The Soul of A New Machine:
The Soccer Robot Team of the FU Berlin

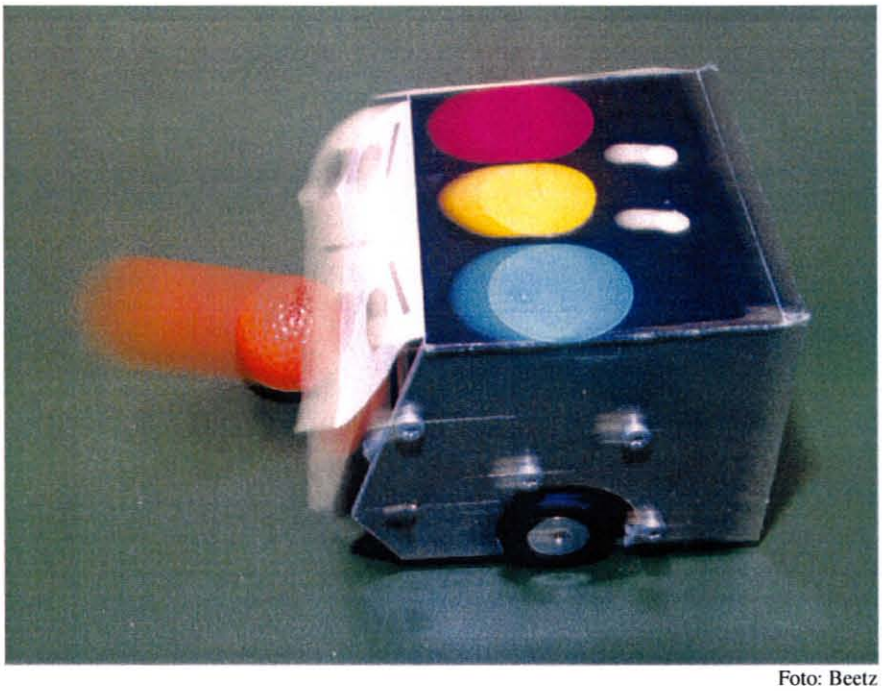

Foto: Beetz

Peter Ackers, Sven Behnke, Bernhard Frötschl, Wolf Lindstrot,
Manuel de Melo, Raúl Rojas, Andreas Schebesch, Mark Simon,
Martin Sprengel, Oliver Tenchio

Technical Report B-12/99
July 1999

Freie Universität Berlin
Department of Mathematics and Computer Science
Takustr. 9, 14195 Berlin, Germany

The Soul of A New Machine: The Soccer Robot Team of the FU Berlin

Peter Ackers, Sven Behnke, Bernhard Frötschl, Wolf Lindstrot,
Manuel de Melo, Raúl Rojas, Andreas Schebesch, Mark Simon,
Martin Sprengel, Oliver Tenchio

Technical Report B-12/99
July 1999

Freie Universität Berlin
Department of Mathematics and Computer Science
Takustr. 9, 14195 Berlin, Germany

Abstract. This paper describes the hardware and software of the robotic soccer team built at the Freie Universität Berlin which took part in the 1999 RoboCup Championship in Stockholm, Sweden. Our team, the *FU Fighters*, consists of five robots of less than 18 cm horizontal cross-section. Four of the robots have the same mechanical design, while the goalie is slightly different. All the hardware was designed and assembled at the FU Berlin. The paper describes the hierarchical control architecture used to generate the behavior of individual agents and the whole team. Our reactive approach is based on the dual dynamics framework proposed by Jäger, but extended with a third module of sensor readings. Fast changing sensors are aggregated in time to form slowly changing percepts in a temporal resolution hierarchy. We describe the main blocks of the software and their interactions.

1. Introduction

Robotic soccer has been gaining popularity in the last years. Obviously, this has to do with the fact that the final objective of the game is well-defined, easy to understand, and the actions of the robots can be watched and judged on the spot. The impact of robotic soccer on the Artificial Intelligence (AI) community has been tremendous and this requires a more careful explanation.

The development of AI has been always led by some benchmark problems which have been regarded as being hard for machines, although easy for humans. Speech recognition, face recognition, understanding of linguistic context, are all

problems in which much research has been done and which are far from having been completely solved. Not long ago computer algebra was regarded also as a subject pertaining to AI. However, once the field became established, computers faster, and the algorithms better, computer algebra metamorphosed into applications and commercial software packages like Mathematica or Maple, moving out of the AI domain. AI research is confronted with a moving horizon: once a problem has been efficiently solved, it becomes uninteresting for the AI community and we proceed to deal with something different. For example, although work on computer chess continues to date, it is not the field many young researchers want to be, since a computer has already bet the world champion!

Robotic soccer is interesting for many different reasons. First of all, it has to deal with coordination of autonomous agents. Each robot is "alone" on the field and has to respond to a changing and almost unpredictable environment. The movement of the opposing team is difficult to compute in advance, so that we need control software capable of reacting to many different circumstances. The actions of the robots in a team, if coordinated, can lead to a higher level of play and ultimately to victory. Coordination of autonomous agents is a well studied problem regarding software agents, but very difficult in the context of robots acting in the real world.

The second interesting problem in robotic soccer is control of individual robots and the constraints imposed by the game. Each robot has to be able to find the correct orientation to stop or shoot the ball, has to find the best path to the ball in a field full of obstacles and has to adapt its "intentions" to the perceived objectives of the own or adversary players. The problem can best be solved using a learning approach, like for example *reinforcement learning*, in which the pertinent actions of a robot are not coded by hand, but are generated automatically by a system that learns from experience and rewards to map situations to actions without manual intervention.

Robotic soccer has also to do with computer vision. The ball has to be found using one or more video cameras and must be tracked continuously during the game. The movement of the other players has to be monitored also. Object tracking is done finding color marks on the robots but in the future this could be abolished, so that only the form of the robots and their motion can serve as a cue for the vision system.

A last interesting problem is how best to balance the computing power in each robot with the external computer power available for processing. Too much processing in the robot can require excessive energy and heavy batteries. No processing can overburden the central computer.

It is for all these reasons that robotic soccer has become a paradigmatic problem of AI. The application domain is simple and well-understood, a robotic solution is therefore feasible even when only small resources are available. At the same time though, robotic soccer points to other more challenging problems and is open ended in its possibilities. Legged robots, for example, require more complex control and power management strategies. Each new robotic championship closes therefore a development period and sets the stakes higher for the next meeting.

2. The RoboCup challenge

RoboCup consists, at the moment of this writing, of four tournaments: the *simulation league*, in which the players and playing field are simulated in a computer, the *small size league* (F180) of wheeled robots with less than 18 cm cross-section, the *middle size league* (F2000) with wheeled robots of less than 50 cm cross-section and the *league of legged robots*, which during RoboCup 99 will mainly consist of off-the-shelf Sony mechanical toy dogs. Our team qualified for the small size league, in which 19 teams, divided in four groups, participate. The tournament is played like a FIFA World Cup, with group games at the beginning that filter out some teams and sudden-death in the following rounds.

2.1 The small league

The playing field for robots in the small league is a green table, the size of those used in ping pong matches (1.525 m by 2.74 m), surrounded by a 10 cm high white wall. Each goal has a width of 50 cm and it is 18 cm deep. The area behind the goal line can be used by the robots. One of the walls of the goal area is colored dark blue, the other yellow. The playing ball is an orange golf ball. Fig. 1 shows a view of the playing field from above. The rules of the game are similar to normal soccer (regarding kick-off, penalties, etc.) but there are some special rules regarding the protected zones around the two goal lines, in which the goalie cannot be attacked by the robots of the opposing team.

