14

Integrating Autonomous Behaviour and Team Coordination into an Embedded Architecture

Bernd Kleinjohann, Lisa Kleinjohann, Willi Richert and Claudius Stern
University of Paderborn, C-LAB
Germany

1. Introduction

Robotic soccer is a challenging research and application field for the combination of real time embedded systems design with intelligent autonomous behaviour as well as tema coordination and learning. The joint investigation of these themes is pursued by the development of the Paderkicker robots described in this paper.

The Paderkicker team (Richert et al., 2006) consists of five robots (Fig. 1) that already participated successfully in the German Open competition in 2004, 2005 and 2007, the Dutch Open 2006 and the RoboCup 2006 World Championship. Our platform asks for the whole range of research issues needed for a successful deployment in the real world. This includes embedded real-time architectures (Beier at al., 2003; Esau at al., 2003b; Stichling, 2004), real-time vision (Stichling & Kleinjohann, 2002a, 2002b, 2003) learning and adaptation from limited sensor data, skill learning (Richert & Kleinjohann, 2007) and methods to propagate learned skills and behaviours in the robot team (Richert et al., 2005). However, our goal is not to carry out research for specific solutions in the robotic soccer domain, but to use and test advanced techniques from different research projects. The Paderkicker platform serves as a test bench for the collaborative research center 614 (funded by the Deutsche Forschungsgesellschaft). Furthermore, the knowledge in vision, motion and object tracking was used in the AR PDA (Bundesministerium für Bildung und Forschung) project (Reimann, 2005).

This paper describes various aspects of the Paderkicker robots. Section 2 gives an overview of the robots' construction. First the actor systems for driving, ball handling and vision system are described followed by the sensor systems for odometry, landmark and ball recognition. Section 3 focuses on various aspects of the modular robot design. Functional design, hardware, software and process architecture are covered as well as the message format used for communication between different robots and robot components. Section 4 describes the real-time image processing module developed for the Paderkicker robots. Our algorithms for colour segmentation and recognition of objects like ball, field or lines are presented. Furthermore, we propose a DCT-based (discrete cosine transform) pre-processing step improving the subsequent colour segmentation. In Section 5 the behaviour based realization of the Paderkicker robots is explained starting from team coordination issues down to simple reactive behaviour. Section 6 deals with learning capabilities we want to introduce into our

Source: Robotic Soccer, Book edited by: Pedro Lima, ISBN 978-3-902613-21-9,pp. 598, December 2007, Itech Education and Publishing, Vienna, Austria

robots. Section 7 concludes the paper with a short resume and an outlook to future developments.

2. Robot outline

The robots of the Paderkicker team were constructed from scratch, since they also should be used for demonstrations in environments which are less restricted concerning lightning and underground conditions than the RoboCup environment. Furthermore they should provide a test platform for different research projects. Nevertheless their main purpose is playing robotic soccer in the midsize league. The Paderkicker robots are mainly developed in the course of student education projects necessitating a modular design approach enabling students to familiarize with the Paderkickers within about two months, after which they should be able to productively contribute to the Paderkickers' development. As already mentioned the main focus of this development is the design of embedded hardware and software and the application of artificial intelligence algorithms in real-time environments.
Size and form of our robots were mainly determined by the RoboCup regulations. The maximum height of 80 cm was chosen for providing the vision system with good overall viewing capabilities. Robot size was motivated by the upper boundary for the area occupied by the entire robot team, which led to the design objective of keeping the robots as small as possible, in order to allow an additional robot in the team. In our case we arrived at an area of 36 cm x 36 cm for each robot.
The carrying structure consists of standard quadratic aluminium profiles with edge length of 2 cm (Fig. 2). At each side the profiles are equipped with T-shaped channels for connecting profiles or fastening functional units for driving, ball handling and orientation. These functional units are realized as mechatronic functional units (MFU) and controlled by separate hardware controllers (see Section 3). The construction of these MFUs, which are built mainly from standard model craft components, will be described in more detail below because of their crucial role in robot soccer.

Fig. 1. The Paderkicker robot team

Fig. 2. Robot construction

2.1 Driving system

The Paderkicker robots are equipped with omnidirectional drives (omniwheels) since they allow for simultaneous translational and rotational robot movements while supporting all three degrees of freedom on a plane. Omniwheels are actively driven by a motor into one direction. Orthogonally omniwheels consist of passive rolls with small rolling friction. Since this depends on their footprint, we selected wheels with a small diameter of 6 cm. With hindsight these rolls are capable of driving over very small splits only. Splits like those between lift shaft and cabin are for instance at the upper boundary. Presently, his attribute does not mean a restriction for RoboCup, but it could become one if for instance sport halls should be used for RoboCup tournaments without a carpet that covers installations for fastening sports equipment on the floor. We decided to use four (instead of three) omniwheels per robot to increase stability. When using four driven wheels rotated by 90 degrees each, all four motors contribute to a translational movement. This enables the use of smaller dimensioned motors in contrast to the use of only three omniwheels, because in that case only two of them contribute to a straight movement. Also the assembly angle is less advantageous for three wheels. However special care has to be taken to guarantee that each wheel touches the ground whereas this feature is automatically guaranteed for three wheels. For this purpose we mounted the wheels movably in rubber blocks. Thus all wheels are pressed onto the ground by a robot's own weight. Fig. 3 shows one omniwheel including attached sensors which are described in Section 2.4.

2.2 Vision system

Our vision system contains three independent pan-tilt cameras that are used for active viewing (Fig. 4) in contrast to omnivision systems that are currently used by many other teams. Each camera may independently focus and track a different object of interest like ball, corner, goal or marker. Proper choice of the aperture angle (up to 120 degrees) guarantees a sufficient view in all directions. Our experience showed that for the two outermost cameras a shorter focal distance is preferable to provide an overview of the playing ground while a standard focal distance for the medium camera allows focusing single objects appropriately. A major advantage of our approach, especially for larger soccer fields, is its greater resolution and easier calibration, since it is not necessary to account for the average conditions of the entire soccer field. According to our experience the processing effort for our vision algorithm uses about 15% of the processing capabilities of a Pentium M ULV running at 1GHz.

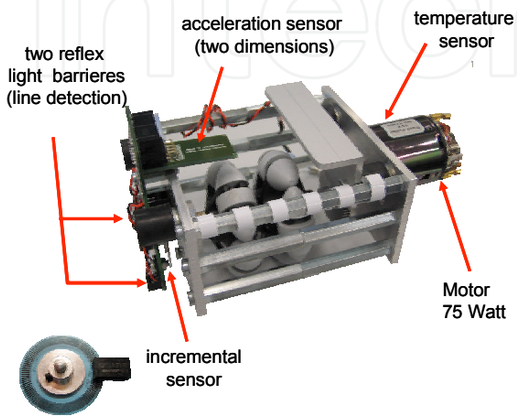

Fig. 3. Omniwheel with attached sensors

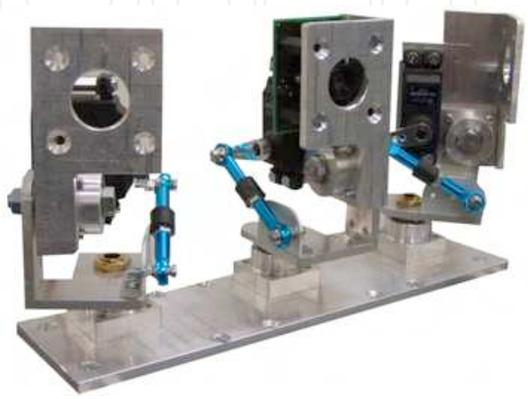

Fig. 4. Active vision system

2.3 Ball handling system

Soccer robots must be able to move the ball in a controlled manner, e.g. for passing it to another robot, taking it away from another one or kicking it towards the goal. The way how they may do it, i.e. how much force may be given to the ball or how long they may touch the ball is restricted by the RoboCup rules to a large extent. For instance, it is not allowed to fix the ball or to enclose more than one third of its diameter. Therefore we developed a mechanism for ball kicking and another one for ball dribbling which is described next.

For ball dribbling we developed a mechanism that allows giving the ball a rotational pulse in diverse directions for rolling it into a desired direction without keeping it in touch with the robot. For this purpose we use three rollers that may act upon the ball in different directions when being in touch with it. Two side rollers are attached horizontally at both sides of the ball handling mechanism and one front roller in the middle as depicted in Fig. 5. The rotational axes are parallel to the ground (side rollers) and parallel to the robot front (front roller). The rollers are fixed at joints supporting an expansion of the side rollers and a lowering of the front roller hereby varying the rotational axis about +/- 45 degrees. Even when driving backwards the rollers can keep the ball rolling backwards and in touch with the robot, although the rules allow this only for a short time. For kicking the ball an electromagnetic solenoid (Fig. 6) is integrated nearly in the bottom center of the robot's chassis. A plunger transfers a directed impulse with adaptable strength to the ball. By adjusting the side rollers correspondingly, the ball may also be hit at its sides and not only in its center, thus allowing a targeting of the ball within an angular range of about 30 degrees. Via this mechanism the robots can kick the ball on the floor. In the future also higher kicking shall be realized.

