Available online at www.sciencedirect.com
SCIENCE DIRECT
International Journal of Approximate Reasoning
41 (2006) 110–127
INTERNATIONAL JOURNAL OF APPROXIMATE REASONING
www.elsevier.com/locate/ijar

Concepts and fuzzy models for behavior-based robotics ☆

Andrea Bonarini *, Matteo Matteucci, Marcello Restelli

Politecnico di Milano Artificial Intelligence and Robotics Lab, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milano, Italy

Received 1 January 2004; received in revised form 1 May 2005; accepted 1 June 2005
Available online 27 September 2005

Abstract

In this paper, we propose a modeling paradigm that uses fuzzy sets to represent concepts on which control modules of a behavior-based autonomous robot operate. The primitives defined in the modeling paradigm are expressive enough to represent the knowledge needed by planning, coordination, and reactive control of a multi-robot control system. At the same time, it provides a well-founded tool to represent in a compact way the data interpretations needed to reason effectively about what is happening in the world and what is desired to happen. This modeling paradigm makes the design of behavior, planning, and coordination modules easy, since its primitives are simple and expressive. Moreover, it provides a sound framework to deal with uncertainty in sensing and world modeling.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Fuzzy behaviors; Autonomous agents; Fuzzy world modeling; Multi-agent; Behavior-based robotics

☆ This research is partially supported by the project PRIN2003, co-funded by the Italian Ministry of University and Scientific and Technological Research.
* Corresponding author. Tel.: +39 02 2399 3525; fax: +39 02 2399 3411.
E-mail addresses: bonarini@elet.polimi.it (A. Bonarini), matteucci@elet.polimi.it (M. Matteucci), restelli@elet.polimi.it (M. Restelli).
URL: http://www.elet.polimi.it/res/air (A. Bonarini).

0888-613X/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.ijar.2005.06.017

1. Introduction

Since some years, most of the architectures of autonomous robots integrate the planning activity, which provides goals for the robot, with behavior-based reactivity, implemented by simple and fast control modules [1,2]. In designing this kind of hybrid architectures, most of the issues arise from the connection between the conceptual and physical level representations used respectively in the deliberative and reactive components of the system [3]. Although this kind of hybrid architectures is now common, only few efforts have been done to formalize, unify, and optimize the underlying knowledge representation model in order to seamless integrate all the modules.

In this paper, we present our approach to knowledge modeling for autonomous robots, aimed at providing a common framework to represent all the knowledge needed by the modules that participate in control and coordination. We define the conceptual aspects needed to represent this type of knowledge and we introduce fuzzy sets as a tool to support this representation. This fuzzy conceptual representation is used by all the modules of our control architecture (see Fig. 1): *MAP* (*Map Anchors Percepts*) [4] that integrates data from sensors and other data sources building an internal representation of the world, *BRIAN* (*Brian Reacts by Inferential ActioNs*) [5] that manages the behaviors and implements all the reactive functionalities of our system, and *SCARE* (*Scare Coordinates Agents*

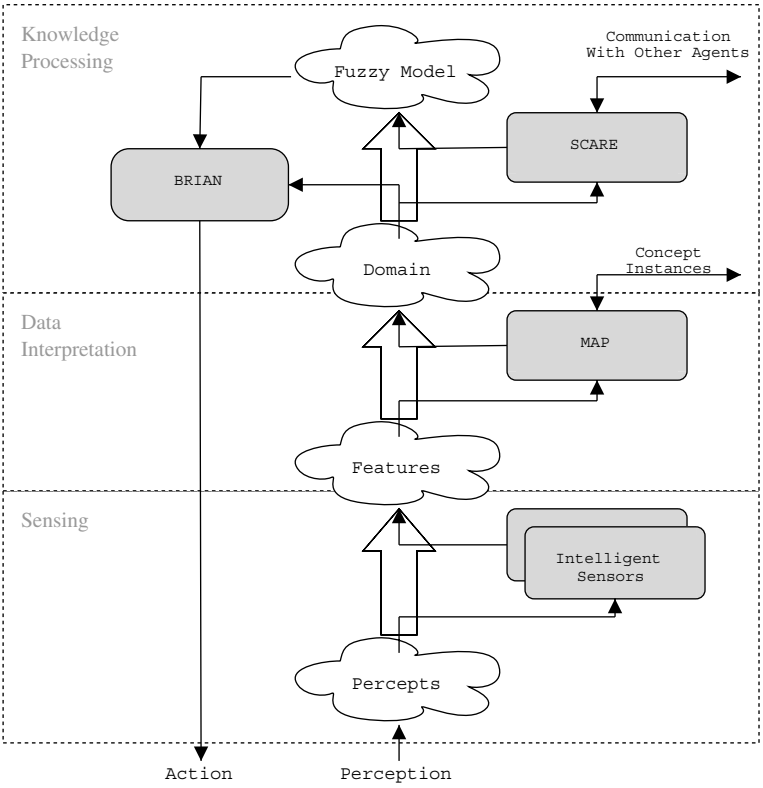

Fig. 1. The reference knowledge model and the architecture modules.

in Robotic Environments) [6] that coordinates the agent's behaviors and plans its activity. In fact, a uniform knowledge representation makes it possible a coordinated design of the modules, and an efficient exchange of information.

We have applied our model both in service and edutainment robotics. Here, we take this last just as an explicative environment. We focus on robot soccer playing, since a Robocup [7] match provides a rich and challenging environment where our approach clearly shows its effectiveness. Robocup is an initiative bringing together thousands of researchers every year, competing on common test-benches that require the implementation of effective autonomous robots. There are different competitions, and we describe here the results that we have obtained in the Medium Size League of real robots. Here, each robot can have a maximum width and length of about 50 cm. It should be able to play soccer autonomously, by interacting in teams of 4–6 members, in a field sized 8×12 m. The ground is a green carpet, the ball is reddish, all robots should have a "mostly" black body and a cyan or purple marker. The ball and the robots can move at more than 4 m/s. At the first Robocup World Championships (Paris-98 [8], Stockholm-99 [7]) the main challenge was to detect the relevant elements on the field (ball, opponents, goals) and decide reasonable actions in accordance. In the last World Championships [9–11] some non-trivial behaviors have been seen, and the issue of designing complex behaviors emerged as the really challenging issue. We participate to Robocup with the Milan Robocup Team (see Fig. 2). All our robots, although mechanically different from each other, are equipped by an omnidirectional vision sensor [12], which provides every 70 ms information about the position of the other robots, the ball, and the relevant elements of the field. This information is fused by the MAP module with analogous information coming from other robots through the radio Ethernet link. The detailed presentation of the complex sensor fusion process is beyond the scope of this paper and has been given elsewhere [4].

