In M. Hannebauer, J. Wendler, and E. Pagello (editors):
Balancing Reactivity and Social Deliberation in Multi-Agent Systems
pp. 125-136, LNCS 2103, Springer, 2001.

A hierarchy of reactive behaviors handles complexity

Sven Behnke and Raúl Rojas

Free University of Berlin, Institute of Computer Science
Takustr. 9, 14195 Berlin, Germany
{behnke|rojas}@inf.fu-berlin.de
`http://www.fu-fighters.de`

Abstract. This paper discusses the hierarchical control architecture used to generate the behavior of individual agents and a team of robots for the RoboCup Small Size competition.
Our reactive approach is based on control layers organized in a temporal hierarchy. Fast and simple behaviors reside on the bottom of the hierarchy, while an increasing number of slower and more complex behaviors are implemented in the higher levels. In our architecture deliberation is not implemented explicitly, but to an external viewer it seems to be present.
Each layer is composed of three modules. First, the sensor module, where the perceptual dynamics aggregates the readings of fast changing sensors in time to form complex, slow changing percepts. Next, the activation module computes the activation dynamics that determines whether or not a behavior is allowed to influence actuators, and finally the actuator module, where the active behaviors influence the actuators to match a target dynamics.
We illustrate our approach by describing the bottom-up design of behaviors for the RoboCup domain.

1 Introduction

The "behavior based" approach has proven useful for real time control of mobile robots. Here, the actions of an agent are derived directly from sensory input without requiring an explicit symbolic model of the world [4, 5, 9]. In 1992, the programming language PDL was developed by Steels and Vertommen as a tool to implement stimulus driven control of autonomous agents [11, 12]. PDL has been used by several groups working in behavior oriented robotics [10]. It allows the description of parallel processes that react to sensor readings by influencing the actuators. Many basic behaviors, like taxis, are easily formulated in such a framework. On the other hand, it is difficult and expensive to implement more complex behaviors in PDL, mostly those that need persistent percepts about the state of the environment. Consider, for example, a situation in a RoboCup [2] soccer game in which we want to position our defensive players preferentially on the side of the field where the offensive players of the other team mostly concentrate. It is not useful to take this decision at a rate of 30Hz based on a snapshot of the most recent sensor readings. The position of the defense has to be determined only from time to time, e.g. every second, on the basis of the average positions of the attacking robots during the immediate past.

The Dual Dynamics control architecture, developed by Herbert Jäger [7, 8], arranges reactive behaviors in a two-level hierarchy of control processes. Each elementary behavior in the lower level is divided into two modules: the activation dynamics which at every time step determines whether or not the behavior tries to influence actuators, and the target dynamics, that describes strength and direction of that influence. Complex behaviors in the higher level do not contain a target dynamics. They only compute an activation dynamics. When becoming active they configure the low-level control loops via activation factors that set the current mode of the primitive behaviors. This can produce qualitatively different reactions if the agent receives the same stimulus again, but has changed its mode due to stimuli received in the meantime.

We extended the Dual Dynamics approach by introducing a multi-level time hierarchy with fast behaviors on the bottom and slower behaviors towards the top of the control hierarchy. We don't restrict the target dynamics to the lowest layer, and add a third dynamics, the perceptual dynamics, to the system.

Dynamical systems have been used by others for behavior control. Gallagher and Beer, for instance, used evolved recurrent neural networks to control a visually guided walking agent [6]. Steinhage and Schöner proposed an approach for robot navigation formulated with differential equations [15]. Although planning and learning have been implemented in this framework [14], it is not clear how this non-hierarchical approach will scale to complex problems, since all behaviors interact pairwise with each other. Steinhage also proposed the use of nonlinear attractor dynamics for sensor fusion [13].

The remainder of the paper is organized as follows: In the next section we explain the hierarchical control architecture. Section 3 illustrates the design of behaviors using examples from the RoboCup domain.

