Hierarchical-map Building and Self-positioning with MonaLysa

Jean-Yves Donnart and Jean-Arcady Meyer
AnimatLab*

Abstract

This paper describes how an animat endowed with the MonaLysa control architecture can build a cognitive map that merges into a hierarchical framework not only topological links between landmarks, but also higher-level structures, control information, and metric distances and orientations. The paper also describes how the animat can use such a map to locate itself, even if it is endowed with noisy dead-reckoning capacities. MonaLysa's mapping and self-positioning capacities are illustrated by results obtained in three different environments and four noise-level conditions. These capacities appear to be gracefully degraded when the environment grows more challenging and when the noise level increases. In the discussion, the current approach is compared to others with similar objectives, and directions for future work are outlined.

Keywords

Hierarchical map. Topological information. Metric information.
Landmarks. Self-positioning. Dead-reckoning. Robustness to noise.

1 Introduction

In robotics or animat research, traditional navigation methods that use internal geometrical representations of the environment (Latombe, 1991) are confronted with various implementation difficulties, due to memory and time requirements, as well as sensory and motor errors (Nehmzow, 1995). If such difficulties may be coped with in environments that are small enough for their most important features to be detectable from a single place, the problem is more difficult in large-scale space, as discussed by Brooks (1985), Kuipers and Byun (1988), Levitt et Lawton (1990) and Kuipers and Byun (1991) , for example.

To overcome the fragility of purely metric methods, many researchers resort to the use of various types of topological models that represent the connectivity of the environment, and several such models have been devised that aim to mimic known nervous architectures and behavioral capacities in animals (Muller et al., 1991; Matarić, 1992; Schmajuk and Thieme, 1992; Penna and Wu, 1993; Bachelder and Waxman, 1994; Nehmzow, 1995; Schölkopf and Mallot, 1995). Basically, these topological models endow an animat or a robot with cognitive abilities that make it possible to "recognize" the place it is situated in and to "know" that, if it performs a given move in a given direction, it will arrive in another "known" place. In other words, such models belong to the category of so-called *world models*, and they encode a variety of *Stimulus-Response-Stimulus* (S-R-S) information (Riolo, 1991; Roitblat, 1994). At the functional level, they allow the animat or the robot to navigate according to a *route-following* strategy (Trullier et al., 1997) and to plan a trajectory from a given starting place to a given goal place, provided that such a trajectory can pass through places already known and involves moves from place to place that have already been experienced. They also facilitate place recognition because such a task may take into account not only the various features that characterize a given place, but also the specific moves that lead to that place, or depart from it, together with the specific places that are thus connected to it. Moreover, additional metric information provided by specialized sensors - like distances and orientations - can also be taken into account and facilitate the place-recognition task. This is, for instance, the case with Matarić's robot (Matarić, 1992) and with Kuipers and Byun's animat (Kuipers and Byun, 1991). In principle, such additional metric information makes possible a more sophisticated navigation strategy than route-following, i.e., that of *metric navigation* or *survey mapping* (Trullier et al., 1997).

*AnimatLab, Ecole Normale Supérieure. 46, rue d'Ulm 75230 Paris Cedex 05, France. E-mail: donnart@wotan.ens.fr, meyer@wotan.ens.fr

This paper is primarily concerned with the place-recognition and self-positioning functionalities. It describes how it has been possible to add to the basic exploratory abilities of an animat endowed with the MonaLysa control architecture (Donnart and Meyer, 1994; Donnart and Meyer, 1996) the faculties of building a so-called *cognitive map* of its environment (Tolman, 1948; O'Keefe and Nadel, 1978; Kuipers, 1982; Thinus-Blanc, 1988; Gallistel, 1990; Poucet, 1993), and of locating itself despite the lack of precision in its sensors and actuators. Such a cognitive map merges into a hierarchical framework not only topological links between landmarks, but also higher-level structures and control information. It also includes metric distances and orientations. In other words, the approach described herein calls upon both topological and metric information to overcome the difficulties of exploration, mapping and self-positioning in large-scale environments. Later, it will be used to implement a full navigation system.

This paper is structured as follows: Section 2 summarizes the characteristics of MonaLysa that have already been described elsewhere, and Section 3 describes the animat's exploration strategy. Section 4 describes how the animat detects *landmarks* and builds its cognitive map. Section 5 describes the animat's self-positioning procedure. Experimental results obtained in various environments and with various levels of metric inaccuracy are shown in Section 6. The last section discusses the advantages and drawbacks of this approach and outlines perspectives for future work.

2 The MonaLysa architecture.

In the present application, the MonaLysa architecture enables an animat to explore a two-dimensional environment that may contain various obstacles. The animat 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 direction of a goal to be reached in each of the eight sectors of space surrounding it. Lastly, it is capable of moving straight ahead, 90° to its right, or 90° to its left. MonaLysa, which relies upon a hierarchical *classifier system* (Holland et al., 1986; Holland , 1986), is organized into eight modules - a reactive module, a planning module, a context manager, an auto-analysis module, an internal reinforcement module, a mapping module, a spatial information processor and a place recognition module (Figure 1).

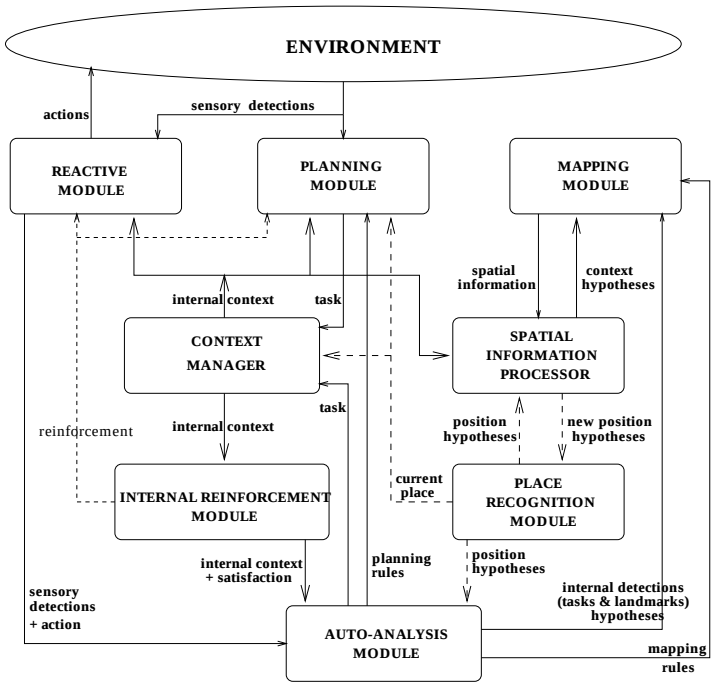

Figure 1: The MonaLysa architecture

The functionalities of the first five modules have been extensively described elsewhere (Donnart and Meyer, 1994; Donnart and Meyer, 1996) and will accordingly be dealt with only briefly. As a whole, they manage a pile

of *tasks* some of which allow the animat to reactively escape from any obstacle it gets trapped into, by skirting around it, and to plan a trajectory that will later allow it to avoid the obstacle from a distance. Such an ability relies upon the animat's capacity to analyze its skirting paths and to detect landmarks that will be used for locating itself in the environment.