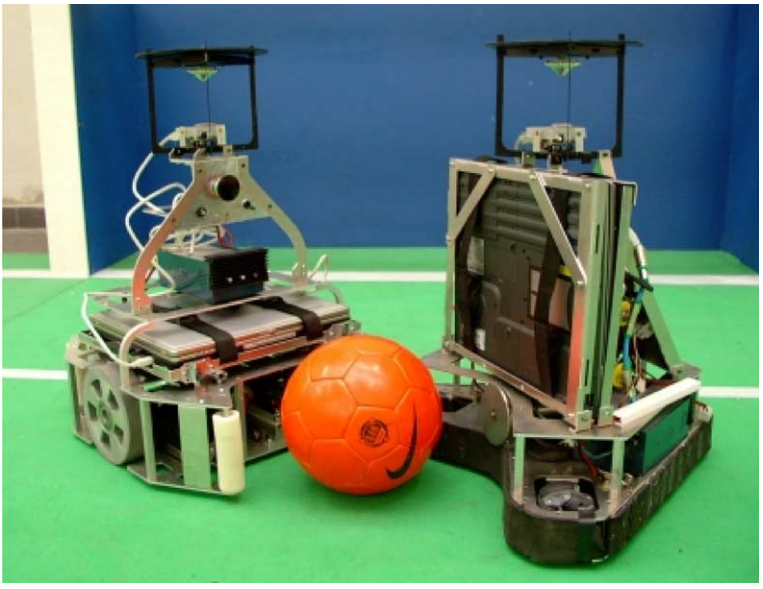

Fig. 2. Robaldo and Recam, two MRT players.

In Section 2 we introduce our knowledge model build and MAP, the module in charge of building the world model. Section 3 describes BRIAN and how this module uses the conceptual model managed by MAP. The SCARE use of the model and its interaction with BRIAN through the use of fuzzy predicates are presented in Section 4.

2. MAP: concepts and the fuzzy world model

In a robotic environment, agents have to interact with several *physical objects* and this interaction is typically implemented as a perception-action loop. Robots are equipped with sensors that perceive physical characteristics of the environment and they use these *percepts* to build an internal representation of the environment. Once this internal representation is formed, it is possible to use it for deliberative or reactive processing which produces actions to be executed in the environment. In this section, we do not go into the theoretical details related to the problem of matching percepts with the corresponding semantic meaning of the physical objects they represent (i.e., the *Symbol Grounding Problem* [13]). Instead, we focus on the knowledge representation we use to face the problem of creating, and maintaining in time, the connection between symbol-level and signal-level representations of the same physical object (i.e., the *Anchoring Process* [3]).

MAP implements a two-stage process for creating the agent internal representation of the environment: first of all, percepts are used to instantiate a real-valued conceptual model of the environment, and, then, this conceptual model is interpreted in terms of fuzzy predicates to be used to reason in coordination and control. As shown in Fig. 1, percepts are processed by sensing modules (i.e., smart sensors) to produce high level *features*. Features referring to a specific physical object are collected in the same internal representation, referred to as its *perceptual image*; this can be seen as the instance of a *concept*. In a formal way, a concept C is described by a set of properties defined as tuples in the form:

$$p \triangleq \langle label, \mathbb{D}, r \rangle, \tag{1}$$

where *label* denotes the property name, $\mathbb{D}$ is the set of all the possible values for that property given a specific representation code, e.g., for the colors we can use either the set of symbols $\{red, green, blue, yellow, magenta, black, \ldots\}$, or the RGB space $\mathbb{N}^3_{[0,255]}$, or a fuzzy classification of this; r represents a restriction of the domain $\mathbb{D}$ for the property in the specific concept. Depending on the concept and on the specific application domain, a property can be classified as *substantial* or *accidental*, respectively $\mathtt{S}$ and $\mathtt{A}$ in Eq. (2) that defines a concept as a set of qualified properties.

$$C \triangleq \{\langle p, \mathtt{x} \rangle\} : \mathtt{x} \in \{\mathtt{S}, \mathtt{A}\}. \tag{2}$$

For instance, in Robocup we can use color and other properties to describe *Ball* concept.

$$\begin{aligned} Ball = {} & \langle \langle color, \{red, green, blue, yellow, cyan, magenta\}, red \rangle, \mathtt{S} \rangle \\ & \times \langle \langle \ldots, \ldots, \ldots \rangle, \ldots \rangle. \end{aligned} \tag{3}$$

In the above example, the *Ball* concept has a substantial property *color* which usually ranges on the mentioned set of values, restricted for this particular concept to be *red*.

Substantial properties characterize the immutable part of a concept: for a given object, their values do not change over time, and they can be used for object recognition since they explain the essence of the object they represent. *Accidental* properties do not characterize a

concept: their values are specific to each conceptual instance, and they can vary over time. They cannot be used for object recognition, but are the basis of instance formation, tracking and model validation. For instance, for a Robocup ball, the *color* property reported above is a substantial property, since only red balls are allowed, however, the *position* of the ball

$$\langle position, \Re \times \Re, ([-6+6],[-3+3])\rangle \tag{4}$$

is an accidental property. During robot activity, data from sensors are matched against concepts in the conceptual model, and, when enough evidence is collected, a concept instance is generated, which inherits from its concept also property values eventually not detected by sensors [4].

According to this concept-based knowledge representation, a property can be either directly perceived, and thus related to a set of high level features coming from sensors, or it can be derived from other properties by inference or computation. This approach allows to draw from properties specific to a concept additional information about the perceptual image, or to infer from the available features characteristics not perceived. Using concepts it is possible to describe both domain specific and general knowledge used by an agent during its activity. To explain how this knowledge is used, we introduce the notion of model $\mathscr{M}$: given D as the set of the known domains, a model $\mathscr{M}_d$ is the set of all the concepts known by the agent referring to the specific domain $d \in D$, linked by relationships—structural (e.g., generalization, and specialization) and domain specific (e.g., colors and landmark in structured environments). A relationship between concepts can represent:

(1) *Constraints*: they must be satisfied by concept instances to include these in the model (e.g., all the red objects on the field are balls, as from the current Robocup rules).
(2) *Functions*: they generate property values for a concept from property values of another (e.g., the fact that a zone of the field is free is computed from the properties stating the position of the detected objects).
(3) *Structures*: structural constraints to be used to reason about classification and uncertainty (e.g., generalization and specialization).

