from robot soccer. A simple soccer playing robot might be able to perform three different moves:

- Patrol: go into your own half and wait
- ApproachBall: approach the ball provided that the position of the ball is known
- WingAttack: dribble the ball along the touch line into the opponent half, turn to the middle of the field and kick. This move can be executed provided that the ego robot is in ball possession

We can implement each of these moves as individual behavior and use a *highest priority first arbitrator* to build a soccer strategy out of these building blocks. While WingAttack is given the highest priority Patrol is given the lowest priority. Hence, the priority arbitrator will always make WingAttack its intention whenever the preconditions are fulfilled, e.g. the ego robot is in ball possession. If the preconditions of WingAttack are not fulfilled but the prerequisites of ApproachBall are met the arbitrator will select ApproachBall while Patrol is executed when the preconditions of the two options of higher priority are not met.

The wing attack move can be further decomposed into the three stages DribbleAlongTouchLine, TurnToMiddle, and Kick which can be implemented as individual behaviors. They can be composed to a sequence using the *sequence arbitrator*. Figure 9 depicts the resulting behavior tree. The invocation and commitment conditions of the three behaviors are chosen appropriately to model the transitions between them. That means, the commitment condition of DribbleAlongTouchLine remains true as long as

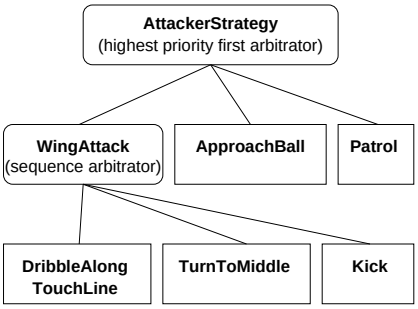

Figure 9: Behavior tree of a simple attacker in robot soccer as described in the text.

the robot does not reach a certain point in the opponent's half at which continuing to dribble is not promising. We declare the option DribbleAlongTouchLine of the sequence arbitrator to be non-interruptible to guarantee that the dribbling is not interrupted until reaching this point. In contrast, we declare the behavior TurnToMiddle to be interruptible which allows us to trigger the transition from TurnToMiddle to Kick by the invocation condition of the latter. This approach is convenient because only the kicking behavior knows which preconditions must be fulfilled for a promising pass. Figure 10 illustrates with an example the course of actions achieved with this behavior tree.

Certainly, the strategy described in this example is quite incomplete but it already shows how decomposition of complex strategies into simple building blocks works using the behavior framework. The actual soccer strategy of the team *Brainstormers Tribots* is built in the same way combining 81 behaviors to a much more sophisticated policy. The fact that arbitration is controlled by boolean invocation and commitment conditions creates a transparent decision making process without the overhead of a fully deliberative approach and with much larger flexibility than decision trees and finite state machines.

5.2. Multi-Agent Coordination

More than just acting as isolated agents the robots of a soccer robot team must cooperate to be successful. They can use wireless communication to exchange information among each other or with the coach computer. However, as described in section 3.2 the reliability of communication is low. That means, it is impossible to use a master-slave architecture in which the scene interpretation and the decisions about the robot and team behavior are made by the coach computer and just executed by the robots. In contrast, a dis-

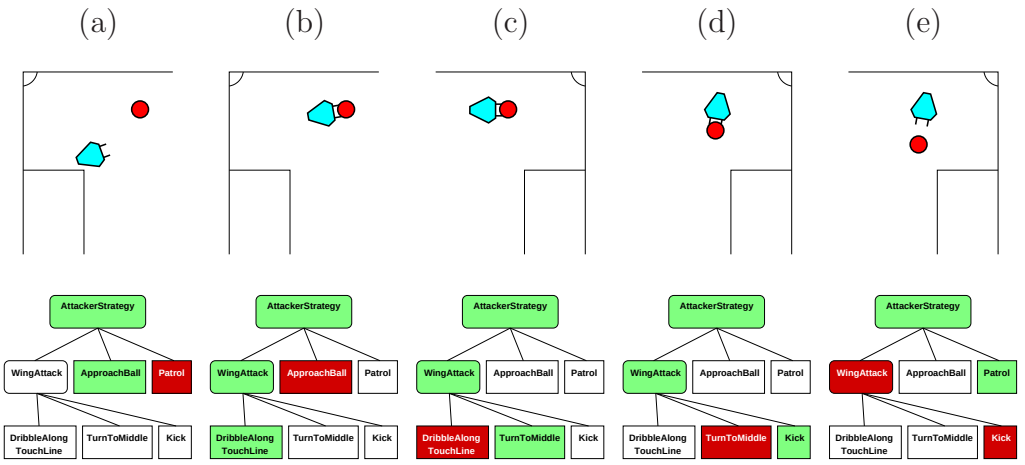

Figure 10: A sequence of situations which illustrates the method of operation of the behavior framework assuming the simple attacker strategy shown in figure 9. The above row shows a sequence of situations on the soccer field while the row below shows which options have become intentions (light green) and which ones lost the status of being the present intention (dark red). Uninvolved behaviors are shown in white. In situation (a) the robot followed the PATROL-Behavior until it perceived the ball so that the invocation condition of the APPROACHBALL-Behavior became true. Hence, the priority arbitrator switches its intention since APPROACHBALL has higher priority than PATROL. (b) As soon as the robot reaches the ball the invocation condition of WINGATTACK becomes true and the priority arbitrator switches its intention again. Since WINGATTACK is a sequence arbitrator it starts with its first option DRIBBLEALONGTOUCHLINE which becomes its intention. (c) After reaching a certain point in the opponent half the commitment condition of DRIBBLEALONGTOUCHLINE becomes false and WINGATTACK makes the next option TURNTOMIDDLE its present intention. (d) As soon as the robot is in a good passing position the invocation condition of the behavior KICK becomes true. Assuming that the previous behavior TURNTOMIDDLE has been declared interruptible the sequence arbitrator in WINGATTACK switches its intention to KICK. (e) Finally, after having kicked the commitment condition of the KICK-behavior becomes false and thereby the commitment condition of WINGATTACK becomes false. Now, the highest priority first arbitrator ATTACKERSTRATEGY selects among its options the one with highest priority and true invocation condition. Here, this might be the PATROL-Behavior again.

tributed decision making framework is needed which can deal with temporal delays in communication. Even if the wireless communication breaks down for some seconds the team of robots must go on working properly.

