Learning Reactive and Planning Rules in a Motivationally Autonomous Animat

Jean-Yves Donnart and Jean-Arcady Meyer

***Abstract*—This work describes a control architecture based on a hierarchical classifier system. This system, which learns both reactive and planning rules, implements a motivationally autonomous animat that chooses the actions it performs according to its perception of the external environment, to its physiological or internal state, to the consequences of its current behavior, and to the expected consequences of its future behavior. The adaptive faculties of this architecture are illustrated within the context of a navigation task, through various experiments with a simulated and a real robot.**

I. Introduction

The work presented in this paper fits into the so-called *animat approach*, which aims at designing *animats*, i.e., simulated animals or real robots whose rules of behavior are inspired by those of animals. The proximate goal of this approach is to discover architectures or working principles that allow an animal or a robot to exhibit an adaptive behavior and, thus, to survive or fulfill its mission even in a changing environment ([13], [35], [36]). The ultimate goal of this approach is to embed human intelligence within an evolutionary perspective and to seek how the highest cognitive abilities of man can be related to the simplest adaptive behaviors of animals ([37], [40], [54]).

An animat is usually equipped with sensors, with actuators, and with a control architecture that allow it to react or to respond to variations in its external or internal environment, notably to those that might impair its chances of survival. This paper describes the control architecture of a *motivationally autonomous animat* ([17], [33], [34]) that enhances its chances of survival by learning to do the right thing at the right time. Action selection [31] in this animat depends upon its perception of the external environment, upon its physiological or internal state, upon the consequences of its current behavior, and upon the expected consequences of its future behavior. In other words, like an animal, such an animat is endowed with a *motivational system* that, according to Toates and Jensen, "selects [at every moment] a goal to be pursued and organizes [the animat's] commerce with it" [48]. This system is said to be *autonomous* because it is very unlikely to be completely controllable and observable by an external agent.

Such a control architecture relies upon reactive and planning rules whose expected utilities in solving a given task can be submitted to a reinforcement learning procedure, thus allowing the animat to improve over time the way it selects its actions. An originality of this architecture is that its capability of analyzing the behavior it generates allows it to learn *situated plans* ([1], [45]). Thus, although the animat's moment-to-moment decisions can be taken at a certain level of abstraction and depend upon a global view of the survival problem to be solved, they are also reactive to the actual state of the environment that, in turn, can modify the expected utilities of the memorized plans. Such adaptive capacities should prove to be very general and to be exhibited in a variety of future applications. They will be illustrated here within the context of a simple navigation task through various simulations. The operational value of these capacities will then be demonstrated through equivalent experiments with a robot.

II. The Control Architecture

In order to survive in a complex environment, an animat may rely on various reinforcement signals that indicate whether or not a given action, or sequence of actions, enhances its chances of survival in given circumstances. However, although various algorithms - like Holland's *bucket brigade* [24], Sutton's *temporal difference learning* [46] or Watkins's *Q-learning* [50]- can be used to let the animat learn how to choose a favorable action in preference to an unfavorable one in each possible circumstance, such a learning process turns out to be very slow. This is due to the fact that reinforcement signals are generally provided in a few particular environmental states and that the step by step propagation of the corresponding information to every other state through a flat organization takes a long time. Among the solutions that have been suggested to improve learning speed - like *generalization procedures* [12] or the use of an *action model* [47] - the most promising seem to be those that advocate a hierarchical organization of state transitions[1], within which the propagation of reinforcement signals is expedited.

The hierarchical control architecture presented here is called MonaLysa[2] and relies upon the same principles. However, the hierarchy it implements, instead of being fixed by the designer ([18], [27],[29], [42], [53]), can be dynamically reconfigured - thus enhancing the animat's adaptive capacities. In particular, the possibility of introducing or sup-

The authors are with the Groupe de BioInformatique of the Ecole Normale Supérieure. 46, rue d'Ulm. 75230 Paris Cedex 05. France (e-mail : donnart@wotan.ens.fr - meyer@wotan.ens.fr) (URL : http://www.ens.fr:80/bioinfo/www/francais/perso/donnart/donnart.html http://www.ens.fr:80/bioinfo/www/francais/perso/meyer/meyer.html)

[1]The operators of such transitions, which will be referred to as *tasks* hereafter, are called *behavioral modules* [53], *skills* [29], *abstract models* [42] or *macro-operators* [27] elsewhere.

[2]MOtivatioNAlLY autonomouS Animat.

pressing tasks within the current hierarchy is afforded by a mechanism that allows the survival value of these tasks to be monitored, according to a heuristic criterion of *internal satisfaction*.

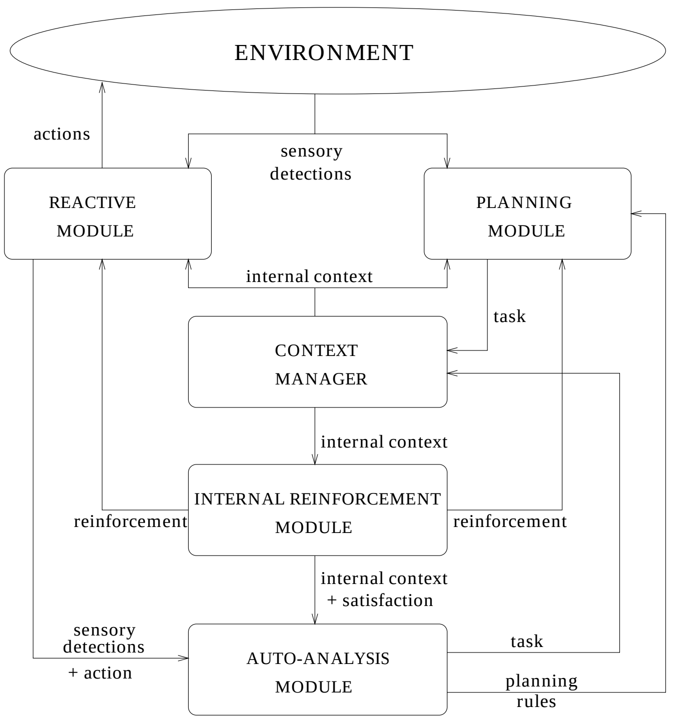

Fig. 1. The MonaLysa architecture

In the present application, the MonaLysa architecture enables a simulated robot to explore a two-dimensional environment that may contain various assorted materials, in particular some obstacles, and to navigate from a given initial place to a given goal place despite such obstacles. The robot is equipped with proximate sensors that keep it informed of the presence or absence of any obstacle in front of it, 90° to its right, or 90° to its left. It is also able to estimate the spatial coordinates of the position it is located in and the direction of a goal to be reached in each of the eight sectors of the space surrounding it. Lastly, it is capable of moving straight ahead, 90° to its right, or 90° to its left. The size of each such move is equal to the length of the robot.

This architecture, which relies upon a hierarchical *classifier system* [24], is organized into five modules - a reactive module, a planning module, a context manager, an auto-analysis module and an internal reinforcement module (Figure 1). It allows the robot to reactively escape from any obstacle it gets trapped into by skirting around it, and to analyze the corresponding skirting path in order to plan a trajectory that will later allow it to avoid the obstacle from a distance.

The role of the reactive module is to decide what action to perform next. The corresponding decisions depend not only upon the robot's sensory detections, but also upon the direction of the robot's current goal. The latter is specified by the robot's current task that is posted on the context manager module either by the planning or the auto-analysis modules. Thus, the context manager contains the pile of all the tasks the animat has to perform. The role of the planning module is to use its planning rules to decompose a given task into simpler tasks. The auto-analysis module either generates a skirting task that is posted on the context manager module when an obstacle is detected, or generates avoidance tasks that are converted into planning rules sent to the planning module. The internal reinforcement module is used to monitor the internal satisfaction of the robot and to provide a reinforcement signal to the reactive and planning modules.

A. The Reactive Module

This module is responsible for choosing which action the robot performs at every moment. It contains a series of rules - or *classifiers* - that allow the robot to react to incoming sensory information from the environment, as well as to the internal context specified by the context manager. In the current configuration, the internal context is the direction of the current goal. The reactive rules take the form:

If <sensory information> and <direction of current goal>
Then <action>

For example, rule $R : 100|001 ==> 01$ can be activated when the robot becomes aware of the presence of a material element in front of it, but not on either side (information coded by "100"), and when the direction of the current goal is 45° to its right (direction "001"). If this rule is activated, the robot performs an elementary move 90° to its right (action "01"). In the simulations to be shown below, to each pair of conditions on the *<sensory information>* and the *<direction of current goal>* correspond, at any moment, three rules capable of being actuated, each of which is associated with one of the three possible actions. The choice of which rule is actually triggered is effected probabilistically on the basis of the *strength* - that will be discussed later (see Section II.E) - of each of the three rules involved. In the case of this simulated robot, there are $8*8*3 = 192$ possible reactive rules. These rules are all created when the system is initialized and are charged into the reactive module, which does not change in size as long as the system is in operation. The strength of each rule is initialized to a given value and is subsequently modified by learning. In other experiments, yet unpublished, a genetic algorithm [20] has been used successfully in the interest of discovering more general rules.

B. The Planning Module

The role of the planning module is to decompose a task into a series of subtasks according to current sensory information. The planning rules it contains take the form:

If <sensory information> and <current task>
Then <subtask>

Sensory information is provided by the sensors and comes from the environment. It consists of information supplied by the proximate sensors and of the robot's coordinates and current orientation. As explained below, the <*current task*> is the one registered at the top of the pile of tasks governed by the context manager and, when a planning rule is triggered, the corresponding <*subtask*> is put above the <*current task*> on the context manager's pile. Such a procedure will be called *task decomposition* hereafter.