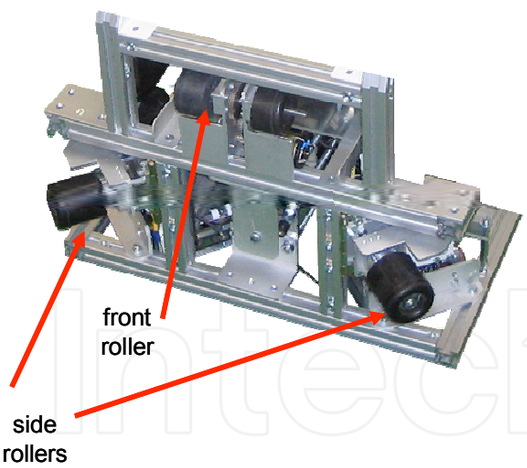

Fig. 5. Rollers for ball dribbling

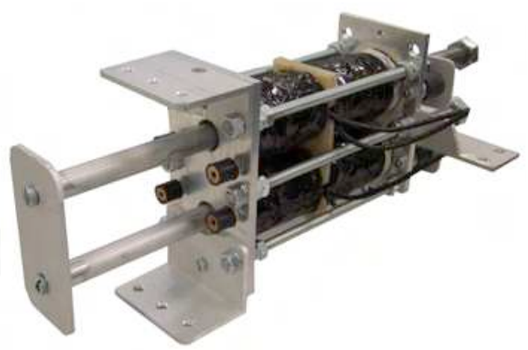

Fig. 6. Solenoid for ball kicking

2.4 Sensory system

Besides the active vision system described above the Paderkicker robots are equipped with several further sensors for monitoring their own status and perceiving their environment. They are located near the actors for ball handling and driving in the bottom layer of the chassis as schematically depicted in Fig. 7.

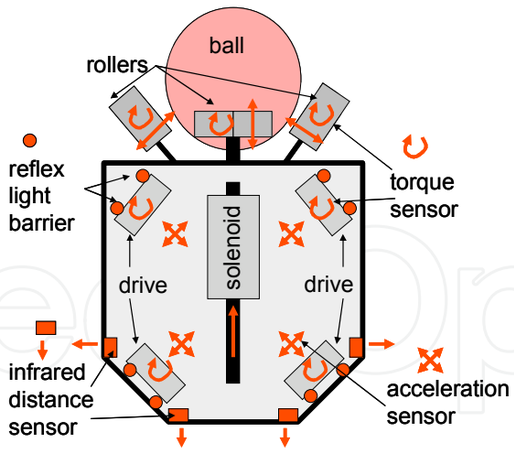

Fig. 7. Sensor and actor location

2.4.1 Motor status

The number of revolutions is monitored for each motor as input for its controller. With help of this value and the current consumption the actual torque of a motor can be estimated. The torque of a motor allows to decide whether for instance a robot is driving against an obstacle (omniwheel motors), whether the ball rollers touch the ball (roller motors) or even during longer driving periods about the floor characteristics and maximum driving velocity on the actual floor. By measuring current and number of revolutions it can also be decided whether one of the omniwheels is spinnning. Furthermore the number of revolutions is used for odometric calculations described next.

2.4.2 Odometry

The sequence of angular movements of the four driving motors is used to calculate a robot's translational and rotational movement on the plane. Together with information about the starting position this allows to estimate the current robot position. It has to be noted that the respective incremental sensors are located near the wheel side and not near the motor (see Fig. 3). They have a resolution of about 1.5 mm. Due to considerable slipping of omniwheels (about 15 to 20%) these calculations are very error prone. Another source for calculation errors are pushes by opponent robots that can be observed frequently although RoboCup rules prohibit them. Since we use four instead of three omniwheels the odometric equation system is over-specified which we use for improving the quality of sensor data.

Further information about a robot's movement is obtained by acceleration sensors at each wheel. They measure the real acceleration over the wheels into two directions entirely independent from the driving motors. Also manual moves are considered. If a wheel spins this leads to faulty large acceleration values. Unfortunately also these measurements are very error prone. Therefore we try to improve odometric data by fusioning the results of several acceleration sensors located at different places within the robot and the data received from the incremental sensors at the wheels. In future the position calculation should be further enhanced by a camera based odometric system that calculates driving direction and velocity from video sequences.

2.4.3 Landmark recognition

Position calculation is most exact if known landmarks can be detected in the environment. For RoboCup lines on the soccer field or goals can be used for this purpose since their position is given. However, if lines are only determined from camera images, the line is usually not detected with sufficient accuracy. This is mainly due to the frame rate of 20 to 30 frames per second and the driving velocity up to 3 m/s that allows only a relatively small resolution regarding time and location. Therefore, the Paderkickers are equipped with two reflex light barriers at each wheel. They are directed towards the floor and can accurately detect driving over a white line on the green floor. A millisecond sampling rate of the eight reflex light barriers' outputs allows to improve the precision of positions determined by odometry considerably.

The goal keeper can further improve its position calculation by taking into account the position of the walls building the goal. If a robot moves within the goal or near to the goal it can measure its distance to these walls. This is particularly helpful since the robot's viewing field is quite restricted in these cases. Therefore, the Paderkicker robots are equipped with four infrared sensors for measuring the distance of the walls into backward direction and to both sides.

3. Modular Paderkicker design

3.1 Functional architecture

The main functional units for driving, ball handling and vision were designed in a modular way as mechatronic functional units (MFU). Each sensor-actor-group is controlled by a separate microcontroller. Originally we planned dedicated designs of mechanical, electronic and embedded system for each MFU to be realized in an independent physical component. Due to cost and time restrictions however, requirements for control of the functional units and electrical motor drivers were gathered and only one microcontroller and driver board was developed to be "universally" used for actor/sensor control and motor driving. Hence, MFUs could not be realized as single physical components. However, logically each microcontroller and driver board belongs to exactly one MFU, which allowed their independent development since resource conflicts are avoided per se.

A Paderkicker robot now consists of a central behaviour module and system control module realized on an embedded Mini-ITX PC board and a microcontroller and the MFUs for driving, ball handling and vision mentioned above, which are further divided into sub-modules as depicted in Fig. 8. The figure also shows the information flow between these modules.

The *driving module* consists of five submodules, one for each of the four wheels and one for central driving coordination. Each wheel is controlled by a separate AVR microcontroller to which also sensors and drivers belonging to this wheel are connected. This microcontroller handles the inputs from several sensors. The actual current consumption and the temperature of the motor as well as the acceleration over this wheel in two axes are calculated. Also two reflex light barriers for recognition of white lines on the soccer field are connected to this board. For goal keepers the AVR controllers for the back wheels also process the inputs from the infrared distance sensors. At the output side an H-bridge is connected for producing the PWM (pulse width modulated) signals driving the motor. This module communicates with its environment via a CAN bus commonly used also in the automotive domain.

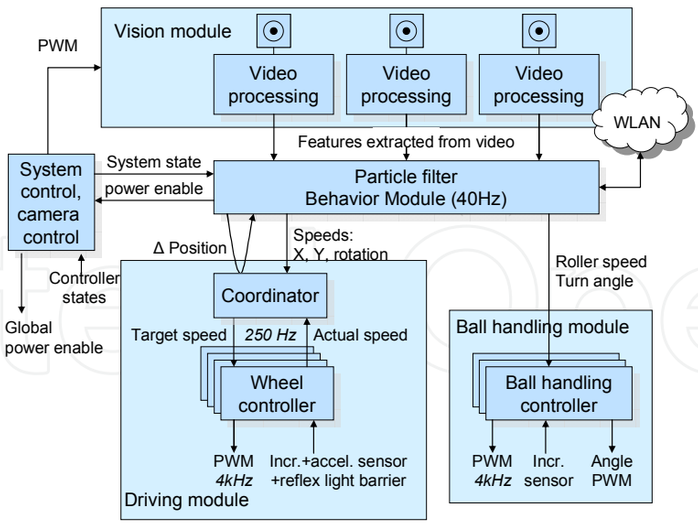

Fig. 8. Paderkicker module structure

The driving coordination submodule is realized on the fifth AVR microcontroller, which is connected to the central PC board via USB. It receives the measurements of the four wheels from the respective microcontroller via CAN, calculates odometric and sensor data and transmits them to the PC via USB.

The *ball handling module* is divided into three submodules two for controlling the two side rollers and a third one for handling the front roller together with the kicking solenoid. For these submodules a separate AVR microcontroller exists one for the side rollers and one for the front roller and solenoid. The third microcontroller (responsible also for kicking) is connected via an H-bridge to a cascaded voltage doubler that generates the fourfold supply voltage (about 100 – 120 Volt). This voltage is needed to load a capacity of 12 mF for moving the plunger. Via an IGBT device the solenoid can be fired. Thus the solenoid cannot only be fired; also its kicking force can be influenced by switching off the power supply while discharging.

The *vision module* has two tasks. On the one hand it separately processes the images perceived by the three cameras as described in Section 4. These results are fusioned by a particle filter. Video processing and particle filter are realized on the PC board. On the other hand it is responsible for generating the PWM signals controlling the pan-tilt units of the cameras which are realized by model craft servo motors. The same technique is also used for controlling the servo motors that adjust the joints (and hence position) for the ball handling rollers. For generating the six PWM signals for all three cameras the same AVR microcontroller is used, which is also responsible for the central system control.

3.2 Hardware architecture

The functional structure described above is mapped onto a hardware architecture as depicted in Fig. 9. The central processing unit is a Pentium M ULV PC running under Linux. Here the vision algorithms (see Section 4), the particle filter and the behaviour-based system (see Section 5) are realized. For control purposes of the modules described above in total eight AVR microcontroller boards are used. This board was developed only once for all modules (Fig. 10, Fig. 11). It is equipped with an AT90CAN128 microcontroller, which al-

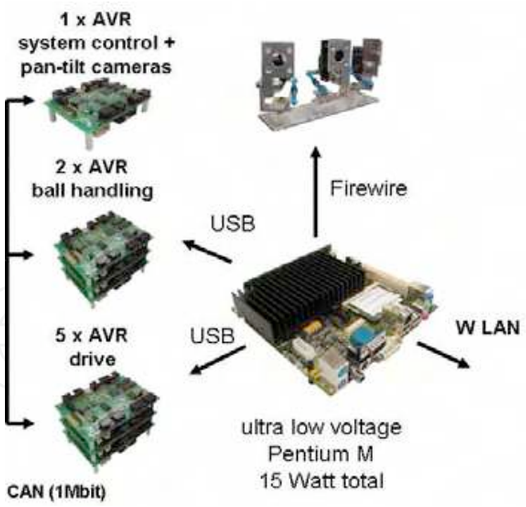

Fig. 9. Paderkicker hardware architecture

ready comes with a CAN bus interfaces. Two USART outputs provided by the microcontroller are connected to an interface component (FTDI) which realizes an USB 1.1 slave interface. This interface is handled by the Linux PC on the Mini-ITX board like a usual file interface. All digital inputs and outputs of the AVR microcontroller are connected via opto couplers (Opt.) or drivers respectively for security reasons. For debugging the status of all inputs and outputs is visualized by LEDs.