The definition of concepts at different levels of abstraction is important to support the classification of percepts, and the instantiation of concepts. Concepts are organized in *ontologies*, which may be partially defined independently from the specific application, at least up to a certain abstraction level. For instance, it is possible to give general properties of *movable objects*, as a concept specializing the more general concept *objects*, and in turn specialize it in *mobile robots* and *human beings*. Such general concepts may also participate in general inferential processes, which allow, for example, to infer that people and mobile robots usually stay on the ground, information useful to compute distances from images. When facing a specific application, it is then possible to complement the general ontology with application-specific information; for instance, it is a substantial property for balls to have a spherical shape, but in a Robocup application we also know that the ball is red and has a given diameter. In case of uncertainty, it is often more reliable to instantiate a more general perceptual image. For example, again in a Robocup application, robots belonging to different teams wear markers of different colors, which may be detected with some uncertainty; therefore, a set of features may be aggregated more reliably as an instance of *robot* than as an instance of *opponent robot*.

During the anchoring process, MAP carries out a three-step process:

(1) *Classification*: each perceived object (i.e., set of features produced by a sensor) is classified according to the conceptual model, thus producing *conceptual instances*.
(2) *Fusion*: the conceptual instances produced by different sources (sensors or robots) that can be associated to the same physical object are merged.
(3) *Tracking*: the conceptual instances generated by the current perceptual inputs are used to update the values of the conceptual instances that are inside the world model.

We assume that smart sensors produce sets of features, where each feature is a triple: $\langle label, v, \rho \rangle$. The *label* of a feature is its name, v is its numerical value, and ρ is its reliability value, i.e., how the data is assumed to be reliable given the specific sensor and the acquisition situation. The classification process produces instances of concepts, by matching features provided by the sensors with the properties of objects in the conceptual model. The reliability of the concept instance is computed as a *min* of the reliability of the features related to substantial properties. The properties have a reliability value equal to the reliability of the corresponding feature. The fusion process merges conceptual instances, thus obtaining new conceptual instances whose reliability is computed as the *max* of the reliability of the same attributes in the different instances. The *perceived instances* produced by the fusion module are used to update the values of the instances in the world model. The reliability of these instances is updated as follows:

- if the perceived instance does not match any instance in the world model, a new instance is created with the ρ values of the perceived instance;
- if an instance in the world model does not match any perceived instance, the reliability values of its attributes are exponentially decreased by a coefficient $\gamma \in (0, 1)$;
- if a perceived instance matches an instance in the world model, their reliability values are composed by an arithmetic mean.

In this way, the reliability of the instances in the world model follows the reliability of the perceived instances, and when an object is no longer perceived, the reliability of the related instance decreases until it is thrown out of the world model. This is needed in dynamical environments, where we cannot expect that data remain the same when we cannot fetch them. The decay is proportional to a constant γ defined by the user, which can be defined by considering the relative rate of change of the environment with respect to the data acquisition rate.

Once the conceptual model $\mathscr{M}$ relative to the agent's established knowledge has been instantiated, we have an internal representation of the environment on which it is possible to evaluate logical predicates, apply inference, or execute behavior control modules. We call this internal representation *Domain* and we will denote it with $\mathscr{D}$. From the design point of view, the presence of a reference model makes it possible a modular design, with people having different competence interacting on the same knowledge. People working on sensors would know that they should produce certain information in a given format, and people working on control could rely on that. Moreover, it is also possible to apply predefined libraries on the application-specific knowledge, thus improving software reliability, and reducing design efforts.

2.1. Fuzzy predicates

We represent features as fuzzy predicates. The *degree of truth* of a fuzzy predicate $P(x)$ ranges on $[0,1]$, and it is computed as the membership of the value of its argument $\bar{x}$ to a fuzzy set F_P. F_P is defined by a *membership function* $\mu_{F_P}(x)$ ranging on the universe U_x of the ordinal variable x. $\mu_{F_P}(x)$ models the semantics of the fuzzy set and of the corresponding predicate. In Fig. 3, you may see three typical membership functions for three fuzzy sets defined on the variable d.

We adopt fuzzy predicates to represent aspects of the world, goals, and information coming from other agents. They are represented by a *label* λ, its *truth value* μ_λ, computed by evaluation of the membership to a fuzzy set, corresponding to the label, of a composition of concept instance properties, and a *reliability value* ρ_λ to take into account the quality of the instance. For example, we may have a predicate represented as

$$\langle ObstacleInFront, 0.8, 0.9\rangle, \tag{5}$$

which can be interpreted as: "It is quite true ($\mu_\lambda = 0.8$, derived from the fuzzification of the composition of real-valued properties such as the distance between the robot and an object in front of it, computed from the relative positions) that there is an obstacle in front of the robot, and this statement has a reliability quite high ($\rho_\lambda = 0.9$), due to the reliability of the sensing conditions".

We consider ground and complex fuzzy predicates. *Ground fuzzy predicates* range on concept instance properties directly available to the agent through $\mathscr{D}$, and have a truth value corresponding to the degree of membership of instance properties to labeled fuzzy sets. The reliability of sensory data is provided by the anchoring process basing on percept analysis, while goal reliability (in case the predicate represents a goal) is stated by the planner. A *complex fuzzy predicate* is a composition of fuzzy predicates obtained by fuzzy logic operators. Complex fuzzy predicates organize the basic information contained in ground predicates into a more abstract model. Their truth values are computed by the classical fuzzy operators. The fuzzy AND is implemented as a T-norm (in our case, the min function), while the fuzzy OR is implemented as a T-conorm (here, the max function). The selection of these specific T-norms does not affect the generality of the proposed approach, and has been done in our example because we prefer taking into account the most critical predicate instead of the contribution of all. The fuzzy NOT operator is implemented as the

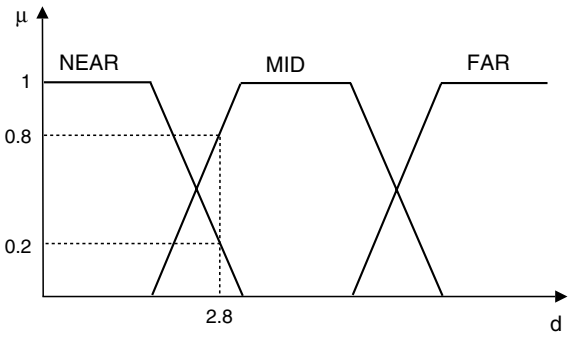

Fig. 3. An example with three fuzzy sets.