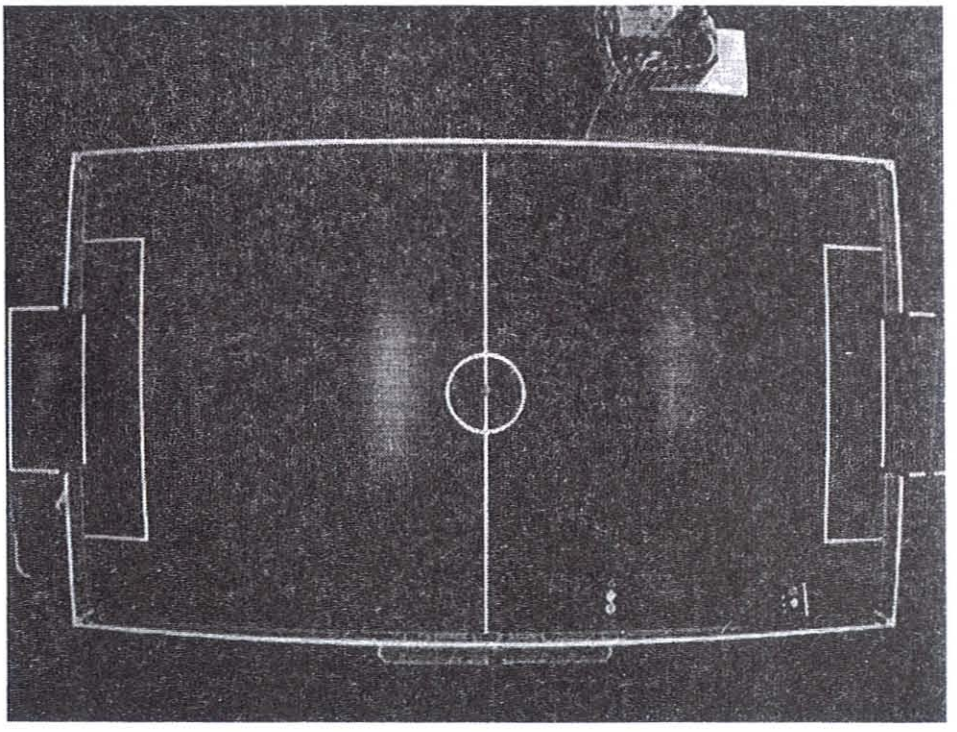

Figure 1: A snapshot of the playing field with the ball at the center

A RoboCup game is divided in two halves of 10 minutes each. Once the game starts, human intervention is not allowed, that is the on-board and off-board computers take full charge. The game can be only stopped by the referee when robots or the ball get stuck, in which case the designated team operator stops the robots and restarts the game when the referee gives the start signal.

2.2 Overview of the FU Fighters team

Our robots were designed in compliance with the new F180 league (the "small league") size regulations. Our team, the FU Fighters, consists of four field player robots with the same mechanical design, and a goalie with different mechanical characteristics, but the same on-board electronics. The robots have stable 4 mm aluminum frames that protect the sensitive inner parts. Each robot has two wheels activated by individual DC motors. A small rolling ball provides the third contact point needed to stabilize the chassis of the field players. The goalie has two extra passive wheels in addition to the active wheels.

Fig. 2 shows the main components of our system. The video camera provides a full view of the playing field from above and delivers 30 frames per second to the main computer. The central computer, an IBM-PC compatible system, processes the video frames, finds the ball and all robots and delivers commands to them using a radio unit attached to the computer. All our robots receive the same messages, but discard those not intended for them (according to the ID of the destination robot contained in the message). The on-board computer processes the packets received by each robot and activates three motors: the two wheel-motors and the motor for the shooting device. It is thus possible to turn a robot on the spot, to make it advance forward or backward, or to combine all these movements. When the robot is in the right position and ready to shoot, the shooting device (a plate) starts rotating.

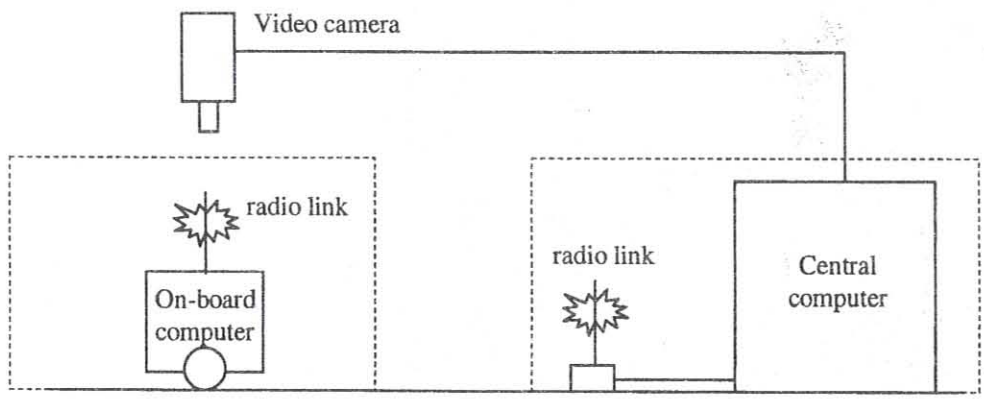

Figure 2: The hardware used by the FU Fighters

Fig. 3 shows three of our robots in a playing situation. The color marks on the top are used to identify the robots. In this image two robots marked with light blue, dark blue and red spots, are playing against a single robot marked with other colors.

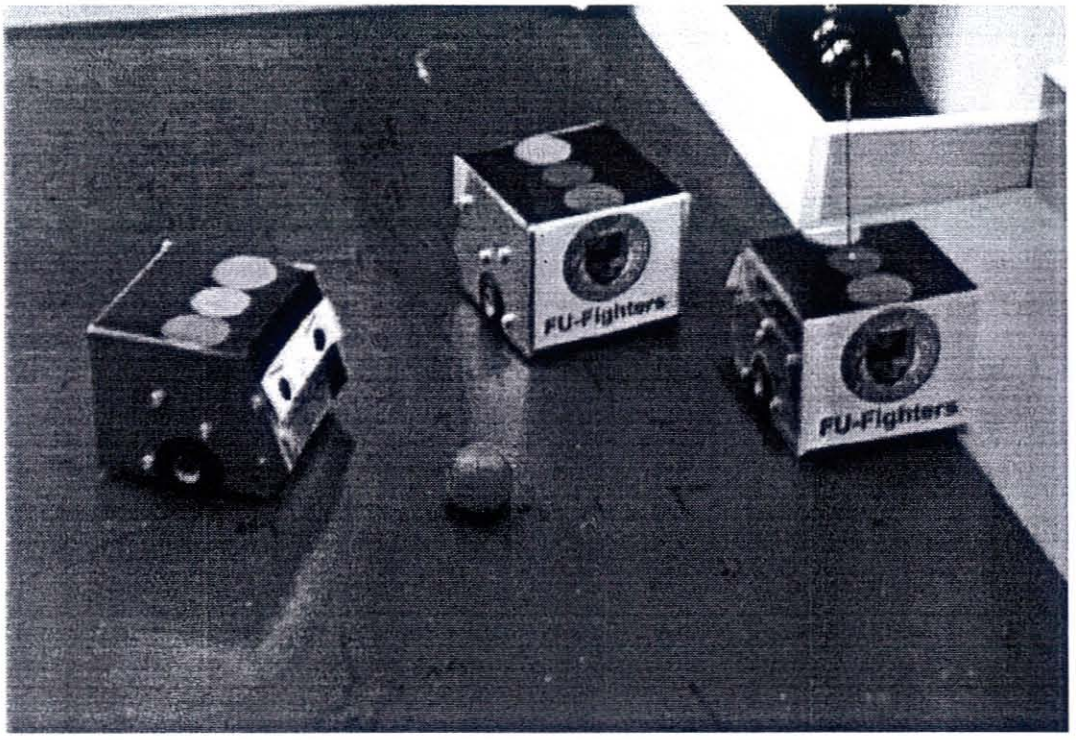

Figure 3: A playing situation. The robot to the left is about to shoot.

2. Mechanical design

2.1 Small-Size League Constraints

A team consists of at least five robots. This means that the rules allow each team to use less than five robots, but this is not the normal case. Robots can get stuck during the game and can be retired during a time off.

The maximum diameter of each robot body is restricted to 18 cm, but the total floor area of the robot must be smaller than 180 cm^2. For robots with local vision the height is restricted to 22.5 cm. Robots using a global camera (like ours) are restricted to a maximum height of 15 cm.