The <*current task*> and the <*subtask*> are each coded as a pair of coordinates, which respectively define an initial position and a final position - i.e., a *goal* - to be reached. Thus, rule $P : 5,1|001|000|3,0;3,5 ==> 5,1;5,2$ can be activated at position 5,1 if the robot is headed in a north-easterly direction (coded by "001"), if it perceives no material element ahead or on either side of it (coded by "000"), and if its current task is to reach the position with coordinates 3,5 when starting at position 3,0. If this rule is activated, the subtask that involves reaching position 5,2 from position 5,1 will be placed on top of the context manager pile.

With each rule of the planning module are associated two strengths - one local, the other global - the evaluation of which will be explained later (see Section II.E). The local strength is used to determine the probability of triggering a rule whose condition part matches the current situation. When the system is initialized, the planning module is empty. During operation, this module can dynamically acquire rules - generated by the auto-analysis module - or loose rules - according to how the global strengths of these rules evolve. The size of the planning module thus varies over time, though it cannot exceed a preestablished upper limit.

C. The Context Manager

The context manager provides an internal context to the other modules and influences their inner working. This internal context is the direction of the robot's current goal in the case of the reactive module; it is the robot's current task in the case of all other modules. The context manager consists essentially of a pile of tasks, at the top of which is the system's current task. New tasks are added to the pile either by the auto-analysis module, when an obstacle is detected, or by the planning module, as just mentioned. The former allow the robot to escape from an obstacle by skirting around it, while trying to cross specific lines that characterize the obstacle. The latter allow the robot to avoid an obstacle from a distance, while passing through specific positions characterized along the obstacle.

The context manager includes an algorithm that transforms the current task into a goal, then supplies the direction of this goal to the reactive module. In the case of a task posted by the planning module, the corresponding goal is simply described by the coordinates of the final position to be reached. In the case of a task posted by the auto-analysis module, the corresponding goal is described by the coordinates of the projection of the robot's current location on the line that must be crossed to skirt around the obstacle. This projection, and accordingly the corresponding direction information, varies whenever the robot moves.

D. The Auto-Analysis Module

The role of the auto-analysis module is to analyze the current behavior of the robot in order to alter its current task dynamically and to create new tasks that will enhance its behavior in the future. In other words, the auto-analysis module is responsible for detecting obstacles, for triggering skirting behaviors and for characterizing *salient states* in the environment, through which it will be useful to travel in the future in order to avoid the obstacles from a distance.

Obstacle Detection:

Within the present application, an *obstacle* is any material element that prevents the execution of the best action the robot can perform in order to move in the direction of its current goal. To escape from such an obstacle, the robot must skirt around it. However, it may happen that some material elements - like a wall at the periphery of the area that is explored - can be detected in the vicinity of the robot by its proximate sensors without impeding movement in the direction of the current goal. In such a case, no specific skirting behavior needs to be triggered. Therefore, the detection of an obstacle and the triggering of the relevant skirting behavior depend upon the following procedure. When the proximate sensors detect a material element in a given position, the robot first searches which reactive rule has the greatest strength among the three rules that could be actuated in the context of the current goal's direction if no material element were actually detected. If the move that this rule would trigger cannot be executed because of the presence of the material element, this element is recognized as an obstacle at the position where that move would lead, and a subsequent skirting subtask is sent to the context manager. Recursively, if the move that corresponds to the second best rule cannot be executed, a second obstacle is recognized at the corresponding position, and so on. Whether the material element has been recognized as an obstacle or not, the robot then probabilistically chooses a rule that matches both the <*sensory information*> in the presence of the material element and the <*direction of current goal*>. If the material element doesn't prevent the corresponding action from being executed, the action is executed and the strength of the rule is changed as explained in section E. If the action cannot be executed, the strength of the rule is set to zero and other reactive rules matching the <*direction of current goal*> in the presence of a material element are probabilistically selected in turn, until one is found that allows motion.

The reason why obstacle detection involves two rule matching procedures, one that takes the actual <*sensory information*> into account and one that operates as if this information were set to "000", is that the strength of rules triggered according to the former procedure can be equal to zero -

thus indicating that these rules would lead the robot in the direction of the obstacle, an action that is not suitable. A zero strength prevents the corresponding rules from being probabilistically selected for action and, if such rules cannot be executed, they cannot be used to recognize the obstacle, nor to skirt around it. Hence, the necessity of relying on the second rule matching procedure for these purposes.

Skirting Behavior:

When an obstacle is detected, the auto-analysis module generates a subtask that specifies that the robot must cross the line that lies parallel to the direction of the move performed, and that passes through the position where the obstacle has been detected. This subtask is coded by the pair <*coordinates of the place*> <*direction vector of the straight line to be crossed*> and is registered at the top of the pile of the context manager. An *emergent functionality* [44] of such subtasks is to enable the robot to skirt around the obstacles it encounters. For example, in the case of Figure 2a, an obstacle is detected in front of the robot when it arrives at position 1. If the robot probabilistically chooses to turn left to try to avoid the obstacle, the task of having to cross the straight line *δ1* is placed on top of the current task - which corresponds to the goal direction *d0* - and the current goal becomes the position marked with a "?" on the figure, in direction *d1*. After its move to the left, the robot arrives at position 2. The current goal, with which direction *d2* is associated, becomes that in Figure 2b, but the pile of tasks doesn't change. If, at position 2, the robot chooses to move forward, it arrives at position 3, and the current goal becomes that of Figure 2c. In this position, if the robot chooses to turn right, it reaches the current goal and the task associated with *δ1* is erased. Then, the robot can resume pursuing its initial goal, in direction *d4* (Figure 2d).

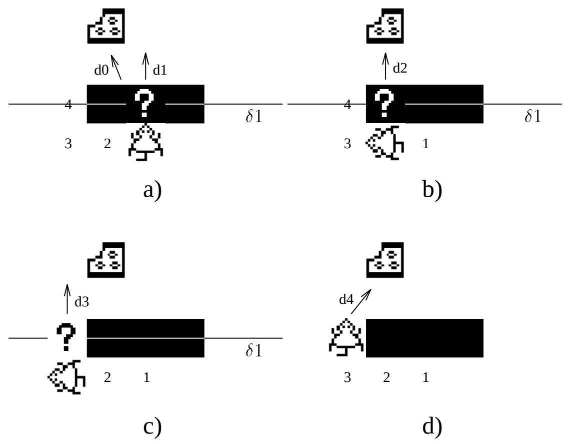

Fig. 2. Different stages in a skirting behavior involving one skirting task.

Following this reasoning, had the robot encountered an obstacle perpendicular to the preceding, preventing it, for example, from reaching position 3, the task of having to cross the line *δ2* would have been placed on top of the task associated with *δ1*. The robot would thus have been directed successively through positions 3 and 4 (Figure 3). In position 4, it would have been able to turn right and reach position 5, where the task linked with *δ2* would have been erased. Likewise, the task linked with *δ1* would in turn have been erased at position 9.

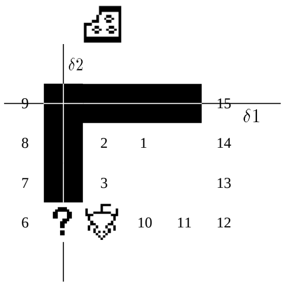

Fig. 3. Skirting behavior involving two skirting tasks.

In some situations, it can happen that the robot, seeking to attain the current goal, erases a task placed farther down than the current task in the pile. In this case, all the tasks situated above the erased task are also erased, as they were generated for the sole purpose of executing this task and no longer have any justification. For instance, if in the situation shown in Figure 3, the robot chooses to move through positions 10 to 15, it will erase the task of having to cross line *δ1* at position 15. As the task of having to cross line *δ2* will no more be justified by the necessity of crossing *δ1*, this task will also be erased from the pile of the context manager.

It will be demonstrated later on that such mechanisms enable the robot to skirt around obstacles and to extricate itself from dead-ends with arbitrarily complicated shapes.

Detection of Salient States:

Another role of the auto-analysis module is to analyze the trajectory followed by the robot in order to detect recursively the *salient states*[3] in the environment. To accomplish this, the module evaluates the internal satisfaction of the robot after each completed action - that is, the success with which this rule brought the robot closer to, or took it farther from, the current goal - and calculates the variation of this satisfaction between two successives actions. At positions where the corresponding gradient is positive, the analysis module detects *satisfaction states*, which are added to the initial and final states of the path in question. These satisfaction states are only detected when the context manager pile contains a skirting task.

The recursive process executed by the auto-analysis module applies first of all to the path actually travelled by the robot, then to the successive fictitious paths that can be abstracted from the satisfaction states detected on these

[3]In a purely navigational task, one could have used the word "landmarks" instead. We prefer to refer to "salient states", as our approach aims at solving more general tasks.

paths. Such a process is thus an illustration of the metaphor which conceives planning as a series of "thought experiments" [15]. When the path obtained by direct connection of the satisfaction states detected on the preceding path generates the same sequence of satisfaction states, the recursion is stopped, and the last satisfaction states discovered are recognized as salient states.

