Using hierarchical dynamical systems to control reactive behavior

Sven Behnke, Bernhard Frötschl, Raúl Rojas,
Peter Ackers, Wolf Lindstrot, Manuel de Melo, Mark Preier, Andreas Schebesch,
Mark Simon, Martin Sprengel, and Oliver Tenchio
Free University of Berlin
Institute of Computer Science
Takustr. 9, 14195 Berlin, Germany

Abstract

This paper describes the mechanical and electrical design, as well as the control strategy, of the *FU-Fighters* robots, a RoboCup F180 league team. It explains how we solved the computer vision and radio communication problems That arose in the course of the project.

The paper mainly discusses the hierarchical control architecture used to generate the behavior of individual agents and the team. Our reactive approach is based on the Dual Dynamics framework developed by H. Jäger, in which activation dynamics determines when a behavior is allowed to influence the actuators, and a target dynamics establishes how this is done. We extended the original framework by adding a third module, the perceptual dynamics. Here, the readings of fast changing sensors are aggregated temporarily to form complex, slow changing percepts.

We describe the bottom-up design of behaviors and illustrate our approach using examples from the RoboCup domain.

1 Introduction

The "behavior based" approach has proved 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 [Brooks, 1991; Christaller, 1999; Pfeifer and Scheier, 1998]. In 1992, the programming language PDL was developed by Steels and Vertommen As a tool to implement stimulus driven control of autonomous agents [Steels, 1992; 1994]. PDL has been used by several groups working in behavior oriented robotics [Schlottmann *et al.*, 1997]. 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 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 based 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 immediate past.

The Dual Dynamics control architecture, developed by Herbert Jäger [Jäger, 1996; Jäger and Christaller, 1997], arranges reactive behaviors in 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 different levels of the hierarchy correspond to different time scales. The high-level behaviors 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 of mode due to stimuli received in the meantime.

The remainder of the paper is organized as follows: The next section describes the mechanical and electrical design of our RoboCup F180 league robots. Then the vision and communication systems are presented. In Section 5 we explain the hierarchical control architecture that we use to generate behaviors for the game of soccer and illustrate it using examples from the RoboCup domain.

2 Mechanical and Electrical Design

Our robots were designed in compliance with the new F180 size RoboCup regulations. We built four identical field players and a goal keeper. All robots have stable aluminum frames that protect the sensitive inner parts.

They have a differential drive with two active wheels in the middle and are supported by one or two passive spheres that can rotate in any direction. Two Faulhaber DC-motors allow for 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.