2.2 Chassis

The chassis of the robots is made of 4 mm aluminum plate. The shape of the sides of each robot as well as the bottom is shown in Fig. 4. The perforations and carvings provide structural support for other components of the robot.

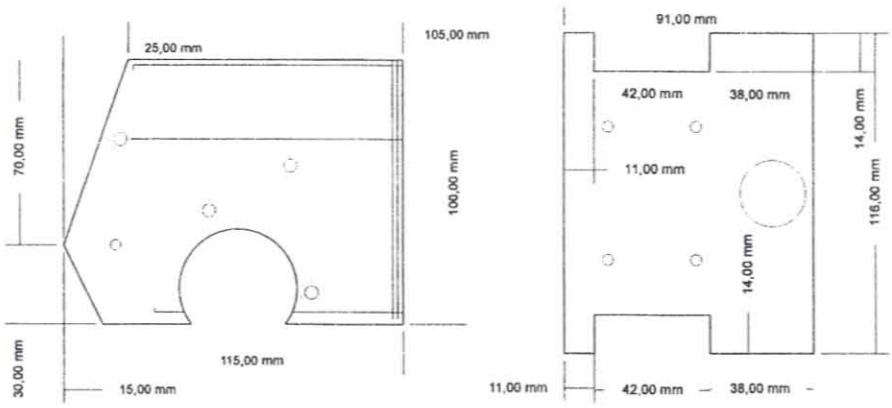

Figure 4: The sides and bottom of the aluminum frame

The robot uses soft wheels that provide good traction. The frame is kept together by bars laid from one side to the other.

2.3 The wheels and motors

Each wheel of the robot uses a different motor. This allows the robot to rotate in place or change direction while going forward or backward. Two DC-motors from Faulhaber provide a maximum speed of about 1 m/s. The motors have an integrated 19:1 gear and an impulse generator with 16 ticks per revolution. The maximum number of revolutions per second, without load, is 9700.

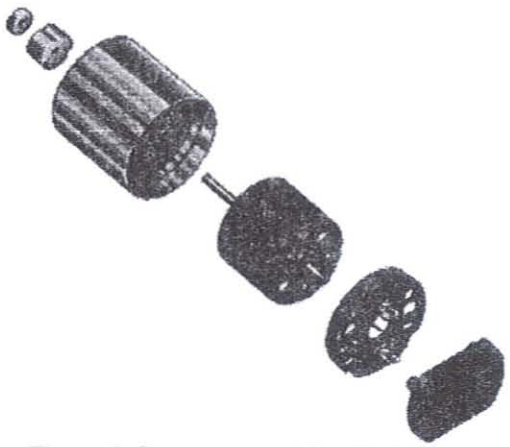

Figure 5: Components of the wheel motors

The motors are activated by sending series of pulses with different width. If the motor has to go faster, the width of the pulses is increased. If it has to slow down, the width is decreased. The motor is activated therefore in discrete steps, but since this is done 122 times per second, it appears as if the motor is being controlled using a slowly continuously varying input.

The microcontroller on the robot sets a target for the motor rotational speed, but since the actual speed varies with the charge of the batteries, we implemented a feed-back loop to check the number of motor revolutions. The 16 impulses per revolution sent by the motors are counted 122 times per second, the difference with the target value is computed, and the width of the control pulses is adjusted accordingly. We use a simple P-control to adapt the motor power.

2.4 The shooting plate

Every robot has in the front a shooting device, which consists of an aluminum plate with a central axis. The plate is moved using a third motor in the robot that makes the plate rotate as shown in Fig. 6. The idea is to store kinetic energy in the plate and release it to the ball when shooting.

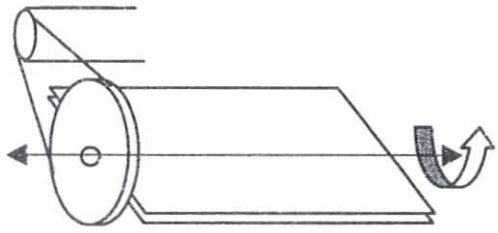

Figure 6: The shooting device

The shooting plate consists of a single aluminum piece machined as shown in Fig. 7. There is no need for fast control of the shooting plate, since the energy is released in the moment that the ball is hit.

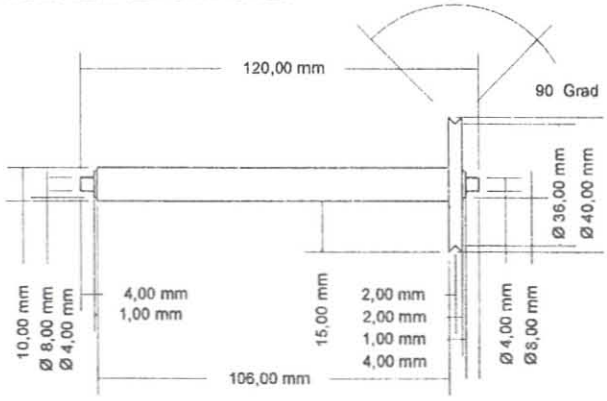

Figure 7: Detailed side view of the shooting plate

3. The electronics

The electronics for our robots is based on some off-the-shelf components and a custom motherboard that integrates all the necessary logic.

3.1 On-board computer

The heart of the on-board electronics is a controller card built and distributed by *Conrad Electronic* in Germany (called C-Control). The board is powered with 5V and consumes about 5 mA. Fig. 8 shows the components contained in the card, which has a ¼ Euroformat size.

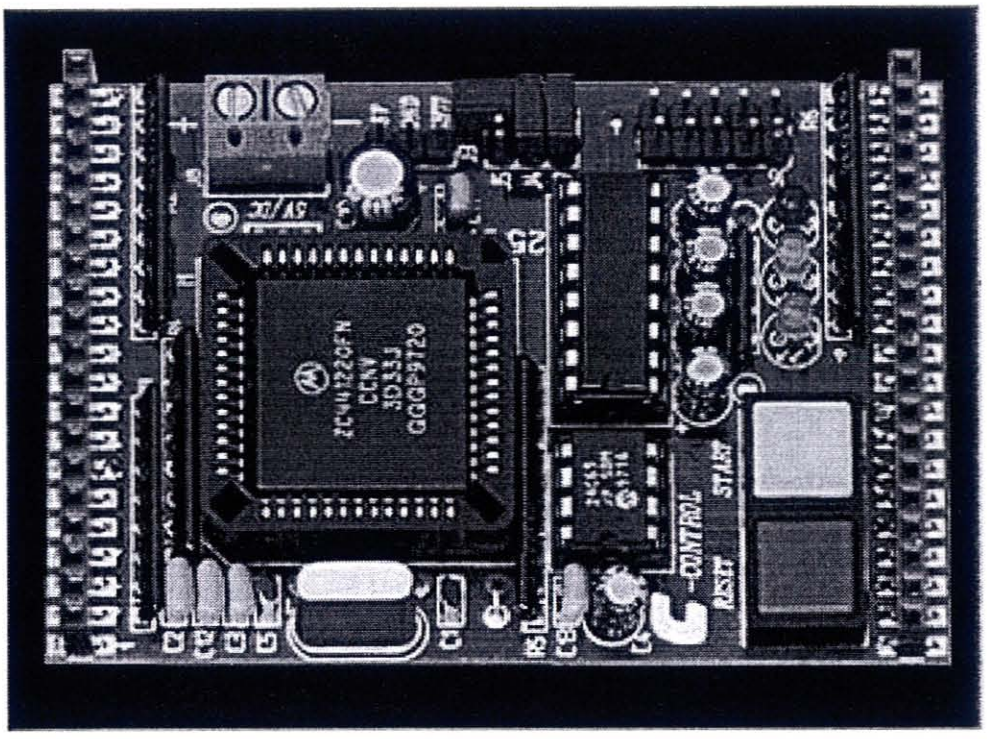

Figure 8: The C-Control unit

The largest chip on the card is a 4 MHz Motorola MC68HC05B16 controller with 256 Bytes free for object code and a built-in 6 Kb operating system. The EEPROM integrated in the control card is an 8K by 8 Bit serial unit. This allows the EEPROM to have a small footprint. The two buttons on the right allow to start (yellow) or reset (red) the program contained in the EEPROM. The three LEDs provide a reading of the status of the controller: the green LED shows the synchronization of the unit with an optional antenna for wireless programming, the yellow LED shows that the processor is ready, while the red LED blinks when the program is running or a program is being transfered to the unit. The connectors to the far right and far left provide access to the following I/O options:

· 16 digital I/O ports (5V/10mA),
· 8 analog inputs,
· 2 analog outputs (pulse-width modulation, PWM frequency of 1953 Hz),

The ten-pin connector on the top is an RS-232 interface for 1200 - 9600 baud.