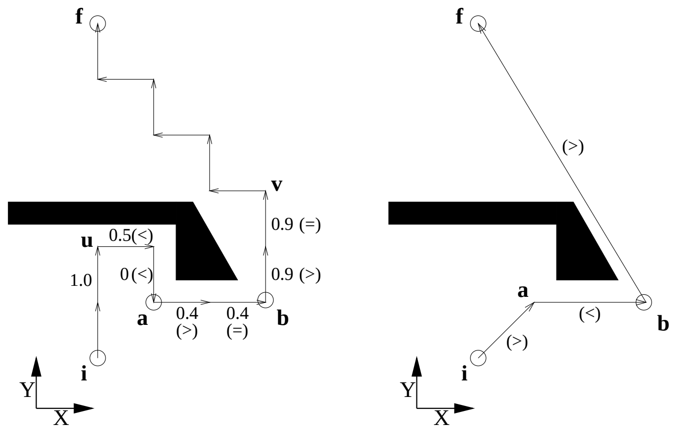

Fig. 4. Detection of satisfaction and salient states. Numerical values indicate the satisfaction brought by each action. $>$, $<$ and $=$ symbols indicate the sign of the satisfaction gradient.

Thus, in the case of Figure 4, the robot has accomplished fourteen actions in order to reach position *f* from position *i*, and it was under the control of a skirting task between positions *u* and *v*. As the gradient of satisfaction is positive at *a* and *b*, the auto-analysis module generates four satisfaction states: *i, f, a* and *b*. At the next stage, the satisfaction gradient associated with each fictitious action, which makes it possible to progress from one satisfaction state to the next, is computed. State *a* can then be eliminated, as the gradient is negative between *a* and *b*. Because no other satisfaction state can be eliminated along the path directly connecting the three remaining states with one another, the recursive process stops at this stage, and the analysis module recognizes and memorizes three salient states, *i, f* and *b*, each characterized by its coordinates, by the sensory information obtained by the robot at this state, and by the corresponding orientation of the robot.

The salient states are then used to generate planning rules. Thus, at the conclusion of the path described in Figure 4, the two rules *P1* and *P2* that decompose path *i-f* into *i-b* and *b-f*:

P1: $1,1|000|000|1,1;1,7 ==> 1,1;4,2$
P2: $4,2|010|000|1,1;1,7 ==> 4,2;1,7$

are created and input to the planning module. The advantage of the redundancy in the description of these rules is to make it possible to recognize the salient states, even when there are imprecisions in the coordinates, or in the sensory information, or in the orientation of the robot, or when the robot reaches one of these states with a new orientation.

E. The Internal Reinforcement Module

The internal reinforcement module works through a process of reinforcement which causes the strengths of the rules of the reactive and planning modules to change. Within the reactive module, this internal reinforcement takes place each time a rule is used and depends on the satisfaction of the robot, that is, on an estimation of the success with which this rule brought the robot closer to, or took it farther from, the current goal:

$$S(R)_{u+1} = (1-\alpha) * S(R)_u + \alpha * satisf$$

where

$$satisf = \frac{dist\_goal(u) - dist\_goal(u+1) + max\_dist}{max\_dist * 2}$$

$S(R)_u$ *: strength of rule R after u triggers*
$S(R)_0$*,* α*,* max_dist *: parameters of simulation (set respectively to 0.1, 0.1 and to the robot's length in the experiments described herein)*
$dist\_goal(u)$ *: estimated distance to the current goal.*

If *max_dist* evaluates the maximum distance by which any move can bring the robot closer or farther from its current goal, the satisfaction brought by a given move varies between 0 and 1. The strength of a given reactive rule is an evaluation of the average satisfaction brought by the move it generates and this evaluation is performed by an *incremental learning* procedure, which is in line with the arguments of biological plausibility put forth by Wilson [52] and Holland [25]. As already mentioned, the strength of a rule that is selected for action and that would move the robot through a material element is set to zero.

Differently from the reactive module in which rules have a single associated strength, the rules of the planning module are characterized by two strengths: a local strength and a global strength. The local strength evaluates the usefulness of decomposing a task into a subtask proposed by the rule. The global strength, on the other hand, is used to detect and suppress the rules which are unlikely to be used by the system.

To calculate the local strength of a rule P1 that decomposes, for example, a task T - such as that of going from *i* to *f* in Figure 5 - into a subtask T1 - such as that of going from *i* to *j* - the reinforcement module evaluates the average cost of all the paths tested by the robot which enable it to reach *f* from *i* without resorting to any decomposition. In the present version of the system, this cost, denoted by AC(T), is an average of the lengths of the paths expressed in terms of the number of elementary moves they necessitate. AC(T) is evaluated incrementally:

$$AC(T)_{u+1} = (1-\alpha) * AC(T)_u + \alpha * C$$

where
u : number of times that task T was achieved without using any planning rule
C : cost (number of moves) of the u+1th path

α : parameter (set to 0.1 in the experiments described herein).

The reinforcement module also evaluates the average cost of all the paths joining *i* to *f* and using rule *P1*, that is, that reach *j* from *i*:

$$AC(P1)_{u+1} = (1-\alpha) * AC(P1)_u + \alpha * C$$

where
u : number of times that task T was achieved using planning rule P1
C : cost (number of moves) of the u+1th path
α : parameter (set to 0.1 in the experiments described herein).

The cost C of the path covered is used as an internal reinforcement and is dispatched to all the rules P1, P2,... that decomposed T. This succession of rules is thus memorized, and a *profit sharing* algorithm [21] is called on to change their average costs AC(P1), AC(P2),... Under these conditions, the local strength of P1, LS(P1), is updated according to the equation:

$$LS(P1) = \frac{AC(T)}{AC(P1)}$$

The shorter the paths joining *i* to *f* and passing through *j* have turned out to be than the paths joining *i* to *f* by way of other paths, the stronger this strength is. For example, while following paths 1 and 2 under the control of task T in Figure 5, the robot has characterized six salient states - *i, j, k, l, m* and *f* - and created six planning rules - P1, P2, P3, P4, P5 and P6 - detailed in Figure 6. When task T has been decomposed into the subtasks associated with these planning rules, paths 3, 4 and 5 have been followed by the robot. The lengths of paths 1 to 5 being respectively of 29, 39, 23, 27 and 23 moves, it follows that AC(T) = (29 + 39)/2 = 34 [4], because AC(T) is the average cost of all the paths - like path 1 and path 2 - that lead the robot from *i* to *f* without using any planning rule. AC(P1) is the average cost of all the paths - like path 3 and path 5 - that lead the robot from *i* to *f* via *j*, under the control of the planning rule P1. Therefore, AC(P1) = (23 + 23)/2 = 23. Likewise, P4 being a rule that decomposes T into the subtask of going from *i* to *l*, AC(P4) is the cost of path 4 and equals 27. It then follows that LS(P1) = 34/23 = 1.478 and that LS(P4) = 34/27 = 1.259. Because LS(P1) turns out to be greater than LS(P4), the robot in position *i* is more likely to choose moving in the direction of *j* than in the direction of *l*. In other words, after having been able to reactively escape from the obstacle via paths 1 and 2 and after having detected the salient states *j* and *l* on these paths, the robot is now capable of directing itself towards each of these states from state *i* and, thus, of avoiding the obstacle from a distance. Moreover, it is capable of choosing the most advantageous among these avoidance trajectories.

[4]For a clearer comprehension of this section, the average values that are actually incrementally estimated by the robot are computed here in a direct way.

Each time a task generated by the planning module is at the top of the context manager pile, the planning module can decide whether or not to decompose this task and to trigger one of various planning rules *P1, P2, ...* it contains according to current sensory information. The decision to decompose is made on the basis of a probabilistic choice depending on the local strength of each rule (a local strength of 1 being assigned to the non-decomposition option), weighted by an exploration-exploitation coefficient. The role of this coefficient is to modify, in the course of operation, the probability that the system will apply rules with a high local strength. Its value is under the experimenter's control for the time being and varies between 0 (pure exploitation mode) and 1 (pure exploration mode). It could subsequently be made to depend upon the progress of the simulation.

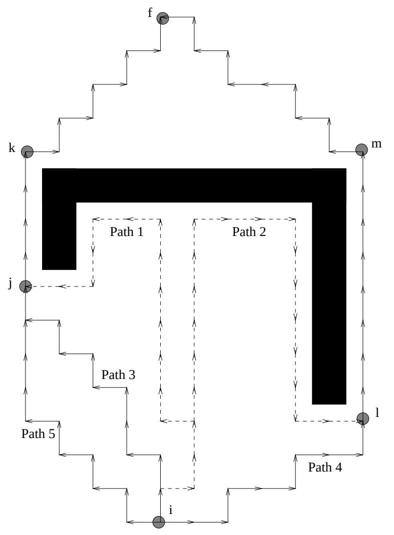

Fig. 5. Several trajectories in a specific environment.

It is clear that, whatever the local strength of rules P1, P2,... that decompose task T, these rules are unlikely to be triggered if task T itself is unlikely to be posted on the context manager pile. The global strength of each rule allows this type of situation to be detected and the less useful rules to be eliminated whenever the size of the planning module exceeds the maximum permissible threshold. A procedure that leads to coherent results consists in evaluating the global strength of P1 that decomposes T into T1 according to the equation:

$$GS(P1) = LS(P1) * GS(T)$$

In such an equation, the global strengh of a task T, GS(T), is the average of the global strengths of all the rules P liable to post T on the context manager pile, this average being calculated incrementally:

$$GS(T)_{u+1} = (1-\alpha) * GS(T)_u + \alpha * GS(P)_{v+1}$$

where
u : number of times that task T was achieved
v : number of triggers of rule P
α : parameter (set to 0.1 in the experiments described herein).