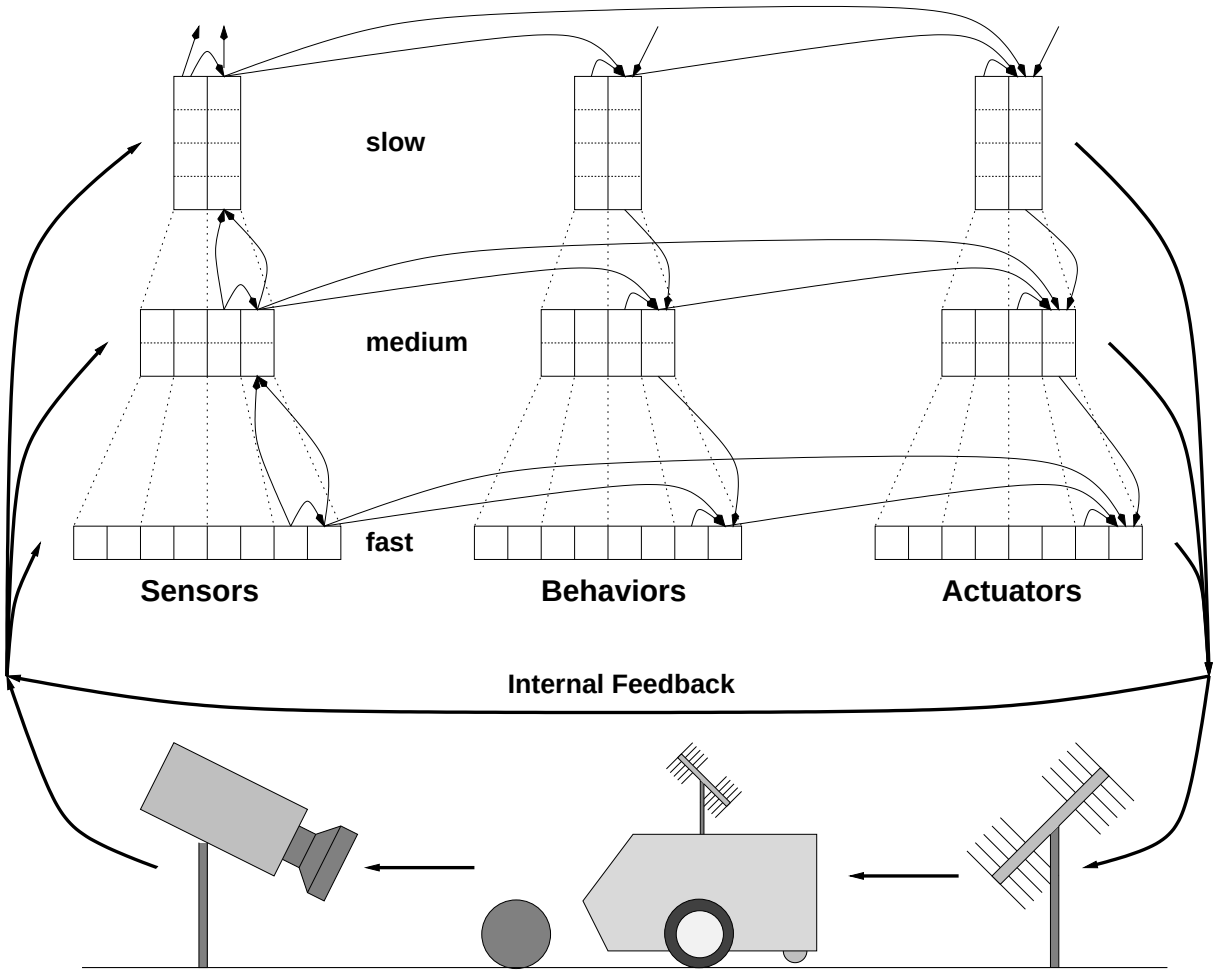

Figure 1: Sketch of the control architecture.

We use C-Control units from Conrad electronics for local processing. They include a Motorola microcontroller HC05 running at 4 MHz with 8 KB EEPROM for program storage, two pulse-length modulated outputs for motor control, a RS-232 serial interface, a free running counter with timer functions, analog inputs, and digital I/O. The units are attached to a custom board containing a stabilized power supply, a dual-H-bridge motor driver L298, a beeper, and a radio transceiver SE200. The robots are powered by 8 + 4 Ni-MH rechargeable mignon batteries.

3 Video

The only physical sensor for our control software is an S-VHS camera that captures the field from above. The camera produces an analog video stream in NTSC format. Using a PCI-framegrabber, we feed images to a PC running MS-Windows. We capture RGB-images of size 640 × 480 at a rate of 30 fps and interpret them to extract the relevant information about the playing field. Since the ball, as well as the robots, are color-coded, we designed our vision software to find and track several colored objects. These objects are the orange ball and all the robots that have been marked with colored dots, in addition to the yellow or blue team ball.

To track the objects we predict their positions in the next frame and then inspect the video image first at a small window centered around the predicted position. We use an adaptive saturation threshold and intensity thresholds to separate the objects from the background. The window size is increased and larger portions of the image are investigated only if an object is not found.

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 geometrical plausibility. When we find the desired objects, we adapt our model of the world using the measured parameters, such as position, color, and size.

4 Communication

The actions selected by the control module 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, control bits, motor speeds, and a checksum. A priority value can be used to transmit more packets to the most active players.

The microcontroller on the robots decodes the packets, checks their integrity, and sets the target values for the 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 battery charge, we implemented locally a closed loop control 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 length ratio for the motor drivers accordingly. We use a simple P-control to adapt the motor power.

5 Behavior

5.1 Architecture

Our control architecture is shown in Figure 1. It is based on the Dual Dynamics scheme developed by H. Jäger [1996; 1997]. The robots are controlled in closed loops that use different time scales and that correspond to behaviors on different levels of the hierarchy.

We extend the Dual Dynamics concept by introducing a third element, namely the perceptual dynamics, as shown on the left side of the drawing. 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, 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 from the interval $[0, 1]$ 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 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.