Lukasiewicz complement: the degree of truth of the predicate $\neg P(x)$ is defined as $\mu_{\neg P}(x) = 1 - \mu_P(x)$. In Robocup, for instance, we can model the concept of ball possession by the *BallOwner* predicate, defined by the conjunction of the ground predicates *BallNorth* and *BallInKick*, respectively deriving from the fuzzification of the direction and distance properties of the ball concept instance in $\mathscr{D}$.

To model the reliability of the data and of the modeling process, also predicates are associated to a reliability value ρ. Since ground fuzzy predicates are computed directly on data contained in the world model, their reliability is equal to the reliability of the datum on which it is evaluated.

Reliability values associated to predicates which are joined by a binary fuzzy operator are composed by the T-norm *min*. The fuzzy NOT operator does not affect the reliability associated to the predicate.

In order to understand the difference between reliability and truth value, let us consider the following example. The robot has perceived a ball at a distance of 2.80 m with a reliability of 0.7, and this means that, according to the fuzzy sets displayed in Fig. 3, we have the following predicates with the related degrees of truth:

- NEAR(ball): $\mu_{\mathrm{NEAR}}(ball) = 0.2$,
- MID(ball): $\mu_{\mathrm{MID}}(ball) = 0.8$,
- FAR(ball): $\mu_{\mathrm{FAR}}(ball) = 0.0$.

Although the degrees of truth of the predicates are different (since they depend on the membership functions that define each fuzzy set), they all have the same reliability value, which depends on the quality of the data used to evaluate them. Since, in the above example, all the predicates are related only to the same percept, the reliability value of the predicates is the same as that of the datum.

3. BRIAN: the behavior management system

Especially in dynamic environments, the main framework to design robot control is the so-called behavior-based architecture [1]. In such a framework, the robot controller is obtained by the implicitly cooperative activity of behavioral modules. Each module operates on a small subset of the input space implementing a relatively simple mapping from sensory input to actions; the global behavior of the robot arises from the interaction among all these modules. One of the major problems in behavior-based robotics is the design of this interaction, often predefined in terms of inhibitory relationships or vector composition of the module output.

3.1. Behavior interaction in BRIAN

In our behavior management system *BRIAN*, integration and coordination among behavior modules are achieved using two sets of fuzzy predicates associated to each module: CANDO and WANT conditions (see Fig. 4). In *BRIAN*, we face the issue of controlling the interaction among modules by decoupling them with these conditions, described in terms of fuzzy predicates, and evaluated over internal state, environmental situation present in $\mathscr{D}$, and goals generated by the strategic module *SCARE* described in Section 4.

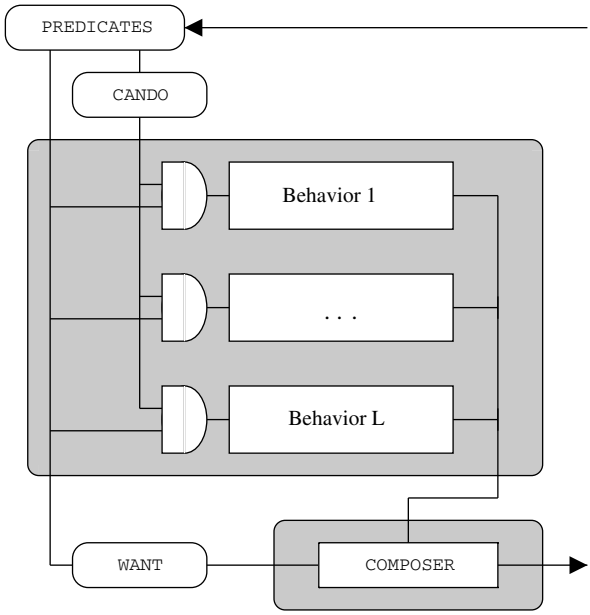

Fig. 4. A schematic view of the behavior management system.

CANDO conditions are used to decide whether a behavior module is appropriate to the specific situation: if they are not verified, the behavior module activation does not make sense. The designer has to put in this set all the fuzzy predicates which have to be true, at least to a significant extent, to give sense to the behavior activation. For instance, in order to consider to kick a ball into the opponent goal, the agent should have the ball control, and it should be oriented towards the goal. This set of conditions has a twofold aim: decoupling behavior design and increasing the computational efficiency of the behavior management system.

WANT conditions represent the motivation for an agent to execute a behavior. They may be related either to the environmental context (e.g., *BallInFront*, "the ball is in front of me"), or to strategic goals (e.g., *ScoreGoal*, "I have to score a goal"). Composition of the actions proposed by behavior modules active at the same time is implemented by the WANT conditions, which represent the opportunity of executing them in the specific context.

The use of these two different sets of conditions allows the designer to set up a dynamic network of behavior modules. This is a sensible improvement with respect to usual behavior-based architectures; we do not have a complex, predefined, interaction schema that has to take into account all possible situations. At any instant, the agent deduces from a high level description of the current situation that it could play only a restricted set of behavior modules (i.e., those enabled by the CANDO conditions), and that it has to select/merge the behaviors coherent with its present motivations.

In *BRIAN*, each behavior module receives data input from MAP and provides output commands to be issued to the environment. We do not make any hypothesis about the implementation of a behavior module, it can be considered as a mapping from input variables to output variables:

$$B_i : I_i \mapsto A_i \quad I_i \subseteq I = \{i_j\}, \quad A_i \subseteq A = \{\mathbf{a}_k\}, \tag{6}$$

where I_i is the specific set of input variables for behavior module i, and A_i is the set of its actions.

In the implementation that we are presenting, behavior modules are realized as fuzzy logic controllers. A set of fuzzy rules match a description of the situation given in terms of fuzzy predicates, and produces actions for the actuators by composing with a T-conorm the output proposed by each rule. Other implementations are possible, for instance neural network modules, mathematical models, precompiled universal plans, etc. We have decided to have fuzzy logic controllers to maintain readability of the control strategy and robustness. Notice that the semantics of the predicates in fuzzy rules is complementary to those of CANDO and WANT conditions. Antecedents of fuzzy rules are matched to identify the situation in which to select the action expressed in the consequent, whereas CANDO conditions are used to select which behavior module (a controller) has to be considered for activations. Consequents of the rules are composed in the traditional fuzzy control way to produce the action proposed by the controller in the current situation, and this action will be then weighted by the WANT conditions to reflect motivations to apply a behavior.

3.2. Behavior reliability

In this section we present our model to manage how the reliability of input data is propagated to each behavioral module. The different semantics of CANDO and WANT predicates call for a different treatment of their reliability, as presented in the next paragraphs.