The requirements of robustness with respect to communication delays, breakdowns, a varying number of active robots, and the limited transmission bandwidth fostered the development of a cooperative structure that is characterized by the following properties:

- every robot makes its own decisions and is able to play soccer, even if the communication breaks down. In such a case, the team performance might be worse but at least a reasonable behavior is exhibited by the robots.

- every robot is assigned with a role that specifies its behavior. Roles are, e.g., goal keeper, attacker, left defender, right defender, etc. The assignment of roles is done by the coach computer which analysis the present situation and the number of active robots and might change the roles of the robots dynamically. As well, the coach computer might change some parameters of the team strategy like the readiness to play passes or the aggressiveness of the team.

- a robot that is in ball possession becomes captain of the team and might send orders and messages to its teammates, like *stay away from the ball* or *be prepared for receiving a pass*. Loosing the ball the robot also looses its captain position.

The communication structure defines two featured agents among all robots: the robot that is in ball possession and the coach computer. The coach computer works like a coach in soccer, it defines the team formation, assigns roles to the robots and defines the overall team strategy. It obtains state information from the robots to be able to make its decisions. However, the decisions of the coach only affect the behavior of the robots in the long run. They are not adequate for short-term team coordination, e.g. for coordinating pass playing.

In contrast, the robot that is in ball possession might affect the behavior of its teammates immediately. As a kind of team captain it controls which move is executed during the next seconds based on its individual perception of the environment. By communicating its decision, it might therefore influence the behavior of its teammates, if appropriate. For instance, in case of playing

a pass the captain is sending a message to its teammates to announce its intention so that its teammates can already go to an appropriate position.

This way of communication does not use negotiation between the robots. Due to the delays in communication, negotiation would need too much time so that successful moves could be interrupted easily by the opponent team. However, since only at most one player is in possession of the ball, we obtain a well-defined process of team coordination even without negotiation.

5.3. Interactions between Behavior and Perception

The design of our software is guided by the idea that the behavior generating software components can rely on accurate information provided by the perception and sensor fusion process and that the perception process is independent of the action that is executed by the robot. This idea is supported by the catadioptric camera sensor that provides a 360° field of view independent of the robot orientation. However, a complete decoupling between behavior generation and perception is not possible due to some limitations of the camera system, i.e.

- the mirror mounting of the omnidirectional camera system generates dead angles in which relevant objects might be hidden.
- the perspective camera of the goalie has a limited aperture angle of 80°. Hence, 3D reconstruction is only possible for objects in front of the robot. The outfield players are not equipped with stereo sensors.
- the catadioptric camera has a very limited field of view. Objects of the size of a ball can be recognized only up to a distance of 5 meters.
- teammates and opponents might occlude relevant objects like the ball.

As consequence of these limitations the robot behavior had to be adapted to avoid undesirable side-effekts for perception. Examples are, among others:

- to benefit from the mixed stereo camera system the goalie always turns towards the ball so that it can see the ball in both cameras.
- when the goalie does not see the ball it slightly turns left and right to avoid that the ball is hidden in dead angles.

- when the ball position is unknown the outfield players spread out over the entire field to guarantee that all areas of the field are observed by at least one robot.

- in case of an opponent set-play the ball is often occluded by opponent robots. Therefore, one of our robots takes a position at which it can see the ball and notify its teammates when the set-play has been executed.

- pass playing requires precise tracking of the ball movement. Since the outfield players are not capable to determine the ball position when the ball leaves the ground we had to modify the kicking devices of the robots to be able to execute flat kicks for passing.

As can be seen from these examples, the perception process has consequences for the strategy of the robots. Not only is an accurate recognition of the scene a prerequisite for sucessful soccer playing but also has the strategy to take into account the consequences of an action for visual perception. Although of general interest for cognitive systems, a systematic analysis of this phenomenon in the context of soccer robots is lacking up to now.

6. Reinforcement Learning

6.1. Background

Reinforcement learning (RL) [33] follows the idea that an autonomously acting agent learns its behavior policy through repeated interaction with its environment on a trial-and-error basis. In the following, we will delineate how this learning methodology can be profitably employed in the context of learning soccer robots.

Reinforcement learning problems are usually described as Markov Decision Processes (MDP) [57]. MDPs model a time-discrete, stochastic state-transition system in which the RL agent can choose from a set of actions to change the course of state transitions. Formally, an MDP is described as a 4-tuple $M = (S, A, p, c)$ where S denotes the set of states and A the set of actions the agent can perform. Function $c : S \times A \times S \to \mathbb{R}$ denotes immediate costs $c(s, a, s')$ that arise when taking action $a \in A$ in state $s \in S$ and transitioning to $s' \in S$. The probability $p_{ss'}(a) = p(s, a, s')$ of ending up in state s' when performing action a in state s is specified by the conditional probability distribution $p : S \times A \times S \to [0, 1]$. Furthermore, we assume that

the transition probabilities are independent of the past, i.e. the system is Markovian.

The RL agent interacts with the MDP and observes the present state and the cost arising from a transition but it does not know the transition probabilities nor the cost function. The agent's goal is to minimize the long-term, expected costs. To do so, it learns a decision policy that is used to determine the best action for a given state. Such a policy is a function $\pi : S \rightarrow A$ that maps the current state onto an action from a set of viable actions. It has been shown [58] that for every MDP there is an optimal policy $\pi^\star$ which yields the lowest expected long-term costs among all possible policies.

A number of algorithms has been developed to find an optimal policy that minimizes the long-term expected costs, including *policy iteration* [59], *value iteration* [60], and *Q-learning* [61]. While policy iteration and value iteration require an explicit knowledge of the transition probabilities and the cost function, Q-learning can be applied without this prior knowledge and allows to learn from interactions with the system.