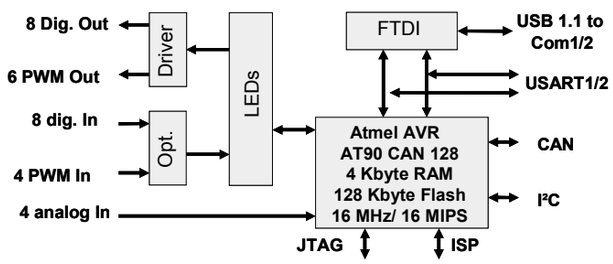

Fig. 10. AVR board (principle structure)

Fig. 11. AVR board (photo)

3.3 Software architecture

3.3.1 Team communication

In a soccer team the robots do not only act autonomously but they have to coordinate their actions. Therefore they need a communication infrastructure. Furthermore they have to react on the referee's commands and their parameters have to be adapted to the actual soccer field and its environmental conditions. The Paderkicker robots use a central team server for this purpose (Fig. 12). Each robot has to register when it is ready to play. Also different master panels (e.g. the referee box) concerning the team as a whole or concerning the initialization or operation of a specific robot can connect to the central team server. The communication is realized by a TCP/IP network with fixed addresses (LAN and WLAN). Particular attention has to be given to a robust WLAN connection, since this connection tends to

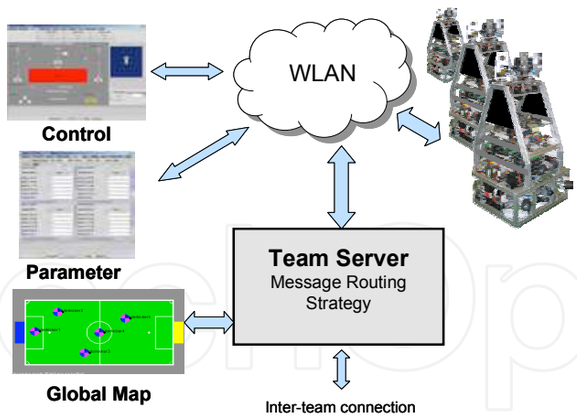

Fig. 12. Paderkicker communication structure

break several times during one halftime of fifteen minutes. A robot that has to restart its entire system from scratch each time it looses the WLAN connection, will not be able to actively participate in the game. Therefore the Paderkickers were designed very robustly against WLAN break down. They carry on with their actual autonomous behaviour and try to re-connect with the network without restart.

3.3.2 Logical robot architecture

The software architecture of a single Paderkicker robot can be described from two viewing points: the processes and threads responsible for processing the different kind of data and the logical or functional separation of tasks and the according data flow.

The *logical software architecture* is structured according to the triple tower model of Nilsson (Nilsson, 1998) that distinguishes between *perception, model* and *action tower* (Fig. 13).

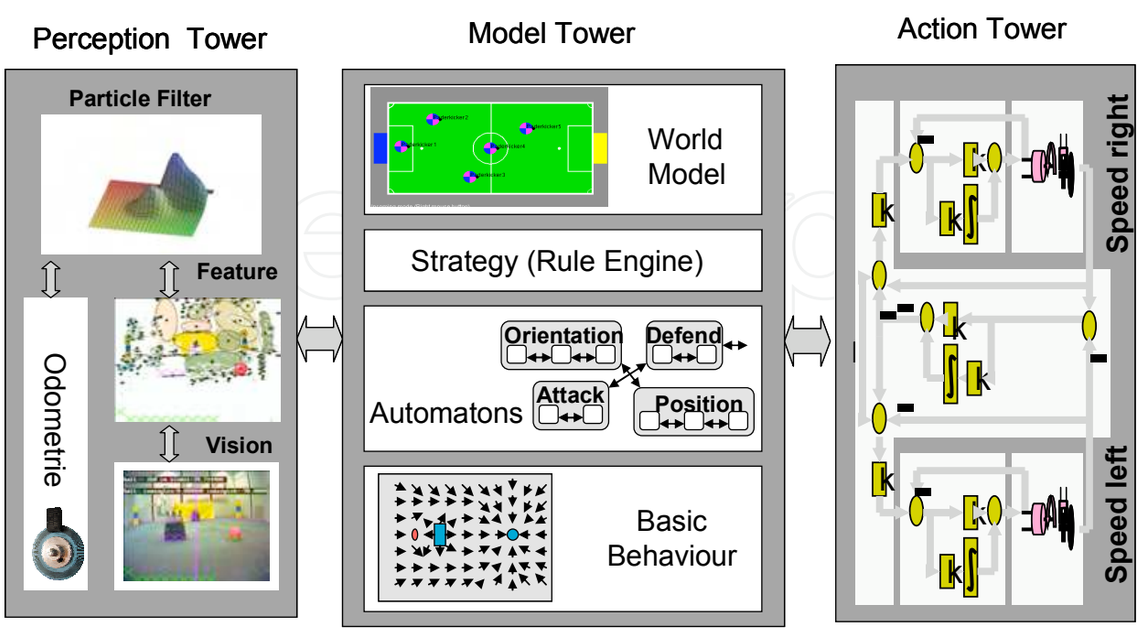

Fig. 13. Paderkicker software architecture

The *perception tower* is responsible for processing and fusioning the sensor inputs and providing them to the model tower in a more compact problem specific way. At the lower level features are extracted from camera images. Also information from the driving system's sensors and from the touch sensors of the ball handling mechanism are evaluated. At the next higher level the sensor information is pre-processed. For instance the debouncing of touch sensors or the different stages of image processing (see Section 4) determining the position of various objects like goal, lines etc. in relation to a single robot belong to this layer. Also the odometric calculations belong to this layer.
Since the Paderkicker robots have four omniwheels, an over-specified system of equations has to be solved. The distance driven by each wheel is sampled with a rate of 100 Hz. Via a table driven model determined by a modelling tool according to a robot's geometry the actual translation and rotation angle of a single robot is calculated. At the next higher level this data is fusioned with the positions calculated for other objects by a particle filter (Bayes filter).
A typical characteristic of the model craft servo motors used in the Paderkicker robots is that they do not deliver any feedback about their actual position. Hence we use a model driven approach to estimate the previously set angle of the servos. This model driven position estimation uses sensor information at various levels of abstraction. If for instance the ball has to be at a proper position for the ball handling system, this position estimation component needs the most recently perceived position information for the ball before it is filtered by the particle filter. In other cases where for instance the reliability of the ball position is more crucial due to its larger distance the filtered position may be preferred although it is received with a small delay. This information may for instance be needed if a robot has to drive towards the ball. Therefore, it has to be decided separately for each behaviour to which degree its sensor information has to be preprocessed.
The *action tower* realizes the control of the actor systems. Here the physical signals are generated for the nominal values in the form needed by the actors. Via specific hardware components on the AVR board PWM signals are generated or analogue values for voltage or current are measured for controlling the motors. Simple control loops are only used for the driving motors and the roller motors. We use cascaded PID controllers for current and revolution control. Their nominal values are calculated by the driving coordination module on the basis of the actual translation and rotation angle for each wheel.
For the future it is desirable that slight changes of the driving system mechanics resulting from maintenance or reconstruction are handled by an automatic calibration. Also adaptive control techniques should be used for adjusting controller parameters automatically. This is particularly desirable, since the maximum speed of a robot depends on the actual underground and its characteristic rolling friction which may change frequently. As always a certain reserve has to be granted by the controller in order to allow for robot turns also at maximum speed, a controller that automatically adapts to the actual rolling resistance would be advantageous.
The *model tower* is responsible for planning and selecting appropriate actions at various layers of abstraction. The lowest level, the so called security level, takes care that a robot is switched off in critical situations, e.g. if due to overheated motors or empty accumulators a robot might damage itself. On top of this security level a reactive layer is realized using the motor schemes developed by Arkin (Arkin, 1998) (see Section 5). This reactive layer is controlled by the next higher level that determines action sequences for a robot. At the topmost

level the playing strategy of a robot is coordinated with its teammates. Via WLAN the information about the positions of goals, opponents or other obstacles are distributed in the Paderkicker team. This information is used to determine which role (attacker, defender, etc.) a robot should take (see Section 5).

3.3.3 Process architecture

From the process point of view there is a main process called Brain running on the ITX PC board that opens several threads. The main process is responsible for action planning. One thread works as internal router for exchanging messages between other processes or threads. Per interface connected to the Brain process another thread is opened. There exist three interfaces one to the driving system, the ball handling and the pan-tilt unit of the vision system. Each of these interfaces is realized via a USB connection of the ITX PC to the respective microcontroller. In principle each microcontroller can be viewed as one additional process, since no operating system is used on the microcontrollers. Among each other the microcontrollers are connected via CAN bus.
For each camera one process is started for feature and object recognition. This information is passed to the router thread of the Brain process. Also the particle filter is realized as separate process that communicates to the Brain process via the router thread.

3.4 Message format

For integration of the separate Paderkicker components that communicate over different bus systems (CAN bus, USB, TCP/IP for communication between robots) a homogeneous message format was designed. Also the different master panels use this format. If a robot component is changed or a new panel should be integrated, a new message has to be defined, if the existing ones should not cover this functionality. For the Paderkicker robots an XML format was defined where all messages are registered. These data is used to generate access routines in different programming languages (C, C++, Java, Python) in a semi-automatical way. The need for a message format that is both robust and computationally cheap led to the following design (Fig. 14).