2.1 The reactive module

This module chooses the next move to perform. It uses production rules that take the form :
If <sensory information> and <direction of current goal> Then <action>
The sensory information that these rules take into account is that provided by the animat's proximate sensors.

2.2 The planning module

This module decomposes a task into a series of subtasks, that is, into a series of planned trajectories connecting a starting place to an end place. It uses production rules that take the form :
If <sensory information> and <current task> Then <subtask>
The sensory information that these rules take into account is that provided by the animat's proximate sensors, completed by the animat's coordinates and current orientation.

2.3 The context manager

It contains a pile of tasks that are either imposed by the experimenter or autonomously generated while the animat moves in its environment. Tasks such as exploring the environment or reaching a given goal belong to the former category. So-called *skirting tasks*, that are posted by the auto-analysis module when the animat has detected an obstacle, or *obstacle-avoidance tasks*, that are posted by the planning module when a trajectory likely to avoid obstacles has been planned, belong to the latter category. At each instant, the task at the top of the pile specifies the *current goal* and the *current task*, i.e., the *internal context* in which reactive and planning rules can be triggered.

2.4 The internal-reinforcement module

This module is used to monitor the *satisfaction* of the animat and to adjust the *strengths* of the reactive and planning rules. The satisfaction is an estimation of the success with which a given rule brought the animat closer to, or took it farther from, its current goal. The strengths are used to probabilistically choose which rule to trigger if the condition parts of several rules match the current situation.

2.5 The auto-analysis module

The role of this module is to analyze the current behavior of the animat in order to alter its current task dynamically and to create new tasks that will enhance its future behavior. It is also responsible for characterizing landmarks in the environment, i.e. places through which it would be useful to travel in the future in order to avoid the obstacles from a distance, or which are used for map building and place-recognition.

3 Spatial Exploration

In various realizations, spatial exploration and mapbuilding are constrained by a movement strategy, like wall-following for instance, that is imposed by the control architecture (e.g., Kuipers and Byun, 1991; Nehmzow and Smithers, 1991; Matarić, 1992). In other realizations, they are constrained by the type of environment to which the control architecture is dedicated, as is the case in a maze for instance (e.g., Schmajuk and Thieme, 1992; Schölkopf and Mallot, 1995). To avoid such constraints and to make MonaLysa's cognitive mapping and self-positioning abilities as general as possible, these abilities are independent from any goal and from any goal-achieving task. Actually they are achieved under the control of a specific exploratory task, through the use of specific reactive rules.

3.1 The exploratory task

When an animat endowed with MonaLysa is first put in a new environment, its principal task becomes that of exploring the environment. This entails a series of moves that are totally random at the beginning, but become

slightly biased towards moving straight ahead as learning progresses. When such moves lead to the detection of an obstacle, a skirting task is put on top of the exploratory task on the context manager's pile, under the control of which the animat starts skirting around the obstacle and building its cognitive map, as explained below. This skirting task may trigger the posting of other skirting tasks on the context manager's pile, which are eventually erased in turn as the animat succeeds in skirting around the obstacle. Exploration resumes as soon as the original exploratory task becomes the current task again. Further obstacle encounters will trigger further skirting tasks and ultimately lead to the completion of the cognitive map.

3.2 The exploratory reactive rules

Some rules in the reactive module are dedicated to exploration. They take the form :

If <sensory information> and <no goal> Then <action>

The strength *S (R)* of each such rule *R* is adjusted by a reinforcement procedure that depends upon an internal satisfaction *satisf*:

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

where

$S(R)_u$ *: is the strength of rule R after u triggers,*

α *: is a learning rate, set to 0.1 in the experiments described herein,*

satisf : equals 1 if <action (R)> = <move straight ahead>, or equals 0.9 otherwise.

It is because moving straight ahead provides a slightly greater satisfaction than turning right or left that the animat eventually favors moving forward as learning and exploration proceed.

4 Map building

The elaboration of its cognitive map by MonaLysa depends mostly upon the auto-analysis and mapping modules. The former is responsible for obstacle detection, skirting-task triggering and landmark recognition. The latter basically records, within a hierarchical classifier system, which landmarks - or combinations of landmarks - have been detected under the control of which skirting task, or combination of skirting tasks.

4.1 The auto-analysis module

The process of landmark detection and map building calls upon the management of two lists within the auto-analysis module, a LC_list and a S_list, which are bound to each skirting task triggered. The LC_list records each landmark and each skirting subtask that are encountered or generated within the context of a given skirting task. The S_list records specific portions of the animat's trajectory, called *structures*, that are traveled within the context of a given skirting task.

4.1.1 Obstacle detection and skirting-task triggering

When the animat moves along in its environment and detects an obstacle - as described in Donnart and Meyer (1996) - the auto-analysis module generates a skirting task that specifies that the animat must cross the line lying parallel to the direction taken to avoid the obstacle and passing through the place the obstacle prevented the animat from reaching. This task is coded by the pair <*coordinates of the place*> < *direction vector of the straight line to be crossed*> and is deposited at the top of the pile of the context manager. The moves the animat makes while it attemps to cross this line may lead to further obstacle detections and, thus, to the specification of further skirting tasks that are piled on each other on the top of the context manager's pile. In every place where the animat succeeds to cross such a line, the corresponding task - and every subtask on top of it - is erased from the context manager. It may also happen that the animat gives up a skirting task when the estimated cost of its current trajectory is too high, as explained in Appendix A.

An emergent functionality of the internal dynamics of the auto-analysis module is to enable the animat to skirt around the obstacles it encounters and to extricate itself from dead-ends with arbitrarily complicated shapes. For example, two skirting trajectories around a specific obstacle are shown on Figure 2. The left trajectory has been generated through the creation of three skirting tasks at places A, B and C, respectively associated with lines $\delta 1$, $\delta 2$ and $\delta 3$. The task associated with line $\delta 2$ - that will simply be called task $\delta 2$ from

now on - has been erased at place C, and tasks $\delta 3$ and $\delta 1$ have been erased respectively at places D and E. The right trajectory has been generated through the creation of task $\delta 1$ at place F and its erasing at place G.

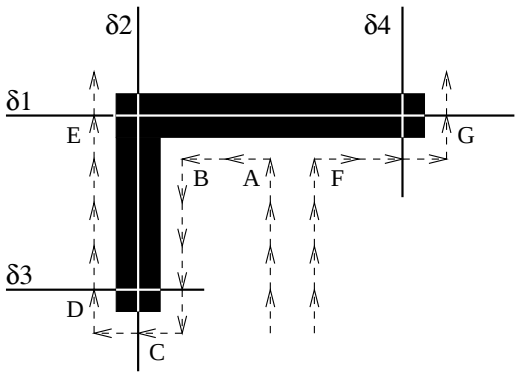

Figure 2: Skirting trajectories around a given obstacle.

(a) Impassable obstacles

When the animat encounters an impassable obstacle, like the periphery of its environment, it tries to skirt around it, as in the case of any other obstacle. However, because the corresponding skirting task cannot be erased, a specific procedure allows MonaLysa to estimate the probability *Prob_impass* that the obstacle actually is impassable and to give up the skirting task. Such a procedure is also described in Appendix A.

