Using Hierarchical Fuzzy Behaviors in the RoboCup Domain

Alessandro Saffiotti and Zbigniew Wasik

Center for Applied Autonomous Sensor Systems
Dept. of Technology, Örebro University
S-70182 Örebro, Sweden
{asaffio,zbych}@aass.oru.se
http://www.aass.oru.se

Abstract. An important reason for the popularity of the behavior-based paradigm in autonomous robotics is the possibility to design complex robot behaviors in an incremental way. We propose a fuzzy hierarchical behavior-based architecture, in which rules and meta-rules are used in a uniform way at all levels of the control hierarchy. This architecture has been successfully used in a number of robots performing autonomous navigation tasks. In this paper, we show the use of hierarchical fuzzy behaviors to implement a set of navigation and ball control behaviors for a Sony four-legged robot operating in the RoboCup domain. We also show that the logical structure of the rules and the hierarchical decomposition simplify the design of very complex behaviors, like the "GoalKeeper" behavior.

1 Introduction

The commercial interest in autonomous robots is rapidly growing. Robots which are able to operate in natural, unmodified environments have an increasing number of potential applications, including: services in home, offices, and industrial sites; help to disabled or elderly people; monitoring and manipulation of hazardous sites and materials; emergency rescue operations; and entertainment.

Autonomous robot operation in natural environments, however, poses a number of challenges which are only partially solved by existing technologies. In particular, the control program of an autonomous robot must be able to cope with high degrees of uncertainty and unpredictability in the environment, with the presence of multiple and time-dependent goals, and with limited perceptual and computational resources. Such a control program necessarily has a high degree of complexity.

A popular way to address this complexity is to design the robot controller in a modular way. According to the behavior-based paradigm [1,2], complex controllers are built by combining in an appropriate way a number of simple behavior-producing units, or *behaviors*. The key to design simplicity is that each behavior is only meant to achieve a simple, elementary goal under a limited set of conditions. Complexity emerges from the combination of different behaviors, resulting in an overall controller that can cope with a larger

In: C. Zhou, D. Maravall and D. Ruan (Eds) Autonomous Robotic Systems. Springer, Berlin, 2003, pages 235-262.

number of goals under a larger set of conditions. However, the fundamental problem of *how* different behaviors should be combined together is not well understood yet. This seriously limits the scalability of behavior-based systems. The two main problems here are: (i) how to effectively *design* effective behavior combination strategies in complex domains, and (ii) how to *prove* that these strategies achieve the intended goals. The second point has been addressed, e.g., in [3]. This chapter contributes to the first point.

We propose a hierarchical approach to the incremental design of complex robot behaviors based on fuzzy logic. (See [4] for an overview of the uses of fuzzy logic in autonomous robotics.) The main points of our approach are:

- Behaviors are defined in terms of fuzzy "if-then" rules; this makes it easier to write complex control strategies based on heuristic knowledge.
- Complex behaviors are obtained by combining simpler ones using fuzzy meta-rules and a mechanism for behavior blending based on fuzzy logic.
- Behaviors inform perception so that the perceptual resources can be used effectively in a task-dependent way.

This approach has been used to implement complex navigation, perception, and ball manipulation behaviors on a team of Sony AIBO robots, and it has been tested in the RoboCup domain in the context of Team Sweden.[1]

The rest of this chapter is organized as follows. In the next section, we introduce the application domain and the overall architecture used to control our robots. Section 3 describes how basic behaviors are defined and implemented within this architecture, while Section 4 shows how complex behaviors are built by combining simpler ones. Section 5 deals with the use of behaviors to control perceptual resources. Section 6 gives a concrete example of how a very complex behavior (a full goal keeper) can be implemented using our approach. Section 7 briefly discusses the main benefits and problems of our approach, and concludes.

2 The RoboCup Domain

RoboCup is an international robot soccer competition held every year since 1996. The RoboCup competition is divided into several leagues, which differ in the type of robots and environment involved. The approach described in this paper was used in the legged robot league. This league is peculiar in that all competing teams use exactly the same physical platform, a special version of the Sony four-legged robot AIBO [6]. Two teams of four robots each compete on a field of approximately 3×5 meters, where all the relevant objects (the ball, the nets, the robots, and six landmarks around the field) can

[1] "Team Sweden" [5] is the Swedish national team that entered the Sony legged robot league at the RoboCup 1999, 2000, and 2001 competitions. Team Sweden currently consists of four universities: Örebro University, Lund University, Umeå University, and the Blekinge Institute of Technology.

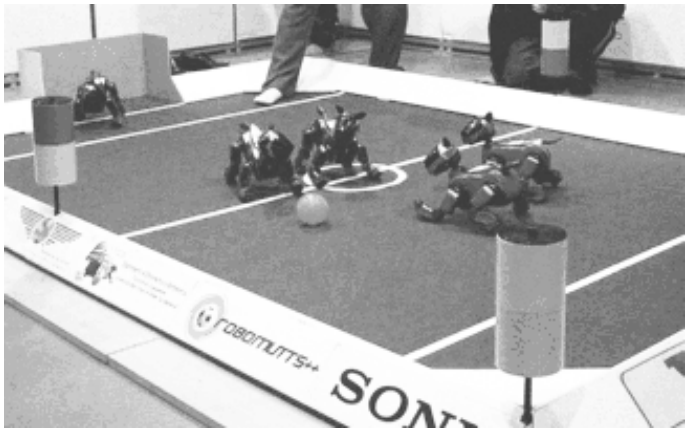

Fig. 1. A snapshot from the 2001 RoboCup competition.

be identified by their color. The robots operate in fully autonomous mode, but can use a radio link to exchange data between them. Fig. 1 shows a snapshot from a game.[2]

Fig. 2 shows one of the robots used in our work. The robot has three degrees of freedom in each leg, and additional degrees of freedom in the neck, head and tail. The principal sensor in this domain is a color camera that allows the robot to detect the objects in the environment and to estimate their position. The estimated position is affected by uncertainty and imprecision due to errors in image segmentation and to partial occlusions. The on-board perceptual and computational resources are limited, so it is important to rely on a cognitive architecture that can best use these resources while effectively copying with the uncertainty and umpredictability that characterize the RoboCup domain.

In our work, we use the layered architecture sketched in Fig. 3. This is a variant of the Thinking Cap, the autonomous robot architecture based on fuzzy logic in use at Örebro University [7]. We outline below the main elements of this architecture.

The lower layer (commander module, or CMD) provides an abstract interface to the sensori-motor functionalities of the robot. The CMD accepts abstract commands from the upper layer, and implements them in terms of actual motion of the robot effectors. In particular, CMD receives set-points for the desired displacement velocity $\langle v_x, v_y, v_\theta \rangle$, where v_x, v_y are the forward and lateral velocities and v_θ is the angular velocity, and translates them to an appropriate walking style by controlling the individual leg joints. This

[2] The snapshot was taken at the RoboCup 2001 games in Seattle. According to the RoboCup regulations at that time, the size of the field was about 2 × 3 meters, teams only had three robots, and no radio communication was allowed.

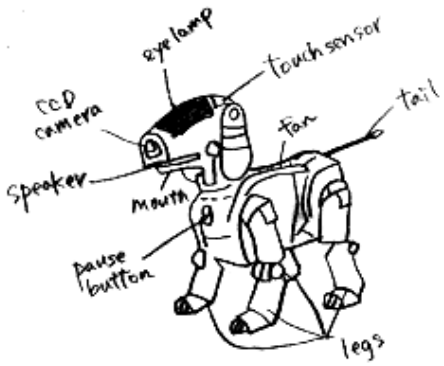

Fig. 2. The Sony AIBO legged robot. ©1999 Sony Corporation.

abstraction strategy allows us to write motion behaviors that can be easily ported across different physical platforms, including both legged and wheeled robots.

The middle layer maintains a consistent representation of the space around the robot (Perceptual Anchoring Module, or PAM), and implements a set of robust tactical behaviors (Hierarchical Behavior Module, or HBM). The PAM acts as a short term memory of the location of the objects around the robot: at every moment, the PAM contains an estimate of the position of these objects based on a combination of current and past observations with self-motion information. For reference, objects are named Ball, Net1 (own net), Net2 (opponent net), and LM1–LM6 (the six landmarks). The PAM is also in charge of camera control, by selecting the fixation point according to the current perceptual needs [8]. The HBM realizes a set of navigation and ball control behaviors, and it is described in greater detail in the following sections.