The controller on the computer is responsible for interpreting the received commands from the central computer and activating the two motors on the unit, as well as the third motor, which drives the shooting plate. The two pulse-width modulated outputs are used to control the wheel motors. The impulse generators of the motors are connected to the interrupt inputs.

The microcontroller updates an internal counter periodically. In case that the counter fails to be updated, because the user program has crashed or after a sudden power drop, the whole unit resets itself. This provides a way of recovering from unexpected electrical problems, for example after a hard collision.

3.2. Custom board

A custom board was designed to provide the necessary power to the microcontroller and to allocate the extra components needed: a dual H-bridge motor driver L298, a beeper, and the radio transceiver SE-200. The robots are powered by 8+4 Ni-MH rechargeable mignon batteries

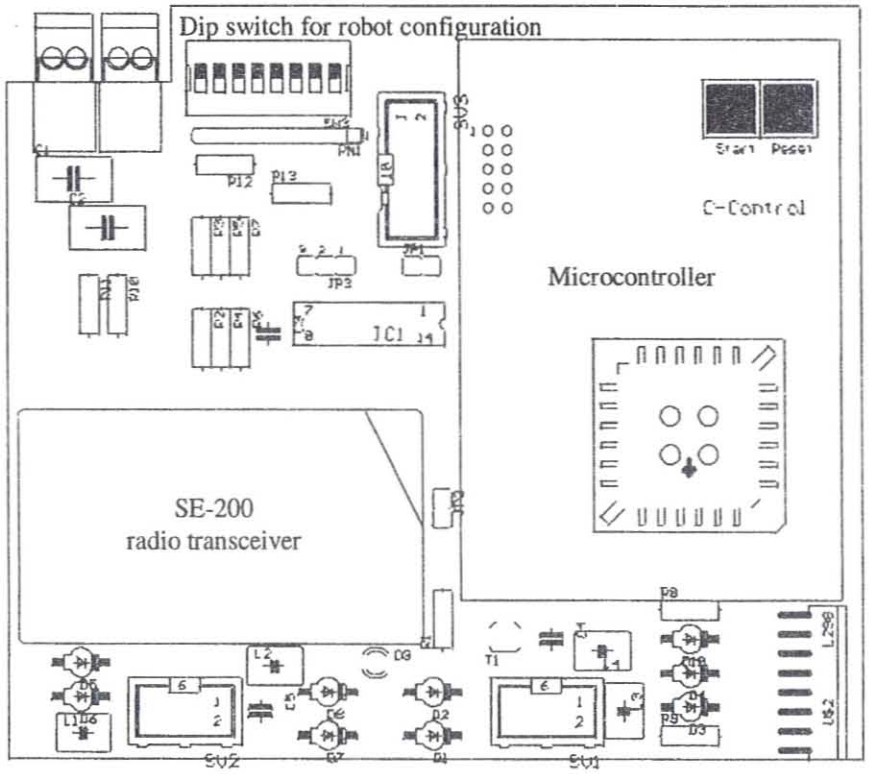

Figure 9: Layout of the custom board

The robots can be configured using eight DIP switches and four jumpers. We use the microcontroller to initialize the radio transceiver and to perform a self-check on start-up.

3.3 Radio link

The actions chosen by the control module of the software are transmitted to the robots via a wireless serial communication link with a speed of 9600 baud. We use radio transmitters operating on a single frequency that can be chosen between 433.0 MHz and 434.5 MHz in 100 KHz steps. The host sends commands in 8-byte packets that include address of the robot, control bits, motor speeds, and a checksum. A priority value can be used to direct more packets to the most active players. The microcontroller on the robots decodes the packets, checks their integrity, and sets the target values for control of the motor speeds. No attempt is made to correct transmission errors, since the packets are sent redundantly. To be independent from the state of the batteries, we implemented on the custom board a feed-back control loop of the motor speeds. The microcontroller counts the impulses from the motors 122 times per second, computes the differences to the target values and adjusts the pulse width ratio for the motor drivers accordingly. We use a simple P-control to adapt the motor power.

3.4 Block diagram of the software

Fig. 10 shows a diagram of the main modules of the control software in the central computer. The vision system is responsible for analyzing the 30 frames arriving each second from the video camera. The result of the analysis is stored in an array of variables used by the behavior module for further calculations (for example, the coordinates of the ball and of each robot in the playing field). The behavior module determines, using the information provided by the vision system, which actions are more adequate for the current situation. Each robot is a different thread in this module and acts "independently" from the others. Once an action has been determined it is passed to the radio communication module, which transmits a packet to the corresponding robot. The packet is interpreted by the on-board computer and is transformed in a movement of the robot.

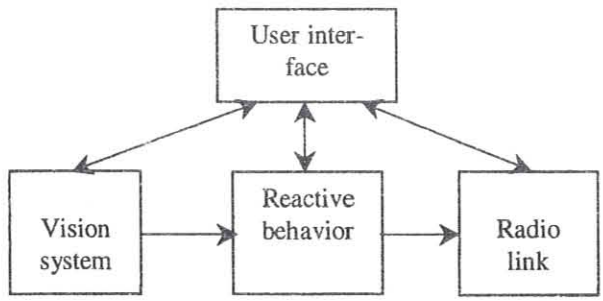

Figure 10: The main modules of the software and their relation

The user interface, finally, allows the programmers to get snapshots of the playing field and of most of the behavior, vision and radio link variables. This is a useful debugging tool in case the actual behavior of the robots differs from the intended one – a common problem in behavior based robotics.

4. Vision and user interface

4.1 The camera

The only physical sensor for our control software is a S-VHS camera placed 3 m above the playing field. Its output is an analog video stream in NTSC format. A PC running MS-Windows captures the images using a PCI frame grabber. We obtain RGB images with 640 x 480 pixels at a rate of 30 fps and interpret them to extract the relevant information.
Since the ball as well as the robots are color-coded, we designed our vision software to find and track multiple colored objects, i.e., the orange ball and the robots marked with colored balls. One of the teams is required to bear a yellow spot on the top, and the other a blue one. There is therefore a "yellow" and a "blue" team.

4.2 Ball and robot tracking

In order to track the objects we predict their positions in the next frame and then inspect a small window centered around the predicted position. We use an adaptive saturation threshold and intensity thresholds to separate the objects from the background. Only if an object is not found, the window size is increased and larger portions of the image are investigated. When we find the desired objects, we update our model of the world using the measured parameters, such as position, color, and size. The decision whether or not the object is present is made on the basis of a quality measure that takes into account the hue and size distances to the model and also geometrical plausibility.
Fig. 11 shows how the vision software works. There is an update module that continuously analyzes the frames arriving from a frame grabber. It locates and tracks the ball using a "ball model", which consist in some variables which describe color and expected position of the ball. The "team" modules do something similar for each robot in each team. There is an individual robot model for each player, that is, a team can consist of entirely different robots. Once located, a robot is tracked continuously during the game. Finally, a translation module transforms the positions of the ball and each robot, as well as the position of obstacles, into normalized sensor readings that can be used by the behavior module. This transformation takes into account the actual position of the field within the image, as well as the distortion caused by the optics of the camera.

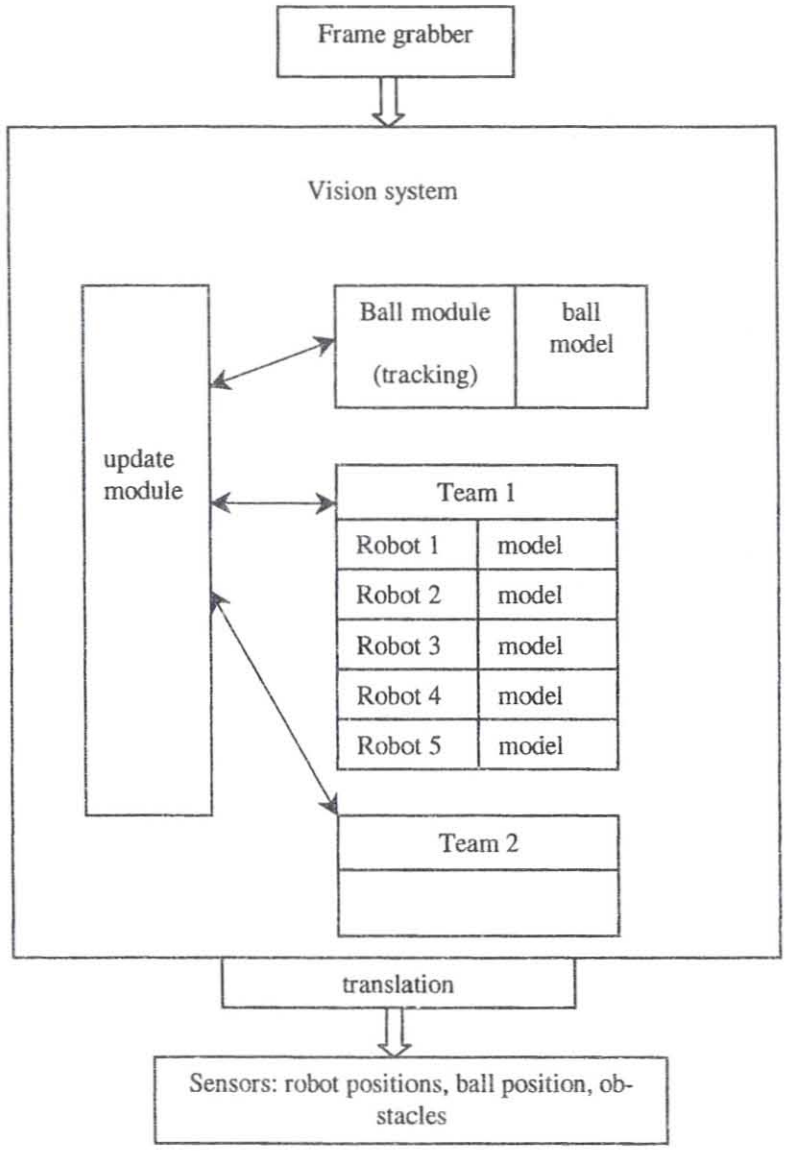

Figure 11: Structure of the vision module

4.3 Visualization of variables

In order to be able to program the system and adapt the behavior parameters, it was necessary to write a user interface which could allow the visual inspection of the system dynamics. Several variables can be monitored at once in a single diagram, so that the actual function of the system can be compared to the intended one. This is a kind of developing and debugging tool which we found very important for a project in which the actual behavior of the robots "emerges" and is not coded explicitly anywhere.

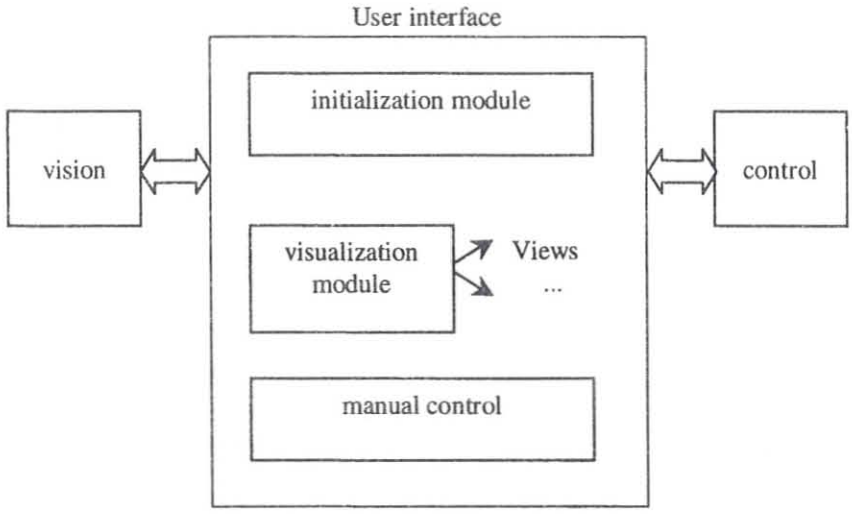

Figure 12: Structure of the user interface

Fig. 12 shows the three main components of the user interface. There is an initialization module that allows to set the parameters needed to start the game, like the position of the relevant marks on the field (goal lines, corners, etc) as well as the marks that will be tracked on the robots (robot models). The visualization module allows to produce multiple views of the internal state of the system, as shown in Fig. 13. The diagrams can show single variables like an oscilloscope and x-y variables as a trace within the field. Since manual intervention is sometimes needed (when the referee stops the game or in special situations), a third module takes care of providing some game controls for the operator.

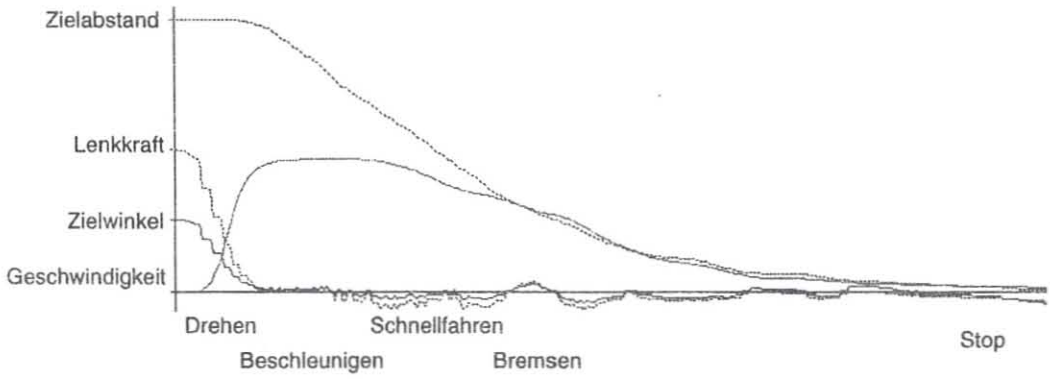

Figure 13: Example of the visualization of some variables

5. Control software

5.1 General approach

The "behavior based"' approach [Brooks 91,Christaller 99,PfeiferScheier 98] has proved useful for real time control of mobile robots. In this framework, so called *reactive agents* do not have a complete internal symbolic model of the world. They act responding to stimuli arriving asynchronously from the environment. Each reactive agent is simple and interacts with others also in a simple way, but complex patterns of behavior emerge from their interaction. This is what Brooks has called "intelligence without reason", i.e. intelligent behavior without a symbolic plan.
Brooks, specially, has put much emphasis in a hierarchical approach to the problem of intelligence in mobile agents. Taking some cues from the evolutionary process, he proposed his "subsumption architecture" (Fig. 14) in which sensory information activates different behaviors that compete to define the final signal to the actuators [Maes and Brooks 90]. In Brooks model, different behaviors can be active simultaneously and together, through excitation and inhibition, they define the final action of the system. The behavior "wander", for example, can make a robot move around in a room, but the subordinated behavior "avoid obstacles" lets it move without bumping into other objects.