In the present implementation, when the system is initialized, a principal task PT - for example, going from position i to position f - is assigned to the robot, and the corresponding global strength GS(PT) is set arbitrarily at 1000. This amounts to considering that an external reinforcement of 1000 (in arbitrary units) is provided in position *f*. In future implementations, the robot will discover by itself the positions where reinforcements are provided and will autonomously assess the values of these reinforcements. Whatever the case, the above equations allow the global strength of the principal task to be propagated down the hierarchy of tasks and subtasks by means of the corresponding planning rules and, if a given task T happens to be generated only rarely, its global strength will be weak, as will those of all the rules that decompose T.

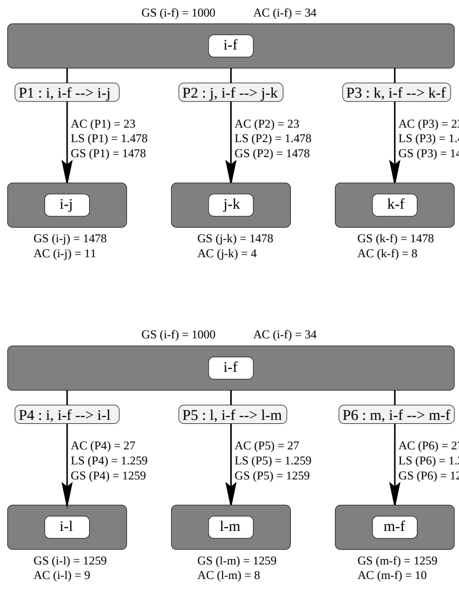

Fig. 6. Tasks and planning rules associated with Figure 5.

Figure 6 describes the tasks and planning rules that are associated with Figure 5. Tasks are characterized by their initial and final positions; planning rules are arrows decomposing tasks into subtasks. Figure 6 also gives the local and global strengths of the planning rules, the global strengths of the tasks, and the associated costs of both rules and tasks. For example, because LS(P1) = 1.478 and GS(PT) = GS(*i-f*) = 1000, it follows that GS(P1) = 1.478 * 1000 = 1478. Then, because the global strengths of rules P4, P5 and P6 are smaller than those of rules P1, P2 and P3, the former rules have a higher likelihood of being eliminated from the planning module than the latter, in the event of memory overflow.

However if, in this situation, a new obstacle is introduced in the environment, as in Figure 7, the robot is committed to skirting around the new obstacle. Having followed paths 6 and 7 under the control of task *i-j*, it characterizes four additional salient states - *n, o, p* and *q* - and creates six new planning rules - P7 to P12 - of which only the first three are detailed in Figure 8. When task *i-j* has been decomposed into the subtasks associated with these planning rules, paths 8 and 9 have been followed by the robot. The lengths of paths 6 to 9 being respectively of 31, 31, 29 and 29 moves, it follows that AC(P1) is now equal to the mean of these four values, i.e., to 30. Therefore, LS(P1) being now equal to 34/30 = 1.133, its value turns out to be lower than that of LS(P4), i.e. 1.259. As a consequence, there are now more chances that the robot at position i will try to avoid the obstacle in Figure 7 by turning in the direction of state l than by turning in the direction of state j. In other words, in the presence of the new obstacle, the robot will now tend to avoid the initial obstacle by turning to the right instead of turning to the left.

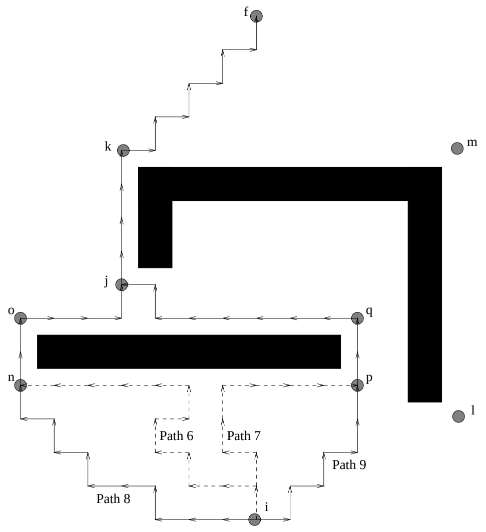

Fig. 7. Several trajectories in the environment of Figure 5 with a new obstacle added.

Figures 6 and 8 also illustrate how task decomposition proceeds within the MonaLysa architecture. In particular, they show that such decomposition is modular, contrary to what happens with classical planning procedures. In the latter case, indeed, the decomposition of a task like *i-f* into the sequence *i-j-k-f* is made once and for all during the planning phase, and it is up to the execution algorithm to figure out a means for successively reaching positions *j, k* and *f* from *i*. Once the execution of such a sequence has started, there is no opportunity to decide, for instance, to go to position other than *k* when arriving at *j*. In contrast, the procedure used here allows for several different decompositions of a task like *i-f*. Thus it becomes possible to decompose *i-f* into *i-j* according to rule P1 and, when arriving at *j*, to opportunistically choose to use other rules than P2 and P3 - these rules not being shown on the figures - that will lead the robot to *f* via another position than *k*. Such opportunities for reactive planning are at the core of the adaptive capacities of the robot.

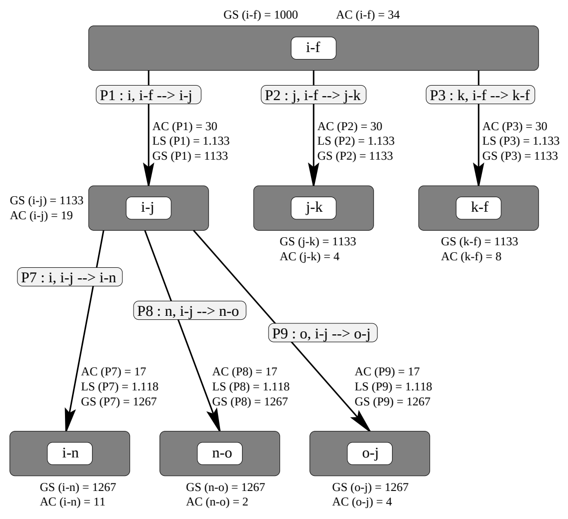

Fig. 8. Tasks and planning rules associated with Figure 7.

III. Simulated Results

This section describes the results obtained in various experiments performed in a square environment, each side of which being 40 times as long as the robot's length. The robot's main task was to reach - within given approximation limits - a goal position situated near the middle of one side, from a starting position situated near the middle of the opposite side. One or several obstacles might be placed in the environment, which forced the robot to deviate from the direct trajectory to the goal.

To illustrate the learning abilities of the robot, each experiment consisted in placing the robot in its starting position and letting it use its rules to choose where to move. When the robot succeeded in reaching the approximate goal position, the number of moves since the starting position was recorded, the robot was brought back in its starting position, and the whole process was iterated again, for a given number of loops called *iterations* hereafter. Because all the reactive rules within the control architecture had the same strengths at initialization, the first moves of the robot were chosen at random. However, because the strengths of the reactive rules that tend to bring the robot closer to its current goal were increased and because the strengths of the reactive rules that tend to bring the robot farther from the current goal were decreased, the former soon acquired a higher likelihood of being activated than the latter. Thus, according to such a learning scheme, the number of moves leading to the goal position or to any subgoal decreased from iteration to iteration and, eventually, an optimal reactive trajectory was followed by the robot.

Each experiment was run in two phases. During the first one, although planning rules were learned since the beginning, they were not allowed to modify the goal of the robot, which accordingly chose its action in a purely reactive mode. During the second phase, the planning rules were allowed to generate a succession of appropriate subgoals, which the robot, now acting in planning mode, tried to reach in turn.

Figures 9 to 11 illustrate the optimal trajectories that were followed by the robot when acting in reactive mode. These trajectories were followed in three environmental conditions, when the robot had to learn to skirt around a barrier obstacle, a dead-end obstacle, and a double-spiral obstacle, respectively.

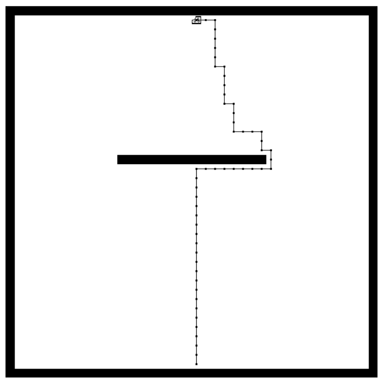

Fig. 9. Optimal reactive trajectory in an environment with a barrier.

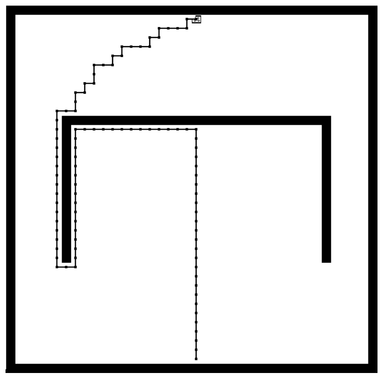

Fig. 10. Optimal reactive trajectory in an environment with a dead-end.

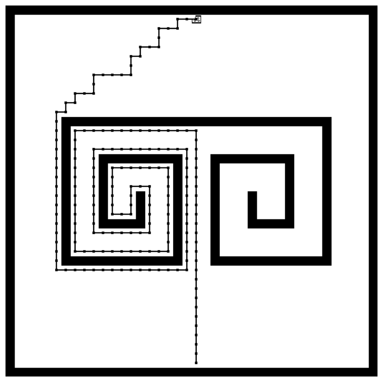

Fig. 11. Optimal reactive trajectory in an environment with a double-spiral.