The higher layer maintains a global map of the field (GM) and makes real-time strategic decisions (RP). Self-localization in the GM is based on fuzzy logic, as reported in [9]. The RP implements a behavior selection scheme based on the artificial electric field approach [10]. We attach sets of positive and negative electric *charges* to the nets and to each robot, and we estimate the heuristic value of a given field situation by measuring the resulting electric potential at some *probe* position — for instance, the position of the ball. This heuristic value is used to select the behavior that would result in the best situation.

In the rest of this chapter we focus on the definition and implementation of the HBM. Additional information about the Team Sweden architecture and its other components can be found on the team web site [5], which also points to the relevant on-line publications.

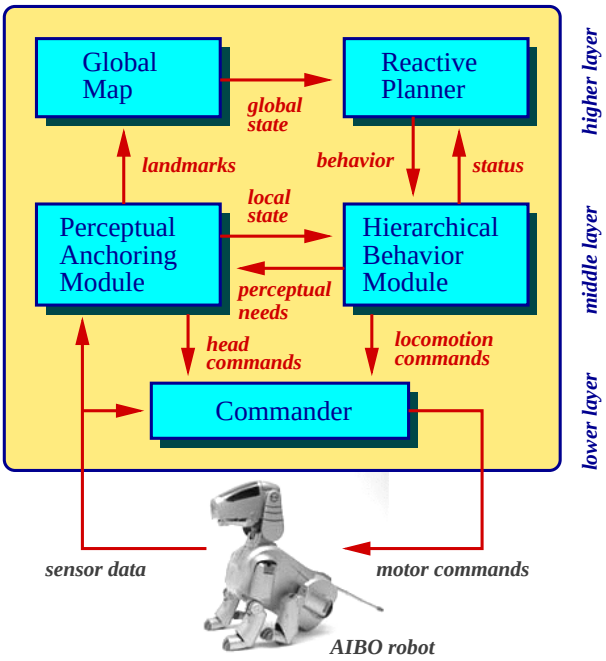

Fig. 3. The variant of the Thinking Cap architecture used by Team Sweden.

3 Basic Behaviors

In this section, we show how we define and implement *basic* behaviors, that is, behaviors that perform elementary types of actions. In our domain, these include turning toward the ball, going to the ball, or moving to a given position. These behaviors constitute the basic building blocks from which more complex types of actions are obtained by hierarchical composition.

3.1 Desirability Functions

In most behavior-based approaches found in the robotic literature, each behavior B acts as a regulator that takes its input from the robot's internal state variables (or directly from the robot sensors) and produces as output set-points for the robot actuators:

$$B : \text{State} \rightarrow \text{Control}\,. \tag{1}$$

Each behavior, then, can be thought of as a decision making agent that generates, in every situation, one "most preferred" control value. Different behaviors *compete* for the control of the actuators, and arbitration is often performed by some sort of winner-take-all mechanism.

Our formal approach to behavior definition is different [3,13]. We describe each behavior B in terms of a *desirability function*

$$Des_B : \text{State} \times \text{Control} \rightarrow [0,1]\,, \tag{2}$$

that measures, for each state vector x and control vector u, the desirability $Des_B(x,u)$ of applying the control u when the state is x. Intuitively, B expresses desirable behavioral traits as quantitative *preferences*, defined over the set possible control actions, from the perspective of the goal associated with that behavior. For example, a behavior for avoiding obstacles could map configurations of sonar readings that correspond to the presence of an obstacle on the left of the robot into a function that prefers actions that steer the robot to the right.

What is important in Equation (2) compared to (1) is that in Equation (2) several control actions can be desirable, to a different extent, for a behavior. This approach recognizes that many alternative controls can generate, to a greater or lesser extent, the same *type* of behavior. Each behavior, then, can be thought of as an agent that generates, in every situation, a set of preferences as to which command to apply. As we shall shortly see, this allows different behaviors to *cooperate* in the control of the actuators by combining their individual preferences into a tradeoff control value. This view is grounded in the formal semantic characterization given by Ruspini [11] after the seminal work by Rescher [12].

In our system, the input space (State) used by all behaviors is the *local state* provided by the PAM (see Fig. 3), which contains the current estimates of the position of all the objects in the field. The output space (Control) consists of the velocity set-points $\langle v_x, v_y, v_\theta \rangle$ which are transmitted to the CMD module (Fig. 3). An additional control variable k is used to indicate that a kick of a given type should be performed.

3.2 Implementing Desirability Functions

In practice, we implement a desirability function Des_B for a given behavior B by a set of *fuzzy control rules* of the form

$$\text{IF } A_i \text{ THEN } U_i, \quad i = 1, \ldots, n\,. \tag{3}$$

where U_i is a fuzzy set on the universe of control values, and A_i is a propositional formula in fuzzy logic whose truth value depends on the current values of the local state variables, e.g., the estimated distance to the ball. Fuzzy control rules allow us to easily express our heuristic knowledge about which actions should be performed in each situation in order to promote the goal of that specific behavior, e.g., to get close to the ball.

For instance, the following fuzzy control rules implement the *GoToBall* basic behavior.

IF (BallOnLeft $\wedge \neg$ BallHere)	TURN(Left)
IF (BallOnRight $\wedge \neg$ BallHere)	TURN(Right)
IF (BallAhead $\vee$ BallHere)	TURN(Ahead)
IF $\neg$ BallHere	GO(Fast)
IF BallHere	GO(Stay)
ALWAYS	SIDE(None)

The left hand side of these rules contain fuzzy formulas obtained from a set of fuzzy predicates, like BallOnLeft, by the standard fuzzy connectives $\wedge$ (min), $\vee$ (max) and $\neg$ (complement to 1). ALWAYS denotes a precondition which is always true. Each fuzzy predicate is defined by a function that computes its truth value, in the $[0, 1]$ real interval, from the value of the internal state variables. Figure 4 (left) shows the truth values of the fuzzy predicates BallOnLeft, BallOnRight and BallAhead as a function of the estimated angle θ between the robot and the ball. Note that the use of general fuzzy formulas to express rule preconditions makes our approach more expressive than standard fuzzy control, where only conjunctions of positive literals are allowed in the antecedent of the control rules.

The right hand side of each rule indicates which control variable should be affected by that rule: the GO and SIDE keywords respectively indicate the v_x variable, corresponding to forward-backward motion, and the v_y variable, corresponding to side-to-side motion. The TURN keyword indicates the v_θ variable, corresponding to rotational motion. The parameters Left, Right and so on are linguistic labels that denote fuzzy sets of control values for each control variable. Fig. 4 (right) shows the five fuzzy sets for the v_θ (TURN) variable.

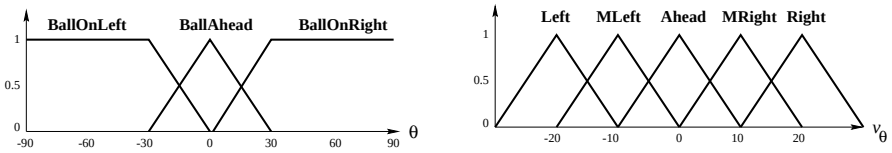

Fig. 4. Left: the three fuzzy predicates related to the ball angle. Right: the five fuzzy sets for the TURN control values.

Given a set R of n fuzzy rules of the form (3), our fuzzy controller computes a corresponding desirability function Des_R by:

$$Des_R(x, u) = [A_1(x) \wedge U_1(u)] \vee \cdots \vee [A_n(x) \wedge U_n(u)]. \quad (4)$$

Intuitively, Equation (4) characterizes a control action u as being desirable in state x if there is some rule in R that supports u and whose antecedent is true in x. This interpretation of a fuzzy rule-set is that of a classical (Mamdami type) fuzzy controller, generalized so as to allow each antecedent A_i to be an arbitrary fuzzy-logic formula. The Des_R desirability function can

be directly used to select a most desired control action $\hat{u}$ by applying a defuzzification technique. In our system, we use Center of Gravity (CoG) defuzzification:

$$\hat{u} = \frac{\int u \, Des_R(x,u) \, du}{\int Des_R(x,u) \, du} \,. \tag{5}$$

In general, however, this desirability function is first combined with the ones produced by other concurrent behaviors, as discussed in Section 4 below.