Behaviors are partitioned into two classes according to the truth value of their CANDOs: if the truth value μ_i^{C} for the CANDO of behavior i is above a given threshold $\bar{\mu}^{\mathrm{C}}$ the behavior is active. In our model, we take into consideration also the reliability of the CANDO predicates, and not only their truth value: among active behaviors (i.e., behaviors having $\mu^{\mathrm{C}} > \bar{\mu}$) only the ones having reliable CANDO conditions (i.e., ρ^{C} above a threshold $\bar{\rho}^{\mathrm{C}}$) are enabled behaviors B_i^{E}. This has a twofold goal: increasing computational efficiency and reduce the influence of unreliable behaviors. In real-time robotic applications where an autonomous robot has to interact with people, we do not wish its behavior to be based on unreliable data. This is a conservative choice that prefers doing nothing, instead of doing unreliable actions, and could be tuned by the choice of an appropriate $\bar{\rho}^{\mathrm{C}}$ made by the designer or even learned on-line.

Also for WANT predicates we have a matching degree μ^{W} to represent the truth value of the predicate and a reliability value ρ^{W} to represent the reliability of the predicate. Composition of actions proposed by enabled behaviors is obtained by a weighted sum that uses the matching degree μ_i^{W} of enabled behavior B_i^{E} to weight the action proposed by it. However, we prevent behaviors having unreliable WANT conditions from interfere with the final action. This is obtained by setting the weight w_i for each action proposed by behavior i to μ_i^{W} if the corresponding ρ_i^{W} is above a given threshold $\bar{\rho}^{\mathrm{W}}$ or 0 otherwise:

$$\mathbf{a} = \frac{\sum_i w_i \mathbf{a}_i}{\sum_i w_i}, \quad w_i = \begin{cases} \mu_i^{\mathrm{W}} & \text{if } \rho_i^{\mathrm{W}} > \bar{\rho}^{\mathrm{W}}, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

In *BRIAN*, also predicates that match rule antecedents have associated μ and ρ values that represent, respectively, the matching degree of the rule with the present situation and the matching reliability. The consequents of these rules are actions proposed to compose the final output of the agent. Inference in this architecture has to take into account for antecedent matching degree and reliability; thus, each proposed action inherits μ and ρ of the present situation.

The same action, proposed by different behaviors with different matching degrees, is considered by taking the maximum μ. Then, *BRIAN* sorts the so-obtained actions according to their reliability value ρ. If the reliability of an action is above a given threshold ρ_H the action is inserted in the list of proposed actions A_p. If the action reliability is below another threshold ρ_L it is not considered to be performed by the agent. If A_p does not contain any action for a possible actuation, we have decided to consider also actions, for that actuation, with a reliability value $\rho_L < \rho < \rho_H$. This means that we may accept a limited degree of unreliability to tend to have at least an action for each actuation.

3.3. An application: playing robotic soccer

In this section, we focus on a simplified version of the behavior system we are using in Robocup, to show in details how *BRIAN* works. We consider a small number of fuzzy behaviors whose composition enables the robots to play effectively in a match, fighting for the possess of the ball, avoiding other robots, kicking in the opponents' goal, keeping in field, controlling the mid-field and taking a defense behavior when the own goal-keeper has problems. The basic behaviors are: *AlignRight*, *AlignLeft*, *GoToGoal*, *Kick*, *AvoidObstacle*, *GetOffOwnArea*, *KeepInField*, *Midfield*, *Defend*.

We can identify subsets of behaviors which are intended to be activated in sequence, and their conditions (reported in Table 1) have been designed in this example to avoid interference. For instance, *AlignRight* aligns the robot coming from the right to the direction of the line between the ball and the opponent goal (all our robots have an omnidirectional vision system, so they almost always know where the ball and the goal are); *GoToBall* brings the robot on the ball when it is in the forward direction. The CANDO

Table 1
CANDO and WANT conditions for each behavior

Behavior	CANDO	WANT
AvoidObstacle	`(ObstaclePresent)`	
AlignRight	`(AND (AND (BallSeeing) (NOT (RightAligned))) (NOT (AND (BallOwner) (Aligned))))`	`(AND (OR (AlonePlayer)(ForwardRole)) (AND (NOT (ObstacleAvoiding)) (NOT (GoalNear))))`
AlignLeft	`(AND (AND (BallSeeing) (NOT (LeftAligned))) (NOT (AND (BallOwner) (Aligned))))`	`(AND (OR (AlonePlayer)(ForwardRole)) (AND (NOT (ObstacleAvoiding)) (NOT (GoalNear))))`
GoToGoal	`(AND (BallOwner)(Aligned))`	`(AND (NOT (ObstacleAvoiding)) (NOT (GoalNear)))`
KickInGoal	`(AND (BallOwner)(Aligned))`	`(GoalNear)`

conditions for *AlignRight* include the fact that the ball is visible and that the robot is not aligned nor controls it. Notice how these conditions are essential to apply this behavior. Among the WANT conditions for *GoToBall* we have that the ball should be in the forward direction. The *AlignRight* and *AlignLeft* behavior tend to make this condition true, and, when this is the case, the context is favorable to the activation of *GoToBall*. Both the behaviors cooperate to bring the agent in a position from where it can take the ball and bring it towards the goal. The *GoToGoal* behavior has among its CANDO conditions predicates representing the fact that the robot has the control of the ball and is aligned both to ball and goal: so it can be activated only when both the other mentioned behaviors have achieved their goals.

We have also defined the *AvoidObstacle* behavior to take care of the integrity of the robot: it should inhibit the other behaviors in order to handle critical situations. It is devoted to solve possible problems due to the presence of obstacles in the desired movement directions. We have designed the enabling conditions for this behaviors to implement exclusive activation. In particular, all the WANT conditions of the incompatible behaviors contain the control predicate stating that the *AvoidObstacle* behavior should not be active. In this way, if the robot has to avoid something, it does this without any interference from the other behaviors, whose action is considered as undesirable when it is active. We have recently implemented a more sound, clean (and complex) solution to design independently behaviors and give the possibility to implement priorities. We will briefly discuss it in the conclusions.

Interesting behaviors emerge from the interaction of behaviors belonging to the two sets. For instance, we have seen one of our robots dribbling a couple of opponents due to the appropriate switching between *AvoidObstacle*, *AlignRight* and *GoToGoal*, as shown in detail in the next section. The co-operation of these behaviors made the robot react, when facing an opponent, by throwing the ball aside (obtained by a fast rotation decided by the *AvoidObstacle* behavior in order to get around the obstacle) and running to catch it again (composition of *AlignRight* first and *GoToGoal* then).

A second emergent behavior is a sort of defense behavior made by the co-operation between *KeepInField*, *AvoidObstacle* and *Defend* behaviors. When the ball is close to the border line with another robot that is trying to catch it, our robot stays still, covering its goal area, watching the other robot and waiting that it gets rid of the situation, since *KeepInField* prevents the robot to go outside the field. Once the opponent brings the ball away from the border line, the defender is ready to close it on the line again, by applying a *Defend* behavior. Notice that the MSL rules state that when the ball is not moving for more than 10 s (e.g., because no robot can or wishes to reach it), then it should be removed by the referee and positioned in another position. The mentioned behavior gives also the robot the possibility to stay away from the border line and be ready to get to the new ball position, when it is re-positioned.

A third emergent behavior we mention here makes the robot catching the ball close to an opponent. It is obtained by the co-operation between *Align* and *AvoidObstacle*. When the ball is close to another robot, the switching between behaviors makes the robot to approach slowly the ball (since it is close to it) with fast and impulsive rotations due to activation of the *AvoidObstacle* behavior which tries to avoid the contact with the opponent by turning the robot body. When it is close enough to the ball, the result of the interaction between these behaviors is a sort of kick with the robot's hands that throw the ball away from the obstacle.

3.4. A detailed experiment

In this section we present in details the results obtained in a specific experiment where the robot executes a standard Robocup challenge, used in this case to show the composition of behavioral modules.

In Fig. 5(A) you can see the experimental setting of the test. The black objects are static obstacles the gray one is the ball to catch and kick in the goal. The track in the plot is taken from real data: the trajectory is projected on the field using the odometry of the robot. The robot starts in position (*a*), after 0.5 s the robot, incidentally touching the ball (position (*b*)), moves it to the position (*c*). So it has to dynamically change its behavior to face this unforeseen situation. The trajectory executed by the robot is obtained by selecting and blending different actions proposed by different behavioral units and reacting to changes in the environment, such as the variation of ball position as shown in Fig. 5(B). In this experiment, we use only five simplified basic behaviors and no planning features, to show the effectiveness of *BRIAN* (see Table 1). The predicates appearing in the fuzzy conditions are computed by evaluating fuzzy sets on data in the domain $\mathscr{D}$, and composing them. In Fig. 6, you may see the definition of some of the fuzzy sets involved in the definition of the above-mentioned predicates. In particular, we have reported the frame of cognition of the distance to the ball, and the direction of the ball.

Complex predicates are computed by composing basic ones. For instance, the *BallOwner* predicate is computed as (*AND BallNord BallInKick*), where *BallNord* comes from (*OR N1 N2*) computed from the second frame of cognition reported in Fig. 6, and *BallInKick* comes from the value of *InKick* from the distance frame of cognition. Analogous computations bring to the evaluation of all the needed predicates.

During the experiment that we are presenting, we have logged the activation level of CANDO and WANT for each behavior module (Fig. 7(A) and (B)) and the final behavior activation level coming from the combination of CANDO and WANT (Fig. 5(B)). From Fig. 7(B) you may notice that, during this experiment, the WANT conditions for *Align*

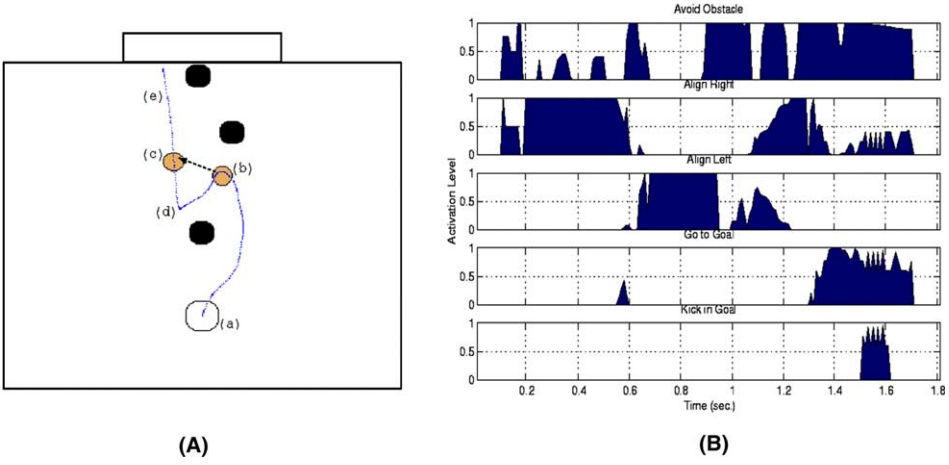

Fig. 5. The trace of robot trajectory during the test and the activation level of the behaviors.

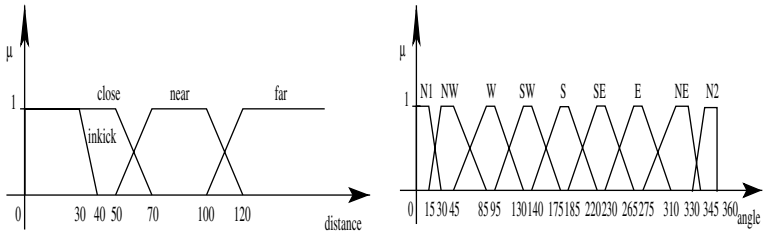

Fig. 6. The definition of some of the fuzzy sets used to compute the predicates involved in CANDO and WANT conditions of the behavior modules developed for Robocup.

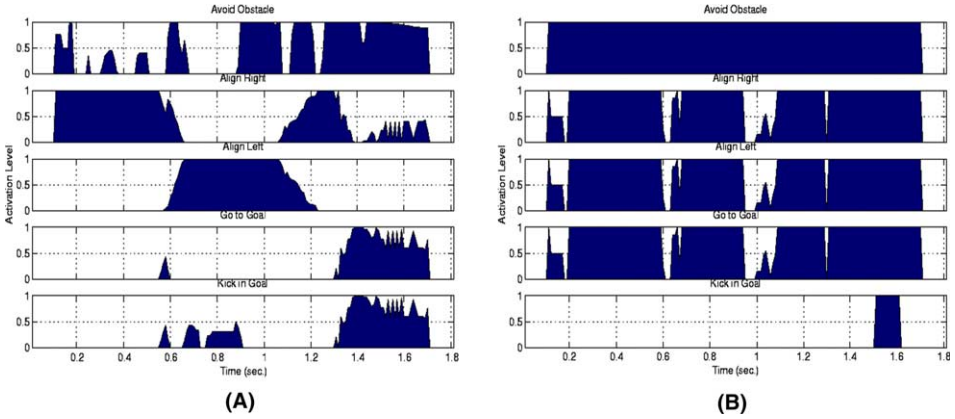

Fig. 7. The activation level of CANDO (A) and WANT (B) conditions during the test.

Left, *AlignRight* and *GoToGoal* have the same activation level, since the predicate (*AlonePlayer*) is always verified and the other conditions are the same for all the behaviors.

As you may notice from the definitions of the WANT conditions of *AlignLeft* and *AlignRight* in Table 1, it is possible to include in them also context description predicates like the actual role of the teammate (i.e., *ForwardRole*) or the explicit reference to other behaviors (i.e., *ObstacleAvoiding*). In the example, the value of *ForwardRole* has been set by the user to be always *TRUE*; in a real situation SCARE, described in the next section, sets this value according to the evaluation of current situation and the planning strategy. This is just an example of how our formalism can be used both for modeling the context and controlling the behavior blending.

4. SCARE, the coordination system

Cooperation holds a very important role in Multi-Agent System (MAS) applications. To face the typical issues of these applications, we have implemented *SCARE* [6] a general architecture for coordination in multi-robot domains. Also the knowledge model used in *SCARE* is based on the same primitives presented in Section 2 and adopted in the other modules of our architecture.

SCARE is able to deal with:

- *Heterogeneity*: when a MAS is made up of agents with different skills, our architecture exploits these differences in order to improve the overall performance.
- *Communication*: coordination policy may change according to the amount of information that can be exchanged among agents and according to network connectivity.
- *Adaptation*: in order to grant the autonomy of the system, the coordination mechanism is able to adapt its parameters in reaction to environment changes.

Using *SCARE*, the MAS application developer has to identify the macro-activities that the agents can carry out; we call *jobs* such macro-activities. Besides jobs, we have defined other macro-activities named *schemata*. A schema is a complex activity consisting of sequences of jobs that require the collaboration of two or more agents. Jobs and schemata are both *activities* and play analogous roles in the architecture.

An agent cannot be assigned to more than one job at a time, while two or more agents may carry out the same job at the same time. A job is defined by the goals that should be achieved, and each agent is free to select the behavior modules to achieve these goals according to its own capabilities.

Since the agents in the MAS can be heterogeneous, our architecture allows to specify their *skills* in order to estimate their *attitudes* towards the various activities. These attitudes toward the jobs define the *role* of an agent. Notice how this role definition is quite different from those adopted in related approaches [14,15], where the concept of role refers only to what an agent is doing in a given moment, and there is no relationship with the agent's characteristics. In our approach, the role of an agent could dynamically change as a consequence of some variation in the skills. For instance, if a robot cannot kick anymore, it may abandon the attacking role to take a more defensive one. Attitude is just one of the parameters that take part in the job assignment process.

The MAS application developer has to identify the macro-activities that the agents can carry out. To cope with uncertainty of the perception and approximate definitions, we adopt the *fuzzy predicates* introduced in Section 2.1. In this way, the states of the world model can be considered as matching a situation σ with a certain degree μ. By introducing a threshold $\bar{t}$ it is possible to define a situation by a fuzzy predicate: $\sigma = \{m \in \mathscr{M} | \mu(m) > \bar{t}\}$ $\bar{t} \in [0, 1]$.

In the job assignment process, in order to establish how much each activity is suited for the agent, we use several parameters implemented by fuzzy predicates, which operate on the domain $\mathscr{D}$:

- *cando*: define when the activity can take part in the assignment process;
- *attitude*: define how much the skills of the agent are useful for the activity;
- *chance*: define the situation where the agent has good chances to succeed;
- *utility*: define the situation where the activity is useful for the agent team;
- *success*: define the goal achievement situation for the activity;
- *failure*: define the situation where the activity should be stopped because of unrecoverable failure.

An activity terminates when the success or failure conditions are verified. If an agent is idle, a job assignment process starts. For each activity, the CANDO predicates are eval-

uated in order to reject those activities that cannot take place. For each remaining activity, utility and chance predicates, and the agent's attitude are considered, thus obtaining indicators to take the decision. Through the application of some multi-objective technique (e.g., weighted sums, goal programming, normal-boundary intersection, multi-level programming, or others), each agent gets an ordered list of activities (*agenda*). Once all the agendas are produced they must be compared in order to achieve a coordinated assignment of jobs (for details, see [6]). At the end of this process, each agent is assigned to a job.

SCARE interacts with BRIAN by sending to it the values of *coordination predicates*, which influence the behavior conditions. Coordination predicates refer to the job assigned to the agent. This information allows to activate only the behavior modules needed to execute the assigned job. At each iteration, *SCARE* also monitors the state of execution of the assigned job: when a termination condition (either success or failure) occurs, *SCARE* starts a new job assignment process, in order to identify the best activity according to the multi-agent context.

5. Conclusion

In this paper, we have presented the fuzzy cognitive model we use to integrate in a uniform framework the deliberative and reactive components of multi-agent systems. The cognitive model we propose integrates coordination, planning and reactive behaviors providing a common cognitive substratum for a team of robots where behavior modules are used as high-level macro-actions, virtually originating structured plans, defined by a flexible, multi-agent coordination system.

All the elements in the knowledge processing level are based on simple fuzzy predicates that can be easily managed, designed and adapted. The control model can be easily designed to be tuned and adapted on-line so that the team strategies and the role of robots in the control schemata can be automatically modified to face different opponent teams, and changes in robot performances [6].

5.1. Comments

Some of the most interesting issues that can be obtained by basing the robot architecture on the mentioned knowledge model are here summarized.

- *Noise filtering*: using a conceptual model of the environment it is possible to eliminate out-layers in percepts and filter in a proper way noisy data coming from sensors; this produces more reliable information for the other modules.
- *Sensor fusion*: percepts coming from different sensors, and referring to the same objects, can be fused enhancing fault tolerance and enabling on-line diagnosis.
- *Virtual sensing*: a model of the environment can be used to infer new features, not perceived by physical sensors.
- *Time consistency*: the instances in the conceptual model represent a state of the environment; it is possible to maintain and monitor its consistency in real-time; this activity can be used to learn and check models.
- *Abstraction*: the use of fuzzy predicates instead of raw data, or features, in the behavior definition gives more abstraction in designing robot behaviors, and robustness to noisy data; it also facilitates design, since gives the designer the possibility to reason

in symbolic terms. Moreover, exchanging this information is more effective for agents of a Multi-Agent System (MAS) that share the same semantics for symbols.

5.2. Related works

Our architecture follows the Saffiotti's approach [2,16] to the use of fuzzy logic in robotics [17], which tries to face the problem of designing an effective controller for mobile robots by combining goal-specific strategies to solve conflicts between multiple objectives. We keep separate the CANDO from the WANT conditions, due to their different semantics, while in [2] they are put together in the *desirability* parameter. Keeping separated these conditions is important for design flexibility and has cognitive plausibility, which makes the design process closer to the designer way of thinking. Our behavior architecture is quite different from the Brooks' one [18]: our behavior management system works on fuzzy predicates, thus achieving the possibility of coordinating the concurrent execution of several behaviors. We consider the adoption of CANDO and WANT predicates as an alternative choice to the implementation of the subsumption architecture. It is more general than Brook's proposal and also more effective in strongly dynamic environments. In our implementation the enabling connections among behaviors are context dependent, so relationships among behaviors are not rigidly defined, but we can adapt the emerging global behavior, depending also on external conditions and motivations. Another reference we have considered is Arkin's schema-based behavior architecture [1]. It is possible to map the basic principles of our and Arkin's approaches into each other. The main difference is the fuzzy model we put at the basis of our architecture, whereas the analogous features are represented by Arkin by different tools such as a *gain* value to weight each behavior contribution, and their representation in terms of potential fields. However, the gain has a different meaning, stating the a priori relevance of a behavior with respect to another, while we blend (or select) behaviors according to their condition values and context, which change at any time instant. Another difference between our architecture and those of Brooks and Arkin is the presence in ours of a world modeler that interfaces the external world with behavior modules. In the mentioned architectures, world modeling is embedded in each behavior definition. We have implemented such a module to achieve efficiency and to provide a unified interface from the environment to the behaviors, as done by many others in these years.

5.3. Future directions

We are currently enhancing our system in two directions: informed hierarchical organization of the behaviors [19] and uncertainty management [20] in such architecture.

As shown in Section 3, the interaction among behaviors having different priorities has to be managed by a careful design of the predicates of CANDO and WANT conditions. We have defined a different interaction model, where behavior modules are organized in a hierarchy: higher priority behaviors receive in input not only the predicates from *MAP* and *SCARE*, but also the actions proposed by lower priority modules. They will decide whether these actions are in contrast with their goals, and eventually eliminate or modify them. This preserves the independence of behavior design, since each module can be designed by considering the proposed actions potentially contrasting with its goals without

any concern about the actual possibility that any other behavior proposing these actions is present in the network. At the same time, it is possible to implement priorities on a conceptual basis, obtaining a more clean and clear design.

References

[1] R.C. Arkin, Behavior-Based Robotics, MIT Press, Cambridge, MA, 1998.
[2] A. Saffiotti, K. Konolige, E.H. Ruspini, A multivalued-logic approach to integrating planning and control, Artificial Intelligence Journal 76 (1–2) (1995) 481–526.
[3] S. Coradeschi, A. Saffiotti, Anchoring symbols to sensor data: preliminary report, in: Proceedings of the 17th AAAI Conference, Austin, Texas, 2000, pp. 129–135.
[4] A. Bonarini, M. Matteucci, M. Restelli, Anchoring: do we need new solutions to an old problem or do we have old solutions for a new problem? in: Proceedings of the AAAI Fall Symposium on Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems, AAAI Press, Menlo Park, CA, 2001, pp. 79–86.
[5] A. Bonarini, G. Invernizzi, T. Labella, M. Matteucci, An architecture to co-ordinate fuzzy behaviors to control an autonomous robot, Fuzzy Sets and Systems 134 (1) (2002) 101–115.
[6] A. Bonarini, M. Restelli, An architecture to implement agents co-operating in dynamic environments, in: Proceedings of AAMAS 2002—Autonomous Agents and Multi-Agent Systems, ACM Press, New York, NY, 2002, pp. 1143–1144.
[7] M. Asada, H. Kitano, I. Noda, M. Veloso, RoboCup: today and tomorrow—what we have learned, Artificial Intelligence Journal 110 (1999) 193–214.
[8] M. Asada (Ed.), RoboCup-98: Robot Soccer World Cup II, Springer-Verlag, Berlin, D, 1998.
[9] P.S.G. Kraetzschmar, T. Balch (Eds.), RoboCup-2000: Robot Soccer World Cup IV, Springer-Verlag, Berlin, D, 2001.
[10] A. Birk, S. Coradeschi, S. Tadokoro (Eds.), RoboCup-2001: Robot Soccer World Cup V, Springer-Verlag, Berlin, D, 2002.
[11] G. Kaminka, P. Lima, R. Rojas (Eds.), RoboCup-2002: Robot Soccer World Cup VI, Springer-Verlag, Berlin, D, 2003.
[12] A. Bonarini, P. Aliverti, M. Lucioni, An omnidirectional vision sensor for fast tracking for mobile robots, IEEE Transactions on Instrumentation and Measurement 49 (3) (2000) 509–512.
[13] S. Harnad, The symbol grounding problem, Physica D 42 (1990) 335–346.
[14] C. CastelPietra, L. Iocchi, D. Nardi, R. Rosati, Coordination in multi–agent autonomous cognitive robotic systems, in: Proceedings of the Second International Cognitive RoboCup Workshop, Amsterdam, NL, 2000.
[15] P. Stone, M. Veloso, Task decomposition and dynamic role assignment for real-time strategic teamwork, in: J. Müller, M.P. Singh, A.S. Rao (Eds.), Proceedings of the 5th ATAL-98, vol. 1555, Springer-Verlag, Heidelberg, D, 1999, pp. 293–308.
[16] K. Konolige, K. Myers, E. Ruspini, A. Saffiotti, The saphira architecture: a design for autonomy, Journal of Experimental and Theoretical Artificial Intelligence 9 (1) (1997) 215–235.
[17] A. Saffiotti, The uses of fuzzy logic in autonomous robot navigation, Soft Computing 1 (1) (1997) 180–197.
[18] R.A. Brooks, A robust layered control system for a mobile robot, IEEE Journal of Robot Automation 2 (1) (1986) 14–23.
[19] A. Bonarini, M. Matteucci, M. Restelli, A novel model to rule behavior interaction, in: Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8), IOS Press, Amsterdam, NL, 2004, pp. 199–206.
[20] A. Bonarini, M. Matteucci, M. Restelli, A model to manage data reliability in behavior-based robotics, in: Proceedings of the Fifth IFAC Symposium on Intelligent Autonomous Vehicles IAV-2004, Elsevier, Amsterdam, NL, 2004, on CD.