2 Hierarchy of Reactive Behaviors

2.1 Architecture

Our control architecture is shown in Figure 1. It was inspired by the Dual Dynamics scheme developed by H. Jäger [7, 8]. In contrast to the two-level original proposal, the robots are controlled in closed loops that use many different time scales and that correspond to behaviors on different levels of the hierarchy. On the lowest level we have few simple and fast behaviors. While the speed of the behaviors decreases when going up the hierarchy, their number, as well as the number of sensors and actuators, increases. This allows to model complex systems.

We extended the Dual Dynamics concept further by introducing a third element, namely the perceptual dynamics, as shown on the left side of figure 1.

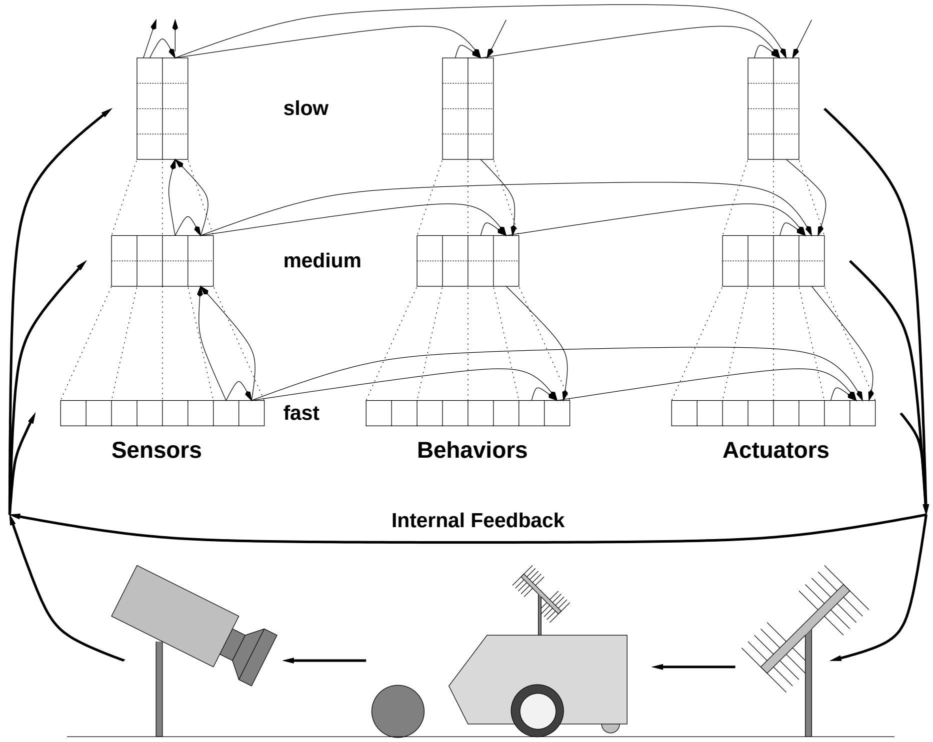


Fig. 1. Sketch of the control architecture.

Here, either slow changing physical sensors, such as the charging state indicators of the batteries, are plugged-in at the higher levels, or the readings of fast changing sensors, like the ball position in soccer, are aggregated by dynamic processes into slower and longer lasting percepts. The boxes shown in the figure are divided into cells. Each cell represents a sensor value that is constant for a time step. The rows correspond to different sensors and the columns show the time advancing from left to right.

A set of behaviors is shown in the middle of each level. Each row contains an activation factor between 0 and 1 that determines when the corresponding behavior is allowed to influence actuators.

The actuator values are shown on the right hand side. Some of these values are connected to physical actuators that modify the environment. The other actuators influence lower levels of the hierarchy when used as parameters of faster behaviors 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 cost. This leads to rich interactions with the environment, and therefore allows for complexity.