3.3 A Suite of Basic Behaviors for RoboCup

Fig. 5 lists the basic behaviors that we have implemented in our robots for the RoboCup domain. Behaviors are classified into three categories: *navigation* behaviors (marked by N) modify the position of the robot in the field; *manipulation* behaviors (M) modify the position of other objects, typically the ball; and *perceptual* behaviors (P) acquire information. The table also shows the number of fuzzy rules used in each behavior. Usually, only a small number of rules is needed to define each behavior, which makes behaviors simpler to write, tune, and maintain. The two key factors for this simplicity are: (i) an accurate design choice about which behaviors should be implemented as basic, and (ii) the fact that we allow arbitrary fuzzy formulas in the rule preconditions.

Type	*Behavior name*	*# rules*	*Typical instance*
N	*GoTo(X)*	7	GoTo(Ball)
N	*GoToStar(X)*	6	GoToStar(Target)
N	*Face(X)*	7	Face(Ball)
N	*Align(X,Y)*	13	Align(Ball,Net2)
M	*Kick*	5	Kick
M	*StealLeft*	3	StealLeft
M	*StealLeft*	3	StealLeft
M	*Block*	2	Block
P	*Search(X)*	4	Search(Ball)
P	*SelfLocalize*	5	SelfLocalize

Fig. 5. The basic behaviors implemented for the RoboCup domain.

Most behaviors take as arguments one or more objects in the environment. For instance, the generic *GoTo* behavior is typically used to go close to the ball by giving it the ball as argument. In a similar way, we typically use *Face(Ball)* and *Align(Ball,Net2)* to perform local position adjustments in order to achieve an adequate posture for kicking. (Net2 denotes the opponent net, and Net1 denotes our own net.) Obviously, the control program cannot be given a pointer to an external physical object. Instead, it is given a pointer to an internal data structure, called *anchor*, whose properties are kept in

synch with the properties of the physical object using perception. The PAM module (Fig. 3) is in charge of implementing this anchoring process — see [8] for more on this point.

The difference between the *GoTo* and the *GoToStar* behaviors is that the latter performs obstacle avoidance while trying to go to the target position. This is done by setting up a local fuzzy occupancy grid around the robot, and performing A* path planning on this grid. The grid contains 15×17 cells, each representing a square of 10×10 cm, and it is filled with occupancy information about the objects to be avoided. Each object is blurred to account for the uncertainty in its estimated position — see Fig. 6. Since the grid is small, filling and planning are inexpensive and they are re-computed at every control cycle (200 msec in our implementation). The *GoToStar* behavior is typically used as a sub-behavior (see below). The grid is filled by the caller behavior, since that behavior knows which objects should be avoided and which ones should not. For instance, navigation behaviors usually need to avoid the ball when the robot is moving in the direction of its own net, but not when it is moving toward the opponent net.

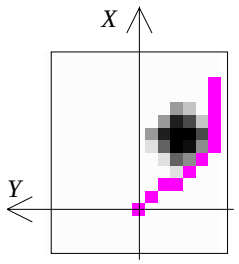

Fig. 6. The local occupancy grid used by the *GoToStar* behavior for obstacle avoidance, and a path leading to a target point behind the ball. The robot is at the origin, in top-view and directed as the X axis.

The *Kick*, *StealLeft* and *StealRight* behaviors respectively kick the ball in the forward direction and to the left/right of the robot. A combination of leg and head motions is used in the CMD to implement each kick. The *Block* behavior causes the robot to stand in a crab-like posture, which is useful to block an incoming ball.

Finally, the *Search* and *SelfLocalize* behaviors visually scan the field until the object is seen, or until enough landmarks are seen to allow establish the robot's location in the field, respectively. These behaviors use a combination of head motion and body rotation.

4 Complex Behaviors

In our approach, we build complex behaviors by combining simpler ones using fuzzy meta-rules. This procedure can be iterated to build a full hierarchy of increasingly complex behaviors. In this section, we discuss the main mechanism used to realize behavior composition, called *Context-Dependent Blending*, and show how we have implemented it in our robots.

4.1 Context-Dependent Blending

In behavior-based approaches, the overall behavior of the system is the result of the coordinated activity of several independent behavior-producing units. While this divide and conquer strategy is the key to the power of these approaches, it is also its main source of difficulty. In fact, the problem of how to coordinate behaviors so that they result in the performance of the intended task still constitutes the Holy Grail of behavior-based robotics.

Following [14], we split the behavior coordination problem into two conceptually different sub-problems: (i) how to decide which behaviors should be activated at each moment; and (ii) how to combine the results from different behaviors into one command to be sent to the robot's effectors. We call these the *behavior arbitration* and the *command fusion* problems, respectively.

The arbitration policy determines which behavior(s) should influence the operation of the robot at each moment, and thus ultimately determines the task actually performed by the robot. Many proposals in the literature use a winner-take-all crisp switching schema: in each situation, one behavior is selected and is given complete control of the effectors (e.g., [1,15]). This simple scheme may be inadequate in situations where several criteria should be taken into account. To see why, consider a robot that encounters an unexpected obstacle while following a path, and suppose that it has the option to go around the obstacle from the left or from the right. This choice may be indifferent to the obstacle avoidance behavior. However, from the point of view of the path-following behavior, one choice might be dramatically better than the other. In most implementations, the obstacle avoidance behavior alone could not know about this, and would take an arbitrary decision.

More flexible arbitration policies can be obtained using fuzzy meta-rules of the form

$$\text{IF } \textit{context} \text{ THEN } \textit{behavior}, \tag{6}$$

meaning that *behavior* should be activated with a strength given by the truth value of *context*, a formula in fuzzy logic. The use of fuzzy meta-rules to express behavior arbitration policies has two main advantages: (i) the ability to express partial and concurrent activations of behaviors; and (ii) smooth transitions between behaviors.

If several behaviors can be simultaneously activated, we need to solve the problem of how to combine the output of different behaviors that refer to the

same control variable. That is, we have to solve the command fusion problem. Our definition of behaviors as preference-producing modules suggests that command fusion can be thought of as the problem of aggregating individual preferences: if Des_{B1} and Des_{B2} denote the desirability functions produced by two behaviors B1 and B2, respectively, then the combined preferences of B1 and B2 are represented by the function Des_B given, for all state x and control value u, as

$$Des_B(x,u) = Des_{B1}(x,u) \wedge Des_{B2}(x,u)\,, \tag{7}$$

where $\wedge$ denotes as usual the minimum operator.

In practice, we represent the output of each behavior $B_i, i = 1,\ldots,n$, at a given state x by a fuzzy set U_i over the space U of control values. We then use a fuzzy conjunction operator to combine the preferences of different behaviors, represented by fuzzy sets on U, into a collective preference U_{fuse} given by

$$U_{\text{fuse}}(u) = \bigwedge_{i=1,\ldots,n} U_i(u)\,. \tag{8}$$

Finally, we chose a command from this collective preference according to a defuzzification strategy, e.g., the CoG Equation (5). Fig. 7 (left) illustrates this process on two behaviors $B1$ and $B2$ both concerned with the turning angle.

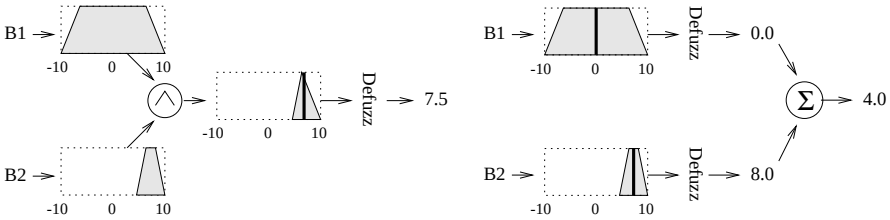

Fig. 7. Two ways to fuse commands: combining individual preferences (left); combining individual decisions (right). The final result may be different.

The fact that defuzzification is performed after combination is crucial, since the decision taken from the collective preference can be different from the result of combining the decisions taken from the individual preferences. Fig. 7 graphically illustrates this point. Intuitively, the individual decision issued by a behavior tells us which is *the* preferred command according to that behavior, but does not tell anything about the desirability of alternatives. Preferences contain more information, as they give a measure of desirability for each possible command. This observation explains why fuzzy command fusion is fundamentally different from potential field methods [2,16], in which

force vectors represent individual decisions.[3] A similar distinction between combining preferences and combining decisions is commonly made in the field of data fusion [17].