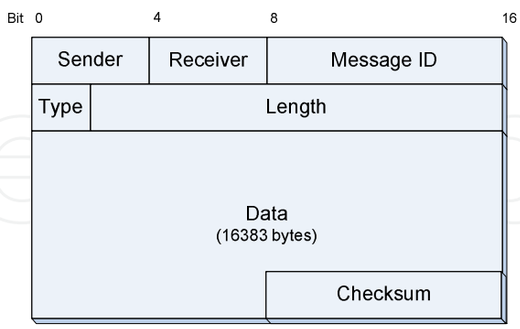

Fig. 14. The Paderkicker message format

Every message is preceded by a message start delimiter. Thereby, each subsystem is able to figure out the starting point of messages in continuous data streams with low processing overhead. Fields to uniquely specify sender and receiver follow. By these fields the different subsystems are addressed, as e.g. the behaviour system, the particle filter, or the TeamServer. The message ID specifies the actual meaning of the following data bytes. Thereby it

is also possible to extend the current message set by not yet foreseen message possibilities. The type describes the nature of the message, which provides the following possibilities: *set*, *request*, *info*, and *poll*. This is needed for certain message IDs where it is possible to set or get the according data, or even request the specified data to be delivered recurrently. Finally, the checksum provides means to recognize errors in the message.

4. Vision

Research regarding computer vision is done mainly in the area of real-time image processing. An optimized algorithm for low latency real-time colour segmentation (Stichling & Kleinjohann, 2002b) now was adapted to run on a PC under an ordinary Linux system. It even performs well on a PDA and was formerly implemented on a Trimedia TM 1100 video processing board running at only 100 MHz. A more detailed description follows later on. Three digital Firewire cameras are mounted on pan-tilt units to cover the whole 360° view being used as an active vision system. This configuration leads to an overall higher resolution and to a larger viewing area in terms of visual depth, compared to an omnivision system. Hence, the robots are capable of seeing distant objects much better. Higher visual coverage of the environment will become more important in the near future when the field will likely be extended in size again. The Firewire cameras are directly connected to the Mini-ITX board running a Linux system where the according video streams are processed simultaneously. In the following first our image processing algorithms will be described. Afterwards we deal with the recognition of RoboCup specific objects like ball, field, lines, etc. and describe a future improvement of image processing based on the discrete cosine transform.

4.1 Colour segmentation

In the environment of RoboCup special colour codes are used to make sure that objects are distinguishable from each other. Therefore a real-time colour segmentation of the camera streams is essential for the system. The above mentioned algorithm is optimized for an extremely low latency, so the overall latency decreases as the depending behaviour system can react faster. The main difference of the used colour segmentation algorithm compared to other segmentation algorithms is its linear processing. The algorithm processes an image line by line and occurring data dependencies refer only to already obtained data. Assuming adequate processing power, the latency of the colour segmentation is the transmission time of one image plus the computing time of the processing steps for the last line.

In a first step the algorithm does a so called "Line-based Region-Growing" in which regions with similar colour are grown while stepping through the individual pixels of the line. In a second step the just created regions are merged together if they meet defined colour-similarity and spatial distance criteria.

As this method was planned to run on embedded devices, the computational need of the algorithm had to be as low as possible. Therefore the underlying data structure has to be simple, especially in terms of computability. In computer vision algorithms moments are often used (Prokop et al., 1992) as they are significant and easy to compute. In this case moments $M(p,q)$ up to second order are used as region descriptors. For a region, the moment $M(p,q)$ is defined as follows:

$$M(p,q) = \sum_{(x,y)\in region} x^p y^q \tag{1}$$

The moments *M(0,0)*, *M(1,0)*, *M(0,1)* describe the number of pixels in x and y direction and the centre of a region. Additionally, to get the orientation of the moment, the central moments *C(p,q)* with the according centre (c_x,c_y) of the region are needed.

$$C(p,q) = \sum_{(x,y)\in region} (x-c_x)^p (y-c_y)^q \tag{2}$$

The central moments *C(1,1)*, *C(2,0)*, *C(0,2)* together with the above mentioned three moments describe one region. For visualization purposes they can be used as a representation for an ellipse, but as ellipses are computationally difficult to handle this is not used for the algorithm's calculations. Additionally the average colour of the regions is part of the descriptor, whereby the YUV colour space is used. The regions are stored as a connected list, whereby the region-growing step assures an ordering of the regions by the y-coordinate of the uppermost pixel of one region. This spatial ordering speeds up the following region-merging step. In this step the regions created before are merged together if they are spatially near to each other and are similar in colour (see Fig. 15). As the moments can be visualized

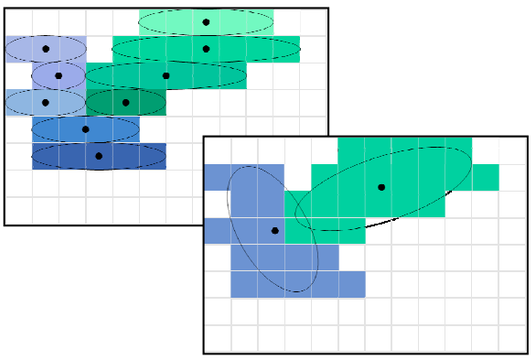

Fig. 15. Region-growing and region-merging

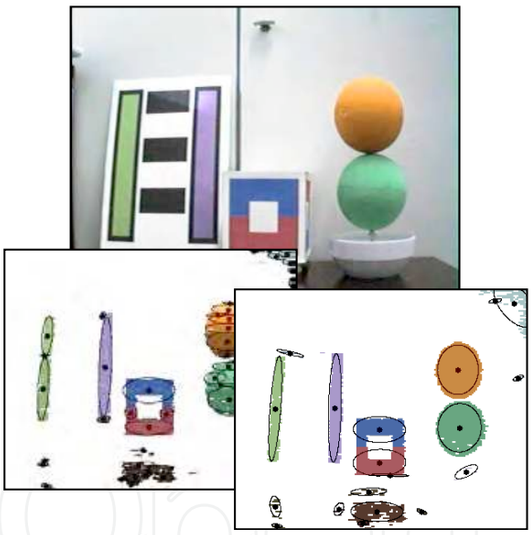

Fig. 16. Influence of segmentation parameters

as ellipses, spatial near can mean overlapping ellipses. The YUV colour space used for region-growing is not very suitable for region-merging because higher Euclidian distances in the colour space are treated as similar. In this step one would like to merge regions of similar colour disregarding the shading, therefore the HSI colour space is suitable for this purpose.

So, regions are merged together if their Euclidian distance in the HSI colour space falls below a specified threshold and if they overlap. The actual region is compared to regions whose lowermost pixel is not higher than the uppermost pixel of the actual region. Hence only the data from the line-based region-growing step of the actual image line and the pre-

viously created regions are needed to do the region-merging. Fig. 16 shows three different resulting images of the colour segmentation algorithm: In the background the original image, above two colour-segmented images. The lower left image shows segmentation with already found regions, but with a lower threshold for the allowed colour difference of regions to merge. In the other segmented image the elimination of shadows and of highlights can be observed.

4.2 General object detection / recognition

The detection and recognition of objects in general is based on the above described colour segmentation algorithm. The coloured regions found by the algorithm and the corresponding two-dimensional moments are checked for defined properties to identify objects like the ball, the goals or other robots. The pre-defined object properties are stored on an USB flash drive which is located at the on-board Mini-ITX board. Every time the robotic system is started the parameters are loaded from the USB flash drive. They can be read out and modified over a control applet which connects over a wireless connection to the central server where all robots are logged in. At the moment the parameters are loaded into the Trimedia respectively into the software vision module on the robots which are equipped with Firewire cameras. Recognized objects are checked for plausibility and deleted if necessary (e.g. when more than one ball has been detected or if the detected ball is outside the field).
In the application area of RoboCup special premises regarding the computer vision apply, e.g. defined colours for different objects. So, some colours are of more relevance than others. The above described colour segmentation algorithm does not benefit from this a priori knowledge, as it only uses the *achromatic threshold* to identify pixels of interest. For this reason, colour lookup-tables were implemented to make use of the prior knowledge of relevant colours. As a result fewer regions are extracted from the image, so that even smaller ones can be taken into account for object recognition. Thus, edge markers can be detected from a greater distance.
Using these lookup tables leads to a robust and fast possibility to assign pixels to an object without the colour space conversion from YUV to HSI for every pixel. This significantly improves the speed of the colour segmentation algorithm resulting in higher and more constant frame rates.

4.3 Field detection

Detecting the field starts at pixel-level of the camera image. Again, the before mentioned YUV lookup tables are used to decide whether a pixel belongs to the field. The camera image is divided into 36 x 44 fragments for which the number of field-pixels is counted. A threshold decides whether this fragment is part of the field. Another matrix of the same size is used to mask the outer dimensions of the field. This method allows detecting the field even if it is hidden by parts of a robot. Another advantage of the field detection is that it can be used to mask out objects outside the field (like ball-like red shoes of a viewer). As this detection runs in parallel to the actual vision system, it does not interfere with it, leading to more stable frame rates.

4.4 Landmark detection

For a robust localization the detection of landmarks is very helpful in particular if the landmarks are unique. In the RoboCup environment landmarks have defined colours and dimensions and additionally they are assigned to a certain side of the field. In Fig. 17 two different landmarks are shown. They normally mark corners of the field. In this vision-system-test-case the two different landmarks are located on the same side of the field and in addition no goal is present.

The vision system draws rectangles around recognized landmarks and writes the corresponding internal number aside. Landmarks are recognized using the coloured moments from the colour segmentation algorithm by successively processing the list of coloured moments, so that landmarks can be detected on-the-fly. Termination criteria make sure that not every moment is checked against all others. A landmark in the RoboCup environment consists of three coloured regions of the same size, whereas the two outer regions are even of the same colour. The outer colours are defined by the side and are of the same colour as the goal. Hence, the list of coloured moments is searched for corresponding moments which match an alignment and colour combination with a specific variance. A weight is assigned to found landmarks, depending on the number of appropriate coloured moments, whereupon a landmark is considered complete when the weight is three. Complete landmarks are no longer used for further combination-searching, avoiding the recognition of multiple landmarks at the same location. The dimension of the landmark is approximated using the height of the middle moment as this turned out to lead to a stable recognition.