Figure 12 shows the corresponding learning curves, where four experiments, all starting from scratch, have been performed during 200 iterations each. The first experiment was performed in an environment with no obstacle. The three others were performed in environments each containing one of the obstacles shown in Figures 9 to 11. Results obtained show that, in each of the four environments, the robot quickly learns to select the reactive rules that are efficient for skirting around the obstacles. Differences in the sizes and numbers of the leaps that characterize these curves indicate that, the more complex the obstacle, the greater the likelihood of probabilistically triggering suboptimal rules, and the greater the likelihood that such rules will lengthen the trajectory to the goal.

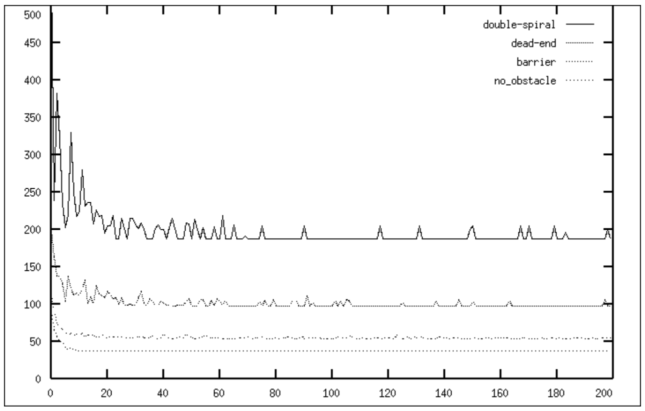

Fig. 12. Evolution of the number of moves necessary to reach the goal in four different environments (average over 10 experiments). Number of moves (ordinate) versus number of iterations (abscissa).

Likewise, Figure 13 illustrates the results obtained when the four experiments were performed consecutively, the environment being changed every 50 iterations, but with no reinitialization of the strengths of the reactive rules. Thus the robot had to capitalize on the rules learned in the previous environment to adapt its behavior to a new environment. These results show that the reactive rules that are found to be efficient in a given environment are general enough to be useful in another environment. Therefore, adaptation from one environment to another is very quick.

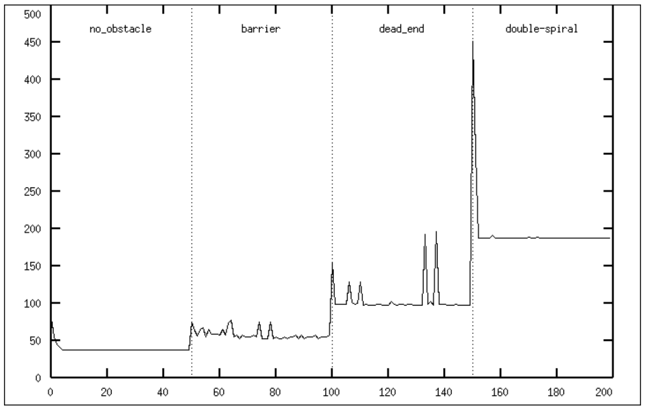

Fig. 13. Adaptation to new environments. Number of moves (ordinate) versus number of iterations (abscissa).

Figure 14 shows three planning rules discovered in the environment containing a double spiral. These rules were created while the robot was moving along the reactive trajectory of Figure 11, i.e., in a single iteration.

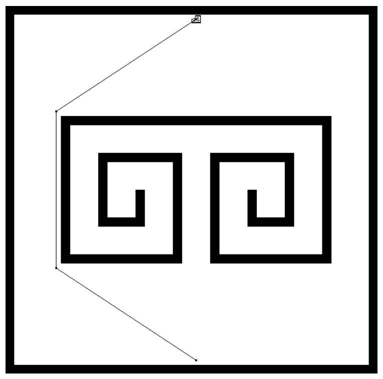

Fig. 14. Three planning rules discovered in an environment with a double spiral.

Figure 15 illustrates the actual trajectory followed by the robot in the environment with a double-spiral when acting in planning mode. From its starting position, the robot successively reached two subgoals that allowed it to avoid entering the obstacle and to finally reach its initial goal.

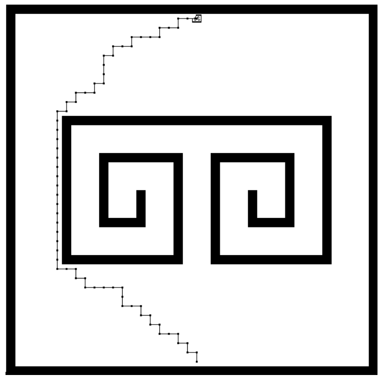

Fig. 15. Actual trajectory followed by the robot in the environment with a double-spiral when acting in planning mode.

Figures 16 to 18 illustrate the fact that the robot retains several plans in its memory and that it is continually updating the local and global strengths of its planning rules according to the procedures described in Section II.E above. Therefore, the robot can switch rapidly from one plan to another, or create new plans, and thus adapt its behavior to new obstacles appearing in its environment.

Thus, after 15 iterations in the environment depicted in Figure 16, the robot has memorized two plans for avoiding the dead-end. The best plan is shown in Figure 16, while the less effective one is shown in Figure 17. If, at iteration 16, a new obstacle is added to the environment along the robot's optimum path, the robot skirts around this obstacle, and the corresponding plan is modified accordingly. However, as the cost of this modified plan exceeds the cost of the second plan stored in memory, this second plan is the one most likely to govern the robot's path from the 22nd iteration on (Figure 17).

Likewise, introducing a new obstacle into the environment at the 30th iteration gives the advantage to the modified version of the first plan (Figure 18). It is thereby seen that the robot is capable of altering its plans as a reaction to modifications in its environment.

It should be noted that the plan presented in Figure 18 is a hierarchical plan with two levels. Indeed, the first task of the plan shown in Figure 16 has been decomposed into a three-task sub-plan. Another exemple of this faculty to generate hierarchical plans can be found in [17].

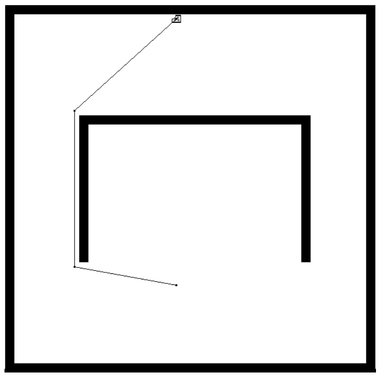

Fig. 16. The best plan in an environment with a dead-end. Iteration 15.

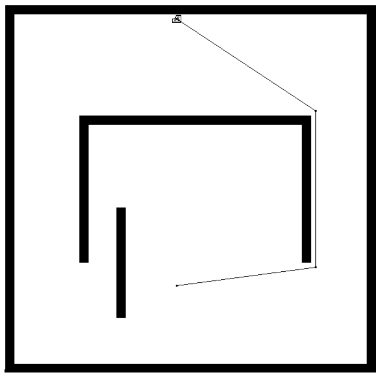

Fig. 17. The best plan in an environment with a dead-end and one obstacle added. Iteration 22.

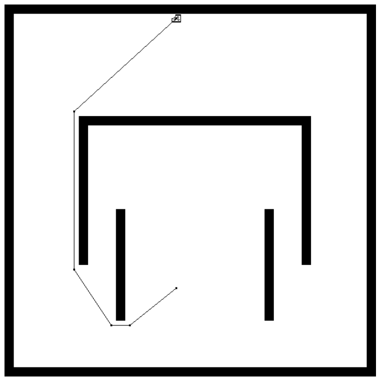

Fig. 18. The best plan in an environment with a dead-end and two obstacles added. Iteration 38.

Finally, results shown in Figure 19, as well as results published elsewhere [17], demonstrate that the decomposition of goals into subgoals can be efficient enough to devise a way of escaping from rather complex mazes.

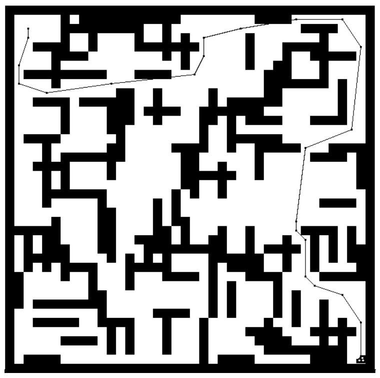

Fig. 19. Overall plan discovered in a maze.

IV. Real Robot Implementation

To reproduce the simulated experiments and results of the previous section, the MonaLysa architecture has been used with a Khepera robot ([28], [38]), shown in Figure 20. The size of each side of the square environment used for such a purpose was set at 70cm. The Khepera robot is equipped with six IR frontal sensors (with two additional sensors in the back) that can detect an obstacle within a range of approximately 5 cm, and with two wheels that can turn forward or backward. For the present application, the signals generated by the frontal sensors have been thresholded to mimick the functioning of the proximate sensors of the previous simulated robot, whose number has thus been raised from three to six. The functionality of obstacle detection has been assigned to the four sensors nearer the frontal axis of the robot, in such a way that, when the signals of two neighboring sensors were at their maximum value, an obstacle preventing any move in its direction was detected. Likewise, because the sensors of Khepera are not evenly distributed over the periphery of the robot, it sometimes happens that they are unable to detect an obstacle before the robot has turned 90° to its right or 90° to its left. Therefore, the Khepera robot was given three elementary actions: move a step forward, turn 45° to the right and move a step forward, or turn 45° to the left and move a step forward. In principle, the size of each step forward has been set at 5.5 cm, i.e., to a value that is approximately equal to the size of the robot. However, if during the corresponding move one or more sensors did detect a modification in the environment (i.e., one or more thresholded sensory readings switched from 0 to 1 or from 1 to 0), the robot stopped until a new reactive rule was selected and actuated. Thus, the actual trajectory of the Khepera robot was made of straight lines of unequal lengths, which followed each other at angles of 0° or 45°. Furthermore, an odometric device connected to each wheel allowed the robot's position and orientation to be monitored at each instant. Finally, with six binary sensors, eight goal directions and three actions, the number of rules in the reactive module of Khepera was 64*8*3 = 1536.