It should be noted that blind application of averaging defuzzification strategies, like CoG, to the combined fuzzy set may produce problematic results. In particular, when this set is not unimodal, defuzzification may result in the selection of an undesirable control value, i.e., a value which lies in the gap between two peaks of the combined set. In the case of robot control, this may mean that the robot, having the option to avoid an obstacle from the right or from the left, decides to go straight. It is the responsibility of the behavior designer to make sure that the rule-set is free from inconsistencies and ambiguities: e.g., rules that propose drastically different controls should have mutually exclusive pre-conditions. Other authors address this problem by defining *ad-hoc* defuzzification schemes (e.g., [18]).

In our system, we use fuzzy meta-rules (6) to represent the arbitration policy, and fuzzy fusion (7) to perform command fusion. The overall desirability function Des^* is then given by

$$Des^*(x,u) = \bigwedge_{j=1,\ldots,m} \left(C_j(x) \wedge Des_{B_j}(x,u)\right) , \tag{9}$$

where C_j and B_j are the *context* and the *behavior* of the j th meta-rule. This general form of behavior combination, originally proposed in [13], is called *context-dependent blending*, or CDB for short.[4] CDB has been used, sometimes under different names, by several authors in several robots. An overview of the uses of CDB in the literature can be found in [14].

4.2 Implementing Context-Dependent Blending

We have implemented CDB in a very simple way. Any behavior can include fuzzy meta-rules of the form

$$\text{IF } C_j \text{ USE } B_j, \quad j = 1,\ldots,m\,. \tag{10}$$

where C_j is a propositional formula in fuzzy logic, called the *context*, and B_j is a sub-behavior to be called. For example, the following rule set implements a simple PenaltyKick behavior, which brings the robot in front of the ball, aligns it with the opponent net (Net2), and kicks.

[3] Vector approaches can be simulated as a special case of fuzzy command fusion, e.g., by using fuzzy numbers for preferences, prod-sum combination, and COG defuzzification. In this case, the order in which defuzzification and combination are performed is irrelevant.

[4] Context-depending blending can be given more general definitions in terms of arbitrary T-norms [3]. Our implementation, however, relies on the distributive property, which only holds when using the min/max pair of norms.

```
IF (BallFar)                  USE GoTo(Ball)
IF (BallNear ∧ ¬ Aligned)     USE Align(Ball,Net2)
IF (BallNear ∧ Aligned)       USE Kick(Ball)
```

This behavior will achieve the intended goal under the assumption that the robot starts somewhere in the part of the field which is behind the ball with respect to Net2. Note that in some situations several sub-behaviors can be (partially) activated and combined by CDB. For instance, as the robot approaches the ball, the Align behaviors becomes more and more active while the GoTo behavior is progressively deactivated. As a result, the robots starts the alignment maneuver while smoothly slowing down as it gets closer to the ball.

Calls to sub-behaviors can be iterated, so that we can build a full hierarchy of behaviors. For instance, the following rules use the above PenaltyKick behavior to implement a simple Score behavior where the robot can start anywhere in the field.

```
IF (BehindBall)      USE PenaltyKick()
IF (¬ BehindBall)    USE GoBehind(Ball,Net2)
```

The GoBehind behavior uses the GoToStar behavior to reach a location behind the ball while avoiding collisions with ball. Fig. 8 shows the corresponding behavior hierarchy.

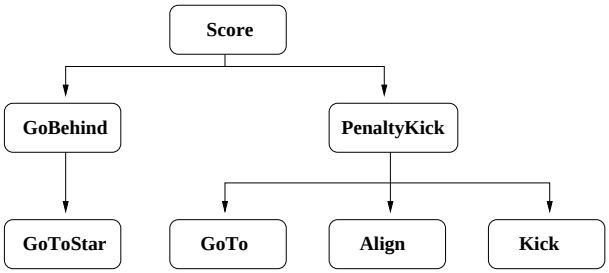

Fig. 8. Behavior hierarchy for a simple Score behavior.

In principle, all of the fuzzy rules in a called sub-behavior B_j are treated as if they were rules inside the caller behavior, except that their consequent is weighted by the value of the context C_j. Suppose that a behavior B contains the meta-rule

$$\text{IF } C' \text{ USE } B' ,$$

where B' is composed of the rules

$$\text{IF } A_i \text{ THEN } U_i, \quad i = 1, \ldots, n .$$

Then, the above meta-rule in B is (virtually) replaced by the set of rules

$$\text{IF } (C' \wedge A_i) \text{ THEN } U_i, \;\; i = 1, \ldots, n \,.$$

If B' itself contains some meta-rules, the expansion process is repeated recursively. The overall desirability functions (one for each controlled variable) are computed from the resulting set of rules using Equation (4). The final control values are obtained from these functions by using COG defuzzification, as per Equation (5). This solution is equivalent to performing CDB according to Equation (9), provided that the min and max pair of T-norm and T-conorm are used to evaluate $\wedge$ and $\vee$ [3].

The above implementation strategy has the advantage that object-level control rules and meta-rules can be intermixed in the same behavior. For instance, the following set of fuzzy rules is used to implement the *PushBall* behavior:

IF (BallInFront)	USE GoTo(Ball)
IF (BallOnLeft)	SIDE(Left)
IF (BallOnRight)	SIDE(Right)
IF ($\neg$ BallInFront)	GO(Stay)
IF ($\neg$ BallInFront)	TURN(Ahead)

The first rule makes the robot move toward the ball when the ball is in front of it, thus pushing the ball. The second and third rules add some side-to-side motion intended to keep the ball centered in front of the robot. The last two rules inhibit forward and rotational motion when the ball is not in front of the robot, and until the lateral (SIDE) motion brings it to a central position again. (Stay and Ahead are linguistic labels that denote zero forward and rotational velocity, respectively.)

4.3 CDB Versus FSM

CDB provides a purely reactive way to combine behaviors: at every moment, the behaviors which are activated depend only on the current perceptual input, as reported by the PAM, and they do not depend on the previous history of behavior activations. Another popular way to define complex behaviors is to use finite state machines (FSM), where states are associated with behaviors and transitions with perceptual events. In a FSM, the behavior(s) to be activated depend on the perceptual input *and* on the previous history. Typically, in these approaches exactly one behavior is activated in every state, and it is given complete control of the effectors.

Fig. 9 shows a FSM implementation of a Score behavior based on the same building blocks as the previous example (Fig. 8). If the robot starts somewhere in the area between the ball and Net2, it will activate in sequence GoToStar, GoTo, Align and Kick. The same would happen using the

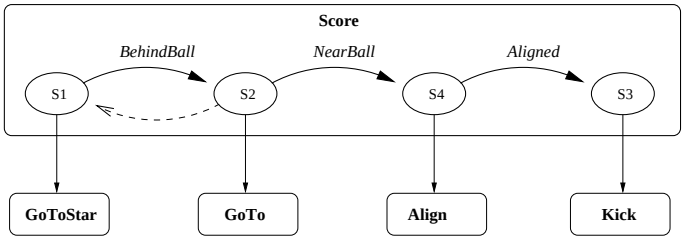

Fig. 9. A FSM for a simple Score behavior.

hierarchical behavior shown in Fig. 8 and implemented by CDB. CDB, however, is intrinsically more robust with respect to sensor noise and unexpected events than the FSM approach. For instance, suppose that after the robot has reached the area behind the ball and has started the GoTo behavior, the ball is moved so that the robot is not behind it any more. The CDB solution naturally copes with this situation, since the BehindBall condition becomes false thus causing the GoBehind behavior to be re-activated. By contrast, in the FSM a new transition should be designed in order to account for this exception (dashed arrow in Fig. 9). A similar problem would arise if a transition were fired by a spurious perceptual event. In general, the FSM needs to incorporate explicit transitions to account for any possible sequence of events.

No matter what the advantages of CDB are, in some cases the introduction of internal state is unavoidable. For instance, we cannot write a purely reactive *Patrol* behavior that goes from one net to the other and back: if the robot is in the middle of the field and perpendicular to the axis between the nets (say, while avoiding an obstacle) it cannot know which way to turn unless it has an internal state to indicate to which net it was going.