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 a sensor. Similarly, the placement of actuators is determined by the time constant they need to produce a change in the environment. 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.

5.2 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 c could be 2, 4, 8, ... In the figure, $c = 2$ 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 $\mathbf{s}_{ci}^{z-1}$, $\mathbf{s}_{ci-1}^{z-1}$, $\mathbf{s}_{ci-2}^{z-1}$, ...
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.

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 $\alpha_{i/c}^{z+1}$. A higher behavior can use multiple layer-z-behaviors and each of them can be activated by many behaviors. For every behavior k on level $(z+1)$ that uses a behavior j from level z there is a term $\alpha_{i/c,k}^{z+1} T_{j,k}^z(\alpha_{i-1}^z, \mathbf{s}_i^z)$ 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. To determine the new activations the 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 a target value $g_{i,j,k}^z = G_{j,k}^z(\mathbf{s}_i^z, \mathbf{a}_{i/c}^{z+1})$.

$\mathbf{a}_i^z$ – *Actuator values:*
The more active a behavior j is, the more it can influence the actuator values

$$\mathbf{a}_i^z = (a_{i,0}^z, a_{i,1}^z, \ldots, a_{i,n_a^z-1}^z) \quad .$$

The desired change for the actuator value $a_{i,k}^z$ is:

$$u_{i,j,k}^z = \tau_{i,j,k}^z \alpha_{i,j}^z (g_{i,j,k}^z - a_{i-1,k}^z) \quad .$$

If several behaviors want to change the same actuator k, the desired updates are added:

$$a_{i,k}^z = a_{i-1,k}^z + u_{i,j_0,k}^z + u_{i,j_1,k}^z + u_{i,j_2,k}^z + \ldots$$

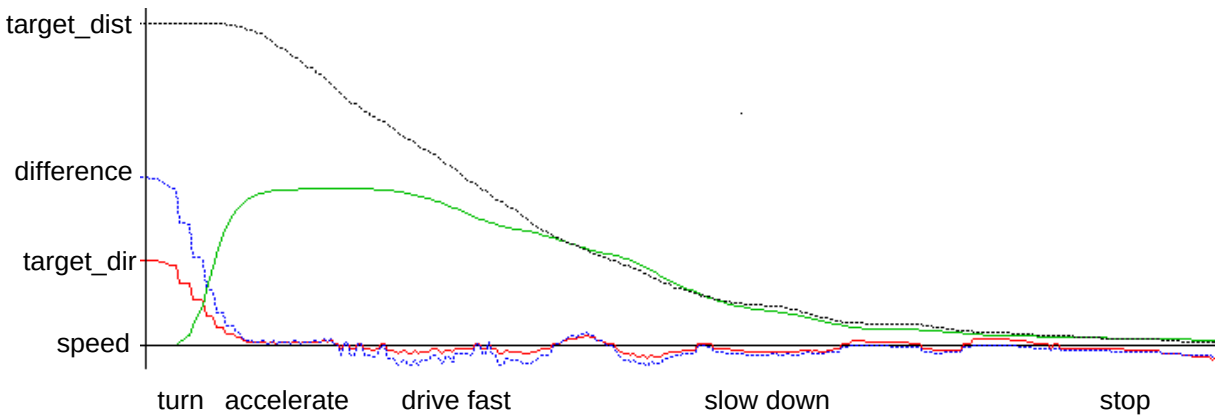

Figure 2: Recording of two sensors (distance and direction of the target) and two actuators (average motor speed and difference between the two motors) during a simple taxis behavior. The robot first turns towards the target, then accelerates, drives fast, slows down, and finally it stops at the target position.

5.3 Bottom-Up Design

Behaviors are constructed in a bottom-up fashion: 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. For this higher level more complex behaviors can now be designed 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 the addition of several layers, fairly complex behaviors can be designed that make decisions using abstract sensors 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 at the topmost level of the hierarchy.

5.4 Examples

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 speed of the motors and the speed differences between them. We choose 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. 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. Figure 2 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.

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 3. 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.

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 no longer valid and the run mode becomes active again. Figure 4 illustrates the trajectory of the field player 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.