6.2. Q Learning

The basic idea of Q-learning is to learn the so-called Q-function first and to derive the optimal policy by evaluating the Q-function afterwards. The Q-function Q^π related to a policy π describes for every pair of state and action (s, a) the expected long-term costs of applying action a in state s and following policy π afterwards. It can be found that the Q-function Q^* of an optimal policy meets the *Bellman-equation*:

$$Q^*(s,a) = \sum_{s' \in S} p_{ss'}(a)(c(s,a,s') + \min_{a' \in A} Q^*(s',a')) \tag{13}$$

Q-learning is derived from (13) observing that the optimal Q-function can be estimated using Robbins-Monroe approximation. Thereto, the algorithm starts with an arbitrary Q-function. By interacting with the MDP the RL agent observes state transitions. Once a transition from a state s_t to s_{t+1} with cost c_t applying action a_t is observed, the algorithm updates the Q-function for the pair (s_t, a_t) in the following way while keeping the Q-function for all other states and actions:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(c(s,a,s') + \min_{a' \in A} Q(s',a')) \tag{14}$$

$\alpha > 0$ denotes a learning rate that decreases over time. For the case of finite state and action spaces there are stochastic convergence guarantees. For the details we refer to [33].

After the optimal Q-function Q^* has been learned an optimal policy π^* can be derived by greedy evaluation:

$$\pi^*(s) \leftarrow \arg\min_{a \in A} Q^*(s, a) \tag{15}$$

6.3. Batch-Mode Reinforcement Learning

For reasonably small state spaces, Q-learning can be implemented using a lookup-table based representation of the Q-function. However, interesting RL problems usually have large and often continuous state spaces, where table-based methods are not applicable any more. One way to deal with that problem is to use function approximation to represent the value function.

Multilayer perceptrons (MLPs) [62] are known to be a very useful and robust regression method to approximate Q-functions in a broad range of different applications [63] [64]. However, some peculiarities have to be considered in order to make them work properly in practical applications. One important property results from the fact that they approximate the function in a global way, which means that – in contrast to local approximation schemes, e.g. lookup tables, radial-basis-function networks or nearest neighbor approaches – changing the value at one point might well have impacts on the outcome of arbitrary other points far away in the input space. Therefore, we consider it as a crucial point for the successful application of multilayer perceptrons for RL, that they are used in a batch-mode type of method, where always a whole set of points is updated simultaneously.

Figure 11 shows a general framework for doing batch RL. It consists of three main steps, namely sampling experience, generating a training pattern set using dynamic programming methods, and finally doing batch supervised learning to approximate the function represented by the training patterns.

During the sampling step new observations of state-transitions are created interacting with the MDP. To efficiently sample from the MDP the Q-function that has been learned so far is exploited to determine the most promising action. The observed state-transitions are stored in a data base together with the associated action and immediate cost.

In the second step, training patterns are built from the observed transitions imitating the Q-learning update rule (14). For an observed state-transition from state s to s' applying action a and creating costs c a training

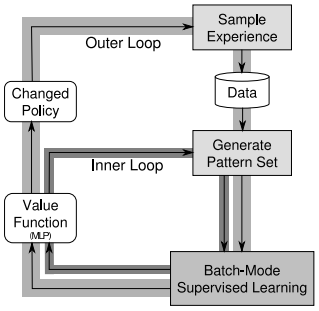

Figure 11: A graphical sketch of the batch RL framework. It consists of three modules (sample experience, generate pattern set, apply batch-mode supervised learning) that are called sequentially in repeated loops.

pattern is created in the following way:

$$(s, a) \mapsto c + \min_{a' \in A} \tilde{Q}(s', a') \tag{16}$$

where (s, a) is the input pattern and the RHS the target pattern. $\tilde{Q}$ denotes the present approximation of the Q-function. For every state transition stored in the data base one training pattern is generated.

In the third step the Q-function is retrained using batch-mode training algorithms for the function approximator. To speed-up convergence of batch-mode reinforcement learning the second and third step can be repeated several times before new samples are generated in the first step.

The convergence of batch-mode reinforcement learning can be guaranteed for the case of linear function approximators [65]. The case of batch-mode reinforcement learning with multilayer perceptrons is known as *Neural Fitted Q-Iteration* (NFQ) [66]. Although for the latter variant no convergence proof is known it has been shown to be very successful in practice.

6.4. Learning to Dribble

While the previous section described the ingredients of a successful reinforcement learning algorithm we want to exemplify its use with two applications from our soccer playing robots starting with the task to dribble a ball. Dribbling in the context of this application means keeping the ball in front of the robot, while driving to a given target position. Since the rules of the middle-size-league do not allow to clamp the ball or to enclose it by

mechanical parts, the only way to dribble the ball is to develop a specialized controller that is able to move the robot in the right way without losing contact to the ball.

From experiments with a manually tuned dribbling approach we found that the most difficult problem in dribbling is to make the robot turn into the target direction without losing the ball. Once the robot has turned it just needs to push the ball towards the target position which can be implemented easily. Therefore, we focussed on the turning problem in our learning approach.

Within the RL framework, we model the dribbling problem as a terminal state problem with both a terminal goal state and terminal failure states, e.g. once the robot reaches the goal state the trial is finished with zero immediate cost while reaching the failure state finishes the trial with immediate costs of $c = 1$. Intermediate steps are punished by small, constant costs of $c = 0.01$.

The state space consists of a six-dimensional vector with real valued entries for (a) the speed of the robot in relative x and y direction, (b) the yaw rate, (c) the x and y ball position relative to the robot, and (d) the heading direction relative to the given target direction. A failure state is encountered if the ball has left a certain area in front of the robot. A goal state is reached whenever the absolute difference between the heading angle and the target angle is less than 5 degrees.

We provided the robot with five possible actions which are characterized as a combination of linear velocities and yaw rates described in table 1. The target velocities specified by the actions are realized exploiting the kinematic model of the holonomic drive and using a PID controller on the level of motor voltages.

Interpreting the sign of the yaw rate situation-dependent we could exploit the symmetry of the problem, i.e. a positive yaw rate turns the robot in such a way that the angle between the robot orientation and the direction of the target position becomes smaller while negative yaw rates increase the angle. Analogously, the sign of the lateral velocity is interpreted situation dependent.