(b) Skirtable obstacle

When the last skirting task on the context manager's pile is erased and when control reverts to the initial exploratory task, as might be the case at place G of Figure 2 for instance, it makes sense for MonaLysa to continue skirting around the whole obstacle and to refine the description of the corresponding part of its map. Therefore, a possibility of detecting that moving to the left would be impossible is afforded to MonaLysa, which accordingly entails the triggering of a new skirting task, like $\delta 4$ on Figure 2. By means of such sequential trigger actions, the animat is able to complete whole skirting trajectories around obstacles. However, in order to avoid endless loops, a specific procedure allows MonaLysa to estimate the probability *Trig_prob* of sequentially triggering a skirting task in such a situation. According to this procedure, when the animat explores its environment, it initially tends to skirt several times around a given obstacle, thus enhancing its chances of detecting landmarks and structures, and of improving its cognitive map. Later, it tends to favor moving away from the obstacle, thus enhancing its chances of discovering new obstacles, which will in turn lead to further improvements of its cognitive map. Again, relevant details are given in Appendix A.

4.1.2 Landmark detection

Within the context of a skirting task, there are four categories of landmarks liable to be detected by MonaLysa. They respectively characterize so-called *satisfying places*, *unsatisfying places*, *task-erasing places* and *marker places*.

(a) Satisfying and unsatisfying landmarks

To detect landmarks belonging to these categories, the auto-analysis module monitors the variation of the animat's satisfaction between two successive moves. It also detects *constrained moves*, that is, moves that aren't the most conducive to reaching the goal, and to which the animat resorts because of the presence of an obstacle. Such moves trigger obstacle detection, as described in Donnart and Meyer (1996). Places where the satisfaction gradient is positive (resp. negative) and that have been reached through a constrained move are categorized as satisfying (resp. unsatisfying) landmarks[1].

(b) Task-erasing landmarks

[1]Some such satisfying landmarks are also used to define the start and end places of the obstacle avoidance subtasks generated by the planning module, according to a recursive procedure that has also been described in Donnart and Meyer (1996).

Task-erasing landmarks are places where a task can be erased from the context manager pile, i.e. when the animat actually succeeds in crossing the corresponding line.

(c) Marker landmarks

Marker landmarks are places that belong to any of the three preceding categories. They are defined as any first landmark the animat is capable of encountering after the triggering of a skirting task, or as any last landmark the animat is capable of encountering in the context of this skirting task. Being bound to the corresponding skirting task, they help to characterize or to recognize it. They make the management of the LC_lists easier and expedite MonaLysa's hierarchical positioning procedure.

When the animat is under the control of a given skirting task, it records - within a LC_list that is bound to this task - a succession of landmarks and skirting subtasks according to a procedure that will be explained later on. Within such lists, marker landmarks are categorized as the first or last landmark, or as any landmark that directly follows another landmark. In the latter case, such landmarks are said to be linked to each other.

4.1.3 Structure detection

Marker landmarks also serve to characterize structures. Any LC_list or any sublist within a LC_list that starts with a marker X and ends with a marker Y is recorded as a structure [XY] within a S_list bound to the corresponding skirting task. It will be shown later that such a structure actually represents a sequence of landmarks and tasks that characterizes a given obstacle, and that it is likely to be recognized and recorded every time the animat triggers the same skirting task around this obstacle.

4.1.4 Management of the LC_list and the S_list

The management of the LC_list and the S_list are illustrated on the specific example of Figure 3. The example assumes that, while the animat is traveling southwards, it encounters an obstacle at place A and triggers three skirting tasks successively, each characterized by lines $\delta 1$, $\delta 2$ and $\delta 3$. Having passed in turn through places A, B, ... I, the animat resumes traveling southwards from place I.

Along its path, the animat's first encounter with the obstacle occurs at place A. Because the animat cannot proceed straightforward, its satisfaction gradient becomes negative at A. However, this place is not characterized as an unsatisfying place because the move that led to A was not constrained.

At place A, the animat chooses to turn right and generates a skirting task characterized by line $\delta 1$. While it moves from A to B, its current goal is to reach the projection of its current position onto line $\delta 1$: the corresponding satisfaction gradient remains constant and no landmark is detected along the path (Figure 4a). However, the corresponding moves are constrained because the presence of the obstacle prevents the animat from turning left, i.e. from reaching its current goal on line $\delta 1$.

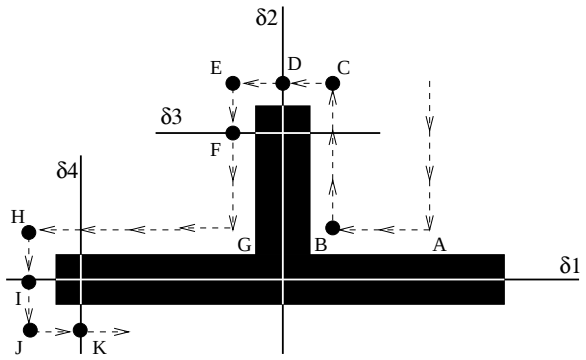

Figure 3: Skirting tasks (δ lines) created and landmarks (black dots) detected along a trajectory

In B, the animat encounters a new obstacle, triggers a new task $\delta 2$, and turns right. Because its satisfaction gradient with respect to line $\delta 1$ diminishes, and because the preceding move was constrained, place B is detected as both an unsatisfying landmark and a marker landmark within the context of task $\delta 1$. B and $\delta 2$ are added to

the LC_list associated with $\delta 1$. Likewise, B is detected as a satisfying landmark and a marker landmark within the context of task $\delta 2$. Thus B is also added to the LC_list associated with $\delta 2$ (Figure 4b).

From B to C, the animat's current goal is to cross line $\delta 2$; the corresponding moves are constrained, and the satisfaction gradient remains unchanged.

In C, the animat can turn towards its current goal on line $\delta 2$ and its satisfaction gradient increases : therefore, place C is detected as both a satisfying landmark and a marker landmark associated with $\delta 2$. Place C is then added to the LC_list associated with $\delta 2$ but, because this list begins and ends with two marker landmarks, B and C, these landmarks are encapsulated as a structure [BC] which is added to the S_list associated with $\delta 2$. The corresponding LC_list shrinks into place C (Figure 4c).

When the animat arrives at place D, task $\delta 2$ is erased from the context-manager pile and place D is detected as a task-erasing landmark and a marker landmark that is added to the LC_list associated with $\delta 2$. However, because, this list begins and ends with two marker landmarks, C and D, these landmarks are encapsulated as a structure [CD] which is added to the S_list associated with $\delta 2$. The corresponding LC_list shrinks into place D (Figure 4d). Then, the two lists associated with $\delta 2$ are erased.

However, because the obstacle still prevents the animat from crossing line $\delta 1$, D is also detected as an unsatisfying landmark associated with $\delta 1$ and the skirting task $\delta 3$ is added to $\delta 1$ on the top of the context-manager pile. Finally, within context $\delta 3$, D is also detected as a satisfying landmark and a marker landmark (Figure 4e).