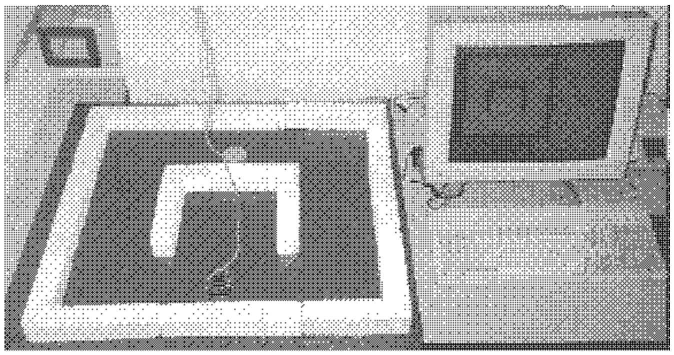

Fig. 20. The robot Khepera and the experimental setting.

Figure 21 shows the learning curves obtained when three experiments of 50 iterations each were performed consecutively. The first experiment used an environment without any obstacle, the second and third experiments used an environment with, respectively, a barrier and a dead-end obstacle. Figures 22 and 23 show the optimal reactive trajectories obtained in the presence of obstacles.

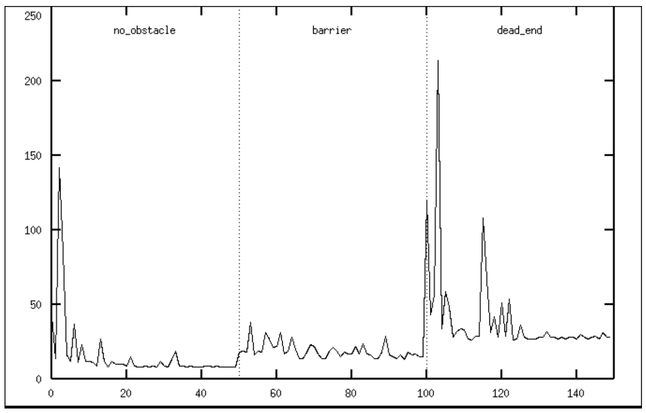

Fig. 21. Khepera's adaptation to new environments. Number of moves (ordinate) versus number of iterations (abscissa).

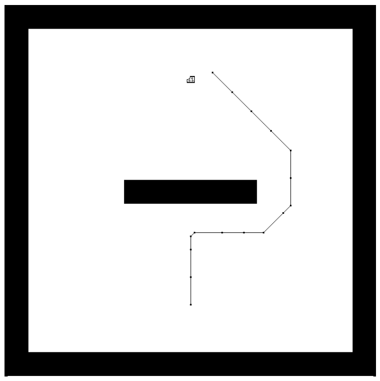

Fig. 22. Khepera's optimal reactive trajectory in an environment with a barrier.

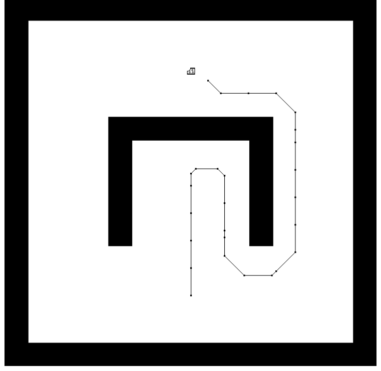

Fig. 23. Khepera's optimal reactive trajectory in an environment with a dead-end.

Results shown in Figure 21 are qualitatively similar to those obtained by simulation (Figure 13). The robot is able to use reactive rules already learned in order to adapt its behavior to new environments. Likewise, the skirting behaviors that are obtained with the simulated robot (Figures 9 and 10) and the real robot (Figures 22 and 23) are qualitatively similar.

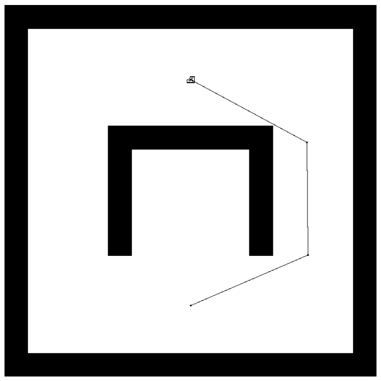

Fig. 24. Three planning rules discovered by Khepera in an environment with a dead-end.

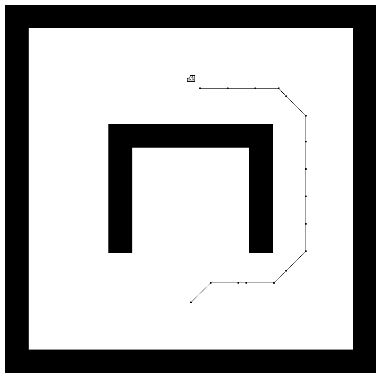

Fig. 25. Actual trajectory followed by Khepera in the environment with a dead-end, when acting in planning mode.

Figures 24 and 25 show a plan generated and a trajectory followed when Khepera was acting in planning mode. Again, the observed behaviors are qualitatively the same as those that have been obtained by simulation in a previous work [17].

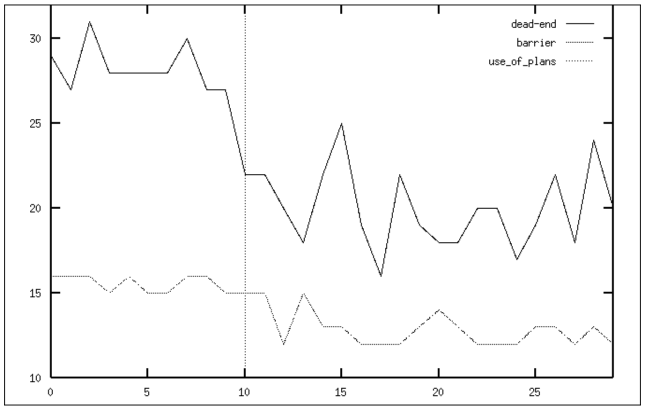

Fig. 26. Improvement of Khepera's performance after shifting to the planning mode at iteration 10. Number of moves (ordinate) versus number of iterations (abscissa).

Figure 26 shows how the performance of Khepera improved when the robot shifted from the reactive mode to the planning mode. As might be expected, a greater improvement is obtained with the dead-end obstacle, because it constitutes a more challenging obstacle than the barrier with regard to progression towards the goal.

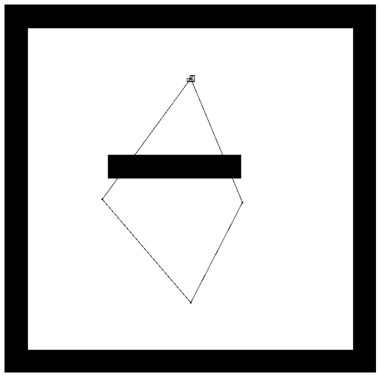

Fig. 27. The best plan (on the right) and an alternative (on the left).

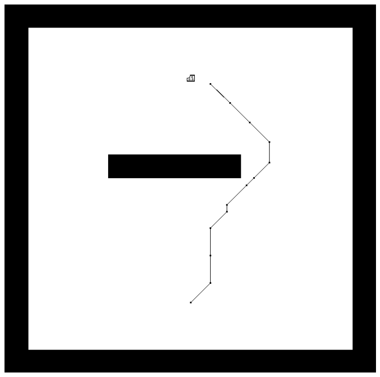

Fig. 28. A trajectory followed when using the best plan of Figure 27. Iteration 20

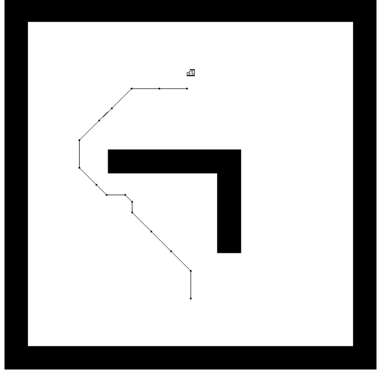

Fig. 29. A trajectory followed when using the alternative plan of Figure 27. Iteration 25

Finally, Figures 27 to 29 illustrate the ability of Khepera to switch adaptively from one plan to another. For instance, the trajectory shown in Figure 28 is that which is followed when the best plan of Figure 27 is used.

If a new obstacle is added to the environment at iteration 21, the best plan kept in memory by Khepera becomes the alternative plan of Figure 27, and this plan generates the trajectory of Figure 29, as might have been expected from the theoretical considerations in Section II.E and from the simulated results shown in Figures 16 and 17.

V. Discussion

The transfer of the MonaLysa architecture from a simulated robot to a real robot has been almost straightforward. Moreover, results obtained in both cases have been qualitatively very similar, as they were in the work of Jakobi, Husbands and Harvey [26] that also involved a Khepera robot. Therefore the common argument (e.g., [8], [9],[10], [43]) according to which no simulation will ever replace actual robot experimentation is certainly not universal, although it probably gets stronger support when a sophisticated robot is used instead of a primitive one.