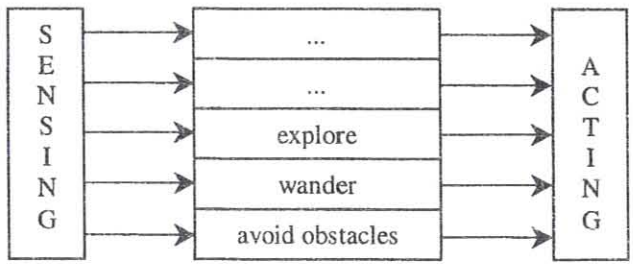

Figure 14: Brook's subsumption architecture

In 1992, the programming language PDL was developed by Steels and Vertommen for the stimulus driven control of autonomous agents [Steels 92,Steels 94]. This language has been used by several groups working in behavior oriented robotics [Schlottmann et al. 97]. It allows the description of parallel processes that react to sensor readings by influencing actuators. Many primitive behaviors, like taxis, are easily formulated in such a framework. On the other hand, it is difficult and computationally expensive to implement more complex behaviors in PDL, specially those that require persistent percepts about the state of the environment, i.e. the handling of different contexts. Consider for example a situation in which we want to position our defensive players preferentially on that region of the field

where the offensive players of the other team mostly attack. It is not feasible to take such a decision based only on a snapshot of sensor readings. The positioning of the defense has to be determined only from time to time, e.g. every minute, on the basis of the average positions of the attacking robots during the last time period.

The *Dual Dynamics* control architecture proposed by Herbert Jäger [Jäger 96, Jäger and Christaller 97], describes reactive behaviors using a hierarchy of control processes. Each layer of the system is partitioned into two modules: the activation dynamics that determines at every time step whether or not a behavior tries to influence actuators, and the target dynamics, that describes strength and direction of that influence. The activation dynamics corresponds to different contexts, leading to different targets. The different levels of the hierarchy correspond to different time scales. Behavior modi at higher levels configure the lower level control loops via activation factors which determine the primitive behavior modus. This can produce qualitatively different reactions if the agent finds the same stimulus again, but has changed its modus due to stimuli received in the meantime.

5.2 The architecture of the system

Our control architecture is based on the *Dual Dynamics* scheme developed by H. Jäger [Jäger 96, Jäger and Christaller 97]. The robots are controlled in closed loops that use different time scales and that correspond to behaviors which sit on different levels of the hierarchy.

We extended the dual dynamics approach by introducing a third dynamics, namely the perceptual dynamics. Sensory data is processed using different time resolutions. The position of the ball, for example, can be registered for every frame coming from the camera. The average position every four frames can be stored in a higher layer. And the average position every sixteen frames in still another layer. Sensory layers contain therefore information which is relevant at different time scales. The predicted ball position, for example, is relevant only if the robot has enough time to react. The behaviors of the system are activated by different sensory readings at different time scales.

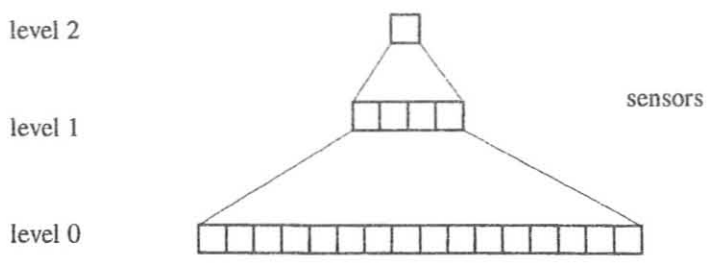

Figure 15: Aggregated sensory readings at three levels of the hierarchy

The complete architecture of our system is shown in Fig. 16. The three main structures visible in the diagram are: sensors, behaviors and actuators. Sensors determine different behaviors, at different levels of the hierarchy, and behaviors, in turn, determine the value of the actuators of the system. In our case there are just three physical actuators: the rotational speed of each of the two wheels and the state of the motor for the shooting plate (on or off). Some of the actuators at the higher levels of the hierarchy are "abstract actuators" in the sense that they do not correspond to physical actuators. For example, an actuator "position to move" at the second level just sets the objective function for the wheel actuators in the lowest level. These abstract actuators can modify some of the sensor variables through an internal feedback loop.

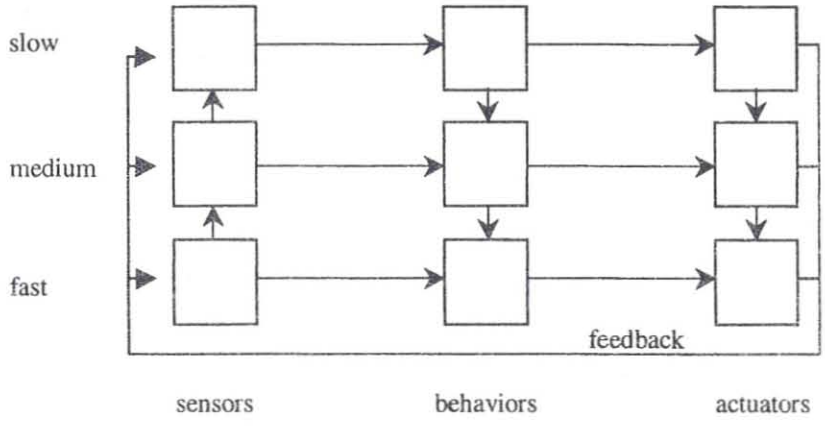

Figure 16: The extended dual dynamics architecture

The boxes shown on the left of Fig. 16 are divided into cells. There are several possible sensors. Since we use a global camera to scan the field, the sensors in our system are virtual sensors. There is for example one to determine the position of the ball. There is a sensor for the position and orientation of every robot on the field which is tracked during the game, etc.

The column of behaviors is shown in the middle of Fig. 16. There are several behaviors in each box. Each of them has an associated activation factor (a number between 0 and 1) that determines when the corresponding behavior is allowed to influence actuators.

The actuators are shown on the right hand side. Some of these actuators are connected to physical actuators that change the environment. The higher-level actuators influence lower levels of the hierarchy or generate sensory percepts in the next time step via the internal feedback loop.

Since we use temporal subsampling, we can afford to implement an increasing number of sensors, behaviors, and actuators in the higher layers without an explosion of computational costs. This leads to rich interactions with the environment.

Each physical sensor or actuator can only be connected to one level of the hierarchy. One can use the typical speed of the change of sensor readings to decide where to connect it. Similarly, the placement of actuators is determined by the time constant they need to be effective.

Behaviors are placed on the level that is low enough to ensure a timely response to stimuli, but that is high enough to provide the necessary aggregated perceptual information and that contains actuators which are abstract enough to produce the desired reactions.

Behaviors are constructed in a bottom up fashion in a way resembling Brook's philosophy for the subsumption architecture: First, the processes that should react quickly to fast changing stimuli are designed. Their critical parameters, e.g. a mode parameter or a target position, are determined. When the fast primitive behaviors work reliably with constant parameters, the next level can be added to the system. More complex behaviors can now be designed for this slower level that influence the environment either directly by moving slow actuators or indirectly by changing the critical parameters of the control loops in the lower level. After adding some layers, fairly complex behaviors can be obtained that make decisions using abstract sensors which are based on a long history and that use powerful actuators to influence the environment. In a soccer playing robot, basic skills, like movement to a position and ball handling, reside on lower levels, tactic behaviors are situated on intermediate layers, while the game strategy is determined on the topmost levels of the hierarchy.

5.3. Update of the dynamics

The state of the sensors, behaviors and actuators is updated using difference equations. Time advances in discrete steps Δt_0 at the lowest level in the control hierarchy. At the higher levels updates are done less frequently: the time step is a multiple of the time step at level 0. Useful choices for the subsampling factor are 2,4, 8, etc., but they can be adjusted as desired.

The variables in each layer of the hierarchy are updated using information from the lower or the upper levels. In the case of the sensors, the i-th sensor at level j in the hierarchy is updated at time t using its last value at time t-1, the state of the relevant physical sensors at time t, as well as the value of the corresponding sensors in the lower level. The update mechanism is shown in Fig. 17.