4.3 Ball detection

For the detection of the ball, again the extracted coloured moments from the colour segmentation algorithm are used. The ball is an object with defined colour and size, and the list of coloured moments is searched for corresponding ones. Moments which are over the bottom line of landmarks recognized to be complete are ignored as this would be a ball outside the field. Suitable regions that are spatially near are merged together. Depending on the illumination of the field a dark shadow is under the ball. This shadow could be detected as an obstacle directly in front of the ball. Hence this shadow is masked to avoid this behaviour. The masking can be seen in Fig. 18 directly below the recognized ball. Under certain circumstances it is possible that multiple coloured moments match criteria of the ball. In this case of multiple possible ball locations, the largest one with the lowermost position in the camera image is taken as the ball position. Unfortunately this is not always correct, so that the actually used ball position is calculated by a particle filter which is fed with all possible ball positions.

4.5 Free Space detection

Another essential algorithm for autonomous robots that are part of a dynamically changing environment is a detection of free space. This information then is used to avoid obstacles and for general path-planning. In general the free space detection is very similar to the field detection but the two algorithms differ in detail. Like in the field detection algorithm the image area again is divided into 36 x 44 fragments. They are checked for “dark” pixels and if the amount of dark pixels exceeds a certain threshold, the corresponding fragment is marked being occupied. Afterwards the matrix is passed through column by column from bottom to top and at the image position of the first occupied matrix-fragment an obstacle

marker is placed. These obstacle markers are visualized as red bars as depicted in Fig. 17. Special attention has to be paid to the fact that our robots are equipped with an active vision system, where all cameras have two degrees of freedom to be moved. Hence, it can happen that a camera sees parts of the robot itself, which are black coated. It must be prevented that these parts are recognized as obstacles. As the actual camera position and the camera's aperture angle is known, a calculation of a suitable *robot-mask* depending on the camera position is possible. This mask is also shown in Fig. 17, but as no part of the robot is visible, all robot-mask markers are at the bottom of the image.

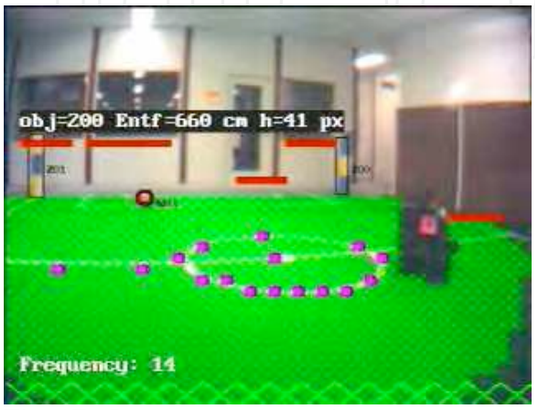

Fig. 17. Landmark recognition, free space detection with obstacle markings and line segment detection

Fig. 18. Ball detection with shadow-masking

4.6 Line recognition

As the lines on the field are defined by the rules of RoboCup, they suit as distinctive attributes for localization. The lines in all divide the field into sections, but one single line is not characteristical for a certain section, because there are the same markings for both sides of the field. As the camera perceives only a section of the field, there are multiple combinations of lines possible, each belonging to another section of the field. Hence, line recognition is inapplicable as the only sensor input for localization; it has to be merged with other sensor data contributing to more overall accuracy in localization.

There are many approaches for line recognition, most of which are very expensive regarding the need for computational power. The gradient based Sobel filter had been implemented on the old Trimedia but was mutually exclusive with the colour segmentation. Hence, another way of recognizing lines had to be invented, which would have less need for additional computational power. With the change to the current PC-based vision system, a lack of computational power is no more existent, but this new method saves computation time which can be used e.g. for the behaviour system. The field-detection algorithm described above seemed to be suitable to calculate a line-recognition-like operation along the way. The image is passed through bottom-up column by column. In the case of a field-fragment which is followed by a mostly white fragment which is followed again by a field-fragment, the inner fragment is assumed to be a so-called line-fragment. The fragments found by the algorithm are marked in the output image as shown in Fig. 17. This algorithm is not line

recognition in terms of vectorizing recognized lines, but rather calculates supporting points of possible lines, which then can be matched by the particle filter.

4.7 Structural simplification of images using a discrete cosine transform

One of the main challenges of computer vision is the extraction of distinct image features. A rule of thumb is: "The simpler the image, the simpler the extraction". One method to simplify images is to reduce the number of used colours before starting the colour segmentation. In the following an according method is described, which utilizes the effect of a discrete cosine transform. The discrete cosine transform is broadly used in image compression algorithms, known from the most popular image compression algorithm "JPEG". The mathematical definition of the DCT-II is as follows:

$$X(n) = \frac{2}{N} C_n \sum_{k=0}^{N-1} x(k) \cdot \cos\left[\frac{\pi(2k+1)n}{2N}\right], \; C_n = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } n = 0 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

n=0,1,…,N-1 and *X(n)* is called the *n*-th spectral component. Applied to a block of pixels, it transforms the pixels into the frequency domain, where high frequencies are sharp edges in

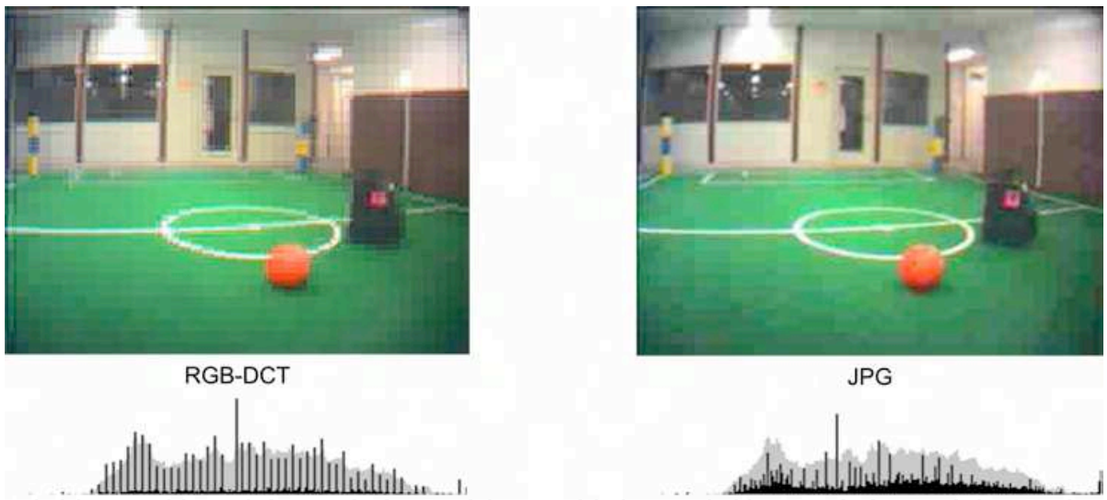

Fig. 19. Resulting colour distributions of DCT applied to the RGB colour space in comparison to ordinary JPEG compression

the image. An image compression is almost always done by cutting off higher frequencies. Talking of frequencies means that the image is transformed into the frequency domain. Then, changes are made to the frequencies, e.g. the higher ones are cut off, and at last the frequency data is transformed back to the time domain. Here, a JPEG-like "compression" is done, but not like in JPEG, the colour space is RGB. Actually the only "compression" consists in reducing the bits used in the frequency domain. Later on this method is referenced as "RGB-DCT". It can be used to reduce the structural complexity of images, e.g. by reducing the amount of used colours, while paying attention to the colour balance. This reduction has to be done in a special way, so that the overall distribution of colours remains unchanged. In Fig. 19 the resulting distributions of the application of the RGB-DCT and the

standard JPEG compression are compared. The resulting distributions are each drawn in black over the distribution of the original image drawn in grey. The RGB-DCT's distribution follows the original one's very accurately unlike the resulting distribution of the JPEG compression. On screen better than printed, the colour distortion caused by the JPEG compression is apparent.

This effect can be utilized as a pre-processing step before the above described colour segmentation algorithm. The RGB-DCT applied to an image leads to smoother images: firstly the image is transformed into the frequency domain. Then, a significant amount of higher frequencies is cut off. At last, the image's data is restored by back-transforming. As now higher frequencies (sharp edges) are reduced, the image is smoother than before. A fact for colour segmentation in general is: the smoother the image, the bigger the moments; as edges can prevent the merging of two regions with similar colours. Of course, there has to be found a balance between over-sharp and over-smooth images. If you use the RGB-DCT prior to the colour segmentation, it can significantly reduce the parameterization effort due to a reduction of the image-complexity. Badly parameterized colour segmentation can lead to an image scattered with many very small moments, and that easily happens. The prior application of the RGB-DCT leads to a significantly better colour segmentation result. The total amount of moments decreases and the average size of a single moment increases (Fig. 20). In this particular example, recognition of the ball is impossible with the left-hand side colour segmentation but is possible with the right-hand side's one. Here, the same colour-segmentation parameters were used, while in the right-hand's image the RGB-DCT was applied before the segmentation – with only 6 bits of relevance in the frequency domain.