The MonaLysa architecture has been conceived in order to generate very general adaptive behaviors, and current work aims at demonstrating its applicability to, for example, traditional *block-world* planning [39]. Likewise, this architecture should prove useful for managing more numerous and varied motivations than those studied here. In particular, it will be used to control the behavioral sequences of a simulated animal facing realistic survival problems, like those described by Tyrrel [49].

Whatever the case, results shown here demonstrate that the MonaLysa architecture allows expedient learning of reactive and planning rules within the context of simple navigation tasks. In particular, a robot equipped with such a control architecture is not only capable of generating and memorizing plans that help to avoid obstacles, but also of altering these plans in reaction to modifications in its environment. In the experiments described here, such modifications were caused by the addition of one or more obstacles in the environment. It turns out that the removal of obstacles leads to the same kind of results as those obtained by Sutton with the DYNA architecture [47]. In particular, MonaLysa allows the animat to discover new shortcuts leading to the goal, but the speed of such a discovery depends upon the current value of the exploration-exploitation coefficient. For instance if, in the situation depicted on Figure 17, the obstacle that has been previously added were removed, less than 100 iterations would allow the animat to favor again the plan on Figure 16, when a very low exploration-exploitation coefficient (0.1) is used. In future implementations, it will be easy to dynamically control such a coefficient and to take advantage of the fact that each salient state is characterized by an associated <*sensory information*>. Thus, the animat would notice that the value of this information shifted from "010" - indicating that the obstacle was present to the left of the robot in this state - to "000" - indicating that the obstacle has been removed. This event would automatically reset the exploration-exploitation to 1, thus increasing the likelihood of shortcut discovery.

As far as the results shown here are concerned, it should also be noted that, although the external contours of the obstacles were always piecewise linear for convenience, it has been shown elsewhere [17] that the same control architecture allows navigation in the presence of more complex obstacles. Likewise, it should also be noted that the MonaLysa architecture is capable of discovering quite intricate optimal trajectories, as exemplified by the results in Figure 19 for instance. However, because the odometric capacities of the Khepera robot are limited, the longer the path to a goal, the greater the chances of getting lost. Therefore, in order to be able to reproduce simulated results like those of Figure 19, which relied on accurate position and orientation estimates, it is necessary to endow the robot with the possibility of relocating itself in its environment. This has been accomplished by means of adequate relocalization rules in an extension of the MonaLysa architecture, and proved to be effective by simulation. Such relocalization rules predict which salient states are expected to be reached from a given salient state, under the control of a specific task, and within the uncertainty margins that are associated with the current position and orientation of the robot. In particular, the use of such rules allow the robot to treat the goal as a salient state among others, with no specific requirements about the precision of its position and direction. The implementation of this extended architecture in a Khepera robot is in progress.

Results shown here also demonstrate that learning is quite efficient within the current version of the MonaLysa architecture. For instance, Figures 12 and 13 show that less than ten iterations are usually needed to substantially decrease the number of moves necessary to reach a goal reactively, and Figures 16 to 18 show that about the same number of iterations allows the robot to switch adaptively from one plan to another. This efficiency is due to the fact that the classical problem of *temporal credit assignment* [47] is avoided in the case of the reactive rules because the strength of each such rule is updated after each utilization, thanks to the management of an internal reward based on the satisfaction criterion. Thus, the internal reinforcement module plays here the role of an *adaptive critic element* [46]. Learning efficiency is also due to the fact that the temporal credit assignment problem is minimized in the case of the planning rules because of the hierarchy of tasks and subtasks they implement. Indeed, as soon as a given task is achieved, the corresponding reinforcement is immediately forwarded to each rule that contributed to this result, and applied either to the task itself or to all its corresponding subtasks. Thus, the strenghts of many rules can be updated at reinforcement time, a logic more efficient than that of the classical *bucket brigade* algorithm [24], as demonstrated by Wilson [53]. However, although Wilson's description involved a hierarchical bucket brigade algorithm, the solution implemented here can be qualified as a *hierarchical profit sharing* algorithm.

Like MonaLysa, the control architectures described by Wilson [53], Shu and Schaeffer [41] and by Dorigo and Colombetti [18] also rely on hierarchical classifier systems, but only the first - which is a theoretical construction and has not given rise to any concrete application - might implement a planning process. Nevertheless, Wilson does not specify how the corresponding tasks and subtasks could be identified by the system. Likewise, though the architecture proposed by Colombetti and Dorigo provides for a classifier system to coordinate the actions proposed by other classifier systems, the hierarchical relationships are predetermined by the programmer. On the contrary, in the present work, the hierarchical relationships among tasks are dynamic, because they are generated internally on the basis of the experience gained by the animat.

As to planning, the MonaLysa architecture does not call on any predefined operators for decomposing problems into subproblems, for the purpose of generating a plan which would then be executed [39]. Such a practice, which implies that planning precedes acting, has shown itself to be singularly ineffective [9]. Conversely, here, acting precedes planning, and the latter does not depend on predefined operators, but rather is abstracted from the paths actually travelled. The plans thus elaborated are initially high level plans and are based on a small number of rules. However, these plans are refined as needed. They are not executed mechanically by the animat, but instead are used as one resource among others to decide which action to perform ([1], [45]). The organization of these plans thus appears as an emergent property, arising from the interactions between the animat and its environment and elicited by the animat's needs. Lastly, the value of these plans is continually reevaluated, which confers considerable adaptive faculties to the system. As already stressed in Section II.E above, these incremental changes contrast with the way other robotic realizations that involve planning solve the problem of reacting as quickly as possible to modifications in the environment. For example, within the AuRA architecture ([4], [5]), any environmental alteration changes the reactive *motor schemas* that are used to generate action, but does not explicitly modify the corresponding overall plan. If this plan does need to be changed, because the challenge of the environment is too great to be dealt with by a mere alteration of motor schemas, then the plan must be rebuilt entirely from scratch.

Albus [2], too, described a hierarchical architecture able to decompose a complex task into a series of subtasks, then into a series of elemental moves, then into a series of motor drive signals which actuate observable behavior in a robot. However, although Albus describes how such an architecture relates to a general theory of intelligence [3], he doesn't state how the corresponding hierarchy might be dynamically generated, nor how it could be modified according to the robot's needs and to the environmental conditions encountered.

In comparison with the literature on animal behavior, it must be stressed that the MonaLysa architecture is not dedicated to navigation tasks only and that it has been conceived for solving general survival problems. In fact this architecture implements a *motivationally autonomous agent* ([33], [34]) that acts in particular ways in order to achieve certain ends. To do so, the agent must decide what action to perform next, according to its physiological or internal state, to the cue state arising from its perception of the external world, to the consequences of its current behavior and to the expected consequences of its future behavior. The latter point requires knowledge of the probable consequences - or *expected utility* - of each possible action. In other words, a motivationally autonomous agent must have some memory of the past consequences of similar activities, and it must be capable of planning - i.e., it must use some form of cognition. Furthermore, as Dennett [16] pointed out, it must want something, it must have goals[5].

An animat endowed with the MonaLysa architecture displays all these characteristics. Indeed, the action it performs at any time depends both on sensors and on what was called here the "internal context". This context actually takes into account the goals that the animat has selected and that it seeks to achieve. There is nothing to prevent this context from subsequently including other information about the internal state of the animat like, for instance, its energy level. The animat's goals are generated by an explicit planning process, and the strengths of the rules memorize the consequences of the various choices that the animat has made in the past. In the navigation task, these consequences were evaluated in terms of their aptitude in bringing the animat nearer to its goal; they may later depend on an appropriate utility function and help the animat decide, for instance, whether it should seek food, seek water, or try to escape from a predator.

In comparison with other approaches aimed at including a motivational system in the architecture of an animat ([6], [7], [11], [19], [23], [30], [51]), this approach is the only one that incorporates a planning process that, as seen previously, substantially enhances the adaptive faculties of the animat. It would accordingly seem that, in the continuum described by McFarland and Bösser [34], which distinguishes *motivated automata* - that choose the action to perform next without taking into account its expected consequences - from *motivationally autonomous agents*, these other approaches tend to be situated in the former category, while the present approach would belong to the latter.

Finally, it is interesting to note that the animat's *behavioral sequences* that are triggered by its motivational system are not random, which could be demonstrated using the same methods that ethologists do [22]. These sequences are organized according to the animat's goals, so that a given action tends preferably to be followed by one action in the context of a particular goal and by another action in the context of a different goal. Such an organization is by no means arbitrary and imposed by the experimenter - like, for instance, in [14] - but rather tends to maximize the utility function. Nor is it determined once and for all, thus allowing the animat to react opportunistically to the surprises of the environment.

[5]In other words, the agent must be *goal-achieving* and *goal-seeking*. Whether its behavior is *goal-directed* or *intentional* is another issue ([16], [32]).

VI. Conclusion

This paper has shown that it is possible to endow an animat with a control architecture inspired from current knowledge about the motivational systems of animals. Such a control architecture is based upon a hierarchical classifier system. It uses reactive and planning rules, which generate both simple stimulus-response behaviors and more cognitive abilities, as demonstrated here within the context of a navigation task. In particular, it has been shown that the animat, although being equipped with very rudimentary sensors, is able to quickly learn to escape from obstacles it may get trapped into, and even to learn plans that will allow it to avoid these obstacles in the future. Moreover, these plans can be rapidly modified if the environment changes. In more general contexts, the MonaLysa architecture could prove useful in reproducing at least some of the adaptive behaviors that allow the most advanced animals to survive, even in quite unpredictible and threatening environments.