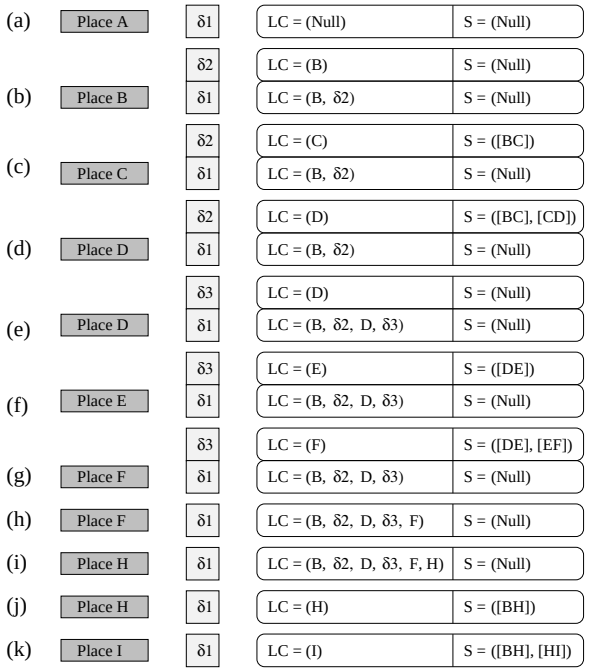

Figure 4: The management of the LC_list and S_list during the exploration of the obstacle on Figure 3.

At place E, the animat can turn right and move towards its current goal on line $\delta 3$. As the preceeding move was constrained, place E is detected as both a satisfying landmark and a marker landmark associated with $\delta 3$. Place E is added to D, within the corresponding LC_list. This list shrinks to E when structure [DE] is created and added to the corresponding S_list (Figure 4f).

The same events that occurred in place D now occur in place F, which is detected as a task-erasing landmark and a marker landmark in context $\delta 3$ (Figure 4g). Then, task $\delta 3$ is erased and F is detected as a satisfying landmark in context $\delta 1$ (Figure 4h).

Place G is not detected as a landmark, because the move that led to it was not constrained. Conversely, place H is detected as both a satisfying and a marker landmark associated with $\delta 1$ and is added to the corresponding LC_list (Figure 4i). Then, the list shrinks to H, while structure [BH] is added to the S_list (Figure 4j).

At place I, structure [HI] is added to the S_list (Figure 4k) and task $\delta 1$ is erased.

In this example, it has so far been assumed that the animat was initially traveling southwards because it was trying to reach a given goal, supposedly situated somewhere in that direction. In such a case, when the animat arrives at place I and erases $\delta 1$, the task of reaching the initial goal reappears at the top of the context manager's pile, and the animat accordingly resumes moving southwards. However, the initial task of the animat might have been an exploratory task as well. Under such circumstances, as it detects an obstacle to its left, it can decide to continue skirting around the obstacle and to post task $\delta 4$ to the context manager according to the procedure described in paragraph 4.1.1.(b) above. As a consequence, the animat will turn left at place J and arrive at place K. In K, task $\delta 4$ will be erased and a new obstacle will be detected on the left. Then, under the control of a new task $\delta 1$', the animat will complete skirting around the obstacle.

4.2 The Mapping Module

The " map " that the animat builds while it explores its environment has nothing in common with a classical two-dimensional map. Rather, it is coded as various production rules that are created by the auto-analysis module, that are memorized into the mapping module, and that post their action part to the spatial information processor. In the present implementation, such rules may be created every time a skirting task is erased from the context manager's pile, depending upon the content of lists LC and S. Rule creation are however limited to specific situations, which will be described later.

Mapping rules belong to six different categories and take the forms :

(1) *If <marker landmark A> and <current task δ > Then <post the pair (A, δ)>*;

(2) *If <marker landmark A> and <current task $\delta 2/\delta 1$> Then <post the pair (A, $\delta 2$)>*;

(3) *If <link between two landmarks (A, B) detected> and <(X, δ) recorded> Then <support structure [XY]/δ >*;

(4) *If <unsatisfying landmark A and subtask $\delta 2$ detected> and <(X, $\delta 1$) recorded> Then <support structure [XY]/$\delta 1$>*;

(5) *If <marker landmark Y> and <(X, δ) recorded> Then <post the structure [XY]/δ >*;

(6) *If <task-erasing landmark A and new task $\delta 2$ detected> and <current task $\delta 1$ to be erased> Then <post the pair (A, $\delta 2$)>*.

Expressions like <task $\delta 2/\delta 1$> and <structure [XY]/$\delta 1$> mean that task $\delta 2$ and structure [XY] are detected within the context of a specific task $\delta 1$.

Thus, such rules are capable of binding a landmark to a skirting task (type 1), or a landmark to two skirting tasks, which can be hierarchically (type 2) or sequentially (type 6) linked. Such rules are also capable of providing support to the recognition of global structures that include two or more landmarks and subtasks (types 3 and 4). Finally, they are capable of recording such structures (type 5), which help to generate position hypotheses, as explained later.

When they involve a skirting task, the corresponding records take into account the coordinates of the place where the task is triggered and the direction vector of the straight line to be crossed. A position error is also recorded. When they involve a landmark, the corresponding records take into account the coordinates of the corresponding place and the orientation of the animat in this place. Such records also take into account on-line estimates of the animat's position error and the category to which the detected landmark belongs.

It must be noted that, although *absolute* coordinates and orientations are recorded into the mapping rules, the matching process between two places or two tasks that will be described later involves *relative* distances or orientations between such places and tasks. Absolute information arbitrarily refers to the animat's initial position. It is taken into account only when matching is ambiguous - for instance when a given place can be matched to two or more other places - or when two places are too far apart to be linked within any mapping rule - for instance when they belong to two distinct objects in the environment. It must also be noted that no sensory information is used in the description of the landmarks, even if such information would certainly increase the animat's self-positioning capabilities. Therefore, the self-positioning capacities of MonaLysa do not depend upon the animat's sensory errors, contrary to what happens in many other realizations (e.g., Kuipers

and Byun, 1991; Nehmzow and Smithers, 1991; Matarić, 1992).

4.2.1 Creation of type 1 and 2 rules

The role of type 1 and type 2 mapping rules is to bind every category of markers except task erasing places to skirting tasks in order to record that these markers are likely to be encountered within the context of these tasks.

Type 1 rules bind together the skirting task and the first marker X of every structure [XY] recorded on the corresponding S_list. Type 2 rules apply to the specific case where the skirting task to be erased has been actually triggered within the context of another skirting task. In this case, the first marker of the first structure recorded on the S_list results in the creation of a type 2 rule.

In the exemple on Figure 3, task $\delta 1$ is to be erased at place I and two structures [BH] and [HI] are recorded on the corresponding S_list (Figure 4k). Thus, two type 1 rules are created:

(1) If < B > and < $\delta 1$> Then <post the pair (B, $\delta 1$)>
(2) If < H > and < $\delta 1$> Then <post the pair (H, $\delta 1$)>

Likewise, task $\delta 2$ is to be erased at place D, where the corresponding S_list is ([BC], [CD]) (Figure 4d). As $\delta 2$ is on top of $\delta 1$, one type 1 rule:

(3) If < C > and < $\delta 2$> Then <post the pair (C, $\delta 2$)>

and one type 2 rule:

(4) If < B > and < $\delta 2/\delta 1$> Then <post the pair (B, $\delta 2$)>

are created.