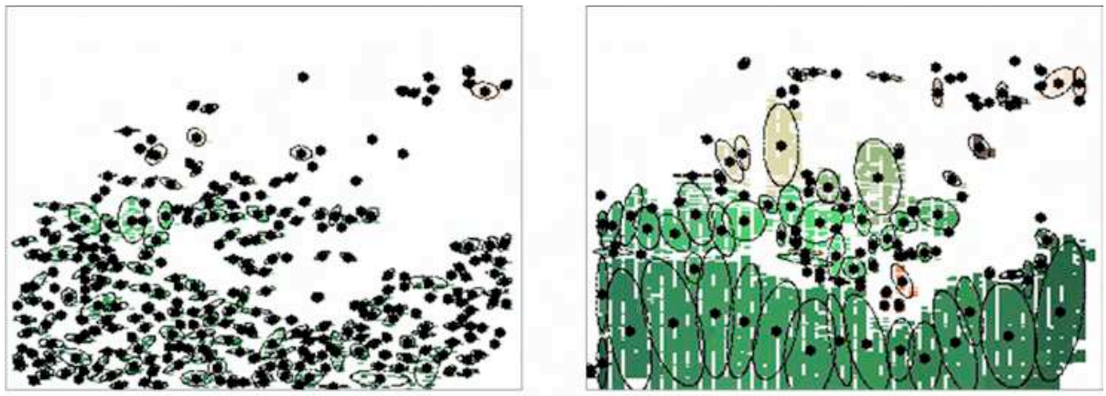

Fig. 20. Resulting colour segmentation moments with and without prior application of RGB-DCT, while using the same colour segmentation parameters

5. Behaviour-based system

Abstracting the hardware perception and action execution, the behaviour-based software is structured as can be seen in Fig. 21. There we distinguish between the *world model, strategy, automaton, and behaviour module*. All modules have to be autonomous in a way that keeps the robot full functioning even in case it has lost wireless network connection to its teammates

which is not uncommon under typical tournament conditions. In this case, the robot sticks to its last committed strategy.
Once the perception is pre-processed, the extracted information is stored in the robot's individual *world model module*. The world model module captures information specific to the robot itself regarding

- the robot's unique identifier
- its pose on the field (x, y, orientation),
- the position's confidence,
- the 2D ball-position as seen by the robot,
- the ball-position's confidence,
- the current role of the robot, and
- possible roles.

In addition it holds team information, like e.g. the global map, the teammates' roles and the game situation. The information regarding the robot's possible roles is a bit mask specifying which role the robot is able to fulfil. A goalkeeper, e.g., is not allowed to switch its role during the whole game. This can be specified by setting possible roles to only the goalkeeper role. All remaining modules (strategy, automaton, and behaviour), are allowed to access the world model module's information.

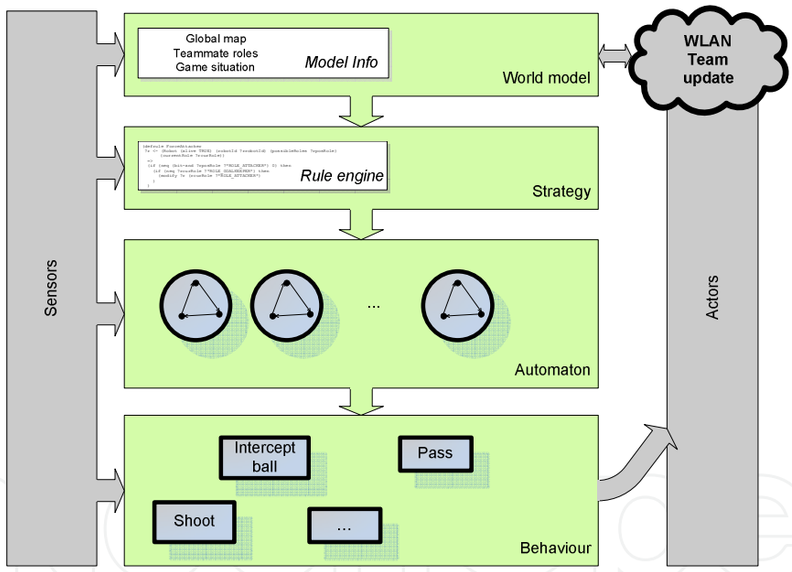

Fig. 21. Behaviour-based control flow in the Paderkicker robot

The world model information is sent to the TeamServer at 4Hz. The TeamServer will be described in the next section. It aggregates the information, decides upon a team strategy, and sends the whole information to each teammate again at 4Hz, so that the individual robot's world model information can also be treated as a life beat. Once, a robot has lost connection and is thus not transmitting its world view to the TeamServer, it is considered as lost and the strategies are reassigned within the team.
Depending on the teammates' world models the *strategy module* then decides on the current strategy for every robot. Strategies include e.g. *Defend* and *Attack*, but also standard situations like *Kick off* or *Penalty*. This is done by the strategy module's rule engine.

After the strategy module has committed to a role, the strategy is realized by an according finite state machine of the *automaton module*. Its states include e.g., ball facing, or staying between goal and opponent.
Every state in the automaton module is mapped to a low-level reactive behaviour like "move to ball" or "kick ball" in the *behaviour module*. This module finally executes the actual actions by sending the calculated action vector to the corresponding hardware boards. It is behaviour based in terms of Arkin's Motor Schemes (Arkin, 1998). Our behaviour system allows for a distinction between cooperative and competitive behaviours and behaviour control through time excited evaluation functions. It consists of a set of low-level behaviours represented by vector fields that have to be combined in order to result in a vector that can be sent to the actuators.

5.1 Team strategy

The strategy of the Paderkicker team is realized by role arbitration based on the robots' propagated world models. There are different ways to actually implement a team strategy module. One way is to use a dedicated server process that distributes roles and commands to the teammates after each of them has provided the TeamServer with its subjective world view. Another way is to design the system responsible for strategy arbitration in a distributed way, so that every robot has the same rule base. Each teammate would then have special rules for the team strategy by which it could independently come to a decision in a given situation. We started with the second solution, implementing our team play as a distributed expert system. However, when the need arose to form a mixed team at the RoboCup world championship in Bremen 2006 we chose the first solution with a dedicated server being able to connect teams with completely different hard- and software. This led to the development of the *Paderkicker TeamServer* running a separate expert system for the cooperation of robots from different teams. The interaction with a TeamServer is particularly important for mixed teams, which will become more and more common in the future. The TeamServer acts on a coach level as in real soccer games. The different robots register at the TeamServer and propagate their individual world model as described before – with the only difference that the whole team thus has only one expert system running responsible for strategy arbitration. Although the teammates now rely on a central server this does not have any impact on the individual robot's robustness: Even if each robot had its own role arbitration module the individual expert system would not trigger any new role without having a network connection in the first described solution.
The core of the TeamServer is the rule-based expert system *Jess* (Friedman-Hill, 2005), which takes as input the teammates' world models. In addition, it is directly connected to the referee box, thereby getting additional game status information, like *game start* and *stop* and special game situations like e.g. *corner kick*. Besides functioning as the team communication router the TeamServer has the task to arbitrate roles for the teammates. This is very intuitive by using the declarative programming methodology of the Jess expert system. Thereby, the expert knowledge is human readable and can be specified as "what is to be solved" and not procedural as "how something should happen". The system relies on the expert system's ability even under incomplete world information. It holds facts in its working memory, which are actually variables of the actual world models sent by the teammates, e.g. "ball is in the perception range of the robot". The rule-base holds the domain specific expert knowledge coded as rules. A typical rule consists of preconditions and actions that are to be exe-

cuted when the preconditions hold. The rule engine's pattern matcher then matches the rule premises against the facts in working memory and creates an agenda for execution of the activated rules.
The rule-based system is fed with the subjective world models of all teammates. In our example if e.g. the ball is close to a robot a rule will fire that changes the role of the robot to attacker under certain conditions and reassigns the previous attacker a different role. As part of that process the rule that forces the robot closest to the ball can be seen in Fig. 22. It uses the facts *alive*, *robotId*, *currentRole*, and *possibleRoles*. If the premise (the part in front of the "=>") matches, the action specified subsequently is executed. This means that in order

```
(defrule ForceAttacker
 ?r <- (Robot (alive TRUE) (robotId ?rrobotId) (possibleRoles ?rposRole)
       (currentRole ?rcurRole))
  =>
  (if (neq (bit-and ?rposRole ?*ROLE_ATTACKER*) 0) then
    (if (neq ?rcurRole ?*ROLE_GOALKEEPER*) then
      (modify ?r (rcurRole ?*ROLE_ATTACKER*)
    )
  )
```

Fig. 22. Example for a role-arbitrating rule in the TeamServer: The teammate with the smallest distance to the ball is enforced to be attacker, but only if the robot is allowed to be Attacker

for the rule to be executed there must be a robot in the working memory, which is alive and has sent its possible roles, its ID and its current role. Side conditions can be easily specified in such a system: In the example the goalkeeper is omitted from this specific rule.
At the moment the Paderkicker team supports the five following roles:

- Attacker
- Goalkeeper
- First and second line defender
- Supporter

As the robots may fade away because of network connection or hardware problems while the game is running, some roles will not be possessed by any teammate. Therefore, the TeamServer has to prioritize the individual roles by their importance for the team. The priorities are dependent on the ball-position on the field. In case the ball is in the team's half of the field, defending the goal is the most important aim, which leads to the following priorities:

1. Attacker
2. Goalkeeper
3. First line defender
4. Second line defender
5. Supporter

In the other case, a more offending style is advisable:

1. Attacker
2. Goalkeeper
3. First line defender
4. Supporter
5. Second line defender

This means, that if one teammate drops out of the team the teammate with the least important role is assigned the lost teammate's role.

5.2 Automatons realizing the strategy

Once the role is assigned to a robot, it executes the finite state automaton for that particular role in the current game situation. It is only dependent on the robot's own perception, i.e. if the TeamServer performs a new phase of role arbitration due to game situation changes or

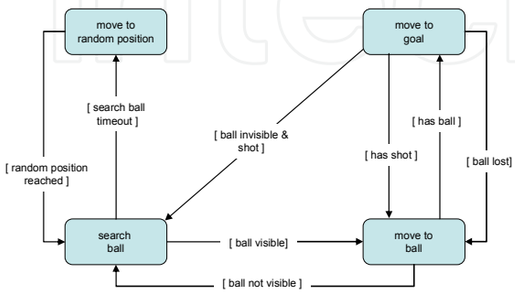

Fig. 24. Attacker's strategy as a finite state automaton

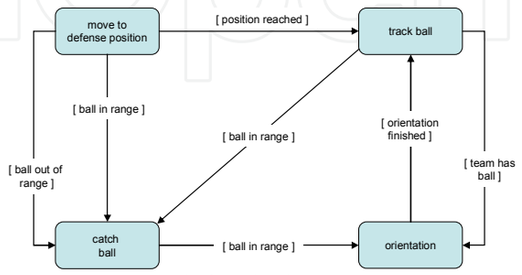

Fig. 23. Line defender's strategy as a finite state automaton

because a robot has been lost, the role change of a teammate results in a switch to the new finite state automaton corresponding to the new role and game situation.