Internal states are easily incorporated into our fuzzy behaviors by maintaining a state variable inside the behavior, and deciding which behavior to call according to the value of this variable. Here is a simple example implementing a Patrol behavior.

IF (State(0))	SetState(1)
IF (State(1) $\wedge$ AtNet2)	SetState(2)
IF (State(2) $\wedge$ AtNet1)	SetState(1)
IF (State(1))	USE GoTo(Net2)
IF (State(2))	USE GoTo(Net1)

The first three rules encode the state transitions of the FSM, the remaining rules associate a behavior with each state. This type of complex behavior can be seen as an instance of a hybrid automaton: a FSM in which each state is associated with a specific control law [19].

5 Behaviors and Perception

Basic and complex behaviors in the Hybrid Behavior Module (HBM) take their input from the local state provided by the PAM (see Fig. 3). This acts as a short term memory of the location of the objects around the robot, stored in robot-centered coordinates. The local state is updated and sent to the HBM at a regular rate (5 Hz in our current implementation). Updating is done by three mechanisms: by *perceptual anchoring*, whenever the object is detected by the robot's camera and its position is measured from the image data; by *global information*, for static objects (e.g., the nets) whenever the robot knows its location in the global map; and by *odometry*, modifying the previous position of the object whenever the robot moves.

In general, estimates based on fresh perceptual data are more reliable than estimates based on odometric update. This is especially true in our domain where: (i) objects can move in unpredictable ways and (ii) odometry is very unreliable when using legged locomotion. Unfortunately, because of limited perceptual and computational resources, the robot cannot keep all the objects in the field under observation at all times. In our AIBO robots, for instance, the head-mounted camera has a limited field of view and limited pan-tilt speed. Keeping track of an object, then, necessarily implies losing track of many others. The design of effective allocation strategies for the perceptual resources is therefore of paramount importance in this domain.

The key observation here is that the robot typically needs to access information about different objects at different stages of execution. For example, in order to kick the ball into the opponent's net, the robot only needs to know the position of the ball and of the net; when it later returns to its home position, it needs the position of the landmarks in order to correctly self-localize, but it does not need to know the position of the ball or of the net any more. In short, perception should be *task-dependent* [20].

To achieve task-dependent perception, we introduce a link in our architecture that feeds the perceptual needs of the controller back to the perceptual module (see Fig. 3) . At every control cycle, the currently active behaviors in the HBM inform the PAM of which objects are "needed" for execution and which ones are not. The perceptual processes in the PAM use this information to decide where to point the camera and which perceptual routines to activate.

Behaviors specify their perceptual needs using fuzzy rules of the form

$$\text{IF } A_i \text{ NEED}(O_i), \tag{11}$$

where A_i is a fuzzy formula, and O_i is the name of one of the objects in the environment. The effect of this rule is to assert the perceptual need for object O_i at a degree that depends on the truth value of A_i.

For example, we can extend the *PenaltyKick* behavior described in the previous section with the following perceptual rules:

IF (BallNear)	NEED(Net2)
ALWAYS	NEED(Ball)

These rules say that the controller needs to have fresh perceptual data for the ball at all times, but only needs to have an accurate estimate of the position of the opponent's net (Net2) when the robot is close to the ball in order to correctly perform the *Align* and *Kick* maneuvers. In a situation where the ball is at 400 mm from the robot, the truth value of BallNear is 0.7, and these rules assert a degree of need of 1.0 for the ball object and of 0.7 for the opponent's net.

We can think of (11) as a rule controlling an additional control variable $needed_i$ that expresses the degree of perceptual need for object O_i. The NEED rules of different behaviors are combined through CDB in the same was as the rules for the other control variables. Since the consequents of these rules are numbers in $[0, 1]$, the result of the combination is simply given, for each variable $needed_i$, by the maximum value computed by any rule that has NEED(O_i) as its consequent.

The values of the $needed_i$ variables so computed are sent to the PAM, which uses them to decide which object should constitute its perceptual focus at every given moment. Intuitively, this is an object which is needed by the currently active behaviors and which has not been updated recently. Full details on our active perception strategy can be found in [8].

During the robot activity, behaviors are dynamically activated and deactivated according to the fuzzy meta-rules. Correspondingly, the perceptual needs change continuously. The combined action of the perceptual rules in the HBM and of the focus selection mechanism in the PAM directs the perceptual resources towards the needs with are most important with respect to the current task at each moment.

Fig. 10 shows how perceptual processes are directed by perceptual needs during an execution of the *PenaltyKick* behavior using the two perceptual rules above. The upper part of the figure plots the temporal evolution of the estimates of the distance and angle (θ, ρ) to the ball, and of the angle θ to Net2. The circles mark the fixation points sent to the camera pan joint by the PAM; the dotted vertical lines mark visual scans done in order to find an object. The lower part plots the degree of perceptual support, on a $[0, 1]$ scale, for the ball and the net, respectively.

Until time 118 the robot was approaching the ball (as testified by the decreasing ρ) while tracking it. Since the "NearBall" predicate was false, the net was never selected as the focus of attention; as a consequence, the camera was never pointed at the net, and θ maintained its default zero value. At time 118, the truth value of "NearBall" was high enough for the net to become the focus of attention. The robot then started a visual scan to search for the net, momentarily losing sight of the ball. The net was soon seen, and the camera was then pointed to the expected position of the ball to re-acquire it. When the robot got even closer to the ball (around time 130) it executed

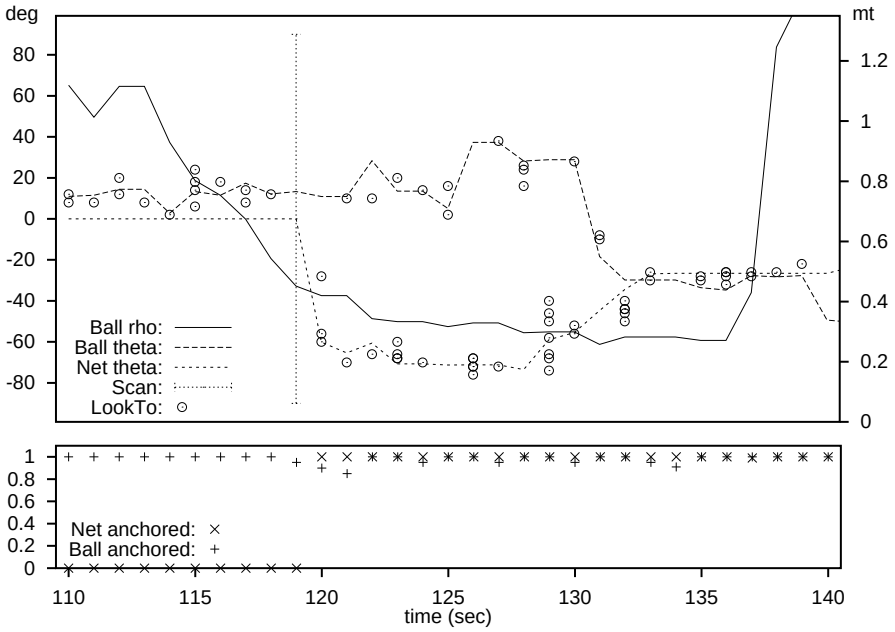

Fig. 10. Task-dependent perception. The robot only tracks the ball until it gets close to it, and then tracks both the ball and the net (see text).

the alignment maneuver while using the camera to alternatively anchor the ball and the net. The alignment maneuver brought the θ of both the ball and the net close to zero. Finally, at time 137, the robot kicked the ball, which headed away from the robot and toward the net.

6 A Case Study: the Goal Keeper

We now give an example of a particularly complex behavior: the *GoalKeeper* behavior. This behavior is run by a robot who is designated as the team goal keeper, and it is intended to perform the full goal keeping task. This behavior is entirely implemented in the HBM using a complex hierarchy of sub-behaviors: the Reactive Planner is not used by the goal keeping robot (see Fig. 3).