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 a sensor's readings to decide where to connect it. Similarly, the placement of actuators is determined by the time needed to make a change in the environment. Behaviors are placed on the level that is low enough to ensure a timely response to stimuli, but high enough to provide the necessary aggregated perceptual information, and that contains actuators which are abstract enough to produce the desired actions.

2.2 Computation of the Dynamics

The dynamic systems of the sensors, behaviors, and actuators can be specified and analyzed as a set of differential equations. Of course, the actual computations are done using difference equations. Here, the time runs in discrete steps of $\Delta t^0 = t_i^0 - t_{i-1}^0$ at the lowest level 0. At the higher levels the updates are done less frequently: $\Delta t^z = t_i^z - t_{i-1}^z = f \Delta t^{z-1}$, where useful choices of the subsampling factor f could be 2, 4, 8, ... In figure 1 $f = 2$ was used.

A layer z is updated in time step t_i^z as follows:

$\mathbf{s}_i^z$ – *Sensor values:*
The n_s^z sensor values $\mathbf{s}_i^z = (s_{i,0}^z, s_{i,1}^z, \ldots, s_{i,n_s^z-1}^z)$ depend on the readings of the n_r^z physical sensors $\mathbf{r}_i^z = (r_{i,0}^z, r_{i,1}^z, \ldots, r_{i,n_r^z-1}^z)$ that are connected to layer z, the previous sensor values $\mathbf{s}_{i-1}^z$, and the previous sensor values from the layer below $\mathbf{s}_{fi}^{z-1}$, $\mathbf{s}_{fi-1}^{z-1}$, $\mathbf{s}_{fi-2}^{z-1}$, ...
In order to avoid the storage of old values in the lower level, the sensor values can be updated from the layer below, e.g. as moving average.
By analyzing the sensor values from the last few time steps, one can also compute predictions for the next few steps that are needed for anticipative behavior. If the predictions in addition take the last few actuator values into account, they can be used to cancel a delay between an action command and the perceived results of that action.

α_i^z – *Activation factors:*
The n_α^z activations $\alpha_i^z = (\alpha_{i,0}^z, \alpha_{i,1}^z, \ldots, \alpha_{i,n_\alpha^z-1}^z)$ of the behaviors depend on the sensor values $\mathbf{s}_i^z$, the previous activations α_{i-1}^z, and on the activations of behaviors in the level above $\alpha_{i/f}^{z+1}$. A layer-$(z+1)$-behavior can utilize multiple layer-z-behaviors and each of them can be activated by many $(z+1)$-behaviors. For every behavior k on level $(z+1)$ that uses a behavior j from level z there is a term $\alpha_{i/f,k}^{z+1} T_{j,k}^z(\alpha_{i-1}^z, \mathbf{s}_i^z)$ that describes the desired change of the activation $\alpha_{i,j}^z$. Note that this term vanishes, if the upper level behavior is not active. If $\alpha_{i/f,k}^{z+1} > 0$, then the current sensor readings and the previous activations contribute to the value of the T-term. To determine the new α_i^z the desired changes from all T-terms are accumulated. A product term is used to deactivate a behavior, if no corresponding higher behavior is active.

$\mathbf{G}_i^z$ – *Target values:*
Each behavior j can specify for each actuator k within its layer z a target value $g_{i,j,k}^z = G_{j,k}^z(\mathbf{s}_i^z, \mathbf{a}_{i/f}^{z+1})$.

$\mathbf{a}_i^z$ – *Actuator values:*
The more active a behavior j is, the more it can influence the actuator values $\mathbf{a}_i^z = (a_{i,0}^z, a_{i,1}^z, \ldots, a_{i,n_a^z-1}^z)$. The desired change for the actuator value $a_{i,k}^z$ is: $u_{i,j,k}^z = \tau_{i,j,k}^z \alpha_{i,j}^z (g_{i,j,k}^z - a_{i-1,k}^z)$, where $\tau_{i,j,k}^z$ is a time constant. If several behaviors want to change the same actuator k, the desired updates are added: $a_{i,k}^z = a_{i-1,k}^z + u_{i,j_0,k}^z + u_{i,j_1,k}^z + u_{i,j_2,k}^z + \ldots$

2.3 Bottom-Up Design

Behaviors are constructed in a bottom-up fashion: First, the control loops 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 these fast primitive behaviors work reliably with constant parameters, the next level can be added to the system. For this higher level, more complex behaviors can now be designed which 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 the addition of several layers, fairly complex behaviors can be designed that make decisions using abstract sensors based on a long history, and use powerful actuators to influence the environment.