Acknowledgements

We are greatly indebted to the anonymous referees and to Marco Dorigo for their helpful comments and constructive suggestions.

References

[1] P.E. Agre and D. Chapman, "What are Plans for ?," *Designing Autonomous Agents. Theory and Practice from Biology to Engineering and Back*, P. Maes, Ed. : The MIT Press, 1990.
[2] J.S. Albus, *Brains, Behavior and Robotics.* Byte Books, 1981.
[3] J.S. Albus, "Outline of a Theory of Intelligence," *IEEE Trans. Syst. Man and Cybernetics*, Vol. 21, No. 3, pp. 473-509, 1991.
[4] R.C. Arkin, "Motor Schema-Based Navigation for Mobile Robot: An Approach to Programming by Behavior," in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 264-271, 1987.
[5] R.C. Arkin, "Navigational Path Planning for a Vision-Based Mobile Robot," *Robotica*, Vol. 7, pp. 49-63, 1989.
[6] R.D. Beer, *Intelligence as Adaptive Behavior: an Experiment in Computational Neuroethology.* Academic Press, 1990.

[7] L.B. Booker, "Classifier Systems that learn internal world models," *Machine Learning*, Vol. 3, No. 2/3, pp. 161-192, 1988.

[8] R.A. Brooks, "Intelligence without Reason," in *Proceedings IJCAI-91*, pp. 569-595, 1991.

[9] R.A. Brooks, "Intelligence without Representations," *Artificial Intelligence*, Vol. 47, pp. 139-159, 1991.

[10] R.A. Brooks, "Artificial Life and Real Robots," in *Proceedings of the First European Conference on Artificial Life*, F.J. Varela and P. Bourgine, Eds. : The MIT Press/Bradford Books, pp. 3-10, 1992.

[11] F. Cecconi and D, Parisi, "Neural Networks with Motivational Units," in *Proceedings of the 2nd Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 346-355, 1993.

[12] D. Chapman and L.P. Kaelbling, "Input Generalization in Delayed Reinforcement Learning: an Algorithm and Performance Comparisons," in *Proceedings of IJCAI-91*, pp. 726-731, 1991.

[13] D. Cliff, P. Husband, J.A. Meyer and S.W. Wilson, Eds. *From Animals to Animats 3. Proceedings of the 3rd Int. Conf. on Simulation of Adaptive Behavior.* The MIT Press/Bradford Books, 1994.

[14] M. Colombetti and M. Dorigo, "Training Agents to Perform SequentialBehavior," *Adaptive Behavior*, Vol. 2, No. 3, pp. 247-275, 1993.

[15] K.J.W. Craik, *The Nature of Explanation.* Cambridge University Press, 1943.

[16] D. Dennett, "Intentional Systems in Cognitive Ethology: the 'Panglossian paradigm' defended," *Behavioral and Brain Science*, Vol. 6, pp. 343-390, 1983.

[17] J.Y. Donnart and J.A. Meyer, "A Hierarchical Classifier System Implementing a Motivationally Autonomous Animat," in *Proceedings of the 3rd Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 144-153, 1994.

[18] M. Dorigo and M. Colombetti, "Robot Shaping: Developing Autonomous Agents through Learning," *Artificial Intelligence*, Vol. 71, No. 2, pp. 321-370, 1994.

[19] L.M. Gabora, "Should I Stay or Should I Go: Coordinating Biological Needs with Continuously-updated Assesments of the Environment," in *Proceedings of the 2nd Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 156-162, 1993.

[20] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, 1989.

[21] J.J. Grefenstette, "Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms," *Machine Learning*, Vol. 2/3, pp. 225-245, 1988.

[22] A. Guillot, "Revue générale des méthodes d'étude des séquences comportementales," *Etudes et Analyses Comportementales*, Vol. 2, No. 3, pp. 86-106, 1986.

[23] J.R.P. Halperin, "Machine Motivation," in *Proceedings of the 1st Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 213-221, 1991.

[24] J.H. Holland, "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems," *Machine Learning: An Artificial Intelligence Approach 2*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell, Eds. : Morgan Kaufmann, pp. 593-623, 1986.

[25] J.H. Holland, K.J. Holyoak, R.E. Nisbett and P.R. Thagard, *Induction: Processes of Inference, Learning and Discovery.* The MIT Press/Bradford Books, 1986

[26] N. Jakobi, P. Husbands and I. Harvey, "Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics," in *Proceedings of the Third European Conference on Artificial Life*, in Press, 1995.

[27] L.P. Kaelbling, "Hierarchical Learning in Stochastic Domains: Preliminary Results," in *Proceedings of the Ninth International Conference on Machine Learning*, pp. 167-173, 1993.

[28] K-Team, *Khepera Users Manual.* EPFL, Lausanne, 1993.

[29] L.J. Lin, "Hierarchical Learning of Robot Skills by Reinforcement," in *Proceedings of the IEEE International Conference on Neural Networks-93*, pp. 181-186, 1993.

[30] P. Maes, "A Bottom-Up Mechanism for Behavior Selection in an Artificial Creature," in *Proceedings of the 1st Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 238-246, 1991.

[31] P. Maes, "Modelling Adaptive Autonomous Agents," *Artificial Life*, Vol. 1, pp. 135-162, 1994.

[32] D. McFarland, "The Teleological Imperative," *Goals, No Goals and Own Goals*, Montefiore and Noble Eds. : Unwin-Hyman, 1989.

[33] D. McFarland, "Defining Motivation and Cognition in Animals," *International Studies in the Philosophy of Science*, Vol. 5, No. 2, pp. 153-170, 1991.

[34] D. McFarland and T. Bösser, *Intelligent Behavior in Animals and Robots.* The MIT Press/Bradford Books, 1993.

[35] J.A. Meyer and S.W. Wilson, Eds. *From Animals to Animats. Proceedings of the 1st Int. Conf. on Simulation of Adaptive Behavior.* The MIT Press/Bradford Books, 1991.

[36] J.A. Meyer, H.L. Roitblat and S.W. Wilson, Eds. *From Animals to Animats 2. Proceedings of the 2nd Int. Conf. on Simulation of Adaptive Behavior.* The MIT Press/Bradford Books, 1993.

[37] J.A. Meyer, "The Animat Approach to Cognitive Science," in *Comparative Approaches to Cognitive Science*, H.L. Roitblat and J.A. Meyer, Eds. : The MIT Press/Bradford Books, 1995.

[38] F. Mondada, E. Franzi and P. Ienne, "Mobile Robot Miniaturisation: A Tool for Investigation in Control Algorithms," in *Proceedings of the 3rd International Symposium on Experimental Robotics*, 1993.

[39] N.J. Nilsson, *Principles of Artificial Intelligence.* Tioga Pub. Co., 1980.

[40] H.L. Roitblat and J.A. Meyer, "Introduction to Comparative Cognition," *Comparative Approaches to Cognitive Science*, H.L. Roitblat and J.A. Meyer, Eds. : The MIT Press/Bradford Books, 1995.

[41] L. Shu and J. Schaeffer, "HCS: Adding Hierarchies to Classifier Systems," *Proceedings of the 4th Int. Conf. on Genetic Algorithms*, Belew and Booker, Eds. : Kaufmann, pp. 339-345, 1991.

[42] S.P. Singh, "Reinforcement Learning with a Hierarchy of Abstract Models," in *Proceedings of AAAI*, pp. 202-207, 1992.

[43] T. Smithers, "On why better robots make it harder", in *Proceedings of the 3rd Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 64-72, 1994.

[44] L. Steels, "Towards a Theory of Emergent Functionality," in *Proceedings of the 1st Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 451-461, 1991.

[45] L.A. Suchman, *Plans and Situated Actions: The Problem of Human-Machine Communication.* Cambridge University Press, 1987.

[46] R.S. Sutton, "Temporal Credit Assignment in Reinforcement Learning," Doctoral Dissertation, Department of Computer and Information Science, University of Massachusetts, 1984.

[47] R.S. Sutton, "Reinforcement Learning Architectures for Animats," in *Proceedings of the 1st Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 288-296, 1991.

[48] F.M. Toates and P. Jensen, "Ethological and Psychological Models of Motivation: Towards a Synthesis," in *Proceedings of the 1st Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 194-205, 1991.

[49] T. Tyrrell, "The Use of Hierarchies for Action Selection," *Adaptive Behavior* Vol.1, No. 4, pp. 387-420, 1993.

[50] C.J. Watkins, "Learning with Delayed Rewards," Ph.D. Dissertation, Cambridge University, 1989.

[51] G.M. Werner, "Using Second Order Neural Connections for Motivation of Behavioral Choices," in *Proceedings of the 3rd Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 154-161, 1994.

[52] S.W. Wilson, "Classifier Systems and the Animat Problem," *Machine Learning*, Vol 2, pp. 199-228. 1987.

[53] S.W. Wilson, "Hierarchical Credit Allocation in a Classifier System," *Genetic algorithms and simulated annealing*, Davies, Ed. : Pitman, pp. 104-115, 1987.

[54] S.W. Wilson, "The Animat Path to AI," in *Proceedings of the 1st Int. Conf. on Simulation of Adaptive Behavior*, The MIT Press/Bradford Books, pp. 15-21, 1991.