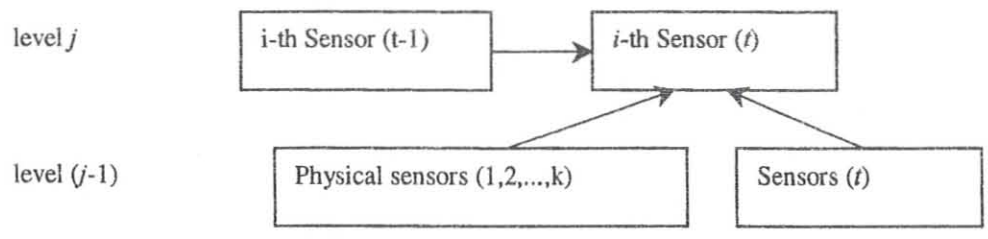

Figure 17: Update of sensors

The activation factors of behaviors depend on the sensor values at the same level of the hierarchy, their previous values and the activation of behaviors in the immediate upper level.

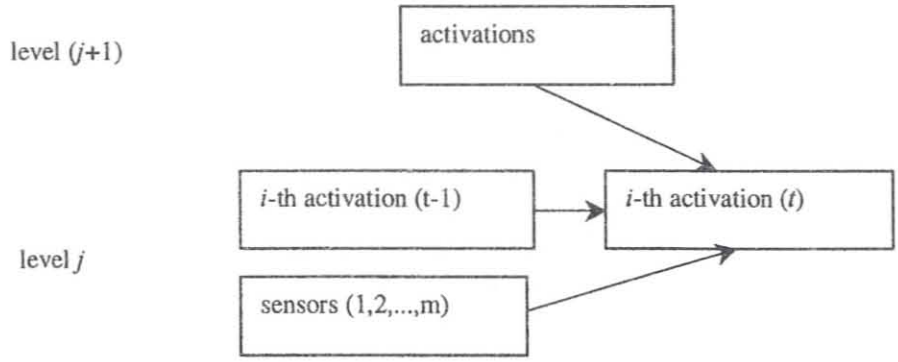

Figure 18: Update of behavior activations

A behavior situated at a higher level can "use" or activate lower level behaviors. For every "connection" from a behavior up in the hierarchy to a behavior at a lower level, there is a connection strength that determines the desired change in the activation factor of the behavior at the lower level. If the upper level behavior is not active, the total connection strength is zero. To determine the new activation of the behavior at the lower level, the changes arriving from all connections to a lower level behavior are accumulated and transformed into the new activation using an adequate function.

Each behavior j at each level specifies for each actuator k a target value T_{jk}. However, the more a behavior is active, the more it can influence the actuator values. The actual change to the actuators is the difference between the present state and the target value, multiplied by the connection strength between the behavior and each actuator and the activation factor of the behavior. Several behaviors can update the same actuators simultaneously, and in this case the total update is the sum of the individual updates.

5.4. Behaviors

The final set of behaviors in our control software has a relatively complex structure, as shown in Fig. 19. There is a team behavior that determines when a player is the one in charge of taking the initiative ("my_turn" variable). The highest level behaviors (in the middle) distinguish between a player who wants to shoot, one who is defending and one guarding his home position. The next level of behaviors decompose these three behaviors in their elementary components: shooting, moving forward, blocking the path to the own goal line, dribbling, going back to the home position. The behaviors at the lowest level are just moving (forward or backward) and steering. The actuators (third column) are two fast ones at the low-

est level for the two wheel motors and one at the immediate upper level for the shooting plate, which has a slower time constant than the wheels.
The goal keeper has a different set of behaviors and sensors as those shown in the figure, since it has to act in a different way. In general, the goal keeper reacts faster to the moving ball in order to block it as soon as it becomes clear that is coming towards the goal line.

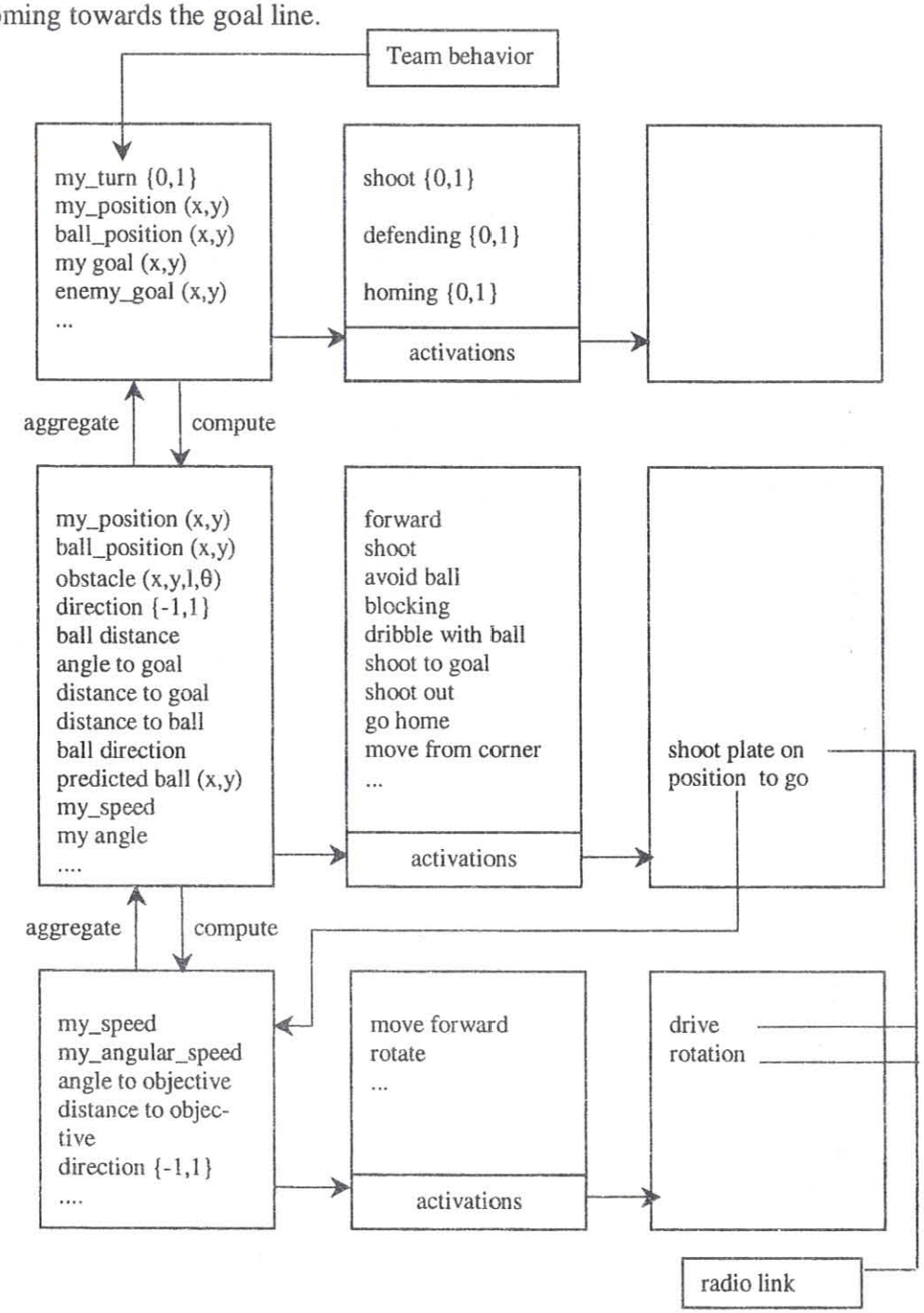

Figure 19: Sketch of the extended dynamic architecture for the FU Fighters

5.5. Standard situations

Finally, a set of special modules had to be written for such standard situations as kick-off, penalties, and reinitialization of the game. There are special rules for each one of these cases which have to be considered in the software.

5.6 Some examples

To realize a Braitenberg vehicle that drives towards a target, we need the direction and the distance to the target as input. The control loop for the two differential drive motors runs on the lowest level of the hierarchy. The two actuator values used determine the average speed of the motors and the speed differences between them. We determine the sign of the speed by looking at the target direction. If the target is in front of the robot, the speed is positive and the robot drives forward, if it is behind then the robot drives backward. Steering depends on the difference of the target direction and the robot's main axis. If this difference is zero, the robot can drive straight. If it is large, it turns on the spot. Similarly, the speed of driving depends on the distance to the target. If the target is far away, the robot can drive fast. When it comes close to the target it slows down and stops at the target position. Smooth transitions between the extreme behaviors are produced using sigmoidal functions. Fig. 13 shows an example of some variables and how they change over time.
This primitive taxis behavior can be used as a building block for the goal keeper. A simple goal keeper could be designed with two modes: block and catch, as shown in Fig 20. In the block mode it sets the target position to the intersection of the goal line and a line that starts behind the goal and goes through the ball. In the catch mode it sets the target position to the intersection of the predicted ball trajectory and the goal line. The goal keeper is always in the bock mode, except for situations where the ball moves fast towards the goal.

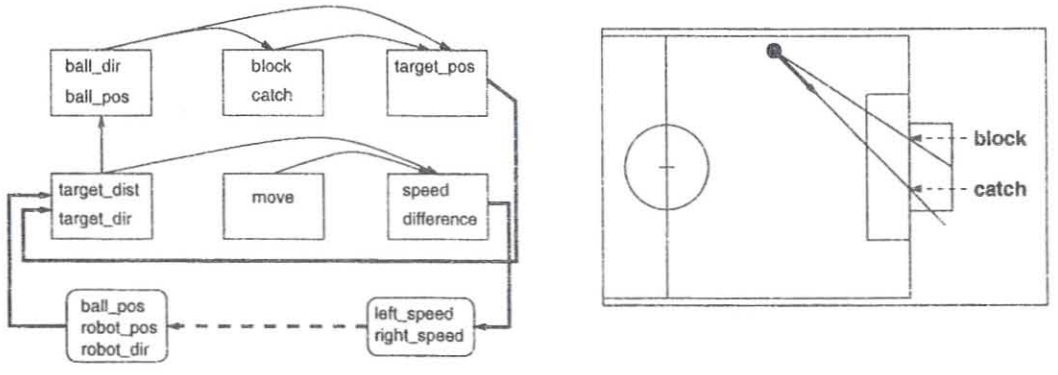

Figure 20: Reactive architecture for the goal keeper

The control hierarchy of the field player that wants to move the ball to a target, e.g. a teammate or the goal, could contain the alternating modes run and push. In the run mode the robot moves to a target point behind the ball with respect to the ball target. When it reaches this location, the push mode becomes active. Then the robot tries to drive through the ball towards the target and pushes it into the desired direction. When it looses the ball, the activation condition for pushing is not longer valid and the run mode becomes active again.

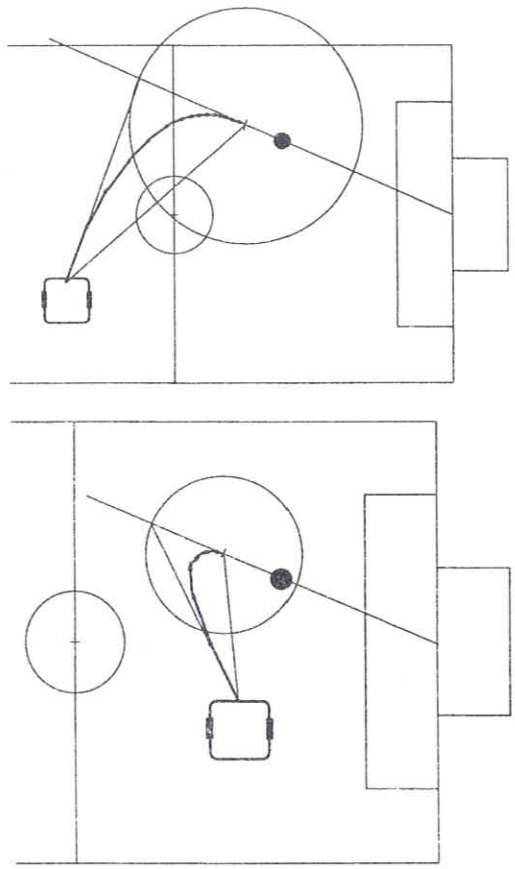

Figure 21: Trajectories generated in the run mode of the field player

Fig. 21 illustrates how the trajectory of the field player is generated in the run mode. A line is drawn through the ball target and the ball. The target point is found on this line at a fixed distance behind the ball. The distance from the robot to this target point is divided by two. The robot is heading always towards the intersection of the dividing circle and the line. This produces a trajectory that smoothly approaches the line. When the robot arrives at the target point, it is heading towards the ball target and can start rotating the shooting plate.

Each of our robots is controlled autonomously by the lower levels of the hierarchy using a local view of the world. For instance, we present the angle and the distance to the ball and the nearest obstacle to each agent. In the upper layers of the control system the focus changes. Now we regard the team as the individual. It

has a slow changing global view to the playground and coordinates the robots as its extremities to reach strategic goals. For example, it could position its defense on the side of the field where the offensive players of the opponent team mostly attack and place its offensive players where the defense of the other team is weak.

6. Conclusions and future work

We described in this paper the design of robust and fast robots with reliable radio communication and a high-speed global vision system. Although our first idea was to use off-the-shelf robots to build our team, the harsh competition requirements of the RoboCup tournament eventually led us to develop customized hardware. The software is divided in three main blocks: a) the vision module, which tracks colored objects on the field and sets the scene for the computation of the activation dynamics, b) the behavior module, and c) the communication module that transforms the desired actions in motor speeds and sends them to the robots using a radio link. To generate actions, we implemented a reactive control architecture with behaviors interacting on different time scales. The relevant control loops were designed in a bottom-up fashion. Lower level behaviors are configured by an increasing number of higher level behaviors that can use a longer history to determine their actions.

Building and programming our own robots was a long but rewarding journey. In the future, more of the software will include learning modules, so that the relevant parameters in the control hierarchy do not have to be set heuristically or using trial and error. Simple actions, like shooting or selecting the best path of approach, can be best solved using neural networks and reinforcement learning. The second generation of FU Fighters robots will feature these advances, as well as improved mechanical and electronic components.

Acknowledgements

We thank the following companies, institutions and individuals for the help they provided to our project: Conrad Electronic and Faulhaber Motoren donated important mechanical and electronic components. Siemens Elektrocom and Lufthansa Systems provided financial help that made possible that the students in our team could travel to RoboCup 1999 in Stockholm. The Freie Universität Berlin financed most of the project. The Chancellor of our University provided travel funds for the team. Georg Heyne from Fritz-Haber Institute provided advice and his laboratory built some parts for the robots.

The Team

Upper row: Wolf Lindstrot, Prof. Dr. Raúl Rojas, Manuel de Melo, Dipl.-Inf. Sven Behnke, Oliver Tenchio.
Lower row: Martin Sprengel, Dipl.-Inf. Bernhard Frötschl, Mark Simon, Peter Ackers, Andreas Schebesch.

References

[1] Brooks, R.A.: "Intelligence without reason". A.I. Memo 1293, MIT AI Lab (1991)

[2] Brooks, R.: "The Behavior Language; User's Guide", AI Memo 1227, MIT AI Lab, (1990).

[3] Christaller, T.: "Cognitive Robotics: A New Approach to Artificial Intelligence". In: *Artificial Life and Robotics*, Springer-Verlag, 3/1999

[4] Jäger, H.: "The Dual Dynamics Design Scheme for Behavior-based Robots: A Tutorial". Arbeitspapiere der GMD 966 (1996)

[5] Jäger, H., Christaller, T.: "Dual Dynamics: Designing Behavior Systems for Autonomous Robots". In: Fujimura, S., Sugisaka, M. (eds:) *Proceedings International Symposium on Artificial Life and Robotics* (AROB '97), Beppu, Japan, (1997) 76—79

[6] Maes, P. and Brooks, R.: "Learning to Coordinate Behaviors", AAAI, Boston, MA (1990).

[7] Schlottmann, E., Spenneberg, D., Pauer, M., Christaller, T., Dautenhahn, K.: "A Modular Design Approach Towards Behaviour Oriented Robotics". GMD 1088 (1997)

[8] Steels, L.: *The PDL reference manual.* VUB AI Lab memo 92-5, Brussels (1992).