In the same manner, the creation of other type 1 and 2 rules would link marker E to $\delta 3$, marker J to $\delta 4$ and marker D to $\delta 3/\delta 2$.

Thus it appears that the creation of such rules acknowledges the fact that the animat's first encounter with the obstacle could occur somewhere to the right of place B, where skirting task $\delta 1$ is likely to be triggered. It also records that, provided the animat chooses to turn right, the first marker it is likely to encounter is marker B. Thus, marker B and task $\delta 1$ are bound by rule 1. Likewise, subtask $\delta 2$ is likely to be posted on top of task $\delta 1$ at place B. Thus, marker B and sub-task $\delta 2$ of $\delta 1$ are bound by rule 4. It is also acknowledged that the animat's first encounter with the obstacle could occur somewhere between B and C, where skirting task $\delta 2$ is likely to be triggered. Assuming that the animat chooses to turn on its right, the first marker it is likely to encounter is marker C (rule 3). Finally, rule 2 acknowledges the fact that the animat might also encounter the obstacle for the first time somewhere between G and H and that it is likely to reach marker H within the context of task $\delta 1$.

4.2.2 Creation of type 3 and 4 rules

The role of type 3 and type 4 mapping rules is to embed specific sub-components of the animat's trajectory within the context of higher-level structures. These rules are created when a skirting task is to be erased and when a structure is recorded on the S_list that is bound to this task. Two linked landmarks recorded in the corresponding LC_list lead to the creation of a type 3 rule, while the succession of an unsatisfying landmark and a skirting task lead to the creation of a type 4 rule.

In the example on Figure 3, when the animat arrives at place I and is about to erase task $\delta 1$, structure [BH] is recorded on the S_list of $\delta 1$ (Figure 4k). As this structure derives from the LC_list *(B, $\delta 2$, D, $\delta 3$, F, H)* (Figure 4i), one type 3 rule and two type 4 rules can be created:

(5) If <F and H> and <(B, $\delta 1$)> Then <support structure [BH]/$\delta 1$>
(6) If <B and $\delta 2$> and <(B, $\delta 1$)> Then <support structure [BH]/$\delta 1$>
(7) If <D and $\delta 3$> and <(B, $\delta 1$)> Then <support structure [BH]/$\delta 1$>

Thus, rule 5 records the fact that, if the animat goes from F to H after having previously bound marker B to skirting task $\delta 1$, then its trajectory is likely to be a sub-component of structure [BH] bound to $\delta 1$. Rules 6 and 7 predict that trajectories passing through markers B bound to $\delta 2$ or D bound to $\delta 3$, provided marker B has previously been bound to task $\delta 1$, are probably sub-components of structure [BH] bound to $\delta 1$. In the same manner, the creation of other type 3 rules would embed sub-trajectories (B,C) within [BC]/δ2, (C,D) within [CD]/δ2, (D,E) within [DE]/δ3, (E,F) within [EF]/δ3, and (H,I) within [HI]/δ1. No other type 4 rules would be created in the case of Figure 3.

4.2.3 Creation of type 5 rules

The role of type 5 mapping rules is to bind structures to skirting tasks in order to record that these structures are likely to be encountered within the context of these tasks. Type 5 rules are created when a skirting task is to be erased and when a structure is recorded on the S_list that is bound to this task.

In the example on Figure 3, when the animat arrives at place F and is about to erase task $\delta 3$, structures [DE] and [EF] are recorded on the S_list of $\delta 3$ (Figure 4g). Two type 5 rules are accordingly created:

(8) If < E > and <(D, $\delta 3$)> Then <post the structure [DE]/$\delta 3$>
(9) If < F > and <(E, $\delta 3$)> Then <post the structure [EF]/$\delta 3$>

Other type 5 rules would respectively bind structures [BC] and [CD] to task $\delta 2$ in place D, and structures [BH] and [HI] to $\delta 1$ at place I.

4.2.4 Creation of type 6 rules

The role of type 6 mapping rules is to record a sequential link between two skirting tasks, when the erasure of a skirting task in a given place is immediately followed by the triggering of a new skirting task in this same place.

In the example on Figure 3, a type 6 mapping rule would be created at place I should the animat decide to trigger skirting task $\delta 4$ just after having erased task $\delta 1$ in order to continue exploring the obstacle and refining the corresponding part of its map:

(10) If <I and $\delta 4$> and < $\delta 1$ to be erased> Then <post the pair (I, $\delta 4$)>

4.2.5 Partial graphic representation of the map

Although a very small portion of the information recorded in MonaLysa's mapping rules can be represented graphically, such a representation is useful for the interpretation of the experimental results in Section 6 below. This sort of representation is shown on Figure 5, which corresponds to the navigation example of Figure 3. Basically, it takes into account the positions of all the landmarks that are recorded within the mapping rules and the links between landmarks that are specifically encoded within type 3 rules. It should be understood that the arrow linking F to H, for example, refers to rule 5 of paragraph 4.2.2 above and means that the animat "knows" that H is the next landmark it is likely to encounter when it is positioned in F.

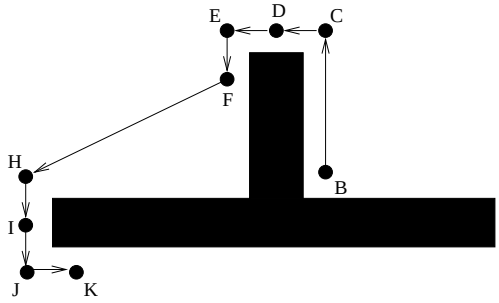

Figure 5: Graphic representation of the map built during the exploration of the obstacle on Figure 3.

4.2.6 Hierarchical spatial representation

However, instead of learning a flat sequence of landmarks, MonaLysa actually learns a hierarchical organization where each element serves as a context for the recognition of others. This is illustrated on Figure 6, which also corresponds to the navigation example of Figure 3.

According to such a hierarchical representation, it is seen that MonaLysa "knows" that structures [BH] and [HI], for instance, are likely to be encountered when skirting task $\delta 1$ is triggered. In particular, this would be the case should the animat first encounter the obstacle at any place other than A to the right of B, then trigger task $\delta 1$, and then turn right in this place. However, if the animat first encountered the obstacle at any place between G and H, and then turn right in this place, it would not recognize structure [BH], but would possibly still travel through and recognize structure [HI] within the context of task $\delta 1$. On the contrary, structure [BH] could be traveled through and recognized normally, while structure [HI] could be missed, should the animat randomly decide not to turn right at place H, thereby missing place I. Likewise, structures [BC] and [CD] are likely to be traveled through and recognized within the context of task $\delta 2$, although this might occasionnally not be the case. Likewise, MonaLysa "knows" that link (F,H) is likely to be encountered as a sub-component of structure [BH] within the context of task $\delta 1$.