6.1 Overall Strategy

The task of the goal keeper is to defend its net from possible scores from any robot. We have adopted a rather defensive strategy: the robot always stays close to its net, tries to be on the way between the ball and the net, and only

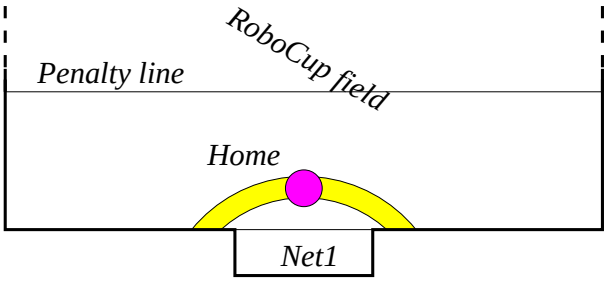

Fig. 11. The workspace of the goal keeper.

acts to send the ball away when there is a good opportunity to do so. Fig. 11 shows the workspace of the goal keeper.

The goal keeper operates in two distinct modes. Under normal conditions, the robot stays in the zone between the net and the penalty line, and it operates in "Play" mode. What is done in this mode depends on the position of the ball. If the ball is far away, or not seen at all, the robot stays at its home position (the circle in Fig. 11) pointing to the opponent net. From this position the robot can see most of the field and it has better chances to localize the ball. Occasionally, the robot turns to the left and to the right to check the blind spots behind it. If the ball is in view, the robot moves left and right at a fixed distance from the center of the net (the circular arc in Fig. 11) in order to be on the line between the ball and the net. If the ball comes too close, it may decide to push or kick it away.

If the robot finds itself well beyond the penalty line, it switches to the "Home" mode of operation. This may happen, for instance, after the robot has performed a long foray to push the ball away, or if it has been displaced by the referees in application of a game rule. Under this mode, the robot tries to reach its home position as quickly as possibly while avoiding collisions with the ball or the other robots on its way.

In addition to implementing the above strategy, we have made a design choice to develop a set of specialized walking and kicking routines for the goal keeper robot, implemented in the CMD module (Fig. 3). We have used walking styles that keep a very low posture so as to improve stability and grip, and we have increased the speed of lateral motion since the goal keeper needs to quickly move left and right. We also keep the legs far apart whenever possible, so as to cover a larger portion of the net. For instance, in the "Block" position, the robot stays still in a crab-like posture that covers about 50% of the size of the net.

6.2 The Behavior Hierarchy

The above strategy has been implemented as a set of specialized behaviors, organized according to the hierarchy shown in Fig. 12.

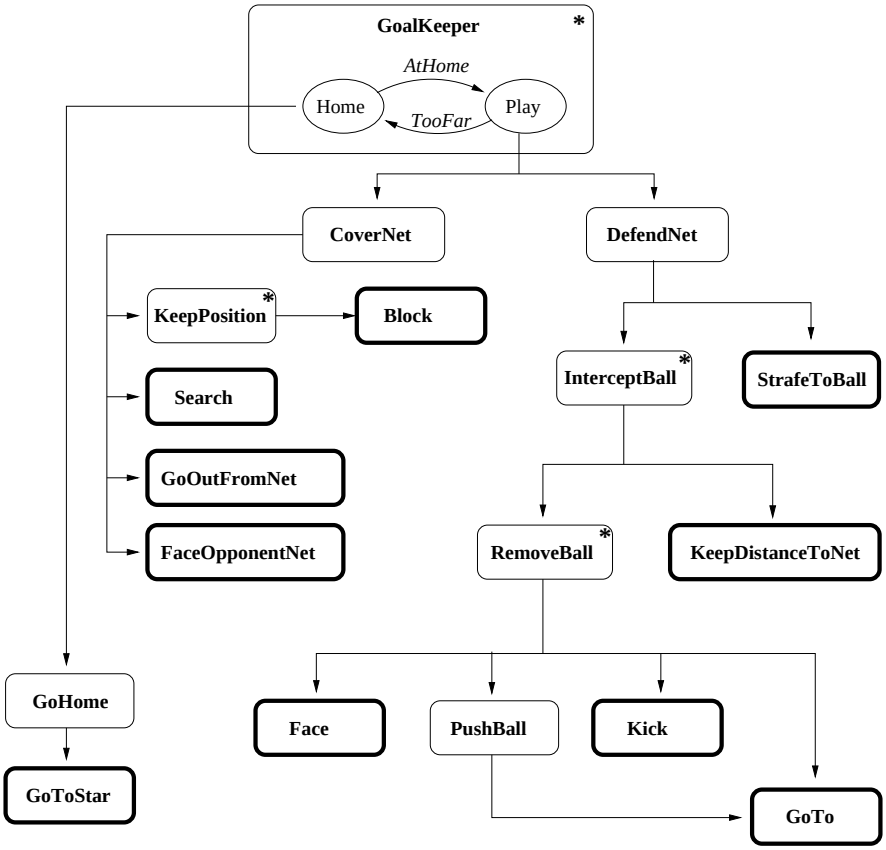

Fig. 12. Behavior hierarchy for the GoalKeeper behavior. The boxes with thick borders indicate basic behaviors. Behaviors with a '*' are detailed in the text.

The boxes with thick borders indicate basic behaviors, that is, behaviors that directly control motion and do not call any sub-behaviors. These are the same basic behaviors listed in Fig. 5 above, plus the following goal-keeper specific behaviors:

FaceOpponentNet Orients the robot towards the opponent's net.
GoOutFromNet Gets the robot out from inside its own net.
KeepDistanceToNet Moves the robot forward or backward in order to keep a fixed distance from the net.
CrabToBall Gets the robot in front of the ball using side-to-side motion.

Notice that the task of *FaceOpponentNet* could be performed by a call to *Face(Net2)*. However, the *Face* behavior is tuned to perform the precise but slow maneuvers typically needed in order to achive a correct position

for an effective kick. We have therefore preferred to implement a specialized behavior that provides faster, albeit less accurate, motion.

The other behaviors in the hierarchy are complex behaviors intended to perform the following tasks.

GoalKeeper This is the top-level behavior, described in the next section.
CoverNet Maintains a protective position while observing the field and occasionally checking the left and right corners.
DefendNet Intercepts the ball when it comes close, and possibly gains control of it.
GoHome Reaches the home position from any place on the field while avoiding collisions.
KeepPosition Maintains the home position.
InterceptBall Places the robot on the line between the ball and the net.
RemoveBall Gains possession of the ball and gets it out of the penalty area.

In addition, the *PushBall* behavior described in Section 4 above is used by *RemoveBall*.

Some of the above behaviors express specific perceptual needs by way of perceptual rules of the form (11) above. For instance, most behaviors express a need for the ball position. The *KeepPosition* and *GoHome* behaviors both need to have accurate information about the robot's own location in the field, and hence they express a need for the most probably visible landmarks, including the opponent's net. In addition, the *GoHome* behavior also needs to know the position of all the robots in order to perform obstacle avoidance while navigating to the home position. In general, the overall perceptual needs of the *GoalKeeper* behavior depend on which sub-behaviors are activated at every moment, and are used to direct perception as discussed in Section 5.

6.3 The Top-Level Behavior

The top-level *GoalKeeper* behavior is implemented by the rules shown in Fig. 13. Rules 1–3 encode a Finite State Machine with two internal states, corresponding to the two modes of operation described above: Home (state 1) and Play (state 2). The use of a FSM here is necessary since in general the action to perform does not only depend on the current perception, but also on the current mode. For instance, consider the situation in which the robot is close to the penalty line and oriented toward its own net and no ball is seen: in the Home mode, the robot should keep going forward toward its home position (using *GoHome*); in the Play mode, however, the robot should turn toward the opponent's net and go backward to its home position (using *CoverNet*).

Each mode is implemented by a call to one or more sub-behavior(s). In the Home mode (rule 4) the *GoHome* behavior is called: this fills up the local occupancy grid with information about the position of all the robots and the ball, and then calls the *GoToStar* behavior to navigate to its home position.

1.	IF (State(0))	SetState(1)
2.	IF (State(1) $\wedge$ AtHome)	SetState(2)
3.	IF (State(2) $\wedge$ FarFromHome)	SetState(1)
4.	IF (State(1))	USE GoHome()
5.	IF (State(2) $\wedge$ ($\neg$ BallInView $\vee$ BallOnOtherSide))	USE CoverNet()
6.	IF (State(2) $\wedge$ BallInView $\wedge$ BallOnThisSide)	USE DefendNet()

Fig. 13. The *GoalKeeper* behavior.