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 5. We present, for instance, the angle and the distance to the ball and the nearest

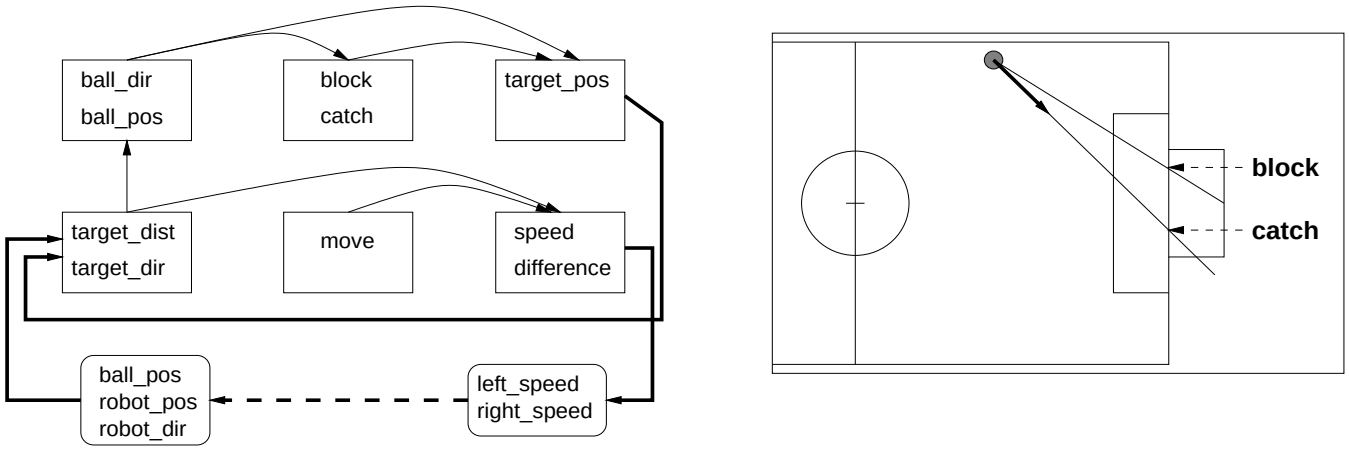

Figure 3: Sketch of goal keeper behavior. Based on the position, speed, and the direction of the ball it decides to either block the ball or to catch it.

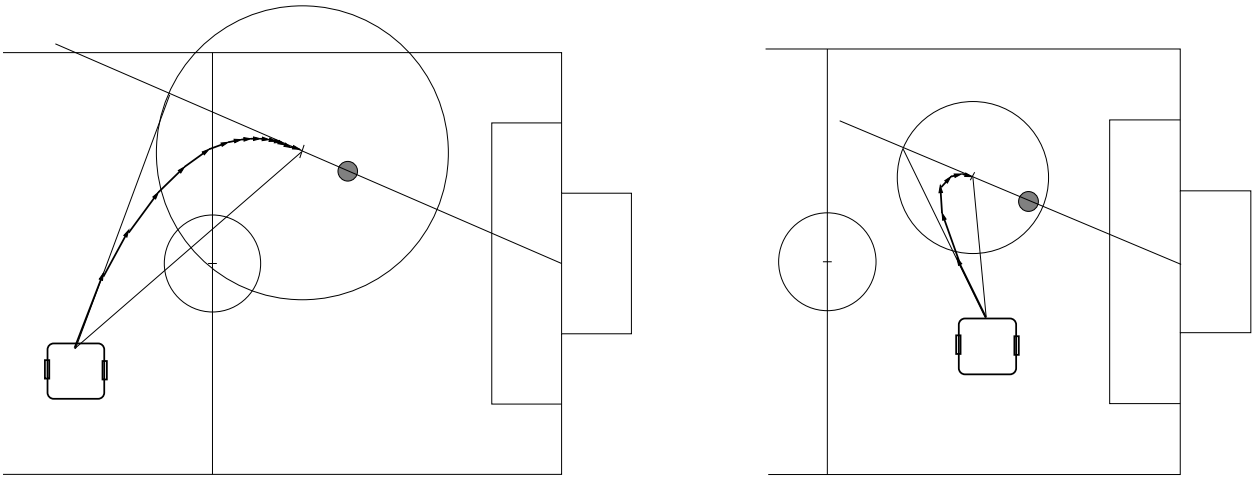

Figure 4: Trajectories generated in the run mode of the field player. It smoothly approaches a point behind the ball that lies on the line from the ball target through the ball.