3 Application to the RoboCup Small Size Competition

In the RoboCup [2] Small Size competition, five on five robots play soccer using an orange golf ball. The area of the robots is restricted to $180cm^2$, and the playground has the size of a table tennis field.

In the Small Size league, a camera is mounted above the field and is connected to an external computer that finds the position of the players and the ball and executes the behavior control software. The next action command for each robot

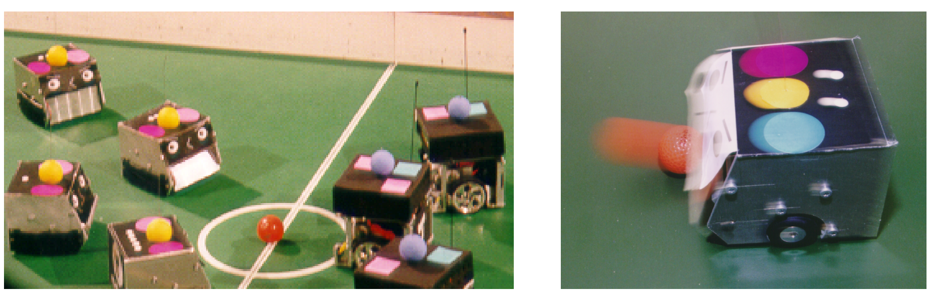


Fig. 2. Kick-off and a FU-Fighters robot kicking the ball (photo: Stefan Beetz).

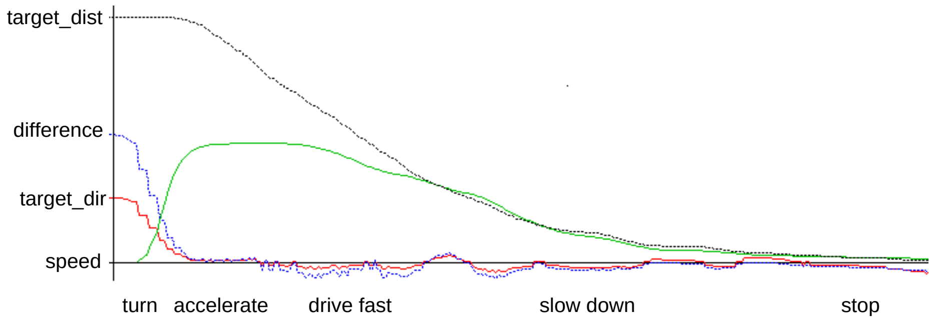


Fig. 3. Recording of two sensors (distance and direction of the target) and two actuators (average motor speed and difference between the two motors) during a simple taxis behavior. The robot first turns towards the target, then accelerates, drives fast, slows down, and finally stops at the target position.

is determined and sent via a wireless link to a microcontroller on the robot. The robots move themselves and the ball producing in this way visual feedback.

We designed the team FU-Fighters for the RoboCup'99 competition, held in Stockholm. We built robust and fast robots featuring a kicking device, as shown in figure 2. Local control is done using a Motorola HC05 microcontroller. The robots receive the desired motor speeds via a wireless serial link at a rate of up to 48Hz as commands. Each robot is marked with three colored dots that are tracked at 30Hz from an NTSC S-VHS video signal. Further details about the design of the FU-Fighters can be found at [1, 3].

The behavior control was implemented using a hierarchy of reactive behaviors. In our soccer playing robots, 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 at the topmost level of the hierarchy.

3.1 Taxis

To realize a Braitenberg vehicle that moves 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 desired speed of the motors and the speed differences between them. We select 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 backwards. 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 ahead. If the difference is large, it does not drive, but 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. Figure 3 shows an example where the robot first turns around until the desired angle has been reached, accelerates, moves with constant speed to a target and finally decelerates. Smooth transitions between the extreme behaviors are produced using sigmoidal functions.

In addition to the coordinates of the target position, we include some parameters of the taxis behavior as actuators on the second level. This allows to configure the driving characteristics of the taxis. The parameters influence the maximal speed driven, the degree of tolerance to lateral deviations from the direct way, the desired speed at the target position, the directional preference (forward/backward), and the use of the brakes.

3.2 Goalie

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 Figure 4. 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 block mode, except when the ball moves rapidly towards the goal.

Since our goalie is equipped with a kicking device, it can actively reflect the ball. This usually moves the ball to the opposite half of the field. Additional behaviors ensure that the longer side of the goalie is faced towards the ball and that the goalie does not leave the goal, although it has been designed to move mostly parallel to the goal line.

3.3 Obstacle Avoidance

Since there are many robots and a ball that move quickly on the field, obstacle avoidance is very important for successful play. We implemented a reactive collision avoidance approach on the lowest level of the control hierarchy. The robots only avoid the closest obstacle, if it is between their current position and the

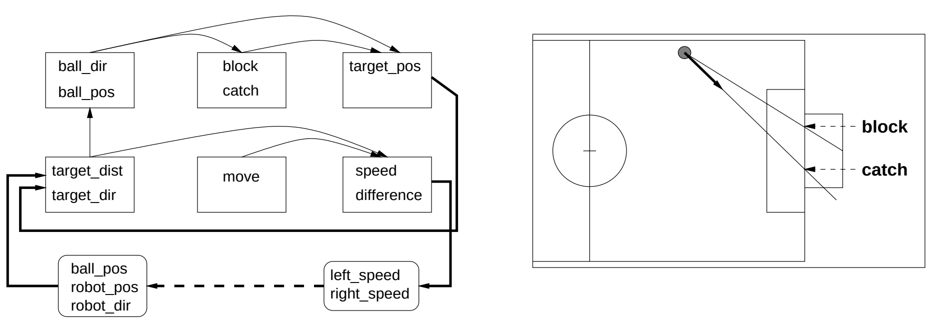


Fig. 4. Sketch of goal keeper behavior. Based on the position, speed, and the direction of the ball, the goalie decides to either block the ball or to catch it.

taxis target. If such a situation is detected and a collision is likely to occur in the next second, then the obstacle avoidance behavior activates itself. This inhibits the normal taxis. The robot now decides whether it should avoid the obstacle by going to the left or to the right. The position of the second closest obstacle, as well as the position of the closest wall point and the taxis target are taken into account for this decision. Since it is not useful to revise the avoidance direction frequently, it is made persistent for the next second. The robot drives on a circle around the obstacle until this is no longer blocking its way to the taxis target.

This fast reactive collision avoidance behavior should be complemented by path planning implemented on higher layers, such that a global view of the field is used and the activation of the collision avoidance behavior is minimized.

3.4 Field Player

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, pushing it into the desired direction. If the line of sight to the goal is free, we activate the kicking device before driving through the ball. This accelerates the ball such that it is hard to catch for the goalie. When the robot looses the ball, the activation condition for pushing is no longer valid and run mode becomes active again.