In the Play mode, the robot reactively chooses a play strategy depending on the position of the ball. If the ball has not been seen for a while or it is in the opposite half of the field, then the *CoverNet* behavior is used (rule 5). This behavior uses the *KeepPosition* and *FaceOpponentNet* to maintain a good observation position. In addition, the *Search(Ball)* behavior is used to turn and inspect blind spots at regular intervals.[5] The *GoOutFromNet* behavior may also be needed if the robot ends up inside its own net — an event which is not uncommon during the RoboCup games!

If the ball is detected on our side of the field, the *DefendNet* behavior is used (rule 6). This in turn uses the *CrabToBall*, *InterceptBall* and *KeepDistanceToNet* sub-behaviors in order to quickly position the robot on the line between the ball and the net at a fixed distance from the net. If the ball comes close to the robot, the *RemoveBall* behavior is used to try and send it out of the penalty area.

6.4 Other Behaviors

In order to give a more concrete impression of how the goal keeper behaviors are implemented, we discuss here three representative behaviors in the hierarchy shown in Fig. 12:

- *KeepPosition*,
- *InterceptBall*, and
- *RemoveBall*.

The *KeepPosition* behavior uses six fuzzy rules to control forward, lateral and rotational motion in order to keep the robot located at its home position and oriented toward the opponent net. These rules are invoked if the robot has moved from its home position, e.g, after a defensive action. In addition, it uses a meta-rule to call the *Block* sub-behavior when this position is achieved. The resulting behavior will cause the robot to move to its home position and wait for the ball in a posture that covers about 50% of the net. The corresponding rules are given as follows:[6]

[5] An internal timer is used for this purpose. In a sense, then, the *CoverNet* behavior also includes a FSM.

[6] For sake of clarity, all the rules shown in this section are given in a slightly simplified form with respect to the actual rules implemented in the robot.

1.	IF (ForwardFromHome)	GO(Back)
2.	IF (BackwardFromHome)	GO(Slow)
3.	IF (RightOfHome)	SIDE(Left)
4.	IF (LeftOfHome)	SIDE(Right)
5.	IF (OpponentNetOnLeft)	TURN(Left)
6.	IF (OpponentNetOnRight)	TURN(Right)
7.	IF (AtHome $\wedge$ FacingOpponentNet)	USE Block()
8.	IF ($\neg$ WellLocalized)	NEED(LM2)
9.	IF ($\neg$ WellLocalized)	NEED(LM4)
10.	ALWAYS	NEED(Net2)
11.	ALWAYS	NEED(Ball)

Fig. 14. The *KeepPosition* behavior.

The intuitive meaning of the fuzzy predicates in the conditions of rules 1–7 should be clear from the drawing in Fig. 11 above. (The actual fuzzy sets used to implement these predicates are not important here.) The meaning of the linguistic labels used in the TURN rules is as shown in Fig. 4 above, the meaning of the other linguistic labels is similar. Note that if no rule for a given control variable is applicable, that variable is given a default "no-motion" value.

In addition to motion control rules, this behavior incorporates perceptual rules 8–11. In order to accurately measure the home position, the robot needs to observe the two opposite landmarks (LM2 and LM4). Rules 8 and 9 say that these should be re-acquired if the self-localization is deteriorating. Rules 10 and 11 say that information about the opponent's net and the ball is always needed.

In contrast to the *KeepPosition* behavior, the *RemoveBall* behavior only uses sub-behaviors to perform its task. This behavior is called when the ball is close to the robot and the robot is positioned on the line between the ball and the net (an effect produced by the *InterceptBall* behavior). It consists of the meta-rules shown in Fig. 15.

1.	IF ($\neg$ BallInFront)	USE Face(Ball)
2.	IF ($\neg$ BallNear)	USE GoTo(Ball)
3.	IF (BallNear $\wedge$ BallInFront $\wedge$ $\neg$ CloseToNet)	USE Kick(Ball)
4.	IF (BallNear $\wedge$ BallInFront $\wedge$ CloseToNet)	USE PushBall()

Fig. 15. The *RemoveBall* behavior.

The first two rules are used to get possession of the ball. Rule 3 and 4 are used to send the ball away from the robot, hopefully out of the penalty area. The choice between these two rules depends on the position of the robot with respect to its own net. If the robot is close to its net, it chooses the *Push* behavior, which will not send the ball very far away but does not incur the

risk of a self-score if, for instance, the ball bounces off the legs of another robot. If the robot is far from its net, it can safely kick the ball away. This strategy was chosen after observing the risk of self-scores in our experiments. Note that no perceptual rules are used in this behavior, since all the relevant perceptual needs are asserted by the sub-behaviors.

Our last example is the *InterceptBall* behavior. This behavior combines the use of control rules to directly control the robot's motion and the use of meta-rules to call sub-behaviors. The purpose of this behavior is to keep the robot on the line between the ball and the net at a fixed distance from the latter. The rules for this behavior are shown in Fig. 16.

1.	IF (BallOnRight)	TURN(Right)
2.	IF (BallOnLeft)	TURN(Left)
3.	IF (BallOnRight $\wedge$ AtRightDistance)	SIDE(Right)
4.	IF (BallOnLeft $\wedge$ AtRightDistance)	SIDE(Left)
5.	IF (Aligned)	USE RemoveBall()
6.	IF ($\neg$ Aligned)	USE KeepDistanceToNet()
7.	IF ($\neg$ WellLocalized)	NEED(LM2)
8.	IF ($\neg$ WellLocalized)	NEED(LM4)
9.	ALWAYS	NEED(Ball)

Fig. 16. The *InterceptBall* behavior.

The AtRightDistance fuzzy predicate is true is the distance between the robot and the net is the correct one (see Fig. 11). Aligned is true if the robot is on the line between the ball and its own net. The alignment motion is obtained by a combination of *KeepDistanceToNet* and of the basic control rules 1–4. As in the case of the *FaceOpponentNet* behavior, we prefer to implement alignment directly instead of using the *CrabToBall* and *FaceBall* sub-behaviors since the latter are tuned to provide accurate but slow motion. Finally, when the robot is aligned with the ball the *RemoveBall* behavior is activated to send the ball away.

The *InterceptBall* behavior needs to know the position of the ball and the location of the robot itself. The latter is used to establish the position of the net, which is behind the robot and therefore is not directly observable. The perceptual rules 7–9 express these needs.

6.5 Critical Assessment

The *GoalKeeper* behavior is probably one of the most complex reactive robot behaviors reported in the literature to this date. It involves the use of navigation, manipulation, and perceptual actions in a highly dynamic and unpredictable environment. The rule-based approach and hierarchical organization allowed us to design, implement and test this behavior with a limited

amount of effort. The full *GoalKeeper* behavior has been decomposed into 17 behaviors, which invove a total of 78 fuzzy rules (including those in the basic behaviors) plus 15 perceptual rules. The development of these rules required about 4 weeks of work by one person. This behavior was used in the RoboCup 2000 and 2001 competitions, resulting in a satisfactory performance.

In our experience with the *GoalKeeper* behavior we have noticed a number of positive and negative points of our approach to building complex behaviors. The main positive ones are as follows.

- The use of fuzzy "if-then" rules made it easy to design and tune several motion strategies involving complex combinations of forward, lateral and rotational motion.
- The hierarchical structure greatly simplified the top-down design process, by allowing us to first concentrate on higher-level behaviors that implement a given strategy, and then on lower-level behaviors that realize the necessary movements.
- The hierarchical structure also simplified the bottom-up tuning and debugging, by allowing us to first focus on the tuning of simple individual behaviors, and then debug the way in which these are combined.
- Behaviors are modular, which simplifies the design of new complex behaviors once a sufficient set of simpler ones is available; for instace, we could build a *GoalKeeper* based on a different strategy by combining the behaviors described in the last section in a different way.
- The use of reactive rules to perform behavior arbitration allowed us to concentrate on *what* should be done in each situation, while ignoring *in what sequence* the events will happen.
- Perceptual rules of the form (11) provided an easy and effective mechanism to orient perception according to the needs of the controller.

As for the observed limitations, the main ones are:

- The tuning of the fuzzy rules used in the behaviors has to be made by trial & error; this is time consuming, and there is no guarantee that the best tuning has been achieved.
- There is currently no way to connect fuzzy predicates which are defined in different behaviors but which should logically related; for example, the *GoToBall* and the *KickBall* behaviors both define internally a predicate called CloseToBall; if these two predicates are tuned differently, then *GoToBall* might stop in front of the ball at a distance which is too large for the *KickBall* behavior to kick it.
- Dealing with potential conflicts between behaviors can be tricky; for instance, two sub-behaviors might suggest drastically different control actions in some situation; in this case, we must make sure that the caller behavior never calls both sub-behaviors simultaneously in this situation; however, there is no systematic way to identify such critical situations.