Learning Procedure. For learning, we use the NFQ framework described in section 6.3. The Q-function is represented by a multilayer perceptron with 9 input units (6 state variables and 3 action variables), 2 hidden layers of 20 neurons each and 1 output neuron. After each batch of 12 trials, we did 10 NFQ iterations. Learning the target values was done in 300 epochs

Table 1: Action set used for dribbling

	a_1	a_2	a_3	a_4	a_5
velocity ahead	$2\frac{m}{s}$	$2.5\frac{m}{s}$	$2.5\frac{m}{s}$	$3\frac{m}{s}$	$3\frac{m}{s}$
lateral velocity	$0\frac{m}{s}$	$0\frac{m}{s}$	$1.5\frac{m}{s}$	$1\frac{m}{s}$	$-1\frac{m}{s}$
yaw rate	$2\frac{rad}{s}$	$1.5\frac{rad}{s}$	$1.5\frac{rad}{s}$	$1\frac{rad}{s}$	$1\frac{rad}{s}$

of supervised batch learning, using the Rprop learning method [67]. After learning was finished, the new controller was used to control the robot during the next data collection phase. After 11 batches (= 132 trials), a very good controller was learned. The complete learning procedure took about one and a half hour, including the time used for offline updating of the neural Q function. The actual interaction time with the real robot was about 30 minutes, including preparation phases.

The batch trials were performed as follows. At the beginning of each trial, the robot waits until the ball is put onto the middle of the field, before moving to a starting position 2m away from the ball. Next, it drives towards the ball and as soon as it gets there, the dribbling trial is started. In every trial, a different target direction is given. Here, we collected batches of 12 trials each without retraining the neural controller within a batch. After each batch, the sampled transitions are added to the data set, and learning is started. If the set of target values used for the 12 trials are the same for each batch, then in parallel to data sampling, the performance of the controllers can be evaluated and compared.

Performance. The neural dribbling controller is implemented as an autonomous behavior within the behavior framework described in section 5. The behavior is initialized with a certain target direction and the current state information. It returns a three-dimensional driving command consisting of the desired forward speed v_x, lateral speed v_y, and yaw rate v_θ. It was the first behavior in our robots that was completely learned on the real robot. without the help of a simulation environment.

The neural dribbling behavior performed significantly better than the previously used, hand-coded and hand-tuned dribbling approach, especially in terms of space and time needed to turn to the desired target direction. Figure 12 shows a trial run. It has been used successfully in our competition team since 2007. With its help, we won the world championship 2007 in

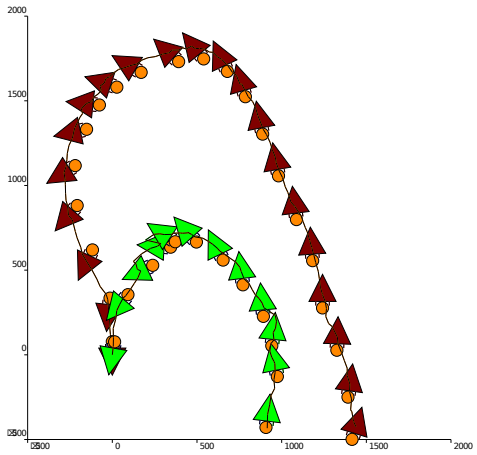

Figure 12: Comparison of hand-coded (dark red) and neural dribbling behavior (light green) when requested to make a U-turn. The data was collected on our real robot. When the robot gets the ball, it typically has an initial speed of about 1.5 to 2 m/s in forward direction. The positions of the robot are displayed every 120 ms. The U-turn of the neural dribbling controller is much sharper and faster.

Atlanta and became third at the world championship in 2008 in Suzhou, China.

6.5. Learning Motor Speed Control

Fast, accurate and reliable motor speed control is a central requirement in many real world applications, especially for mobile, wheel-based robots like soccer robots. This low level motor control behavior is a crucial prerequisite for higher level tasks to be efficient and successful. Especially changing load situations depending on the overall system behavior are challenging and often require immense effort to design an appropriate control law. Thus, being able to learn such control laws automatically would be a great step forward.

On the omnidirectional robot, we have three DC motors in a specific configuration, each driving a mecanum wheel. The motion and dynamics of the robot depend directly on the speeds of the three motors. This gives the robot a high mobility and avoids non holonomic constraints in motion planning. However, the interaction of the three motors causes highly changing load situations. Our goal is to learn a fast, accurate and reliable controller for regulating the speed of each DC motor of the omnidirectional mobile robot, solely by interaction with the robot.

Instead of learning a specialized controller for each motor, we show a setup where we learn a single controller that operates independently on each of the three motors. From the point of view of the controller, it gets the state information and set point of a single motor and answers with an action that will regulate this motor from its current state to the desired speed. In other words, there is only one single DC motor in the view of the controller that has to be controlled in a broad range of dynamical load situations. This procedure is legitimate, since the three motors and wheels are of the same type of construction. In the following, we will describe the learning system setup in more detail. For doing so, we take the viewpoint of the controller, so the task is DC motor speed regulation, based on the state information of a single motor, to arbitrary set points under a broad range of load situations.

The control problem considered can be described as a Markov decision process. The state of a general DC motor can be described sufficiently by two variables, the present motor speed $\dot{\omega}$ and the armature current I. Since our final controller has to deal with arbitrary target speeds, the information about the desired speed must also be incorporated into the input. Here we are using the error between the actual speed and the target speed, $E := \dot{\omega}_d - \dot{\omega}$.

We are facing a set point regulation task, since no terminal states exist, but instead regulation is an ongoing, active control task. The control objective is to first bring the motor speed close to the target value as fast as possible and then to actively keep it at the desired level which can be described by the following choice of the immediate costs c:

$$c(s, a, s\prime) = c(s) = \begin{cases} 0 & \text{if } |\dot{\omega}_d - \dot{\omega}| < \delta \\ 0.01 & \text{otherwise} \end{cases} \tag{17}$$

The first line denotes the desire to keep the motor velocity $\dot{\omega}$ close to its target value $\dot{\omega}_d$, where the allowed tolerance is denoted by $\delta > 0$. Otherwise (second line) the controller is punished.

The accurate regulation of the motor speed at arbitrary target values would, in principle, require the output of voltages from a continuous interval by the controller while reinforcement learning is limited to finite, usually small action sets. To overcome this problem, we use an integrating output approach [68]. The idea is that the controller does not return the desired voltage, but just the decrease or increase of the voltage by a certain amount $\triangle U$. By applying this trick, a wide range of resulting voltages can be produced, whereas the set of actions available to the RL controller remains relatively

small. The final action set of the controller is

$$\Delta U \in \{-0.3, -0.1, -0.01, 0.0, 0.01, 0.1, 0.3\}$$

As a consequence, we have to extend the state of the MDP by the present motor voltage U so that the controller is able to distinguish whether the voltage is already high or low.