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 Summary

We designed robust and fast robots with reliable radio communication and high speed vision. To generate actions, we implemented a reactive control architecture with interacting behaviors 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.

This framework could be used in the future to implement mechanisms, like adaptation and learning using Neural Networks [Rojas, 1996]. We will participate in the RoboCup'99 F180 league competition to benchmark our approach. Until then, a richer set of behaviors will be available.

Acknowledgments

We thank the companies Conrad ELECTRONICS GmbH, Dr. Fritz Faulhaber GmbH & Co KG, and Siemens ElectroCom Postautomation GmbH for their support that made this research possible.

References

[Brooks, 1991] R.A. Brooks. Intelligence without reason. A.I. Memo 1293, MIT Artificial Intelligence Lab, 1991.

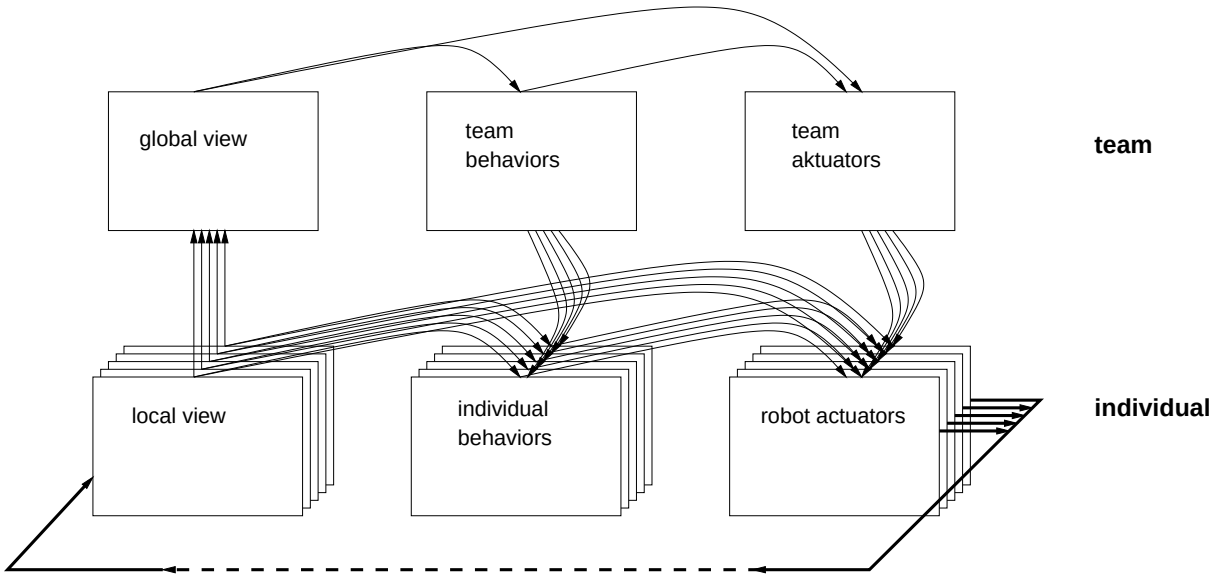

Figure 5: Sketch of the relation between the team and the individual robots.

[Christaller, 1999] T. Christaller. Cognitive robotics: A new approach to artificial intelligence. *Artificial Life and Robotics*, (3), 1999.

[Jäger and Christaller, 1997] 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.

[Jäger, 1996] H. Jäger. The dual dynamics design scheme for behavior-based robots: A tutorial. Arbeitspapier 966, GMD, 1996.

[Pfeifer and Scheier, 1998] R. Pfeifer and C. Scheier. *Understanding Intelligence*. MIT press, Cambridge, 1998.

[Rojas, 1996] R. Rojas. *Neural Networks*. Springer, New York, 1996.

[Schlottmann *et al.*, 1997] E. Schlottmann, D. Spenneberg, M. Pauer, T. Christaller, and K. Dautenhahn. A modular design approach towards behaviour oriented robotics. Arbeitspapier 1088, GMD, 1997.

[Steels, 1992] L. Steels. The pdl reference manual. AI Lab Memo 92-5, VUB Brussels, 1992.

[Steels, 1994] 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.