- The mechanism to express perceptual needs is somewhat limited; in particular, we can specify how much we need information about a given object, but not how often this information should be updated; this limitation should be easy to fix.

In addition to these general considerations, it is interesting to note a few specific limitations in the current *GoalKeeper* behavior. First, the performance of the goal keeper strongly depends on the quality of the perceptual data. Although the fuzzy rules tolerate some noise in the input data thanks to the smooth switching between different rules, bad decision can still be taken in case of transient large errors (e.g., a bad estimate of the ball position) or persistent smaller errors (e.g., inaccurate self-localization). Probably the only solution to this problem is to improve the quality of the perceptual data.

A second limitation is related to our overall strategy. The actions performed in the "Play" mode rely on the assumption that the goal keeper robot is somewhere between the net and the ball. If the ball ends up between the net and the robot, then our strategy (try to reach a location on the line between the ball and the net) might result in hitting the ball and causing a self-score. The solution to this problem would be to extend our strategy.

As a third limitation, the *GoalKeeper* behavior only looks at the current situation, and does not try to make any prediction. In order to improve the performance, especially in a fast game, the goal keeper should be able to predict likely future state, reason about them, and take actions that prevent possible future scores. This should probably be done using the Reactive Planner (Fig. 3), which include mechanisms for prediction and short-term planning.

Finally, the *GoalKeeper* behavior currently does not take into account the position of the other robots in the field. The introduction of this element would considerably increase the number of possible types of situation that have to be considered, calling for an even more complex behavior.

We speculate that the *GoalKeeper* behavior presented here is close to the frontier of complexity that can be managed by using reactive rules and a hierarchical organization: more complex behaviors would probably need the use of state-based systems to reason about the past states (e.g., a FSM) or the future states (e.g., the Reactive Planner) or both. However, these systems could still use complex fuzzy behaviors as atomic actions. In fact, this is how the player robots have been implemented in Team Sweden: the Reactive Planner decides between high-level actions like *GoBehindBall* or *GoToBall*, and these actions are implemented by complex reactive behaviors in the HBM. Making high-level actions available to the planner has two advantages: to reduce the complexity of the planner since this only has to plan at a coarse level of granularity; and to alleviate the need for careful monitoring and replanning since behaviors already provide a good deal of reactivity to unexpected events.

7 Conclusions

The hierarchical approach presented in this chapter greatly simplifies the design of complex reactive behaviors. In this approach, design, implementation and debugging are done in a modular and incremental way based on heuristic knowledge. We have shown how we have used this approach to build a full set of navigation, ball manipulation and perceptual behaviors for a team of soccer playing four-legged robots. These behaviors include particularly complex ones, like the *GoalKeeper* behavior analyzed in the last section. We believe that the use of fuzzy rules and hierarchical organization are the keys factors that allowed the behavior designer to manage this complexity. In particular, the hierarchical organization answers the criticism of non-scalability often moved to fuzzy control techniques [21–23].

Our approach has been used in several other platforms and domains besides the ones described in this work, including wheeled robots for office navigation [3,13] and a robot manipulator performing pick-and-place tasks [24]. In all cases, the use of fuzzy rules and of CDB to define behavior combination strategies provided ease of design and high reactivity to unexpected events.

The critical assessment in the last section revealed several points in our approach that still need to be improved. Of these, the most critical one is perhaps the development of principled techniques to simplify the tuning and testing of the fuzzy rules. Since unpredictable domains like RoboCup are hard or impossible to model, we cannot hope to use classical design and validation tools for this. Instead, we are currently considering the use of statistical simulation techniques to estimate the properties of a set of behaviors.

Acknowledgements

The work reported in this chapter was partly supported by the Swedish KK Foundation. Tom Duckett provided useful comments on a draft of this chapter. Our warmest thanks to all the past and present colleagues of Team Sweden, listed at http://www.aass.oru.se/Agora/RoboCup/.

References

1. Brooks, R.A. (1986) A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, **2**, 1, 14–23
2. Arkin, R.C. (1998) Behavior-Based Robotics. MIT Press, Cambridge, MA
3. Saffiotti, A., Konolige, K., and Ruspini, E.H. (1995) A multivalued-logic approach to integrating planning and control. Artificial Intelligence, **76**, 1-2, 481–526. Online at http://www.aass.oru.se/~asaffio/
4. Driankov, D. and Saffiotti, A. (2001) Fuzzy Logic Techniques for Autonomous Vehicle Navigation. Physica-Springer Verlag, Berlin, DE. Online at http://www.aass.oru.se/Agora/FLAR/

5. Saffiotti, A. Team Sweden web site. http://www.aass.oru.se/Agora/RoboCup/
6. Sony Corporation. Entertainement robot AIBO. http://www.aibo.com/
7. Saffiotti, A. The Thinking Cap. http://www.aass.oru.se/~asaffio/Software/TC/
8. Saffiotti, A. and LeBlanc, K. (2000) Active perceptual anchoring of robot behavior in a dynamic environment. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), San Francisco, CA, 3796–3802. Online at http://www.aass.oru.se/~asaffio/
9. Buschka, P., Saffiotti, A., and Wasik, Z. (2000) Fuzzy landmark-based localization for a legged robot. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Takamatsu, Japan, 1205–1210. Online at http://www.aass.oru.se/~asaffio/
10. Johansson, S. and Saffiotti, A. (2001) Using the Electric Field Approach in the RoboCup domain. In: Proc. of the Int. RoboCup Symposium, Seattle, WA. Online at http://www.aass.oru.se/~asaffio/
11. Ruspini, E.H. (1991) Truth as utility: A conceptual synthesis. In: Proc. of the 7th Conf. on Uncertainty in AI, Los Angeles, CA, 316–322
12. Rescher, N. (1967) Semantic foundations for the logic of preference. In: The Logic of Decision and Action (ed. Rescher, N.). Univ. of Pittsburgh Press, Pittsburgh, PA
13. Saffiotti, A., Ruspini, E.H., and Konolige, K. (1993) Blending reactivity and goal-directedness in a fuzzy controller. In: Proc. of the 2nd IEEE Int. Conf. on Fuzzy Systems, San Francisco, CA, 134–139. Online at http://www.aass.oru.se/~asaffio/
14. Saffiotti, A. (1997) The uses of fuzzy logic in autonomous robot navigation. Soft Computing, **1**, 4, 180–197. Online at http://www.aass.oru.se/~asaffio/
15. Payton, D.W. (1986) An architecture for reflexive autonomous vehicle control. In: Procs. of the IEEE Int. Conf. on Robotics and Automation, San Francisco, CA, 1838–1845
16. Khatib, O. (1986) Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research, **5**, 1, 90–98
17. Bloch, I. and Hunter, A., editors (2001) Special issue on data and knowledge fusion. International Journal of Intelligent Systems, **16**, 10
18. Yen J. and Pfluger, N. (1995) A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation. IEEE Trans. on Systems, Man, and Cybernetics, **25**, 6, 971–978
19. Henzinger, T.A. (1996) The theory of hybrid automata. In: Proc. of the 11th Symp. on Logic in Computer Science (LICS), New Brunswick, New Jersey, 278–292
20. Ballard, D.H. (1991) Animate vision. Artificial Intelligence, **48**, 57–87
21. Berenji, H., Chen, Y-Y., Lee, C-C., Jang, J-S., and Murugesan, S. (1990) A hierarchical approach to designing approximate reasoning-based controllers for dynamic physical systems. In: Proc. of the 6th Conf. on Uncertainty in AI, Cambridge, MA, 362–369
22. Raju, G.V.S., Zhou, J., and Kisner, R.A. (1991) Hierarchical fuzzy control. Int. J. Control, **54**, 5, 1201–1216
23. Berenji, H.R. (1994) The unique strength of fuzzy logic control. IEEE Expert, August, page 9. Response to Elkan's "The paradoxical success of fuzzy logic," same issue
24. Wasik, Z., and Saffiotti, A. (2002) A fuzzy behavior-based control system for manipulation. Submitted, manuscript available from the authors