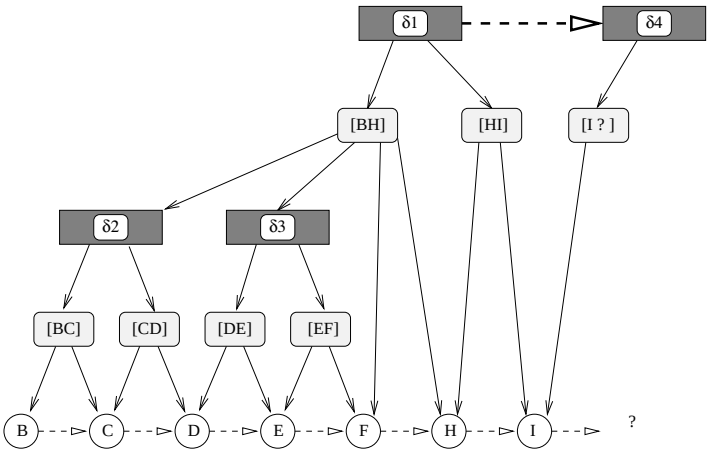

Figure 6: Hierarchical spatial representation built by MonaLysa during the exploration example on Figure 3. Circles correspond to landmarks, light-grey rectangles to structures and dark-grey rectangles to skirting tasks. Regular arrows represent hierarchical links, and dotted arrows represent sequential links that are recorded in specific mapping rules.

MonaLysa's hierarchical representation also indicates that landmark D, for instance, can be detected as a marker landmark at the beginning of a structure [DE] within context $\delta 3$, or as a task-erasing landmark ending structure [CD] within context $\delta 2$, or as an unsatisfying landmark bound to $\delta 3$ that belongs to structure [BH] within context $\delta 1$. It will be shown later that such different ways of recognizing landmark D substantiate each other and contribute to make the animat's self-positioning easier.

5 Localization

As soon as it starts learning a map of its environment, the animat must be able to use it to locate itself, even if its dead-reckoning capacities are noisy. This entails a capacity to monitor the uncertainty with which spatial information is acquired and recorded, and a capacity to match ambiguous estimates like, for instance, the place the animat is currently in and a place previously recorded on the map.

5.1 Matching procedure

5.1.1 Spatial uncertainty

In the present application, the animat is assumed to be equipped with a specific device that correctly estimates its orientation, but also with a rudimentary dead-reckoning procedure that generates incorrect estimates of its position, the corresponding error increasing with the distance traveled. In particular, although each elementary

move is assumed to be equal to the animat's length (*Animat_length*) by the dead-reckoning procedure, it is actually greater or shorter than this estimate, according to a gaussian distribution whose mean is equal to *Animat_length* and whose standard deviation is equal to $Noise\_level * Animat\_length$. Thus, after n such elementary moves, the corresponding standard error is that of a sum of normal random variates, i.e., it is equal to $\sqrt{n} * Noise\_level * Animat\_length$. *Noise_level* is a parameter that will be given various values in the experiments described below.

Due to this faulty dead-reckoning capacity, the position *Pos* of every place is characterized in MonaLysa by a vector of coordinates *Pos = (x,y)* and by an *Err(Pos)* value, which quantifies the uncertainty with which the position is known. In other words, it is assumed that the actual place that corresponds to an estimated position *Pos* may be situated anywhere inside an uncertainty circle having a center *Pos* and a radius *Err(Pos)*.

Likewise, every skirting task that is managed by MonaLysa is characterized not only by the position *(Pos)* and the position error *(Err(Pos))* of a place, but also by a direction vector and an associated uncertainty stripe with a width of *2*Err(Pos)*.

Similarly, every link between two landmarks is characterized within MonaLysa, not only by the positions and position errors of each landmark, but also by a relative-displacement vector, which originates in the first landmark and ends in the second. To the end of this vector is associated a relative-uncertainty circle, centered on the second landmark, and whose radius is equal to the increase of the position error that occurs when the animat moves from the first landmark to the second.

5.1.2 Matching conditions

Two places are considered as matching when their uncertainty circles intersect, within an error margin of *Animat_length*.

Two skirting tasks are considered as matching when their uncertainty stripes intersect, within an error margin of *Animat_length*.

To decide whether two links between pairs of landmarks match, MonaLysa first checks whether each place in a pair matches the corresponding place in the other pair. If such is the case, it assigns the same origin to the corresponding relative-displacement vectors and checks whether their relative uncertainty circles intersect, within an error margin of *Animat_length*.

5.1.3 Position and error resets

Usually, when the animat moves about in its environment, the estimates of its coordinates change accordingly. For instance, when a given action moves the animat by a vector (dx, dy), then every position estimate (xi, yi) in MonaLysa is changed to (xi + dx, yi + dy).

Likewise, drawing upon common statistical practice, the position error *Err(Pos)* is set to twice the above-mentioned standard deviation. It is therefore updated after each elementary move according to the equation :

$$Err(Pos)_{u+1} = \sqrt{Err(Pos)_u^2 + (2 * Noise\_level * Animat\_length)^2}$$

where $Err(Pos)_u$ is the size of the uncertainty radius after u elementary moves.

However, when the animat arrives at a place that matches a place already recorded in its map, the position and error estimates of the former can be reset according to the self-positioning procedure described in the next section.

Conversely, the position *Pos(L)* of a landmark L on the map is adjusted every time it matches the *CPH*, i.e. what will be later termed the animat's *current position hypothesis*, according to equations:

$$Pos(L)_{u+1} = (1 - \beta\_Maj) * Pos(L)_u + \beta\_Maj * Pos(CPH)$$

$$Err(Pos(L))_{u+1} = (1 - \beta\_Maj) * Err(Pos(L))_u + \beta\_Maj * Err(Pos(CPH))$$

where $Pos(L)_u$ is the position of landmark L after u matchings and β_Maj is a weighting factor whose computation is described in Appendix B.

5.2 Self-positioning procedure

Four modules, each managing a specific list of spatial information, allow MonaLysa to determine its current position in the environment (Figure 7). The mapping module contains a list of mapping rules. The spatial-information processor contains a CH_list of context hypotheses that concern the current skirting task and its associated markers and structures. The place recognition module contains a PH_list of position hypotheses, one of these being the animat's CPH. Finally, the auto-analysis module contains a LC_list of internal detections, i.e., a list of landmarks and skirting tasks.

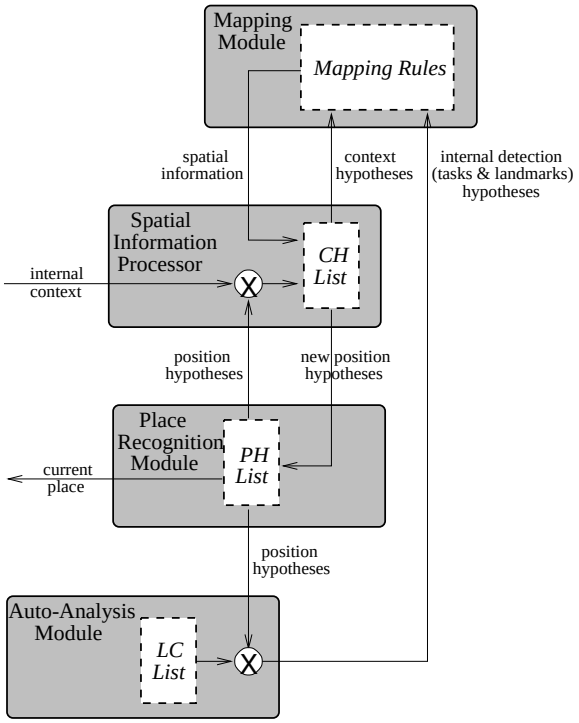

Figure 7: The four modules and their associated lists of spatial information that are used for self-positioning. Circles refer to an expansion process that is described in the text.