The attacker's duty is to find the ball (with the help of all other teammates' world models including the ball-position), intercept it, drive to the opponent's goal and shoot (Fig. 24). The line defenders are assigned two lines of defence orthogonal to the goal and at different distances (Fig. 23). The line defenders have to work together in order to intercept the ball before it goes into the own goal or leaves the field (Fig. 25). If only one defender is available in the team it tries to stay between the ball and the goal. The supporter has the task to stay near the attacker to be the first choice to become the new attacker, if the old one has lost the ball.

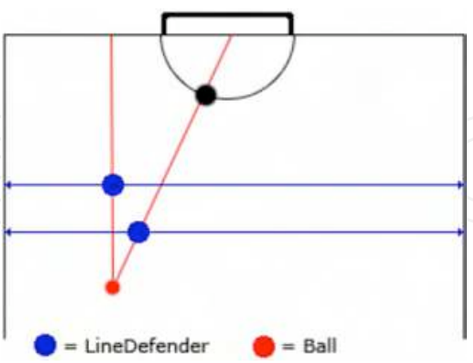

Fig. 25. Positioning of the two line defenders

5.3 Low-level behaviours

Each state in the automaton is realized by a reactive behaviour assembly which is mixed out of one or more low-level behaviours like, for instance, ball dribbling, tracking an object with

the cameras or moving towards a specified object. After all behaviour assemblies have been processed with the actual perception input, a result vector $R = (r_s)$ is generated out of the assemblies' individual outputs, specifying a value to set for every servo s of the Paderkicker, which has the following capabilities:

- 2D vector and orientation for the omniwheel motor
- Position and speed of the three ball handling rolls
- Direction for the three pan-tilt cameras to look at
- Whether the shooting device should be activated or not.

For the mixing process of the behaviour assembly, two different modes exist:

- *Competitive mixing* chooses the result of the strongest low-level behaviour as the output for the whole behaviour assembly, also known as the *Winner-Takes-All* strategy.
- *Cooperative mixing* combines all low-level behaviour's by a weighted sum.

E.g. two-dimensional movements are typically generated by using cooperative behaviour mixing, while competitive mixing is more appropriate for calculating the direction of the pan-tilt camera.

The mixing operator takes a fixed-sized vector $\vec{C}_b$ for every low-level behaviour b as input which specifies the desired parameters for each available actor possibility:

$$\vec{C}_b = \begin{pmatrix} (c_{1,b}, v_{1,b}, m_{1,b}) \\ \vdots \\ (c_{n,b}, v_{n,b}, m_{n,b}) \end{pmatrix} \tag{4}$$

The elements contain the actual behaviour value c, a vote v denoting the desired strength the value should be given in the final vector field, and the mode m telling the mixer whether this should be mixed competitively or cooperatively. In the same vector $\vec{C}_b$ modes can be different, whereas the modes for the same servo must be the same for all behaviours. If a behaviour does not want to set a certain servo, it can set the corresponding vote to 0. In addition to $\vec{C}_b$ the mixing operator needs so-called *gain* values g_b for each behaviour by which the automaton's state can configure the diverse low-level behaviours in the behaviour assembly. Together with the servo's value c, the mixing operator calculates the weight for it. Putting all together the result r_s for servo s in one cycle is done as follows:

1. Determine the weight $w_{s,b}=g_b \cdot v_{s,b}$
2. Calculate $r_s = \begin{cases} c_{s,b}, \text{if } \mathrm{m_{s,b}} \text{ is competitive} \\ \sum_{b=1}^{B} \mathrm{k_{s,b}} \cdot c_{s,b}, \text{otherwise} \end{cases}$, $\mathrm{k_{s,b}} = \mathrm{w_{s,b}} / \sum_{b=1}^{B} \mathrm{w_{s,b}}$ is the normalized weight of behaviour b for a servo s.

6. Adaptation and learning

Especially in the RoboCup domain the need to increase the intrinsic robustness of the robots is obvious. Even more, as the environment gets more complex and uncertain the need for

the robots to consider every bit of information regarding the environment and themselves grows, in order to stay functioning in case of breaks, environmental changes, or other unforeseen events. By robustness we stick to the definition of the IEEE Standard Glossary of Software Engineering Terminology (IEEE, 1990):

> *The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.*

When considering societies of robots, each individual robot thereof has several different information sources to increase its own robustness (Fig. 26): It can foremost analyse its past behaviour and derive knowledge about e.g. which behaviour caused damage to the robot. If the robot has recognized a discrepancy between its expected behaviour and the actual result in its environment, it is even better, if it actively carries out experiments with the goal to find new behaviours that are suited better. Thereby, the robot in fact has to recognize dynamic entities like other robots as such so that it does not derive wrong hypotheses. In a society the robots even can propagate the knowledge within the robot group. And, as a last resort in case the robots cannot communicate, they can imitate each other so that the same behaviours do not have to be found out cumbersomely by each robot individually.

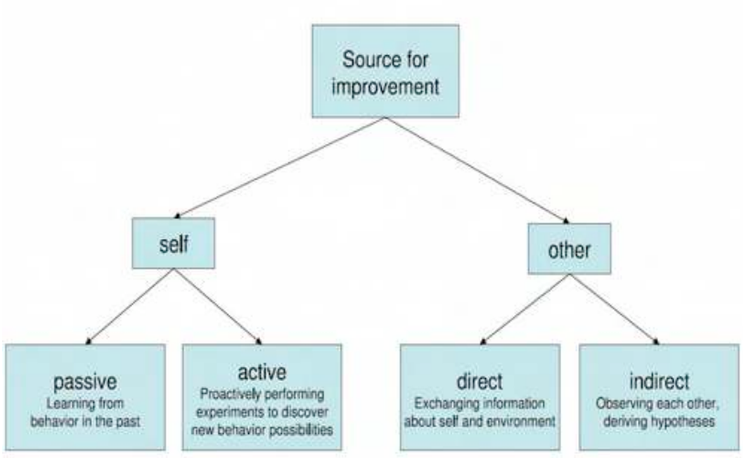

Fig. 26. A robot has several sources to improve its robustness available

In the RoboCup domain, however, the short duration of the game (2 x 15min) allows only for learning efforts that can quickly be conducted. In addition, the most environmental circumstances are guaranteed to not change during the game. Taken this into account, the changes the robot has to react on robustly are

- changes at the robot due to wear out and
- sudden breaks because of a collision with another robot.

In this section we will give an overview of our current research progress which addresses robustly learning low-level behaviours called "skills". The robust skill learning module (Richert & Kleinjohann, 2007) described in this section is meant to be placed between the behaviour module and the actual hardware components. It maps the vector resulting from the behaviour module to the actual servo possibilities. Thereby, the behaviour assemblies do not have to be adjusted, when something has changed at the hardware layer. Normally, the skill layer is turned off and the manually programmed behaviour assemblies are directly used to set the servos. The skill layer is turned on if the system detects discrepancies be-

tween the expectation and the reality regarding the environment's reaction to the system's action. The previously presented behaviour-based architecture (Fig. 21) thus has to be slightly modified, which results in the architecture in Fig. 27.

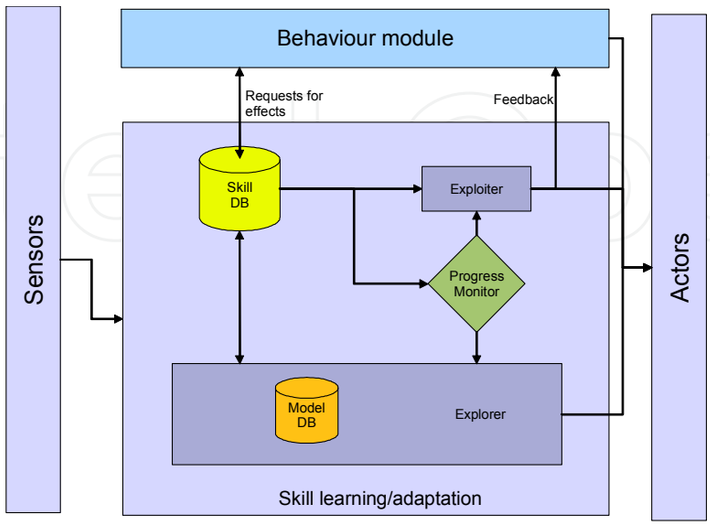

Fig. 27. Behaviour-based architecture with robust skill learning capabilities

The heart of the skill learning module is the skill database, which stores skills as tuple *(pc, a, e)* being the precondition the action and the expected effect in the environment. Normally, skills are requested by the upper layer in terms of effects that the behaviour module wants to happen, e.g. the decrease of the distance to a goal. If according skills are found in the database they are checked whether a skill's precondition holds true for the current situation. In the positive case the skill is directly executed and the corresponding effect is checked in the next cycle. In the other case, a change has been detected and the precondition of the skill is updated in a way that it is not executed in a similar situation the next time. If no skill can be found in the database that could achieve the requested effect the system starts tinkering with its actors until it finds an action that affects the requested effect. With the recorded perception data it has collected in this phase the following steps are processed:

1. Segmentation of the perception stream
2. For every segment
    a. Determine the model function most suitable and test it with the according model invariant
    b. Approximate the segment with the chosen model function
    c. Generate the skill capturing the segment's information

At first the perception stream is segmented into consecutive chunks of perception data according to the model function, which will be used to approximate the expected effects. For each segment a different function will be approximated resulting potentially in a new skill. An individual skill will be created to account for that. The choice of the model functions is subject to future research. As a first straightforward solution the system tries all model functions available and takes the one that is able to approximate the perception data with the least error. In the current implementation only polynomials are used. Experiments simulating both, gradual degradation and abrupt breaks, showed that though the modest learning speed, the skill learning module managed to robustly adapt to environmental changes.