Learning Procedure. To train the controller, we use NFQ, described in section 6.3, with a multilayer perceptron whose topology consists of 5 neurons in the input layer (4 for state description, one for the action), 2 hidden layers with 10 neurons each, and a single output denoting the Q-value of the respective state-action pair.

To collect training data with typical load situations for the controller in its final working range, we had to collect them directly in interaction with the real robot. This was done by putting the robot on the ground and driving it around by applying the controller for each of the three motors. Following this approach, we can collect three different transition samples in different load conditions in each time step. In contrast to the transition sampling procedure that interleaves learning phases and data collection, we decided to pursue a random strategy here, i.e. random control signals were emitted to each of the three motors on trajectories of an episode length of 150 time steps (5 seconds). A new set point was randomly selected for each motor after each data collection episode in order to collect a broad range of set points. After all data-collecting episodes are finished, the controller is trained by NFQ in a purely off-line mannor.

We ran 50 data collection episodes which gave an overall of $50 \cdot 150 = 7500$ transition samples collected for each motor. Since the data was collected simultaneously for all three motors, this resulted in an overall of 22500 transition samples that could be used for training the controller within the NFQ framework. The whole process of data collection on the real robot needed only $250s$. After only 30 iterations through the NFQ loop, a highly effective controller was learned.

Performance. In Figure 13, the learned controller is shown running on the real robot. The global driving commands used as a demonstration here are 'drive forward with $0.5\frac{m}{s}$' and then 'rotate by $2\frac{rad}{s}$'. The inverse kinematics are used to deliver the respective target speeds for each motor. The task of

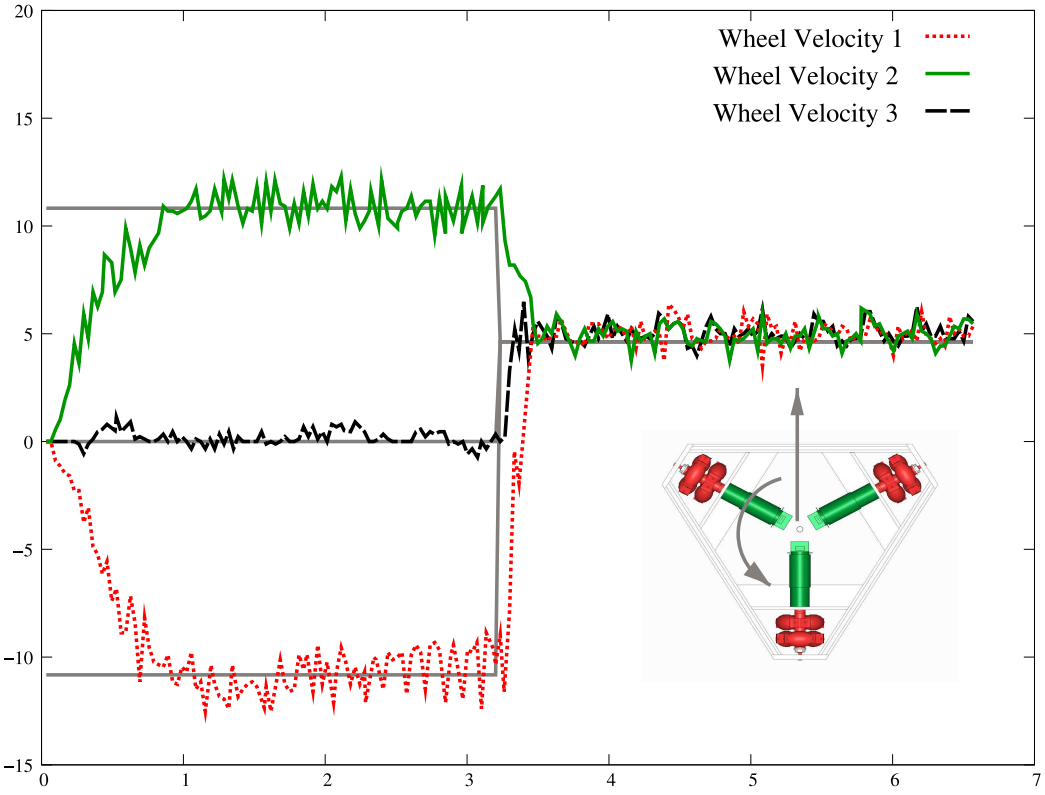

Figure 13: The learned controller tested on a follow-up control on the real robot. The robot was driven forward with $0.5\frac{m}{s}$ changing to a rotational velocity of 2 $\frac{rad}{s}$. The controller is able to achieve the velocities for all three motors under the presence of noise generated from the wheels.

	'00	'01	'02	'03	'04	'05	'06	'07	'08
simulation league									
hard & precise kicking	●	●	●	●	●	●	●	●	●
intercepting the ball	●	●	●	●		○	○		
going to position	●	●	●	●	●				
1-vs-1 aggressive defense								●	●
7-vs-8 attack	○	●	●	●	●		●	●	●
penalty kick				●	●	●	●	●	●
middle-size-league									
motor speed control								○	○
going to position							○	○	○
intercepting the ball							●	●	●
dribbling								●	●

Table 2: Overview of a selection of behaviors that were learned by neural reinforcement learning methods for the Brainstormers simulation league and middle-size-league teams over the years from 2000 to 2008. Filled dots denote the application in the competition team, empty dots denote that the skill was successfully learned, but finally did not make it into the competition team. Many of the skills have been improved from year to year.

the learned controller is then to regulate each motor to the desired motor speed.

As shown in Figure 13, the neural controller has learned to control the motors very quickly and reliably to their desired target speeds. A highly satisfying fact is that the learned speed controller works reliably under the wide range of actual loads that occur within the real robot movement. It has even learned to deal with the considerable noise that occurs in the measured data due to the shape of the wheels. The whole approach is described in more details in [64].

6.6. Extensions to the Strategy Level

The case studies described in the previous sections show how reinforcement learning techniques can be used to derive elaborated control policies for autonomous robots only from interactions with the environment. No physical model of the environment must be available nor do we need to do system identification. This concept has been applied very successfully in our robot soccer team. Beside the approaches to learn dribbling and motor control we also made the robots learn to intercept the ball [69] and to drive to a target position in a time-optimal way.

Our work also shows that reinforcement learning can be applied on very different levels of abstraction. First, we were able to learn a motor controller on the lowest level of control. Secondly, we were able to learn individual

skills like dribbling and ball interception. What remains is the strategy level and the team level. On these abstraction levels the immense effort to run experimental trials requiring two full teams of soccer robots prevented the use of reinforcement learning up to now.

However, a look to the RoboCup simulation league shows that also on these levels of abstraction learning is possible. In our affiliated simulation league team which participates in RoboCup since 1998 we started to apply reinforcement learning techniques in 2000 with very big success. While in the beginning we were learning individual skills like kicking, going to a target position, and intercepting the ball we extended the learning approach to skills interfering with opponents (e.g. an aggressive defense behavior [70]) and to the team strategy level where we learned 2-versus-2 defense strategies and 7-versus-8 attack strategies [71].

In order to finally yield a competitive software agent, the individual skills must also prove their superiority to alternative approaches. Therefore, all the learned skills that made it inside the competition code have proven their superiority compared to previous hand-coded versions. When all the learned skills are activated in our simulation league agent, up to 80 percent of the decisions in a game are triggered by neural networks[5].

In most cases, we used neural networks for representing the value function, using the batch RL framework described in Section 6.3 as the general learning framework. The state dimension typically ranged from 5 to 10 real valued state variables, the number of discrete actions was typically in the range of up to 10 for real robot learning and up to several hundreds of actions for learning in the simulation league agent. An overview of reinforcement learning approaches that we implemented in our simulation league and middle-size-league team is given in table 2.

7. Discussion

This paper described how cognitive techniques of learning and behavior generation can be used in a team of autonomous robots. Using as example the soccer robot team *Brainstormers Tribots* we were able to demonstrate how these techniques can be integrated into a complex robot system and

[5]e.g. when activating all neural skills in our 2005 agent, a neural network is involved in decision-making on average in 56.8% (defender), 73.0% (sweeper), 84.4% (midfielder), 82.6% (attacker), of its total number of actions

how they can contribute to an elaborated performance of the overall system. The successful participation of the team at *RoboCup* emphasizes the gain in performance using cognitive techniques.

The main challenge in developing integrated, complex systems is to bring together the scientific concepts and the demands of the application. One important consideration is to find appropriate methods to perceive and represent the environment adequately so that the representation meets the requirements of the approaches for behavior generation and learning. In our case building a consistent geometric and dynamic model of the environment enabled us to predict the state of the environment into short-term future to overcome the problem of latencies in sensors and actuators. Hence, the environment can be modeled as a Markov decision process which is the basis for efficient learning.

Based on the accurate representation of the world we were able to make the robot learn individual skills like intercepting the ball and dribbling the ball only from success and failure using reinforcement learning. An important breakthrough could be achieved making those approaches work even on real robots using neural networks as function approximators to represent the value function and reusing old experiences in a memory-based reinforcement learning approach. By doing so we were able to efficiently represent value functions of six dimensions or more and to reduce the training time reasonably.

Individual robot skills like dribbling the ball could be learned within a training time of three hours. The learned control policy performed better than a hand-coded dribbling approach. We could treat the robot as a black-box system and did not need to develop a precise physical model nor perform system identification. We could use the robot as best model of itself and generate emergent behavior by learning on the robot.

To integrate various techniques of behavior generation into a complex strategy a transparent software concept is essential. We developed a behavior-based framework in this paper which combines ideas from various existing behavior architectures into a common framework and which establishes a software design pattern for behavior creation.

Its main idea is to encapsulate in a single software structure the execution of a behavior and the preconditions under which the behavior can be applied, and to combine many behaviors using generic arbitrators which select an appropriate behavior on the basis of the invocation and commitment conditions of all options. Hence, the annoying and error-prone process of

developing situation-dependent arbitration mechanisms can be encapsulated in a small set of generic, general-purpose arbitrators. Furthermore, since the arbitrators are equipped with generic preconditions they can be interpreted as behaviors on a higher level of abstraction and can contribute themselves to other arbitrators. Thus, it is possible to build behavior hierarchies of variable depth and branching factor in an iterated software development process.

What is common to all techniques developed for robot soccer is that they realize a balance between the ideas of cognitive modeling and goal-oriented software engineering for a dynamic real-time application. They must contribute to an integrated system including a team of embodied agents. This means, instead of just optimizing a single, isolated technique it is essential to optimize an integrated system and to reveal the conceptual relationships between different areas like perception and behavior generation or hardware design and cognitive abilities.

Beyond the cognitive techniques discussed in this paper there is ongoing research to make soccer robots smarter. These ideas include reinforcement learning on the level of cooperative behavior, an intelligent coach that analyzes the games automatically, and methods of self-monitoring and self-repair. First approaches for all areas exist, however, up to now they are very specific and do not allow generalization to other domains.

We believe that the techniques introduced in this paper – especially sensor fusion on a geometric and dynamic level, a modular behavior framework, and reinforcement learning as technique for behavior creation – are important building blocks for cognitive architectures and can contribute to autonomous systems in a wide variety of applications.

References

[1] R. Pfeifer, J. Bongard, How the Body Shapes the Way we Think: a New View of Intelligence, MIT Press, 2007.

[2] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, H. Matsubara, RoboCup: A challenge problem for AI, AI Magazine 18 (1) (1997) 73–85.

[3] H.-D. Burkhard, D. Duhaut, M. Fujita, P. Lima, R. Murphy, R. Rojas, The road to RoboCup 2050, IEEE Robotics & Automation Magazine 9 (2) (2002) 31–38.

[4] P. Lima (Ed.), Robotic Soccer, I-Tech Education and Publishing, 2007.

[5] A. A. F. Nassiraei, Y. Takemura, A. Sanada, Y. Kitazumi, Y. Ogawa, I. Godler, K. Ishii, H. Miyamoto, A. Ghaderi, Concept of mechatronics modular design for an autonomous mobile soccer robot, in: Proceedings 7th IEEE International Symposium on Computational Intelligence in Robotics and Automation, 2007, pp. 178–183.

[6] A. Ghaderi, A. Sanada, A. A. Nassiraei, K. Ishii, I. Godler, Power and propulsion systems design for an autonomous omni-directional mobile robot, in: Proceedings 23rd Annual IEEE Applied Power Electronics Conference and Exposition, 2008, pp. 267–272.

[7] M. Riedmiller, R. Hafner, S. Lange, M. Lauer, Learning to dribble on a real robot by success and failure, in: Proceedings IEEE International Conference on Robotics and Automation, 2008, pp. 2207–2208.

[8] P. Heinemann, H. Becker, A. Zell, Improved path planning in highly dynamic environments based on time variant potential fields, in: Proceedings 37th International Symposium on Robotics, 2006, pp. 177–178.

[9] D. Bruijnen, J. van Helvoort, R. van de Molengraft, Realtime motion path generation using subtargets in a rapidly changing environment, Robotics and Autonomous Systems 55 (6) (2007) 470–479.

[10] A. Ferrein, C. Fritz, G. Lakemeyer, Using golog for deliberation and team coordination in robotic soccer, KI Künstliche Intelligenz 19 (1) (2005) 24–43.

[11] H. Fujii, M. Kato, K. Yoshida, Cooperative action control based on evaluating objective achievements, in: RoboCup 2005: Robot Soccer World Cup IX, 2006, pp. 208–218.

[12] O. Zweigle, U.-P. Käppeler, R. Lafrenz, H. Rajaie, F. Schreiber, P. Levi, Situation recognition for reactive agent behavior, in: Artificial Intelligence and Soft Computing, 2006, pp. 92–97.

[13] V. A. Ziparo, L. Iocchi, D. Nardi, P. F. Palamara, H. Costelha, Petri net plans: a formal model for representation and execution of multi-robot plans, in: Proceedings 7th International Joint Conference on Autonomous Agents and Multiagent Systems, 2008, pp. 79–86.

[14] N. Lau, L. S. Lopes, G. Corrente, N. Filipe, Multi-robot team coordination through roles, positioning and coordinated procedures, in: Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009.

[15] F. Pin, S. Killough, A new family of omnidirectional and holonomic wheeled platforms for mobile robots, IEEE Transactions on Robotics and Automation 10 (4) (1994) 480–489.

[16] S. Baker, S. K. Nayar, A theory of catadioptric image formation, in: Procedings 6th International Conference on Computer Vision, 1998, pp. 35–42.

[17] G. Mayer, H. Utz, G. Kraetzschmar, Towards autonomous vision self-calibration for soccer robots, in: Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002, pp. 214– 219.

[18] P. Heinemann, F. Sehnke, F. Streichert, A. Zell, An automatic approach to online color training in robocup environments, in: Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 4880–4885.

[19] A. Treptow, A. Masselli, A. Zell, Real-time object tracking for soccer-robots without color information, in: Proceedings European Conference on Mobile Robotics, 2003, pp. 33–38.

[20] D. A. Martins, A. J. R. Neves, A. J. Pinho, Real-time generic ball detection in robocup domain, in: IBERAMIA'08 - Ibero-American Conference on Artificial Intelligence: IROBOT'08 - 3rd International Workshop on Intelligent Robotics, 2008, pp. 37–48.

[21] D. Bruijnen, W. Aangenent, J. van Helvoort, R. van de Molengraft, From vision to realtime motion control for the robocup domain, in: Proceedings IEEE International Conference on Control Applications, 2007, pp. 545–550.

[22] J. Silva, N. Lau, J. Rodrigues, J. L. Azevedo, A. J. R. Neves, Sensor and information fusion applied to a robotic soccer team, in: RoboCup 2009: Robot Soccer World Cup XIII, 2010.

[23] M. Lauer, Ego-motion estimation and collision detection for omnidirectional robots, in: RoboCup 2006: Robot Soccer World Cup X, 2006, pp. 466–473.

[24] P. Heinemann, J. Haase, A. Zell, A combined monte-carlo localization and tracking algorithm for RoboCup, in: Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 1535–1540.

[25] D. Stronger, P. Stone, A comparison of two approaches for vision and self-localization on a mobile robot, in: Proceedings IEEE International Conference on Robotics and Automation, 2007, p. 3915–3920.

[26] A. Ferrein, L. Hermanns, G. Lakemeyer, Comparing sensor fusion techniques for ball position estimation, in: RoboCup 2005: Robot Soccer World Cup IX, 2005, pp. 154–165.

[27] M. Lauer, S. Lange, M. Riedmiller, Motion estimation of moving objects for autonomous mobile robots, Künstliche Intelligenz 20 (1) (2006) 11–17.

[28] M. Taiana, J. A. Gaspar, J. C. Nascimento, A. Bernardino, P. U. Lima, 3D tracking by catadioptric vision based on particle filters, in: RoboCup 2007: Robot Soccer World Cup XI, 2008, pp. 77–88.

[29] A. Strack, A. Ferrein, G. Lakemeyer, Laser-based localization with sparse landmarks, in: RoboCup 2005: Robot Soccer World Cup IX, 2005, pp. 569–576.

[30] A. Bonarini, M. Matteucci, M. Restelli, Automatic error detection and reduction for an odometric sensor based on two optical mice, in: Proceedings IEEE International Conference on Robotics and Automation, 2005, pp. 1675–1680.

[31] J. de Best, R. van de Molengraft, An active ball handling mechanism for robocup, in: Proceedings 10th International Conference on Control, Automation, Robotics and Vision, 2008, pp. 2060–2065.

[32] M. Lauer, M. Riedmiller, Participating in autonomous robot competitions: Experiences from a robot soccer team, Proceedings Workshop on

Competitions in Artificial Intelligence and Robotics at the International Joint Conference on Artificial Intelligence (2009).

[33] R. Sutton, A. Barto, Reinforcement Learning. An Introduction, MIT Press, Cambridge, USA, 1998.

[34] B. Meyer, Applying "design by contract", IEEE Computer 25 (10) (1992) 40–51.

[35] M. Goebl, G. Färber, A real-time-capable hard- and software architecture for joint image and knowledge processing in cognitive automobiles, in: Proceedings IEEE Intelligent Vehicles Symposium, 2007, pp. 734–740.

[36] T. Kawanishi, K. Yamazawa, H. Iwasa, H. Takemura, N. Yokoya, Generation of high-resolution panoramic images by omnidirectional imaging sensor using hexagonal pyramidal mirrors, in: Proceedings 14th International Conference on Pattern Recognition, 1998, pp. 485–489.

[37] J. Gluckman, S. K. Nayar, K. J. Thoresz, Real-time omnidirectional and panoramic stereo, in: Proceedings DARPA Image Understanding Workshop, 1998, pp. 299–303.

[38] L. Matuszyk, A. Zelinsky, L. Nilsson, M. Rilbe, Stereo panoramic vision for monitoring vehicle blind-spots, in: Proceedings IEEE Intelligent Vehicle Symposium, 2004, pp. 31–36.

[39] A. Voigtländer, S. Lange, M. Lauer, M. Riedmiller, Real-time 3D ball recognition using perspective and catadioptric cameras, in: Proceedings 3rd European Conference on Mobile Robots, 2007.

[40] P. J. Huber, Robust statistics, Wiley, 1981.

[41] A. Gelb, Applied Optimal Estimation, MIT Press, 1974.

[42] M. Lauer, S. Lange, M. Riedmiller, Calculating the perfect match: an efficient and accurate approach for robot self-localization, in: Robocup 2005: Robot Soccer World Cup IX, 2005, pp. 142–153.

[43] S. Thrun, D. Fox, W. Burgard, F. Dellaert, Robust monte carlo localization for mobile robots, Artificial Intelligence 128 (1-2) (2001) 99–141.

[44] A. Merke, S. Welker, M. Riedmiller, Line base robot localisation under natural light conditions, in: Proceedigs ECAI Workshop on Agents in Dynamic and Real-Time Environments, 2004.

[45] A. E. Hoerl, R. W. Kennard, Ridge regression: Biased estimation for nonorthogonal problems, Technometrics (1970) 55–67.

[46] X.-L. Meng, D. B. Rubin, Maximum likelihood estimation via the ECM algorithm: A general framework, Biometrika 80 (2) (1993) 267–278.

[47] S. Behnke, A. Egorova, A. Gloye, R. Rojas, M. Simon, Predicting away robot control latency., in: RoboCup 2003: Robot Soccer World Cup VII, 2003, pp. 712–719.

[48] M. Wooldridge, Reasoning about Rational Agents, MIT Press, 2000.

[49] D. Klahr, P. Langley, R. Neches (Eds.), Production system models of learning and development, MIT Press, 1987.

[50] R. A. Brooks, Intelligence without representation, Artificial Intelligence 47 (1991) 139–159.

[51] R. C. Arkin, T. R. Balch, AuRA: principles and practice in review, Journal of Experimental and Theoretical Artificial Intelligence 9 (2-3) (1997) 175–189.

[52] D. Harel, Statecharts: A visual formalism for complex systems, Science of Computer Programming 8 (3) (1987) 231–274.

[53] R. C. Arkin, Behavior-based robotics, MIT Press, 2000.

[54] M. L. Littman, Markov games as a framework for multi-agent reinforcement learning, in: Proceedings International Conference on Machine Learning, 1994, pp. 157–163.

[55] M. Georgeff, A. Rao, Rational software agents: From theory to practice, in: N. R. Jennings, M. J. Wooldridge (Eds.), Agent Technology: Foundations, Applications, and Markets, Springer, 1998.

[56] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison-Wesley, 2004.

[57] M. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley-Interscience, USA, 2005.

[58] D. Bertsekas, J. Tsitsiklis, Neuro Dynamic Programming, Athena Scientific, Belmont, USA, 1996.

[59] R. A. Howard, Dynamic Programming and Markov Processes, The M.I.T. Press, 1960.

[60] R. Bellman, Dynamic Programming, Princeton University Press, Princeton, USA, 1957.

[61] C. Watkins, P. Dayan, Q-Learning, Machine Learning 8 (1992) 279–292.

[62] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall, 1998.

[63] M. Riedmiller, M. Montemerlo, H. Dahlkamp, Learning to Drive in 20 Minutes, in: Proceedings of the FBIT 2007 Conference, Springer, Jeju, Korea, 2007.

[64] R. Hafner, M. Riedmiller, Neural Reinforcement Learning Controllers for a Real Robot Application, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 07), IEEE Press, Rome, Italy, 2007.

[65] D. Ernst, P. Geurts, L. Wehenkel, Tree-Based Batch Mode Reinforcement Learning, Journal of Machine Learning Research 6 (1) (2006) 503–556.

[66] M. Riedmiller, Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method, in: Machine Learning: ECML 2005, 16th European Conference on Machine Learning, Springer, Porto, Portugal, 2005.

[67] M. Riedmiller, H. Braun, A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm, in: H. Ruspini (Ed.), Proceedings of the IEEE International Conference on Neural Networks (ICNN), San Francisco, 1993, pp. 586–591.

[68] M. Riedmiller, Generating Continuous Control Signals for Reinforcement Controllers Using Dynamic Output Elements, in: Proceedings of the European Symposium on Artificial Neural Networks (ESANN 1997), Bruges, Belgium, 1997.

[69] H. Müller, M. Lauer, R. Hafner, S. Lange, A. Merke, M. Riedmiller, Making a robot learn to play soccer using reward and punishment, in: KI 2007 Advances in Artificial Intelligence, 2007, pp. 220–234.

[70] T. Gabel, M. Riedmiller, F. Trost, A Case Study on Improving Defense Behavior in Soccer Simulation 2D: The NeuroHassle Approach, in: L. Iocchi, H. Matsubara, A. Weitzenfeld, C. Zhou, editors, RoboCup 2008: Robot Soccer World Cup XII, LNCS, Suzhou, China, 2008, pp. 61–72.

[71] M. Riedmiller, A. Merke, Using Machine Learning Techniques in Complex Multi-Agent Domains, in: I. Stamatescu, W. Menzel, M. Richter, U. Ratsch (Eds.), Adaptivity and Learning, Springer, 2003.