Figure 5 illustrates the trajectory of the field player generated in run mode. A line is drawn through the ball target, e.g. the middle of the goal line, and the ball. The target point for taxis is found on this line at a variable distance behind the ball. The position behind the ball for activating the push mode is chosen at a fixed offset from the ball. Half the distance of the robot to this position is added to the offset to determine the distance of the taxis target from the ball. The taxis behavior makes the robot always head towards the taxis target. As the robot comes closer, the taxis target moves to the push mode point. This

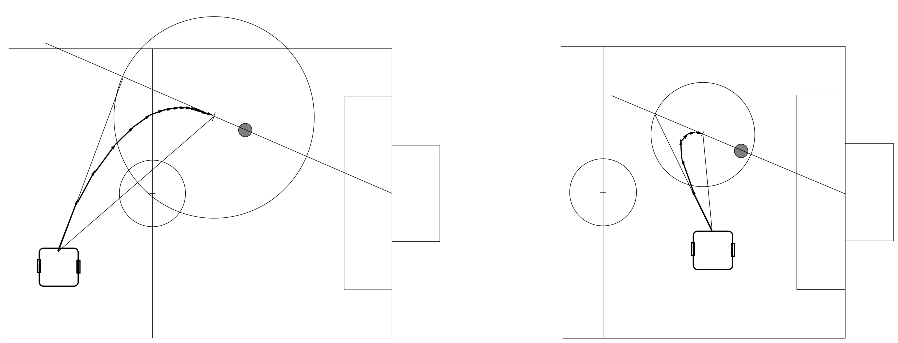


Fig. 5. Trajectories generated in the run mode of the field player. It smoothly approaches a point behind the ball that lies on the line from the goal through the ball.

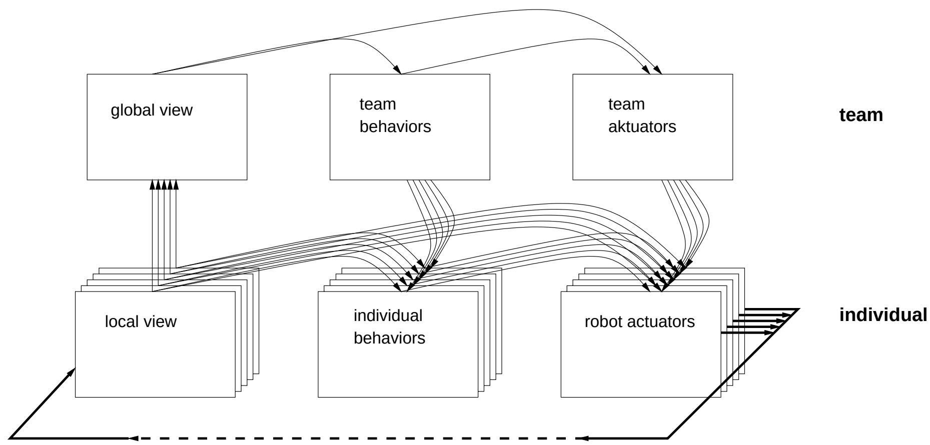


Fig. 6. Sketch of the relation between the team and the individual robots.

dynamic taxis target produces a trajectory that smoothly approaches the line. When the robot arrives at the push mode point, it is heading towards the ball target, ready to kick.

3.5 Team Play

Each of our robots is controlled autonomously by the lower levels of the hierarchy using a local view of the world, as indicated in Figure 6. We present, for instance, 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 of the playground and coordinates the robots as its extremities in order to reach strategic goals.

One simple strategy used to coordinate the four field players is that only one of them is allowed to go for the ball. The team evaluates the positions of the players relative to the ball and the goal and selects the one that gets the highest score. This player takes the initiative and tries to get behind the ball, dribbles and kicks it towards the goal. The robots not chosen in this selection do different things. If they are defenders, they cover the attacking robots of the other team. If they are offensive players, they position themself to be able to receive passes and then have a free path towards the goal. If the chosen robot is not able to get to the ball, the player with the second highest score is selected to take the initiative.

Although we did not implement explicit passes, some implicit passes have emerged during games. The most impressive ones are produced by a behavior that tries to free the ball from corners by quickly rotating the robot. If the direction of the rotation is chosen such that the ball is moved towards the goal, the ball frequently moved slowly across the field just in front of the goal. The offensive player waiting near the other corner of the goal area can now easily intercept the ball and kick it into the goal. These short distance kicks are extremely hard to catch for the goalie.

We also implemented a selective obstacle avoidance between the teammates. The player that goes for the ball does not avoid its teammates. They must avoid the active robot and move out of its path to the goal.

The field players are assigned different roles, like left/right wing, offender, defender. We implemented a dynamic assignment of roles, depending on the actual positions of the robots, relative to home positions of the roles. This allows to have more roles than robots. Only those roles most important in a situation are assigned to players. This feature is needed when a robot detects that it does not reach it's target position for a longer time. Then the robot signals the team that is defect and the team does not assign further roles to this player until the next stoppage in play.

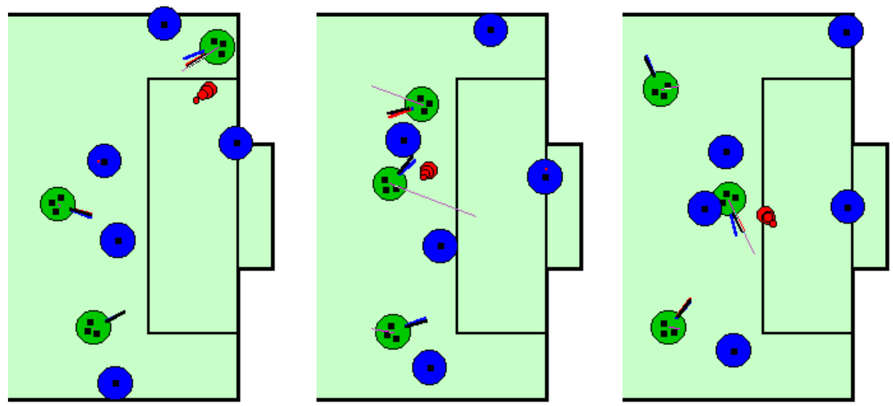
3.6 Complex Behaviors

We implemented some complex behaviors for the RoboCup competition. They include, for instance, dynamic homing, where the home positions of our defensive players are adjusted such that they block the offensive robots from the other team, and the home positions of our offensive players are adjusted, such that they have a free path to the goal. Another example is ball interception, where we predict the ball trajectory and the time it takes for the robot to reach successive points on this trajectory. We direct the robot towards the point where it can first reach such a point earlier than the ball. This results in an anticipative behavior. We also detect when a robot wants to move, but does not move for a longer time, e.g. because it is blocked by other robots or got stuck in a corner. Then we quickly rotate the robot for a short time, in order to free the player.

For presentations, we added an automated referee component to the system. It detects when the ball enters a goal and changes the mode of the game to kickoff. Then the robots move automatically to their kickoff positions. When the ball is detected in the middle of the field for some seconds, the game mode is changed back to normal play.

In our current system, the deliberation of common goals among the autonomous agents is not explicitly modeled. There is coordination among the robots. The highest level in the hierarchy, the team level, assigns each robot a role and keeps track of the robots. It is through their specific role that the robots collaborate.

One example can illustrate this. When the left wing player drives the ball through the field, the right wing player moves parallel to it. Once the left player reaches the corner of the field, it rotates in order to free the ball and produces a pass to the right. The pass will be taken by the right wing player or the central offensive player that is waiting in front of the goal. The result is a situation in which deliberation as bargaining is not present, but coordinated team play produces results that mimic deliberation. Figure 7 illustrates such a successful pass. It has been produced using the log-file from the final game in the Melbourne RoboCup competition.



(a) (b) (c)

Fig. 7. Successful pass: (a) the player in the upper corner frees the ball and sends it towards the middle; (b) the center player receives the ball; (c) the center has kicked towards the goal.

4 Summary

In the paper we described a hierarchical architecture for reactive control. This architecture contains interacting behaviors residing on different time scales. These control loops are 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.

We illustrated the design of behaviors using examples from the RoboCup domain. Our successful participation in the RoboCup'99 and '2000 F180 league competitions, where we finished second (next to Big Red, from Cornell University) and in the European RoboCup'2000, where we won, shows that the architecture can be applied to complex multi-agent control problems.

One remaining problem is the design complexity. The higher the design process advances in the hierarchy, the larger the number of sensors, behaviors, and actuators becomes. It is therefore increasingly difficult to determine the free parameters of the system. To design larger systems, automated design techniques, such as reinforcement learning, are needed.

References

1. P. Ackers, S. Behnke, B. Frötschl, W. Lindstrot, M. de Melo, R. Rojas, A. Schebesch, M. Simon, M. Sprengel, and O. Tenchio. The soul of a new machine. Technical Report B-12/99, Freie Universität Berlin, 1999.
2. M. Asada and H. Kitano, editors. *RoboCup-98: Robot Soccer World Cup II.* Lecture Note in Artificial Intelligence 1604. Springer, 1999.

3. S. Behnke, B. Frötschl, R. Rojas, P. Ackers, W. Lindstrot, M. de Melo, M. Preier, A. Schebesch, M. Simon, M. Sprengel, and O. Tenchio. Using hierarchical dynamical systems to control reactive bahaviors. In *Proceedings IJCAI'99 - International Joint Conference on Artificial Intelligence, The Third International Workshop on RoboCup – Stockholm*, pages 28–33, 1999.
4. R.A. Brooks. Intelligence without reason. A.I. Memo 1293, MIT Artificial Intelligence Lab, 1991.
5. T. Christaller. Cognitive robotics: A new approach to artificial intelligence. *Artificial Life and Robotics*, (3), 1999.
6. J.C. Gallagher and R.D. Beer. Evolution and analysis of dynamical neural networks for agents integrating vision, locomotion and short-term memory. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99) – Orlando*, pages 1273–1280, 1999.
7. H. Jäger. The dual dynamics design scheme for behavior-based robots: A tutorial. Arbeitspapier 966, GMD, 1996.
8. H. Jäger and T. Christaller. Dual dynamics: Designing behavior systems for autonomous robots. In S. Fujimura and M. Sugisaka, editors, *Proceedings International Symposium on Artificial Life and Robotics (AROB '97) – Beppu, Japan*, pages 76–79, 1997.
9. R. Pfeifer and C. Scheier. *Understanding Intelligence*. MIT press, Cambridge, 1998.
10. E. Schlottmann, D. Spenneberg, M. Pauer, T. Christaller, and K. Dautenhahn. A modular design approach towards behaviour oriented robotics. Arbeitspapier 1088, GMD, 1997.
11. L. Steels. The PDL reference manual. AI Lab Memo 92-5, VUB Brussels, 1992.
12. L. Steels. Building agents with autonomous behavior systems. In L. Steels and R.A. Brooks, editors, *The 'Artificial Life' route to 'Artificial Intelligence': Building situated embodied agents*. Lawrence Erlbaum Associates, New Haven, 1994.
13. A. Steinhage. Nonlinear attractor dynamics: A new approach to sensor fusion. In P.S. Schenker and G.T. McKee, editors, *Sensor Fusion and Decentralized Control in Robotic Systems II: Proceedings of SPIE*, volume 3839, pages 31–42. Spie-publishing, 1999.
14. A. Steinhage and T. Bergener. Learning by doing: A dynamic architecture for generating adaptive behavioral sequences. In *Proceedings of the Second ICSC Symposium on Neural Computation NC2000 – Berlin*, pages 813–820, 2000.
15. A. Steinhage and G. Schöner. The dynamic approach to autonomous robot navigation. In *Proceedings of the IEEE International Symposium on Industrial Electronics ISIE'97*, pages SS7–SS12, 1997.