There a robot had to repeatedly approach a goal while artificial breaks were introduced into its hardware.

In the first experiment the robot had to start with an empty skill base. After the 15th run a break was simulated by switching the first and second element in the actor. The robot thus had to cope with a skill base of which the most skills have become obsolete and new skills had to be learned. The result of how the robot copes with such a sudden break can be seen in Fig. 28. The steep increase of the running time is needed to delete skills that are not usable any more. At the 30th run it again has arrived at the old performance of run 15. As can be seen in the number of totally stored skills, not all skills had to be relearned.

The second experiment forced the robot to cope with gradual degradation of the omniwheel motor by decreasing its power in one dimension at every run. This was done according to the formula $(dx, dy) \leftarrow (dx, dy * \theta * (1 - run/30))$, with $\theta \in [0.1, 0.9]$ and run being the number of times the robot already drove to the goal. Thereby, the speed to drive left or right degraded to the fraction θ of the original capability at the last run. The result of how the robot copes with a continuous degradation over its whole lifetime for different degradation rates is shown in Fig. 29. It can be seen that the skill handles gracefully the degradation according to the grade of robot's damage: The bigger the damage the lower the performance. However, even at only 20% of its original power the robot still manages to cope with the impairment.

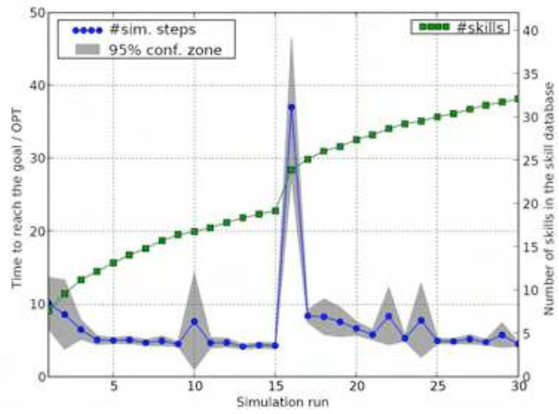

Fig. 28. Performance of robot coping with a sudden break after the 15th run: number of steps needed to reach the goal condition and the learned skills at each run. The grey background is the 95% confidence interval

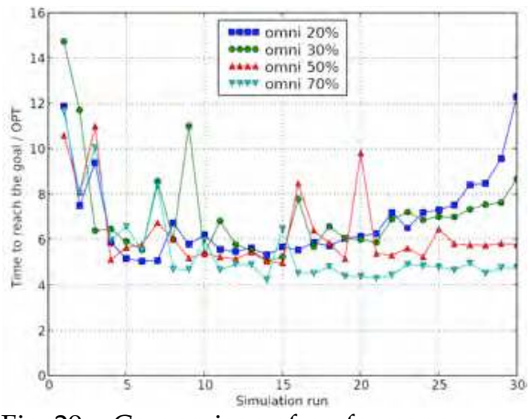

Fig. 29. Comparison of performance development for the robot coping with gradual degradation of one servo down to 20%, 30%, 50% and 70% of its original power (averaged over 400 trials)

The results are encouraging by showing how robust behaviour can be achieved at the skill level of mobile autonomous robots. They show that the robot is able to cope with sudden breaks in the hardware as well as gradual decrease of an individual component's performance. It is able to learn low-level skills guided by behaviour dependent predictions it recorded while the skill was learned. Even with simple model functions the system is able to learn skills without having a priori information about its servos. As an effect of the skill learning module not paying attention to the servo semantics and the way it dynamically creates, chooses, or deletes skills, it has all necessary properties of being robust in unforeseen environments. This also holds if the environment or the robot itself changes due to

wear out or harm caused by other robots. As long as there is a way to proceed and fulfil the goal the skill learning module will find skills to reach it.

7. Conclusion and Outlook

While realizing our soccer robot platform we achieved the two conflicting goals in the soccer domain needed to reach true autonomy: quick response rates for high reactivity and more complex but less frequent deliberation processes. This has led to robots that show autonomous behaviour while keeping attention to changes of the game situation demanding coordinated reaction of the whole team in a timely manner.
Components that are subject to quick changes in the environment as e.g. sensor and actor capabilities of the robots are arranged in a decentralized manner wherever appropriate and are located on specialized hardware. Those processes that do not need that fast update cycles like team communication, planning or processing at the higher levels are located at the Mini-ITX, allowing for faster development cycles but slower execution time. This architecture has evolved naturally out of the needs to combine a fault tolerant embedded system with fast development cycles for behaviour exploration.
In the near future we plan to endow the robots with the capabilities to self-calibrate the individual behaviours, meaning that the robots themselves adapt the behaviour parameters based on their perception of the environment. Furthermore, the TeamServer functionality will have to be decentralized to comply with the upcoming RoboCup soccer rules for the Middle Size League. A first step could be that one robot at a time hosts the TeamServer. If it looses connection to the other teammates another robot then could take over. Another big issue is the calibration automation for the various vision aspects. The usual practice to align the RoboCup rules year by year a little bit more to the FIFA rules has also a deep impact on the near-term Paderkicker development. The abandonment of the corner posts together with the colours of the goals, which are currently used by the Paderkickers to localize themselves, necessitates a partially rework of the vision and localization algorithms.

References

Arkin, R. C. (1998). Behaviour-Based Robotics, MIT Press

Beier, D., Billert, R., Brüderlin, B., Kleinjohann, B. & Stichling, D. (2003). Marker-less vision based tracking for mobile augmented reality. In *Proceedings of the Second International Symposium on Mixed and Augmented Reality (ISMAR 2003)*

Esau, N., Kleinjohann, B., Kleinjohann, L. & Stichling, D. (2003a). MEXI - machine with emotionally extended intelligence: A software architecture for behaviour based handling of emotions and drives. In *Proceedings of the 3rd International Conference on Hybrid and Intelligent Systems (HIS'03)*

Esau, N., Kleinjohann, B., Kleinjohann, L. & Stichling, D. (2003b). Visitrack - video based incremental tracking in real-time. In *6th IEEE International Symposium on Object-oriented Real-time Computing (ISORC '03)*

Friedman-Hill, E. (2003). Jess in Action: Java Rule-Based Systems (In Action series). Manning Publications, December 2002

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology.

Nilsson, N. (1998). Artificial Intelligence: a New Synthesis. Morgan Kaufmann

Prokop, R. J. & Reeves, A. P. (1992). A survey of moment-based techniques for unoccluded object representation and recognition. *Computer Vision, Graphics and Image Processing. Graphical Models and Image Processing*, Vol. 54, (Sept. 1992), pp. 438-460

Reimann, C. (2005). Kick-Real - a mobile mixed reality game. In *ACE2005, ACM SIGCHI International Conference on Advances in Computer Entertainment Techno*logy

Richert, W., Kleinjohann, B., Kleinjohann, L. (2005). Evolving agent societies through imitation controlled by artificial emotions. In M. Huang, X.-P. Zhang, and M. Huang, editors, *ICIC 2005*, number 3644 in LNCS, pages 1004–1013. Springer-Verlag Berlin

Richert, W., Kleinjohann, B., Koch, M., Bruder, A., Rose, S., Adelt, P. (2006). The Paderkicker Team: Autonomy in Realtime Environments. *Proceedings of the Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006)*

Richert, W. & Kleinjohann, B. (2007). Towards Robust Layered Learning. In *IEEE International Conference on Autonomic and Autonomous Systems (ICAS'07)*

Stichling, D. & Kleinjohann, B. (2002a). CV-SDF - a model for real-time computer vision applications. In *IEEE Workshop on Application of Computer Vision*

Stichling, D. & Kleinjohann, B. (2002b). Low latency color segmentation on embedded real-time systems. In Bernd Kleinjohann, K.H. Kim, Lisa Kleinjohann, and Achim Rettberg, editors, *Design and Analysis of Distributed Embedded Systems*. Kluwer Academic Publishers

Stichling, D. & Kleinjohann, B. (2003). Edge vectorization for embedded realtime systems using the CV-SDF model. In *Proceedings of the 16th International Conference on Vision Interfaces (VI 2003)*

Stichling, D. (2004). VisiTrack - Inkrementelles Kameratracking für mobile Echtzeitsysteme. PhD thesis, Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, 2004.

Robotic Soccer

Edited by Pedro Lima

ISBN 978-3-902613-21-9
Hard cover, 598 pages
Publisher I-Tech Education and Publishing
Published online 01, December, 2007
Published in print edition December, 2007

Many papers in the book concern advanced research on (multi-)robot subsystems, naturally motivated by the challenges posed by robot soccer, but certainly applicable to other domains: reasoning, multi-criteria decision-making, behavior and team coordination, cooperative perception, localization, mobility systems (namely omni-directional wheeled motion, as well as quadruped and biped locomotion, all strongly developed within RoboCup), and even a couple of papers on a topic apparently solved before Soccer Robotics - color segmentation - but for which several new algorithms were introduced since the mid-nineties by researchers on the field, to solve dynamic illumination and fast color segmentation problems, among others. This book is certainly a small sample of the research activity on Soccer Robotics going on around the globe as you read it, but it surely covers a good deal of what has been done in the field recently, and as such it works as a valuable source for researchers interested in the involved subjects, whether they are currently "soccer roboticists" or not.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Bernd Kleinjohann, Lisa Kleinjohann, Willi Richert and Claudius Stern (2007). Integrating Autonomous Behaviour and Team Coordination into an Embedded Architecture, Robotic Soccer, Pedro Lima (Ed.), ISBN: 978-3-902613-21-9, InTech, Available from: http://www.intechopen.com/books/robotic_soccer/integrating_autonomous_behaviour_and_team_coordination_into_an_embedded_architecture

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen