# Behavior Analysis and Training—A Methodology for Behavior Engineering

Marco Colombetti, Marco Dorigo, *Member, IEEE*, and Giuseppe Borghi, *Student Member, IEEE*

***Abstract*— We propose *Behavior Engineering* as a new technological area whose aim is to provide methodologies and tools for developing autonomous robots. Building robots is a very complex engineering enterprise that requires the exact definition and scheduling of the activities which a designer, or a team of designers, should follow. Behavior Engineering is, within the autonomous robotics realm, the equivalent of more established disciplines like Software Engineering and Knowledge Engineering. In this article we first give a detailed presentation of a Behavior Engineering methodology, which we call *Behavior Analysis and Training* (BAT), where we stress the role of learning and training. Then we illustrate the application of the BAT methodology to three cases involving different robots: two mobile robots and a manipulator. Results show the feasibility of the proposed approach.**

## I. INTRODUCTION

THIS paper is concerned with problems related to the development of autonomous robots. A major obstacle in the realization of an autonomous robot is programming its control system so that the robot carries out its task in a reasonably flexible and adaptive way. In fact, the real world is so complex and unpredictable that directly programming a robot's controller soon becomes an almost impossible job. Recently, machine learning techniques have emerged as an interesting attempt to overcome this difficulty; however, it is not at all clear how machine learning should be integrated with more traditional design methodologies.

The main purpose of this paper is to analyze machine learning as part of an integrated methodology for designing and developing autonomous robots. We regard our work as a contribution to a new technological discipline, that we call *Behavior Engineering*, whose main concern is to establish methodologies, models and tools for shaping the behavior of robots. The name 'Behavior Engineering' is reminiscent of similar terms, like 'Software Engineering' and 'Knowledge Engineering,' the use of which is by now well established. By introducing a new term, we want to stress the specificity of the problems involved in the development of autonomous robots, and the strict relationships holding between the development of artificial agents and the natural sciences' study of the behavior of organisms. Indeed, the relevance of the notion of behavior in robotics has already been explicitly recognized, in particular by Brooks [4], Arkin [1], Maes [20], Maes and Brooks [21], and Saffiotti, Konolige and Ruspini [24]. However, we believe that the links between robotics and behavioral sciences have not yet been fully appreciated. There is no doubt that the mechanisms of learning, and the complex balance between what is learned and what is genetically determined, are the main concern of behavioral sciences. In our opinion, a similar situation arises in autonomous robotics, where a major problem is in deciding what should be explicitly designed and what should be left for the robot to learn from experience. Besides the intrinsic interest of machine learning mechanisms, we believe that the whole issue is likely to become of primary importance for practical applications.

So far, learning agents have mainly been studied in simulation. But simulated worlds cannot be as complex and unpredictable as the real world, for the simple reason that a simulated environment implements a *model* of a physical environment. Some researchers have therefore begun to experiment on the application of learning mechanisms to physical robots. Just to name a few, Mahadevan and Connell [22] have proposed a subsumption architecture in which basic behaviors are learned by a reinforcement learning technique; Beer and Gallagher [2] train a six-legged robot to walk by a genetic algorithm which evolves neural-net based controllers; Dorigo [9], and Dorigo and Colombetti [11] apply a learning classifier system to the control of a small autonomous robot. However, from scientific experiments to engineering practice there is a huge gap. Helping to fill this gap is the main objective of this article.

Engineering is based on models, tools, and methodologies. As regards models, today there is a wide repertoire of well-understood methods that can be applied to robot learning, and many software tools are available to develop learning systems. In particular, our past work has concentrated on Learning Classifier Systems (LCS's) [3]; as a tool, we have used various versions of ALECSYS, a parallel implementation of a LCS running on a network of transputers [8], [9], [14], [15]. In contrast, very little is known about the methodological issues involved in the effective use of learning as a component of practical robot development. In this paper, we concentrate on such issues. In Section II, we propose a methodology for Behavior Engineering, that we call BAT (Behavior Analysis and Training), based on the experience acquired in our past

Manuscript received July 10, 1994; revised May 27, 1995. This work was supported in part by a MURST "60%" grant for the years 1992–1994 to M. Colombetti, and by an Individual CEC Human Capital and Mobility Programme Fellowship to M. Dorigo for the years 1994–1996.

M. Colombetti and G. Borghi are with the Progetto di Intelligenza Artificiale e Robotica, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 20133 Milano, Italy (e-mails: colombet@elet.polimi.it; www.elet.polimi.it/people/colombet/; borghi@elet.polimi.it, www.elet.polimi.it/people/borghi/).

M. Dorigo is with IRIDIA, Université Libre de Bruxelles, 1050 Bruxelles, Belgium (e-mail: mdorigo@ulb.ac.be; URL: http://iridia.ulb.ac.be/dorigo/dorigo.html).

Publisher Item Identifier S 1083-4419(96)03236-0.

research. Section III is devoted to an analysis of the training that plays a fundamental role in our methodology. Sections IV, V, and VI support the methodology with the description of three practical cases. Finally, Section VII presents our conclusions and suggests directions for future work.

## II. The Bat Methodology

The goal of any engineering methodology is to help engineers to design high quality products—where, of course, the term 'quality' must be understood in a very broad sense. To this purpose, a methodology must at least tell us:

- what are the relevant aspects of quality for the product being designed;
- how to specify the product to be designed;
- what design choices should be made, in what sequence, and on what grounds;
- how to assess the quality of the final product.

In this section, we shall discuss all these facets. To keep the treatment within reasonable limits, we assume that the problem faced by the engineer is to develop an autonomous robot in a situation in which:

- The *robot shell* (i.e., the robot's "anatomy") is essentially predefined[1]. For example, one may be constrained to use a specific commercial manipulator or mobile robot platform.
- The *initial environment* is predefined. For example, the robot may be meant to operate on the lunar surface, or at the bottom of a lake, or in a parking lot.
- The *robot's controller* will include a learned component, and the learning system has already been chosen.

In addition we assume that the main stages in the development of a robotic application are:

- To describe the robot shell and the initial environment, and to define the robot's *target behavior*, that is, the desired pattern of interaction between the robot and its environment. For example, we may want the robot to collect mineral samples from the lunar surface.
- To analyze the target behavior and decompose it into a structured collection of *simple behaviors*.
- To provide a complete *specification* of the various components of the complete robot. In particular, one has to specify:
    - The *sensors* and *actuators* interfacing the robot with its environment, possibly together with some artificial *extension of the environment*, to make perception and action possible. For example, we may decide to place bar-code signs in specific locations of a parking lot, to help the robot monitor its position.
    - The *controller architecture*, that is, the overall structure of the robot's control program.
    - A *training strategy*, that is, a systematic procedure for training the robot to perform the target behavior.

[1]In our terminology, a robot is made up of a *shell*, a *sensorimotor interface*, and a *controller*. The "anatomy" of the robot is the structure of its shell.

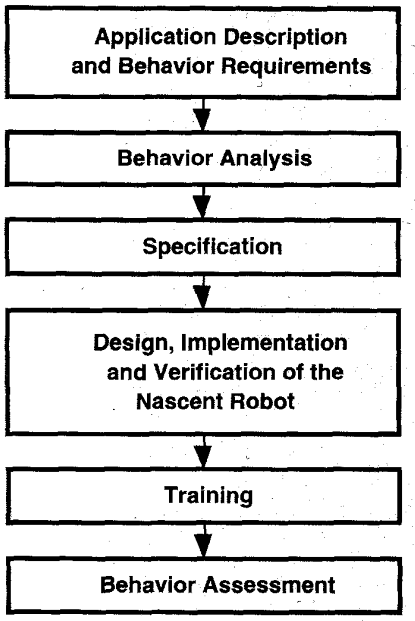

Application Description and Behavior Requirements
Behavior Analysis
Specification
Design, Implementation and Verification of the Nascent Robot
Training
Behavior Assessment

Fig. 1. The sequence of stages in the Bat methodology.

- To design, implement and verify the *nascent robot*, that is, the robot prior to training.
- To carry out *training* until the target behavior is learned.
- To *assess* both the *learning process* and the *final behavior* produced by the robot, the latter with respect to the specified target behavior.

These stages can be arranged in a logical sequence, reminiscent of the well-known *waterfall model* of Software Engineering (probably first introduced by Royce [23]); the sequence is shown in Fig. 1. In the following, we separately analyze the six stages.

### A. Application Description and Behavior Requirements

As behavior is the interaction of the robot's body with its physical environment, a specification of the target behavior presupposes a clear description of both such entities. In this paper we shall consider different types of robots, namely moving platforms (equipped with either wheels or tracks) and a two-link manipulator. The environments will be laboratory rooms containing different kinds of objects.

In fact, neither the robot nor the environment can be thoroughly described before we complete the Specification stage (see Fig. 1). The reason is that the sensorimotor apparatus of the robot can be fully designed only after the robot's behavior has been analyzed in detail; furthermore, the use of certain types of sensors or actuators may require us to modify the environment to achieve a satisfactory coupling between the robot and the environment. Therefore, in this initial phase we can only describe the *robot shell* (with incomplete or no sensorimotor apparatus) and the *initial environment* (prior to possible modification).

The requirements on the target behavior are usually informally stated in natural language. For example, we may want our robot to inspect an area, collecting objects of a given type.

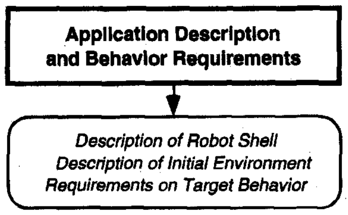

Application Description and Behavior Requirements
Description of Robot Shell
Description of Initial Environment
Requirements on Target Behavior

Fig. 2. Output of the Application Description and Behavior Requirements stage.

An informal description of the target behavior may well be clear and unambiguous, but is not sufficient as a basis for a later quantitative assessment of the robot's actual behavior. Therefore, a complete specification of the target behavior should include a formal, quantitative component. Continuing our previous example, we may specify that the robot has to collect at least $K \cdot D$ objects per time unit (where $K$ is a constant and $D$ is the average number of objects existing in the environment per surface unit), returning to its station at most $N$ times per time unit.

In Fig. 2 we show the products of the first stage of our methodology.

## B. Behavior Analysis

As a support for the subsequent Design stage, the target behavior must be analyzed in detail. In fact, the analysis of behavior is one of the key aspects of our methodology. Our approach involves decomposing the target behavior into simpler ones, in such a way that the target behavior results from the execution of the simpler behaviors and from their interactions. In turn, component behaviors can be further decomposed into even simpler ones, and so on.

Consider again the object-collecting task introduced in the previous subsection. To accomplish such a task, the robot will have to explore the given area, identify and locate the relevant objects, reach for them, grasp them and put them in a container onboard. Moreover, the robot will have to avoid obstacles during all of its activity and to go back to its station when the container is full or the batteries need recharging. Thus the overall behavior (e.g., collecting objects) is decomposed into lower level components (e.g., exploring, identifying objects, locating objects, etc.).

There is no general rule about how to carve simple behaviors out of more complex ones. For example, that "exploring" is a single, basic behavior can only be established on the basis of general knowledge about robot engineering and past design experience. However, once the basic behaviors have been singled out, their interactions can, and must, be completely defined. For example, we must specify that "going back to recharging station" will have to inhibit "exploring" and "identifying objects," but should not inhibit "avoiding obstacles." The product of such an analysis is what we call *structured behavior*.

What types of interactions among behaviors should one take into account? Again, there is no general answer, because this issue is strictly connected to the kind of controller architecture one is going to design. In a previous paper [11], we have singled out the following types of interaction:

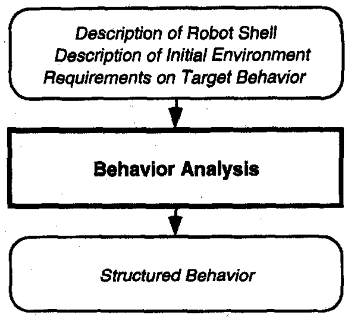

Description of Robot Shell
Description of Initial Environment
Requirements on Target Behavior
Behavior Analysis
Structured Behavior

Fig. 3. Input and output of Behavior Analysis.

- *Independent sum:* two or more independent behaviors are performed at the same time; for example, a robot may explore while trying to locate objects. The independent sum of behavior $\alpha$ and behavior $\beta$ is written as
$$\alpha \mid \beta.$$
- *Combination:* two or more homogeneous behaviors (i.e., behaviors involving the same actuators) are combined into a resulting behavior; for example, a robot may have to avoid an obstacle while attempting to reach for an object. The combination of $\alpha$ and $\beta$ is written as
$$\alpha + \beta.$$
- *Suppression:* a behavior inhibits a competing one; for example, the robot may give up looking for objects in order to reach its recharging station as soon as possible. If $\alpha$ suppresses $\beta$, we write
$$\frac{\alpha}{\beta}.$$
- *Sequence:* a behavioral pattern is built as a sequence of simpler behaviors; for example, an object is reached for only after having located it. The sequence of $\alpha$ and $\beta$ is written
$$\alpha \cdot \beta;$$
moreover, if a sequence $\sigma$ is repeated forever, we write
$$\sigma^*.$$

As we shall see, a clear description of the interactions between simple behaviors is essential for the following Specification stage.

The input and the output of the Behavior Analysis stage are shown in Fig. 3.

## C. Specification

As we have already said, we assume both the robot shell and the initial environment to be given. In general, however, the sensorimotor interface of the robot has to be redesigned for each specific behavior. For example, we may find out that objects in the environment can be roughly located through a sonar belt, and identified through a "chromatic analyzer," that is, a virtual sensor classifying objects on the basis of their color.

In turn, the chromatic analyzer can be physically realized through a rotating color camera and an appropriate software interface. Analogously, requirements on motor control lead to decisions on the kind of output that the robot controller will have to send to the actuators. For example, suppose we need to control a platform moving on two tracks, connected to two independent motors. Assuming that the controller sends a message to each motor at every control cycle, we may decide that the controller's output can be one of the following: *move forward, do not move, move backward.*

Given the state of the art in robotics, robots are often unable to deal directly with the initial environment. For example, it may be necessary to modify the objects with which the robot has to interact, so that they can be detected, identified and located. It is part of the Specification stage to find out what must be added to the environment to make the target behavior possible. The environment thus equipped will be called Extended Environment.

Another task of the Specification stage is to establish the controller architecture. In our work, we have experimented with various kinds of architecture, implemented via ALECSYS (see Section III-A). In a previous paper [11], we contend that the architecture of the controller should parallel the organization of the structured behavior, as established by Behavior Analysis. This result is obtained by allocating a *behavioral module* (BM) to each simple behavior; interactions among behaviors are then dealt with in various ways, that we now briefly describe.

We classify architectures in the following way:

- *Monolithic architectures.* These are built by only one behavioral module, directly connected to the robot's sensorimotor interface (see Fig. 4).
- *Distributed architectures.* These include all architectures built by more than one behavioral module. In this case we distinguish between two subclasses:
  - *Flat architectures*, built by a number of behavioral modules, all directly connected to the robot's sensorimotor interface. In turn, different behavioral modules may have: *independent outputs*, i.e., they send their motor messages to different actuators (Fig. 5(a)); or *integrated outputs*, i.e., their motor messages are integrated in an appropriate way, and then sent to the same actuator (Fig. 5(b)): for example, the integration can be the vector sum of two movements.
  - *Hierarchical architectures*, built by a hierarchy of levels, where the modules at level 1 are connected to the sensorimotor interface, and modules at level $N > 1$ are connected to at least one module of level $N - 1$, and to no module of level higher than $N - 1$ (see Fig. 6).

In fact, there is a natural correspondence between architectures and types of behavior interaction. Referring to the classification of interactions proposed in the previous subsection, we have the following correspondence:

- *Independent sum:* flat architecture with independent outputs.
- *Combination:* flat architecture with integrated outputs, or hierarchical architecture.

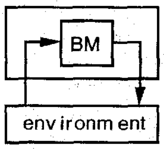


Fig. 4. The monolithic architecture.

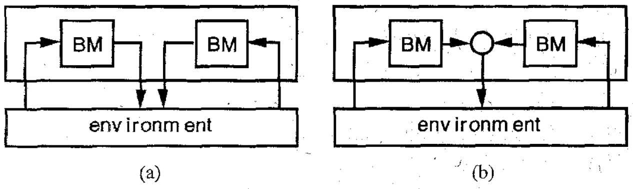


Fig. 5. Flat architectures.

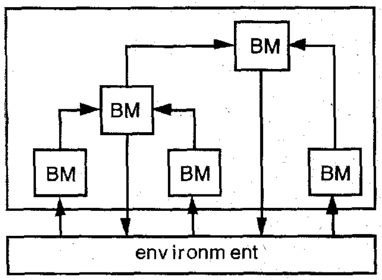


Fig. 6. An example of hierarchical architecture.

- *Suppression:* switch architecture (i.e., a kind of hierarchical architecture in which higher-level modules are only responsible for enabling lower-level modules).
- *Sequence:* hierarchical architecture.

Returning to our running example, the simple behaviors of exploring, avoiding obstacles, locating objects, reaching for objects, going to the recharging station, etc., can be allocated to corresponding behavioral modules. As regards the interactions among different behaviors: exploring and locating objects will have independent outputs; exploring and avoiding obstacles will have integrated outputs; exploring and going to the recharging station will be connected through a switch; and so on.

Finally, there is one more goal for the Specification stage. As we have already suggested, the BAT Methodology assumes that behaviors are allocated to behavioral modules by design, but the function of each module is developed by machine learning techniques. From now on we assume that a *reinforcement learning* system will be used. We still have to make a decision on the *training strategy*, including:

- the *reinforcement program*, that is the information that will be provided to the learning system to make the robot converge to the target behavior; and:
- the *shaping policy*, that is whether the structured behavior should be learned in a one-shot learning, or by a sequence of learning sessions, and in this last case in which order the various behaviors and their interactions will have to be learned.

Such decisions are critical, because training is strongly responsible for the final performance of the robot. Section III is devoted to a more detailed analysis of this fundamental issue.

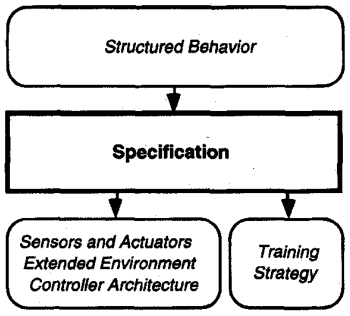


Fig. 7. Input and output of the Specification stage.

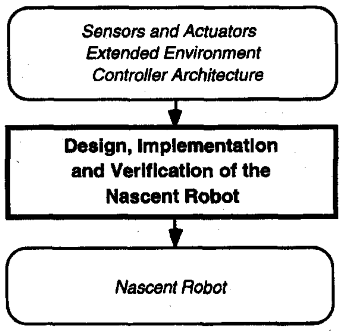


Fig. 8. Input and output of Design, Implementation and Verification of the Nascent Robot.

Fig. 7 shows the input and the outputs of the Specification stage.

## D. Design, Implementation and Verification of the Nascent Robot

At this point we have sufficient information to design, implement and verify the nascent robot, that is, the robot endowed with all its hardware and software components, but prior to training. We have no special remarks on this stage. Designers should exploit known methodologies for hardware and software engineering. The input and output of this stage are shown in Fig. 8.

## E. Training

Given that the training strategy has already been established, the current stage is only responsible for the implementation of such a strategy. The main problem of training is that it can be an extremely expensive task: even the most efficient learning systems converge slowly, and therefore training may require numerous lengthy sessions.

In many cases of practical interest, it is possible to speed up training through the use of simulators. Training a simulated robot is obviously faster than training a real one, and involves only a fraction of the effort. In general, this is not sufficient, because in simulated environments much of the richness and unpredictability of the real world is lost. A reasonable compromise involves using a simulator to develop a first approximation of the final controller, that can be refined through direct training of the real robot. Another advantage of simulation is that it allows one to carry on training in environments that are more manageable than the real one in

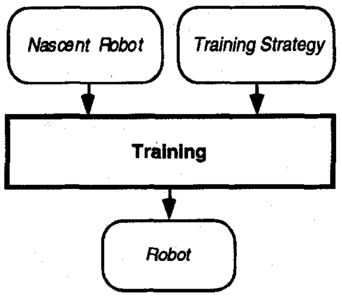


Fig. 9. Input and output of the Training stage.

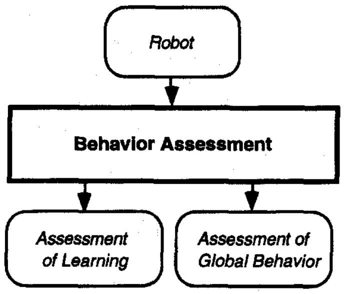


Fig. 10. Input and outputs of Behavior Assessment.

some important respect; we have exploited this aspect in the case presented in Section V.

As shown in Fig. 9, the output of the Training stage is the final robot, whose behavior must now be assessed.

## F. Behavior Assessment

Before introducing an assessment procedure, we need to clarify our notion of the quality of behavior. In general, the quality of a product is made up of different components.

We say that the robot's actual behavior is *correct* if it conforms to the target behavior, in the environmental conditions that have been assumed in the specification. For example, if the target behavior includes reaching location $P$ in environmental conditions $C$, the robot's behavior will be correct if the robot actually reaches $P$ when conditions $C$ hold.

The robot's behavior is said to be *robust*, if it conforms to the target even when the structure or the dynamics of the environment change with respect to what has been assumed in the requirements (without any change to its controller). For example, the robot may still be able to reach location $P$ even if the environment contains obstacles that were not considered in the original description of the target behavior.

Related to robustness is the concept of *adaptiveness*, which is understood as the robot's ability to modify its controller so that its behavior adapts in real time to changes in the structure or the dynamics of the environment. In general, this kind of adaptiveness can be guaranteed by learning mechanisms.

As with all artifacts, it is desirable to realize a system with a high degree of *modularity*. Modularity should be achieved at different levels:

- At the *robot level:* this means that the nascent robot should be a modular system. This aspect of modularity affects the design of the nascent robot, and will not be discussed here.

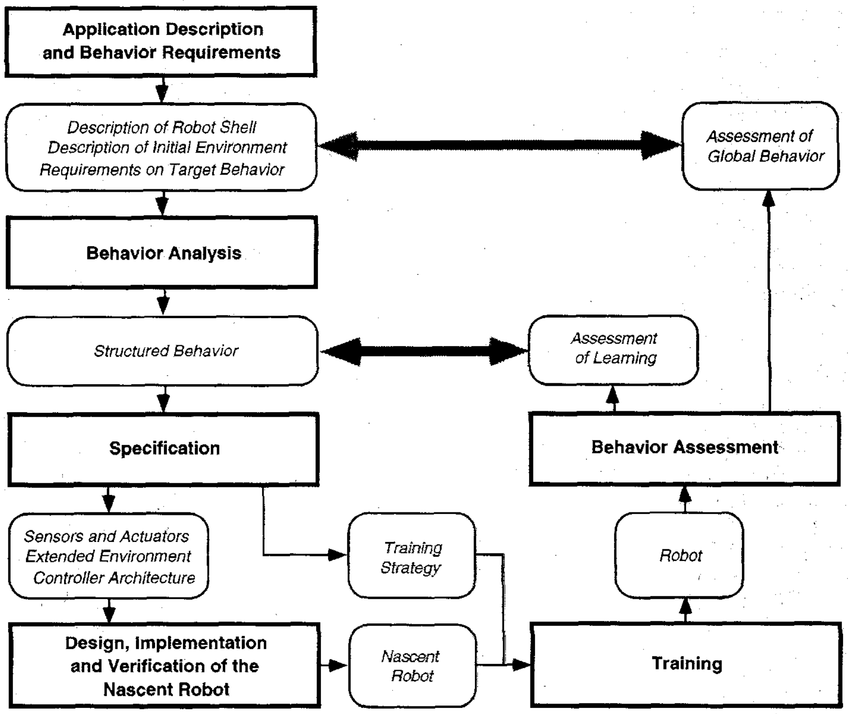

Application Description and Behavior Requirements
Description of Robot Shell
Description of Initial Environment
Requirements on Target Behavior
Assessment of Global Behavior
Behavior Analysis
Structured Behavior
Assessment of Learning
Specification
Behavior Assessment
Sensors and Actuators
Extended Environment
Controller Architecture
Training Strategy
Robot
Design, Implementation and Verification of the Nascent Robot
Nascent Robot
Training

Fig. 11. A global sketch of the BAT methodology.

- At the *behavior level:* this means that the controller should realize the target behavior in a modular way. This aspect of modularity can be enforced by good behavior analysis, and by adopting the controller architecture that best fits the structured behavior.
- At the *training level:* this means that the training strategy should be independent of low-level details about the robot's architecture. This aspect will be discussed in Section III.

In general, the performance of the robot can and should be evaluated in quantitative terms. We distinguish two types of performance indexes:

- *Local performance indexes* (or *learning indexes*), that measure the effectiveness of the learning process (that is, the correspondence between what is taught and what is learned). These indexes allow for the assessment of the learning process.
- *Global performance indexes*, that measure the correspondence of the robot's behaviors with the target behavior (as defined by the corresponding requirements). These indexes allow for the assessment of the robot's global behavior.

In this paper the local performance index is defined as

$$L(t) = \frac{R(t)}{t}$$

where $t$ is the number of robot moves from the beginning of the experiment and $R(t)$ is the number of moves that have been positively reinforced from the beginning of the experiment. The global performance index was computed as

$$G(t) = \frac{t}{A(t)}$$

where $A(t)$ is the number of achievements from the beginning of the experiment (i.e., the number of times the agent has reached the goal). Typically, this measure is used when $A(t)$ is not fixed in advance, as in the experiments of Section IV. When the number of achievements is predefined, as in the experiments of Section V, $G$ is computed as the total number of moves necessary to obtain them.

Fig. 10 shows the input to and output from the present stage.

To conclude this section, Fig. 11 summarizes all the stages previously described. The two bidirectional arrows connect the outputs of the Assessment stage with the documents against which the robot's behavior has to be evaluated.

## III. About Learning and Training

### A. The Role of the Trainer

In the BAT methodology, training plays a fundamental role. It involves four components: the robot to be trained, its environment, a learning system, and a trainer. Kinds of learning systems that have proved to be particularly fit for robot training are the ones based on *reinforcement learning*. In reinforcement learning, all or some of the actions performed by the robot receive a positive or negative *reinforcement* from the trainer. Positive reinforcements, or *rewards*, indicate that the action performed is correct, given the predefined target behavior and the current environmental situation. Negative rewards, or *punishments*, indicate that the action performed is considered incorrect. The information contained in reinforcements is exploited by the learning system to converge toward a robot controller that implements the target behavior in the given environment.

A first classification of reinforcement learning systems distinguishes between *immediate reinforcement* learning and *delayed reinforcement* learning. In the former, reinforcements are provided by the trainer for each single action performed by the robot. In the latter, delayed reinforcements refer to a whole sequence of actions, that in principle can be arbitrarily long. As an example, let us assume that a robot is being trained to reach object $A$. In immediate reinforcement, the trainer will have to reinforce the robot at each elementary movement; typically, the robot will be rewarded if it gets closer to $A$, and punished otherwise. In delayed reinforcement, the robot is rewarded if and only if it reaches $A$; such a reward has to be considered as a positive evaluation of the movements that led it to reach $A$. Both Learning Classifier Systems[2] (LCS's) [3], the model we used to implement our learning system, and $Q$-learning [25] are typical examples of reinforcement learning systems[3]. Both methods can be given immediate or delayed reinforcements, the main difference being the time they require to converge on good action policies, usually shorter with immediate reinforcement, and the quantity of information the trainer needs to evaluate the robot moves, often greater for immediate reinforcements.

There are deep differences between immediate reinforcement and delayed reinforcement that should be thoroughly appreciated. Delayed reinforcement learning has a great conceptual advantage that can be illustrated through the previous example. If the robot has to be rewarded only when it reaches object $A$, it is sufficient to endow the robot with a specific sensor to detect that A has been reached. On the contrary, immediate reinforcement involves an evaluation of the distance between the robot and $A$, to be repeated after each movement, which may be difficult in many practical applications.

On the other side, delayed reinforcement learning has the drawback of being more complex and converging much more slowly than immediate reinforcement; in fact, it converges so slowly that it is completely useless in most practical cases. A good solution to this problem might be to exploit both mechanisms, in order to get the best out of each. Even if a few steps in this direction have already been made [5], [26], no final solution has been proposed yet. Throughout this paper we shall assume that training is achieved through immediate reinforcement learning (henceforward simply called *learning*).

Given that a sufficiently efficient learning system is available, the main problem of training is to teach the right thing—that is, to make the robot converge precisely to the desired behavior. This means that the most critical component is the trainer.

Often, the target behavior cannot be taught directly. Consider again the behavior of reaching object $A$. In fact the trainer

[2] See the brief description of ALECSYS in the next subsection, or [9], [10] for more details.

[3] The similarities existing between the LCS and $Q$-learning have been recently discussed in Dorigo and Bersini [10].

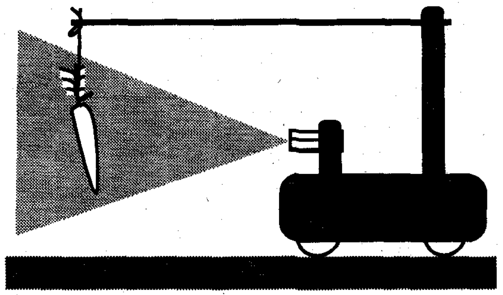


Fig. 12. Why moving in the right direction may not be enough.

cannot just reward the action of reaching $A$: rather, it has to reward the robot when it gets closer to $A$. But this means that the robot is being taught *to approach* $A$, not *to reach* it. This is a consequence of a limitation of any immediate learning strategy.

Moreover, the robot might be able to identify the direction in which $A$ is located, without knowing $A$'s distance. In this case, the robot cannot even learn *to approach* $A$: at most, it can learn *to move in* $A$*'s direction*. This is a consequence of a sensory limitation.

Suppose now that the robot has learned to move in $A$'s direction. Doing so does not logically imply that the robot will eventually reach $A$. In fact, $A$ might just move too fast to be ever caught by the robot! The point is that the learned behavior (moving in $A$'s direction) converges to the target behavior (reaching $A$) only if the environment satisfies a number of constraints (e.g., $A$'s average velocity could be known to be lower than the robot's one).

This simple example shows that:

- Some target behaviors cannot be directly taught through immediate reinforcement learning. Instead of teaching a target behavior $\alpha$, the trainer has to teach a behavior $\beta$, such that doing $\beta$ achieves $\alpha$ if the environment satisfies a number of constraints.
- What can be directly taught depends on general limitations of both the immediate training strategy and the robot's sensory apparatus.

As a consequence, assessing the effectiveness of learning and assessing the correspondence of the robot's final behavior with the target behavior are two different things. A robot may have a local performance index of 100%, and completely fail to carry out the target behavior (for example, the robot may have learnt to move in $A$'s direction, but still be unable to reach $A$—see Fig. 12).

Another problem related to the trainer is that rewarding or punishing an action presupposes the ability to perceive its effects. In principle, the trainer could be a human being, observing the robot's behavior and providing reinforcements according to his or her understanding of the target behavior. In practice, such a solution is seldom viable because human beings tend to be rather inaccurate in their evaluations, and too slow with respect to the robot's perception-action cycle. Therefore, a trainer is generally a computer program, designed to give reinforcements according to the robot's actions. As we shall discuss in Section IV-C, this may require additional

winning is proportional to the amount of strength offered. Winning rules acquire the right to send their messages to actuators or to other rules at the next time step. The rules that win the auction pay their bids to those rules which posted at the preceding time step the messages by which they were activated, or to the external environment if the message was sent by sensors. Rules which post a message (i.e., win the auction) and whose message causes an action, receive reinforcement by the trainer. Reinforcement can be either positive, i.e., *rewards*, or negative, i.e., *punishments*. Rules which post a message and whose message matches a winning rule at the next time step, receive the amount of strength paid by the matching rule. In this way reinforcement flows backward from the trainer to the external environment in the form of rule strengths. In addition rules lose a small amount of their strength at each cycle (this is the so-called *life tax*), so rules that never participate in the auction and that never win the auctions will slowly decrease in strength.

- *The rule discovery algorithm*: The rule discovery algorithm overwrites low strength rules with new rules generated by a genetic algorithm [16]: new rules are created by the recombination and mutation of high strength rules.

## IV. Case 1: AutonoMouse V

In this section we instantiate the BAT methodology using a simple robot, called AutonoMouse V, as a running example. As this section will be paradigmatic of the way the BAT methodology should be applied, its organization strictly follows the sequence of stages previously discussed in Section II. The reader will therefore find in Subsection IV–X the description of the output of the activity carried out, within the AutonoMouse V application, following the methodology as described in Subsection II–X.

### A. Outputs of the Application Description and Behavior Requirements Stage

*Description of Robot Shell:* AutonoMouse V (AM hereafter) is a small (35 cm long plus 26 cm for the tail, 15 cm wide, and 28 cm high) robot (Fig. 13(a)). Its sensory apparatus is: two light sensors, one sonar, three whiskers, and a "change of direction" sensor (see Fig. 13(b)). It is also provided with two motors which control two tracks. The robot carries a battery which provides it with three hours of autonomy.

The light sensors are two photodiodes which are positioned within a structure which make them partially directional devices. The two eyes together cover a 270° zone, with an overlapping of 90° in front of the robot (see Fig. 14). They can distinguish 256 levels of light intensity.

The sonar is highly directional (it detects obstacles in front of AM) and can sense an object as far as 10 meters away. It returns a number between 0 and 256 which is an estimate of the distance to the obstacle.

The three whiskers, placed on the front of AM (see Fig. 15), are devices that change state when AM bumps into an obstacle. The directional sensor is a rod, which we call a tail, with

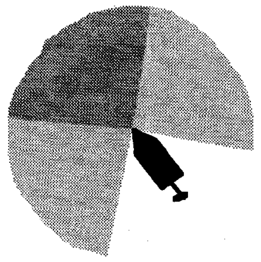


Fig. 14. AutonoMouse V light sensors.

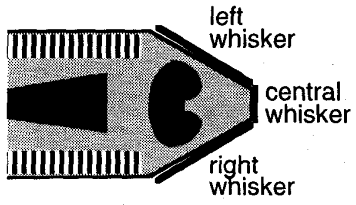


Fig. 15. AutonoMouse V whiskers.

a wheel at its end which can rotate around the rod axis. It therefore does not rotate when AM moves forward without turning, while it rotates when AM turns. It's resolution is about 1.2°.

Each AM motor has nineteen activation speeds: nine forward, nine backward, and one for not moving.

AM has some onboard computing capabilities to transform sensory input into digital messages in the format used by the learning algorithms. The learning algorithms run on a transputer board in a host computer connected to AM via a 4800-baud infra-red link.

*Description of the Initial Environment:* AM moves in a office-like environment lightened by artificial lights. The terrain is smooth, and people do not interfere with the robot's movements. In AM's environment there are no obstacles, except for the walls and for the obstacle specifically used in experiments.

*Requirements on Target Behavior:* In this simple application we want our robot to learn to search and follow a light moving at a speed which is comparable to the speed of AM. Now and then the light disappears behind an obstacle; in these occasions AM should go around the obstacle to see whether the light is on the other side and then start to follow it again (see Fig. 16). We call the target behavior the *Search&FollowLight* behavior.

### B. Output of Behavior Analysis

The target behavior of our robot, which was described informally in Section IV-A, can be expressed as the following structured behavior

$$\text{ReachLight} = \frac{\text{ApproachLight}}{\text{SearchLight}}.$$

Light-approaching and light-searching are the two basic behaviors comprising the structured behavior, and the interaction among them is suppression. Light-approaching is active when the light is seen. Light-searching is a rather complex behavior,

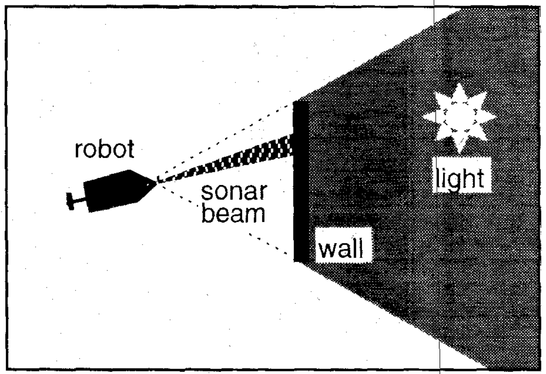


Fig. 16. The experimental environment for the *Search&FollowLight* behavior.

which includes: turning around to look for the light or the wall; approaching the wall when it is sensed by the sonar; trailing around the wall when it is sensed by the whiskers.

## C. Outputs of the Specification Stage

*Sensors and Actuators:* Sensors as defined in Section IV-A are too fine grained for our application. In fact, when using a learning algorithm it should be remembered that the greater the amount of sensory information, the greater the dimension of the search space. It is therefore advisable to choose a sensorial granularity that is the coarsest granularity which still allows the learning algorithm to distinguish among environmental states which are different from the task point of view.

We chose to put a threshold on both light sensors and the sonar sensor, transforming them into ON/OFF devices. The output of the light sensors is therefore either *I see a light* or *I do not see a light*, while the output of the sonar is either *I sense an object*, or *I do not sense an object* (the threshold for the sonar was chosen such that objects are sensed if they are closer than 1.5 meters).

The direction sensor, that is the tail, was thresholded so that the AM knows that it has changed its direction whenever it turns by an angle greater than 1.2°. For the same reasons, namely not to burden the learning system with useless extra search space given by too detailed control on movements, motor moves have been thresholded so that tracks are moved by sending two bits messages which correspond to the following possible movements of each track: *move backward at speed one, stay still, move forward at speed one, and move forward at speed two.* (Speed two corresponds to the maximum velocity of the robot, while speed one to one half of maximum velocity.)

*Extended Environment:* The initial environment has been extended by a 50 W bulb lamp which is moved around by one of the experimenters, and by a pleated wall obstacle (see Fig. 17). The choice of a pleated wall was due to the incapacity of our single and highly directional sonar to perceive smooth surfaces under all the possible angles of incidence of the sonar signal.

*Controller Architecture:* Given the kind of structured behavior described in Section IV-B, the switch architecture is the most appropriate, and was used for experiments (this

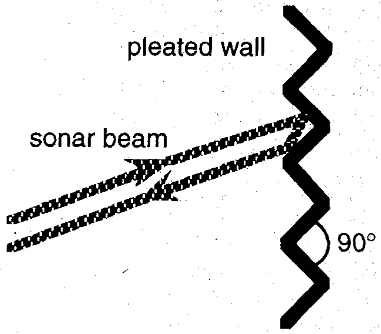


Fig. 17. The pleated wall.

architecture was compared with other possible solutions both in real world experiments and in simulation; see [7]). In this architecture the light-approaching and the light-searching behaviors (basic behaviors) are coordinated by a switch module, which learns to choose which basic behavior to give priority to.

*Training Strategy:* As we said in Section II-C, there are two determinations to be made regarding the training strategy: the reinforcement program and the shaping policy.

*Reinforcement Programs:* The reinforcement program is in charge of giving reinforcements after each move of AM. It is usually composed of a different subprogram for each basic behavior. In AM we defined the following reinforcement programs for each of the two behaviors identified in Section IV-B:

*Light-approaching reinforcement program*: After each move, the robot is rewarded if its distance from the light decreases, otherwise it is punished. Changes in distance are evaluated by measuring the change in intensity of light as seen by the two AM eyes. AM eyes have therefore a double use: as AM sensors (in this case they are thresholded), and as trainer sensors (in this case their full granularity is used to evaluate changes in light intensity).

*Light-searching reinforcement program*: Light-searching is not as easy to define as light-approaching, in that it allows the definition of a few different searching strategies. The main goal of this behavior is to search behind obstacles to see whether the light is there. For example, the agent could first use sonars to locate the edge of the obstacle, and then move directly in that direction. Another possible strategy could be to use the sonar to locate an obstacle, then approach it, and finally turn around it using bumpers to maintain contact with the obstacle (this could be done with a reinforcement program that rewards a right turn when the left bumper is on, and a left turn when it is off; in this way AM will learn to move along the obstacle with a zig-zag kind of motion). These, and a few other, strategies were investigated by Dorigo and Maesani [7]. In this paper we used the first strategy, which works as follows: if the sonar senses an obstacle, then AM is rewarded if it moves forward and at the same time it turns in the same direction of the previous move. As soon as the sonar does not sense the obstacle any longer, AM is rewarded if, while still moving forward, it turns in the opposite direction ("move forward

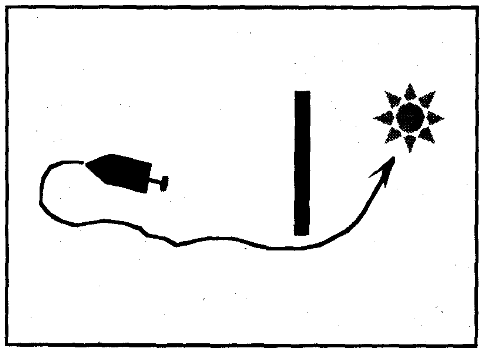


Fig. 18. The light-searching behavior.

while turning" moves are obtained by setting one motor to speed two and the other to speed one). This reinforcement policy generates an obstacle approaching trajectory like the one presented in Fig. 18.

*Shaping Policy:* The shaping policy used was modular. First we trained the two basic behaviors, then we froze them (that is, their learning algorithms were switched off), and we let the coordination behavior learn.

### *D. Outputs of the Design, Implementation and Verification Stage*

As we said in Section II-D, this step will exploit standard hardware and software engineering methodologies. All the design and implementation details regarding this stage of the BAT methodology are documented in [7].

### *E. Output of the Training Stage*

AM was trained using the training strategy defined in Section IV-C. We also compared the results obtained using the switch architecture with modular shaping, with those obtained training the two basic behaviors in a simulated environment and then copying the two rule sets into the two basic behavioral modules used to control the real robot. After transferring the two rule sets, training continues using the same training and shaping strategies as in the previous experiment.

### *F. Output of the Behavior Assessment Stage*

This phase of the Bat methodology is devoted to assessing the degree of learning achieved by the learning agent. Two tools are useful for this assessment: the local and the global performance indexes. Figs. 19 and 20 report the behavior of $L(t)$ and $G(t)$, as defined in Section II-F, for both the switch architecture (SA) and the switch architecture with initial knowledge (SAIK) respectively.

Both indexes show that, given the same amount of learning cycles, starting with some initial knowledge developed in simulation helps. None of the architectures achieves very high local performance. This is mainly due to limited learning time; in fact, we stopped the experiment when learning was still going on. A longer experiment we ran showed that after 12 000 AM moves the system was still learning. We stopped experiments after 6 000 AM moves because of time

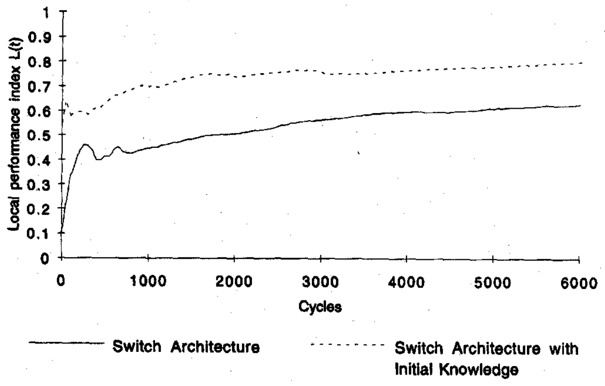


Fig. 19. Local index: Comparison between the SA and the SAIK architectures (with the local index, a higher value reflects a better performance).

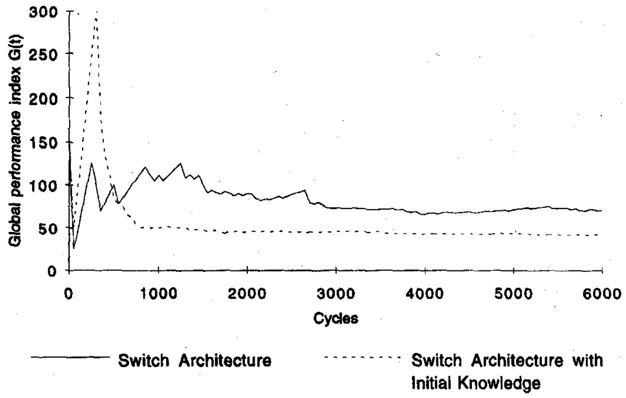


Fig. 20. Global index: Comparison between the SA and the SAIK architectures (with the global index, a lower value reflects a better performance).

constraints (6 000 moves take about one hour of time). Graphs are averaged on 5 runs.

In this experiment the assessment of global behavior is somewhat arbitrary, given the highly research-oriented nature of the task. $G(t)$ decreases as $L(t)$ increases, and this is an indication of the fact that the training strategy is somewhat successful in driving the learning system toward better global performance. Whether the given training strategy is a good one or not can be evaluated only by comparing it with other training procedures such as the one proposed in Section IV-C. In a more application-oriented task, $G(t)$ would be compared against quantitative requirements on the target behavior.

## V. CASE 2: HAMSTER

A second example of the application of the BAT methodology is HAMSTER, a mobile robot based on a commercial platform, whose task is to bring "food" to its "nest."[5] We shall not describe all steps in the development of this robot, but confine ourselves to the description of the main differences with respect to AM.

[5]This robot's name is mainly justified by its target behavior. It is also an acronym, namely Highly Autonomous Mobile System TrainEd by Reinforcements.

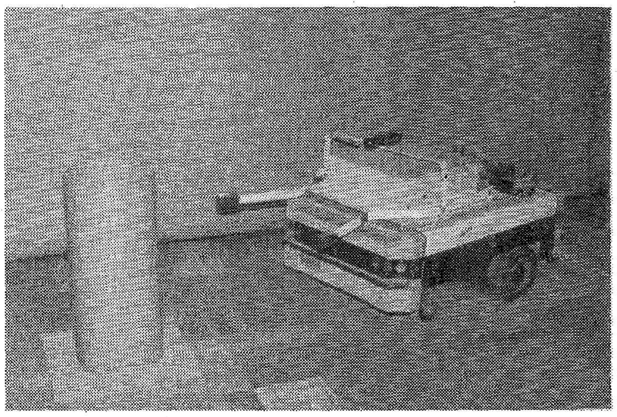


Fig. 21. A picture of HAMSTER.

The chief features of HAMSTER are that it combines "innate" (i.e., prewired) and learned behaviors, and that training was carried out in a simulated environment and then transferred to the physical robot.

### A. *HAMSTER's Shell*

HAMSTER's shell is Robuter, a commercial platform produced by RoboSoft (see Fig. 21). It is 102 cm long, 68 cm wide and 44 cm high. The configuration we used has a belt of 24 Polaroid sonars, surrounding the whole platform. Motion is produced by two motors acting on two independent wheels.

### B. *The Hoarding Behavior*

The target behavior is food hoarding, that is, to collect food pieces and to store them into HAMSTER's nest. The environment is a room of size 14 × 13.3 m, with various obstacles (see Fig. 22). Each piece of food is a cylinder (diameter 30 cm, height 70 cm) which slides on the floor when pushed by HAMSTER. The nest is located in a corner of the room.

The target behavior can be decomposed as follows:

$$\begin{aligned} HoardFood = {} & (LeaveNest \cdot GetFood \cdot ReachNest)^* \\ & + AvoidObstacles. \end{aligned}$$

### C. *Specification*

*Extension to the Robot's Shell and to the Environment:* Two rigid metal bars have been adapted to the Robuter's front so that food cylinders do not slip to either side when they are pushed. A frontal proximity sensor, based on the frontal sonars, allows HAMSTER to sense whether an object in front is far, fairly close, very close, or at contact. The food cylinders are wrapped into violet paper, and the nest's position is marked by another cylinder (diameter 30 cm, height 130 cm) wrapped in pink paper. HAMSTER uses a frontal color camera to identify the position of food cylinders and of the nest, which are distinguished on the basis of color. Moreover, the nest sensor exploits an odometer (that is, a sensor that estimates the robot's position and heading), to approximately identify the position of the nest when this is not visible.

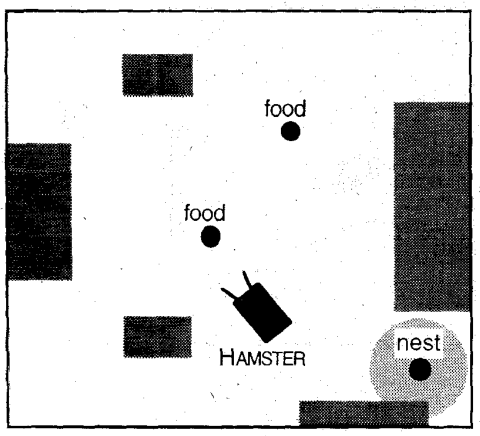


Fig. 22. Map of HAMSTER's environment.

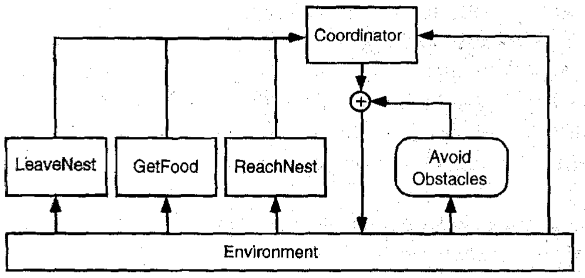


Fig. 23. Controller architecture for HAMSTER.

*Controller Architecture:* A main concern of the HAMSTER project was to combine learned and innate behavior modules. We chose to program *AvoidObstacles* directly, implementing a potential-based avoidance mechanism exploiting Robuter's sonars (see for example [19]).

The remaining part of the hoarding behavior, that is the food-fetching cycle, can be analyzed as a *pseudo-sequence* [6]. This means that at each control cycle there is enough information coming from the sensors to decide which of the three sub-behaviors should be active. More precisely:

- When the robot is in the nest: leave the nest, in such a way that previously captured food is left in the nest.
- When the robot is out of the nest and no food is captured: get a piece of food.
- When the robot is out of the nest and a piece of food is captured: reach the nest, pushing the piece of food.

We adopted a hierarchical controller architecture, with four behavioral modules at Level 1, and a coordinator module at Level 2. Each Level-1 module proposes a direction for the robot's movement. The coordinator chooses one of the proposed moves on the basis of the current situation, which is then combined with the move proposed by *AvoidObstacles* (see Fig. 23). In general, this combination amounts to some kind of vector sum of the moves proposed by *AvoidObstacles* and by the rest of the system. There is, however, a more complex case. When HAMSTER is nearing a piece of food, the front part of the *AvoidObstacle* behavior has to be inhibited, otherwise the food could never be captured. Inhibiting obstacle avoidance in such cases is part of the coordinator's task.

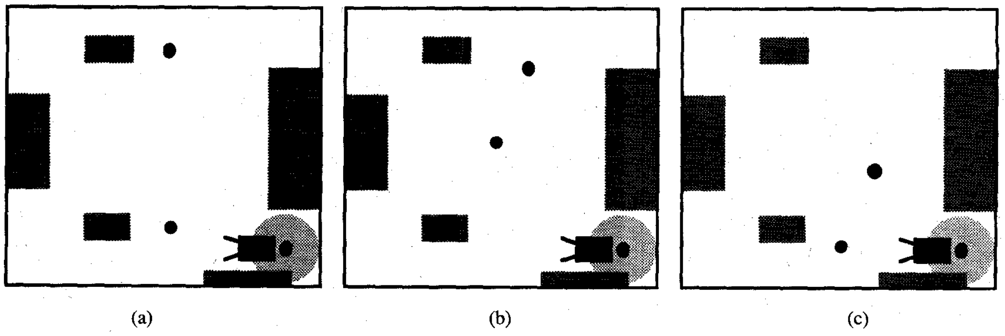


Fig. 24. The three experimental situations. Rectangular shaded areas are obstacles, black circles are food pieces. The HAMSTER is shown at its initial location, that is in the nest (shaded circle).

*Training:* HAMSTER was trained through a modular shaping policy: that is, each learned Level-1 module was trained separately, and then frozen. The Coordinator was then trained to achieve the target behavior.

We decided to use very simple reinforcement programs (RP's). For example, the *ReachNest* behavior was rewarded if and only if the distance from the nest decreased. This decision, however, had a very important consequence, in that it obliged us to carry out training in a simplified environment, that is, one without obstacles. The point is that training the robot while obstacle avoidance is active would have forced us to embed a model of obstacle avoidance into the RP. The reason is that the RP would be unable to evaluate HAMSTER's move properly without knowing the effects of obstacles on the robot. But then, the RP would have been very complex.

In order to eliminate the interference of the innate *AvoidObstacles* behavior, training had to be carried out in an obstacle-free environment. As we had no such physical environment available, all training was carried out in a simulated environment, and the resulting modules were then transferred to the real robot. In fact, we consider the transfer of the learned controller from a simulated environment to be an interesting option in its own respect.

### D. Assessment[6]

The degree of learning was evaluated through the local performance indexes of the basic behaviors, computed in the simulated environment with no obstacles. As expected, we obtained reasonably high values (from .75 to .85, depending on the specific behavior and on different RP's we have experimented with). We did not compute the local performance index in the environment with obstacles, because it would have been meaningless: For example, during the *ReachNest* behavior the robot would have been punished if it moved away from the nest in order to avoid an obstacle.

We then transferred the controller onto the real robot, and ran some experiments in the real environment (with obstacles). In this environment it is meaningful to evaluate the global performance, computed as the number of moves necessary to accomplish the task, averaged over a set of sample situations.

[6]Given that they are not essential to this example, we do not report here on the Design, Implementation and Verification of the nascent robot, nor on the details of the Training task.

TABLE I
GLOBAL PERFORMANCE INDEX OF HAMSTER IN THE REALENVIRONMENTS

| | situation a | situation b | situation c |
|---|---|---|---|
| mean | 365.3 | 450 | 313.5 |
| standard deviation | 27.77 | 127.30 | 34.33 |

More precisely, we chose three different initial situations (see Fig. 24), all including two pieces of food, and ran ten trials for each of them, recording the total number of cycles necessary to complete the hoarding of both pieces of food.

Finally, we computed the mean and the standard deviation of such data (Table I). In fact, in situation *c*, HAMSTER was unable to accomplish the task five times out of ten, due to the difficulty of getting the piece of food in front of the initial robot position and then avoiding the close obstacle. For this situation, the data reported are relative to the five successful trials only.

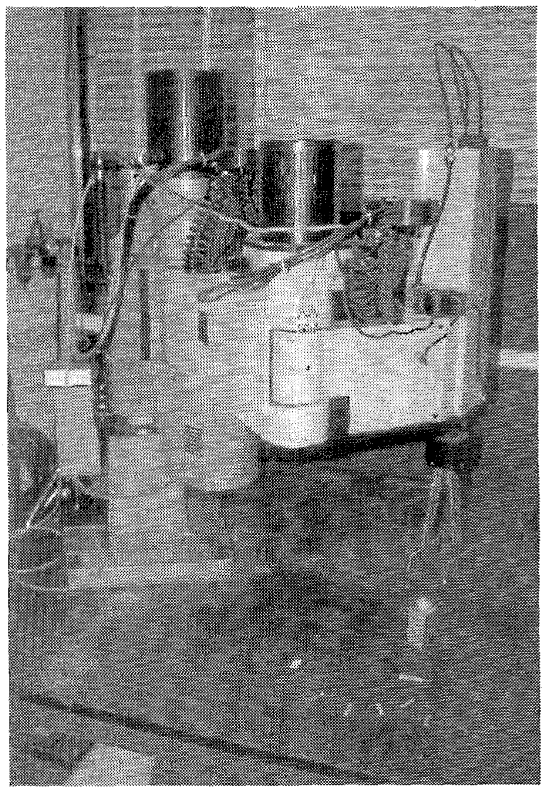
### E. Conclusions

The HAMSTER project shows that it is feasible to implement a robot's controller starting from both innate and learned behavioral modules. However, to keep the RP's as simple as possible it is necessary to carry out training in environments where the innate behaviors are not active. In our case, this required that we carry out training in a simulated environment, and then transfer the learned controller to the physical robot for the final assessment.

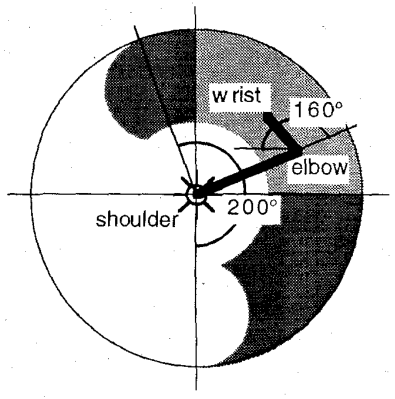
Whether the global performance of HAMSTER is to be considered acceptable is difficult to say. In a real application, some minimum performance level would have been established in advance. In our case, we can only say that an informal observation of the behavior gave us the impression that HAMSTER was performing reasonably well.

## VI. CASE 3: THE CRAB ROBOTIC ARM

For our final example of the BAT methodology we used an industrial manipulator, namely an IBM 7547 with a SCARA geometry (Fig. 25(a)). In this section, we shall refer to this robot as CRAB (Classifier-Based Robotic Arm), a name that indeed fits well the shape of its arm.



(a)

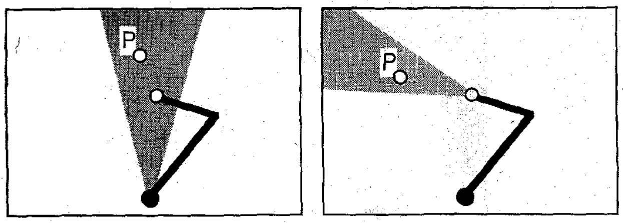


(b)

Fig. 25. The CRAB manipulator: (a) picture, and (b) schematic drawing of the action space.

CRAB is a two-link robotic arm. The first link can rotate 200° around the shoulder joint, and the second link can rotate 160°, with respect to the first link, around the elbow joint. As a result the end effector, attached to the wrist, can cover the gray area shown in Fig. 25(b). The actuators are two motors, acting on the shoulder and the elbow joints, that respectively rotate the first link (with respect to the fixed base) and the second link (with respect to the first link).

The target behavior we taught to CRAB is a very simple one, namely to have the end effector reach a still object ("food"),



(a) (b)

Fig. 26. Why a sector-based sensor must be placed on the end effector. (a) the position of $P$ cannot be distinguished from the position of the end effector; (b) $P$ can be approached with any degree of precision.

placed in the light gray area of Fig. 25(b) (the "foraging area"). In this case, the target behavior itself is a simple behavior, and therefore Behavior Analysis is absent. The reason why we were interested in such a simple behavior is that, given the polar geometry of CRAB, ALECSYS has to learn behavior rules that are very different from the ones controlling the mobile robots described in the previous sections. In particular, the relation between the actions (i.e., elementary rotations of the two links) and the resulting movement of the end effector involves complex trigonometric transformations.

In CRAB's environment, there is exactly one piece of food in the foraging area at any moment. Each time SCARA reaches it, the piece of food is moved to a new random position within the foraging area. To be sensed by the robot, the piece of food emits infrared light that can be received by an appropriate sensor. Deciding where to place such a sensor is a simple, but interesting Specification problem.

A natural agent using arms to grasp objects would be endowed with two types of sensory capacities: *proprioception*, to know the relative position of the limbs; and *vision*, to identify the relative direction and distance of the object to be grasped. Moreover, the visual sensor would be more or less rigidly coupled with the shoulder.

The problem with our artificial agents is that ALECSYS can only deal with a very limited amount of input information, largely insufficient to determine relative distances and angles of objects with the required degree of precision. Therefore, for both proprioception and "infrared vision" we adopted sector-based sensors as we did with our mobile robots. However, a sector-based sensor placed on the shoulder is structurally unable to precisely locate an object (see Fig. 26(a)). There is only one possibility for a sector-based sensor to guide the end effector exactly to a given point: namely, that the sensor itself is placed on the end effector (Fig. 26(b)).

As regards proprioception, experiments carried on by simulation showed that it is enough to have information about the elbow angle, that is, the angle between the first and the second link [13]. The sensorimotor apparatus of Scara was then specified as follows:

- A proprioceptive sensor providing information on whether the elbow angle is between 0 and 80°, or between 81 and 160°.
- An infrared sensor, placed on the end effector, telling the robot in which of 8 equal sectors the food is located (all sectors are of 45°).

- Two motor effectors, one each for the elbow and shoulder joints. These motors can rotate right (4° for both joints), rotate left (4° for both joints) or stay still.

Finally, for the controller architecture we chose a monolithic architecture, given that the target behavior was made up by a single, simple behavior.

The infrared sensor we implemented allowed us to locate the food piece in the correct sector with good precision, but gave us a rather noisy estimate of the distance between the food piece and the end effector. Note however that information about this distance is not used by the learning agent, but only by the RP to compute reinforcements. In fact, each move was reinforced proportionally to the variation of the distance between the end effector and the food piece, as estimated by the infrared sensor.

So far we run only a few pilot experiments with CRAB.[7] In a typical experiment, to speed up learning we started from an initial rule base obtained from a training session of 25 000 cycles run in a simulated environment, and ran a supplementary training session of 2,500 cycles with the real robot. At the end of this training session we obtained a local performance index of 0.86, showing a limited increase in performance from the initial rule base obtained through simulated pre-training (a test run on the real robot with such an initial rule base gave us a performance of 0.83). The result suggests that the supplementary training session on the real robot adapted the rule base learned in the simulated environment to the more noisy physical sensor. To confirm this hypothesis, we are presently running a number of experiments involving longer training sessions.

As a whole, it appears that ALECSYS is able to learn how to reach a still object with a polar motor apparatus. This is a very promising result, if one considers the importance of manipulators in robotic applications.

## VII. Conclusion and Future Work

In this article we have presented a methodology for the development of autonomous robots. The main features of such a methodology are the attention paid to the analysis of behavior, the integration of machine learning techniques with other aspects of robot design, and the independent assessment of learning and of the global behavior. We believe that the three cases we have analyzed show that the methodology is sound and can lead to realizations of practical interest. In particular, we have shown that complex behavior can be taught through modular shaping, that learned behavioral modules can be integrated with hand-coded modules, that at least in some cases it is possible to use behavioral modules learned in simulated environments as a starting point for real robot training, and that our methodology is applicable to robots of different types (like mobile platforms and robotic arms).

Needless to say, much work remains to be done. A first extension will have to be in the direction of more complex behaviors. This will require a larger amount of input information to be processed, and therefore will call for more powerful learning mechanisms. In order to relieve designers from part of their burden, learning techniques might be extended to other aspects of robot development, like the architecture of the controller. This means that the structure of behavioral modules should emerge from the learning process, instead of being predesigned. Another interesting extension to learning would be the possibility of using delayed reinforcements, that is, reinforcements that are not given at each single step, but only when a certain goal is achieved (see [12]).

In our work, we have noticed that a major bottleneck in the achievement of complex behavior is the richness of input information. In particular, realistic applications in complex environments will require the robot to recognize and locate "passive objects," that is, objects that are not specially predesigned in order to be sensed by the robot.

As regards the BAT methodology, we expect it to remain fairly stable in the future. This will allow us to design and implement a number of software tools to help the designer in the process of robot development. Tasks that strongly need such tools are Behavior Analysis and Specification, with particular regard to the definition of sensors.

Finally, aspects of Behavior Engineering that deserve a deeper analysis are the concept of quality of behavior, and the related issue of behavior assessment. An appropriate set of behavior metrics will have to be developed, if this area is to find its way into the field of industrial applications.

[7]More experiments are currently being carried out, in order to gather a significant sample.

## Acknowledgment

AutonoMouse IV was designed and built by Franco Dorigo. The experiments were run: on AutonoMouse V, by F. Dorigo and A. Maesani; on HAMSTER, by E. Radice; on CRAB, by M. Papetti. We wish to thank L. Meeden for the useful comments on an earlier draft version of this article.

## References

[1] R. C. Arkin, "Integrating behavioral, perceptual, and world knowledge in reactive navigation," *Robot. Auton. Syst.*, vol. 6, no. 1–2, pp. 105–122, 1990.
[2] R. D. Beer and J. C. Gallagher, "Evolving dynamical neural networks for adaptive behavior," *Adaptive Behavior*, vol. 1, no. 1, pp. 92–122, 1992.
[3] L. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," *Artif. Intell.*, vol. 40, no. 1–3, pp. 235–282, 1989.
[4] R. A. Brooks, "Intelligence without representation," *Artif. Intell.*, vol. 47, no. 1–3, pp. 139–159, 1991.
[5] P. V. C. Caironi and M. Dorigo, "Training $Q$-Agents," Tech. Rep. IRIDIA/94-14, Université Libre de Bruxelles, Belgium, 1994.
[6] M. Colombetti and M. Dorigo, "Training agents to perform sequential behavior," *Adaptive Behavior*, vol. 2, no. 3, pp. 247–275, 1994.
[7] F. Dorigo and A. Maesani, "Realizzazione e controllo di un robot mobile: integrazione di tecniche di progetto e di apprendimento automatico," *Master's Thesis*, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy, 1993.
[8] M. Dorigo, "Genetic and nongenetic operators in ALECSYS," *Evolutionary Comp. J.*, vol. 1, no. 2, pp. 151–164, 1993.
[9] ——, "ALECSYS and the AutonoMouse: Learning to control a real robot by distributed classifier systems," *Machine Learning*, vol. 19, no. 3, pp. 209–240, 1995.
[10] M. Dorigo and H. Bersini, "A comparison of $Q$-learning and classifier systems," in *Proc. From Animals to Animats, Third Int. Conf. on Simulation of Adaptive Behavior (SAB94)*, 1994, pp. 248–255.
[11] M. Dorigo and M. Colombetti, "Robot shaping: Developing autonomous agents through learning," *Artif. Intell.*, vol. 71, no. 2, pp. 321–370, 1994.

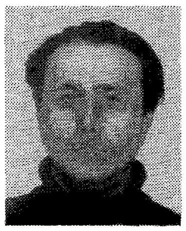
[12] ——, "The role of the trainer in reinforcement learning," in *Proc. MLC-COLT '94 Workshop on Robot Learning*, New Brunswick, NJ, July 1994, pp. 37–45.
[13] M. Dorigo, M. J. Patel, and M. Colombetti, "The effect of sensory information on reinforcement learning by a robot arm," in *Proc. ISRAM'94, Fifth Int. Symp. Robotics and Manufacturing*, Maui, HI, Aug. 1994, pp. 83–88.
[14] M. Dorigo and U. Schnepf, "Genetics-based machine learning and behavior-based robotics: A new synthesis," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 1, pp. 141–154, 1993.
[15] M. Dorigo and E. Sirtori, "ALECSYS: A parallel laboratory for learning classifier systems," in *Proc. Fourth Int. Conf. Genetic Algorithms*, 1991, pp. 296–302.
[16] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. of Michigan Press, 1975.
[17] ——, "Adaptive algorithms for discovering and using general patterns in growing knowledge-bases," *Int. J. Policy Anal. Inform. Syst.*, vol. 4, pp. 217–240, 1980.
[18] ——, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in *Machine Learning II*, R. S. Michalsky, J. G. Carbonell, and T. M. Mitchell, Eds. San Mateo, CA: Morgan Kaufmann, 1986.
[19] J. C. Latombe, *Robot Motion Planning*. Dordrecht: Kluwer, 1991.
[20] P. Maes, "A bottom-up mechanism for behavior selection in an artificial creature," in *Proc. From Animals to Animats: The 1st Int. Conf. on Simulation of Adaptive Behavior SAB-90*, 1990, pp. 239–246.
[21] P. Maes and R. A. Brooks, "Learning to coordinate behaviors," in *Proc. 8th Nat. Conf. on Artificial Intelligence AAAI-90*, 1990, pp. 796–802.
[22] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artif. Intell.*, vol. 55, no. 2, pp. 311–365, 1992.
[23] W. W. Royce, "Managing the development of large software systems," in *Proc. WESTCON*, CA, 1970.
[24] A. Saffiotti, K. Konolige, and E. Ruspini, "A multivalued logic approach to integrating planning and control," *Artif. Intell.*, vol. 76, no. 1–2, pp. 481–526, 1995.
[25] C. J. C. H. Watkins, "Learning with delayed rewards," *Ph.D. Dissertation*, Dept. Psychology, Univ. of Cambridge, Cambridge, England, 1989.
[26] S. D. Whitehead, "A complexity analysis of cooperative mechanisms in reinforcement learning," in *Proc. 9th Nat. Conf. Artificial Intelligence (AAAI-91)*, 1991, pp. 607–613.

**Marco Colombetti** was born in Milan, Italy, in 1951. He received the Laurea (Master of Technology) degree in electronics engineering in 1976 from the Politecnico di Milano, Milan, Italy.

Since 1991, he has been an Associate Professor of Computer Science, Engineering Faculty, Politecnico di Milano, where he currently teaches a course on knowledge engineering and expert systems. His research interests include knowledge representation and behavior engineering, with particular regard to the application of machine learning techniques to the development of robot behavior.

He is a member of the Artificial Intelligence and Robotics Project of the Politecnico di Milano, the Italian Association for Artificial Intelligence (AI*IA), and took part in several European and National research projects.

**Marco Dorigo**, (S'92–M'93) for a photograph and biography, see this issue, p. 364.



**Giuseppe Borghi** (S'95) was born in Milan, Italy, 1964. He received his Laurea degree in electronics engineering in 1992 from the Politecnico di Milano, Milan, Italy, with a thesis on intelligent control of mobile robots based on distributed control architectures. Currently, he is a Ph.D. student in industrial electronics and computer science, Dipartimento di Elettronica e Informatica, Università degli Studi di Padova, Italy, and he collaborates with the LADSEB-CNR Laboratory, Padova.

Since 1992, he has been member of the Artificial Intelligence and Robotics Project, Politecnico di Milano. He is involved in several National research projects concerning autonomous mobile robotics. His current research interests include map learning, motion planning based on the potential field approach, application of machine learning to autonomous robotics, and optimal sensor exploration for mobile robot self-localization.