5.2.1 Role of the mapping module

The role of the mapping module is to refine the animat's estimates about its possible current position by embedding them within the broader contexts of given skirting tasks or given structures recorded on the map. This is performed by the triggering of one or more mapping rules, when the condition parts of these rules match the description of the current situation. This description is provided through two different channels:

- Firstly, the place-recognition module forwards a set of hypotheses about MonaLysa's internal detections, that is, hypotheses about the positions of a marker (which is likely to match the corresponding condition part of type 1, 2 and 5 mapping rules), of two linked landmarks (type 3 rules), or of a landmark and a skirting task (type 4 and 6 rules);
- Secondly, the spatial-information processor forwards a set of hypotheses about the context in which the previous detections have been made, that is, hypotheses about the current skirting task (which is likely to match the corresponding condition part of type 1, 2 and 6 mapping rules), about a higher level skirting task (type 2 rules), or about the position of the last marker encountered (type 3, 4 and 5 rules).

When a mapping rule is triggered, it posts its action part on the CH_list of the spatial-information processor, thus refining the description of the current context by providing hypotheses about the last marker encountered (type 1, 2, 5 and 6 rules) or about the structure to which current detections may belong (type 3, 4 and 5 rules). This refinement of the CH_list may entail confirming a previous context hypothesis - because the action part matches the hypothesis - or suppressing a hypothesis that is not confirmed - because there is no matching.

5.2.2 Role of the auto-analysis module

Besides its role in the creation of planning and mapping rules, the auto-analysis module uses the content of its LC_list for forwarding to the mapping module the landmarks and tasks that it detects. However, as the coordinates used by the auto-analysis module are those of the CPH and as it can be known - from the content of the PH_list within the place-recognition module - that the animat could actually be situated in other places, the auto-analysis module accordingly expands to these other places the list of landmarks and tasks that it forwards to the mapping module.

5.2.3 Role of the spatial-information processor

The role of the spatial-information processor is to provide hypotheses about the context in which MonaLysa's internal detections occur. Such hypotheses may concern the position of the obstacle the animat is currently trying to skirt around (through information about the current skirting task), the position of this obstacle with respect to another obstacle (through information about a higher level skirting task), or the specific sub-component of a structure it is engaged into (through information about structures encountered along this trajectory). To this end, the spatial-information module manages a CH_list of marker landmarks and skirting tasks bound to each skirting task of the context manager. This list can be transmitted as a series of context hypotheses to the mapping module.

Another role of the spatial-information processor is to provide the place-recognition module with new position hypotheses that may, in particular, help to reset the animat's CPH. The greater the confidence the animat has in each such hypothesis, the more such an event is likely to occur. It depends upon the values of two variables, *Max_supports* and *Nb_supports*, that are used to evaluate how successfully a given marker has been recognized as belonging to a recorded structure in the map. The variable *Max_supports* characterizes a given CH_list and thus the structure the animat is currently traveling through within the context of a given skirting task. It evaluates the number of sub-components that make up the structure. The variable *Nb_supports* is attached to every structure that is recognized on the map. It counts how many supports such recognized structures get from mapping rules that are triggered when specific sub-components are recognized on the map.

The CH_list may be supplied with an expansion of the description of the skirting task the animat is currently engaged in - which is provided by the context manager in reference to the current position hypothesis - to skirting tasks that are defined by the same direction vector, but in reference to the other places where the animat could also be situated.

The CH_list may also be supplied by postings from the mapping module, as explained in the following sub-sections.

(a) Postings from type 1 and type 2 rules

To locate itself in the environment, the animat uses type 1 and type 2 mapping rules for recognizing markers within the context of specific skirting tasks and for binding together these markers and tasks. Later, every such bind can be used as a broader context in which further recognitions may occur.

When there is only one skirting task on the context manager's pile, the spatial information processor forwards to the mapping module the expanded description of this task, i.e. a list δi of tasks. When the auto-analysis module detects a marker M within the context of this task, it forwards to the mapping module an expanded description of the marker, i.e. a list Mj of markers. Every pair (Mj, δi) matching the condition part of a type 1 mapping rule entails its triggering, thus implying that the pair has been recognized on the map. As a consequence, the corresponding δi is replaced by the pair (Mj, δi) on the CH_list. Conversely, every δi that cannot be bound to the marker is not recognized on the map and is accordingly erased from the CH_list.

When the context manager's pile contains a skirting task δb on top of another skirting task δa, which means that the former has been triggered within the context of the latter, the spatial information processor forwards to the mapping module the expanded description of this context, i.e., a list of δbi/δai. Every pair (Mj, δbi/δai) matching the condition part of a type 2 mapping rule entails the triggering of the rule, thus implying that Mj, δai and δbi have been recognized on the map. As a consequence, the corresponding hypothesis δbi is replaced by the pair (Mj, δbi) on the CH_list. Likewise, every couple δbi/δai that cannot be bound to the marker is not recognized on the map and δbi is accordingly erased from the CH_list.

The recognition of marker Mj on the map entails the sending of a new position hypothesis to the PH_list of the place-recognition module. This hypothesis suggests that the animat is situated at place Mj, the coordinates and position error of which are those that are recorded in the map. It is passed to the place-recognition module

with *Max_supports* and *Nb_supports* values that are set to 0.

(b) Postings from type 3 and type 4 rules

The animat may also resort to type 3 and 4 rules to self-locate in its environment. Indeed, these rules are used to support hypotheses about the likelihood that some markers belong to sub-components of some structures.

Type 3 rules may be triggered when a link between two landmarks A and B is detected within the LC_list of the auto-analysis module. When this occurs, the auto-analysis module forwards to the mapping module an expanded list of hypotheses (Ak, Bk), while the spatial-information processor forwards the pairs (Xi, δi) that are currently recorded in its CH_list. The corresponding *Max_supports* value is incremented by one unit, thus recording that the structure the animat is currently taveling through contains a sub-component that could be recognized on the map. If the information sent to the mapping module matches the condition part of a type 3 rule, this rule is triggered, thus implying that the link has been recognized on the map and that it is probably a sub-component of a known structure [XiYj]. This, however, still needs to be confirmed, by the recognition of the whole structure, through the triggering of a type 5 rule.

Type 4 rules may be triggered when an unsatisfying landmark A is detected within the context of a new skirting task δb within the LC_list of the auto-analysis module. When this occurs, the auto-analysis module forwards to the mapping module an expanded list of hypotheses (Ak, δbk), while the spatial-information processor forwards the pairs (Xi, δai) that are currently recorded in its CH_list. Again, the corresponding *Max_supports* value is incremented by one unit. If the information sent to the mapping module matches the condition part of a type 4 rule, this rule is triggered, thus implying that the unsatisfying landmark and the new skirting task have been recognized on the map and that they probably belong to the sub-component of a known structure [XiYj]. This also requires confirmation by the triggering of a type 5 rule.

In both cases, the structure [XiYj] that has thus been recognized is sent to the CH_list of the spatial-information processor, where it is attached to the context hypothesis that matched the corresponding condition part of the triggered mapping rule. The variable *Nb_supports* attached to the structure is set to 1 if the structure wasn't already attached to the list. Otherwise, this variable is incremented by one unit.

(c) Postings from type 5 rules

Type 5 rules are used to confirm that a whole structure has been recognized, when a new marker is detected within the context of a skirting task that is already bound to another marker.

When the auto-analysis module detects a marker Y in the context of skirting task δi, it forwards to the mapping module an expanded description of the marker, i.e. a list Yj of markers, while the spatial-information processor forwards the pairs (Xi, δi) that are currently recorded in its CH_list. Every triple (Xi, Yj, δi) matching the condition part of a type 5 mapping rule entails its triggering, thus implying that Xi and Yj have been recognized on the map and actually belong to a known structure [XiYj] in context δi. The corresponding pair (Xi, δi) is consequently replaced by the pair (Yj, δi) on the CH_list. In other words, instead of being carried out within the previous context of (Xi, δi), future detections will be made within the updated context (Yj, δi).

The recognition of marker Yj on the map entails the sending of a new position hypothesis to the PH_list of the place-recognition module. This hypothesis suggests that the animat is situated at place Yj, whose coordinates and position error are those that are recorded in the map. It is passed to the place-recognition module with the *Max_supports* value that characterizes the CH_list of the current task. It is also passed with the *Nb_supports* value that is attached to structure [XiYj]. Then every structure is cleared from the current task's CH_list, whose *Max_supports* value is reset to 0.

(d) Postings from type 6 rules

The triggering of type 6 rules is another means for changing the context of future detections.

When the auto-analysis module detects a task-erasing landmark E and when task δa is about to be replaced by task δb, an expanded list of pairs (Ei, δbj) is sent to the mapping module. This module also receives an expanded list δai from the spatial-information processor. Every type 6 rule whose condition part matches the current situation is triggered, thus implying that Ei, δai and δbj have been recognized on the map. The pair (Ei, δbj) is therefore recorded on the CH_list bound to task δb. This task replaces task δa on top of the context-manager's pile.

The recognition of landmark Ei on the map entails sending a new position hypothesis to the PH_list of the place-recognition module. This hypothesis suggests that the animat is situated at place Ei, whose coordinates

and position error are those that are recorded in the map. It is passed to the place-recognition module with *Max_supports* and *Nb_supports* values that are set to 0.

5.2.4 Role of the place-recognition module

The place-recognition module manages a PH_list of position hypotheses, each such hypothesis being characterized by a *position error* and a *confidence estimate*. At every time step, one of these hypotheses is the animat's CPH and designates the place where the animat estimates it is the most likely to be. Nevertheless, the animat also considers that it could be situated in other places, although the corresponding hypotheses are less likely to be true. The CPH is forwarded to the context manager - which uses it to estimate the direction of the current goal. The whole list of position hypotheses is sent to the auto-analysis module - which uses it to specify the coordinates associated with landmarks and tasks, notably those that are recorded in MonaLysa's planning and mapping rules - and to the spatial-information processor - which uses it to specify the skirting task the animat might currently be engaged in.

The PH_list of position hypotheses originates from actual moves in the environment. When such an hypothesis, embedded within the context of a given skirting task or a given structure, matches the condition part of some mapping rule, this rule is triggered, thus implying that the corresponding position is recognized on the animat's map. Such a recognition entails the updating of the CH_list within the spatial-information processor and the sending of new position hypotheses - each characterized by their *Max_supports* and *Nb_supports* values - to the place-recognition module. In turn, this entails the strengthening or weakening of former position hypotheses or the generation of new hypotheses within the PH_list.

(a) Strength of new hypotheses

Each position hypothesis that is forwarded by the spatial-information processor to the place recognition module is assigned a strength *Hyp_strength* that evaluates the influence the hypothesis should have on the updating of the PH_list. This influence depends mostly upon three factors, that are described in Appendix C:

- a factor that takes into account the values of *Max_supports* and *Nb_supports* and that gives precedence to hypotheses deriving from well-recognized structures;
- a factor that gives precedence to hypotheses close to the CPH;
- a factor that gives precedence to hypotheses close to other hypotheses already present on the PH_list.

(b) Updating of the PH_list

Each new position hypothesis that is sent to the place-recognition module is compared to the current content of the PH_list. Three possibilities, the details of which are given in Appendix D, can then occur:

- hypothesis generation

  When the distance between the new position and any position in the PH_list is greater than *Animat_length*, the corresponding hypothesis is added to the PH_list, with a confidence estimate that is proportional to its *Hyp_strength*.

- hypothesis strengthening

  When the distance between the new position and a given position in the PH_list is smaller than *Animat_length*, the confidence estimate of the matched hypothesis is increased by a factor proportional to the new hypothesis's *Hyp_strength*. This factor is not constant and depends upon *Max_supports*, in order to favor hypotheses deriving from large-sized structures.

- hypothesis weakening

  Every position hypothesis in the PH_list that is farther than *Animat_length* from any hypothesis sent by the spatial-information processor is not confirmed by the map. Its confidence estimate is accordingly decreased.

(c) Current position resetting

In principle, after each move the animat's CPH is derived from the previous current position hypothesis that has been updated according to the estimated size and direction of the move. The current position is said to be reset when, instead of being derived from the previous current hypothesis, it switches to another hypothesis of the PH_list. This occurs under two conditions:

- the confidence estimate of this other hypothesis is greater than that of any other hypothesis in the PH_list
- the confidence estimate of this other hypothesis is greater than a given threshold (which is set to 50 in the present application).

(d) Limiting the size of the PH_list

MonaLysa tries to limit the size of the PH_list to *Nb_hyp* items. To this end, position hypotheses with the lowest confidence estimates or with a confidence estimates smaller than a given threshold (set to 10 herein) may be canceled, provided that this procedure doesn't entail erasing the CPH. As a consequence, it may happen however that the CPH corresponds to the lowest confidence estimate and that the size of the PH_list exceeds *Nb_hyp*, as long as no other hypothesis has a confidence estimate high enough to allow position resetting. *Nb_hyp* is set to 5 in the current application.

(e) Limiting the number of mapping rules

Because MonaLysa simultaneously builds its map and uses it to self-locate, it is mandatory that mapping rules not be created that would be characterized by overly imprecise positions. Likewise, its seems a priori advisable to limit the number of mapping rules, in order to avoid generating too many position hypotheses. To accomplish this, mapping rule creation is allowed only when the confidence estimate of the CPH is the highest of the PH_list and exceeds a given threshold (which is set to 50 in the present application).

5.3 Management of the CH_list and PH_list

Figures 8 and 9 help to describe the self-positioning procedure from a specific example. They assume that the animat, after a preliminary exploratory stage, has detected several obstacles in its environment and built the corresponding map, according to the mapping procedure previously described. Such a map involves landmarks B',...N', B",... N" and skirting lines like $\delta 1$', ...$\delta 4$' ,$\delta 1$", ...$\delta 4$" (Figure 8).

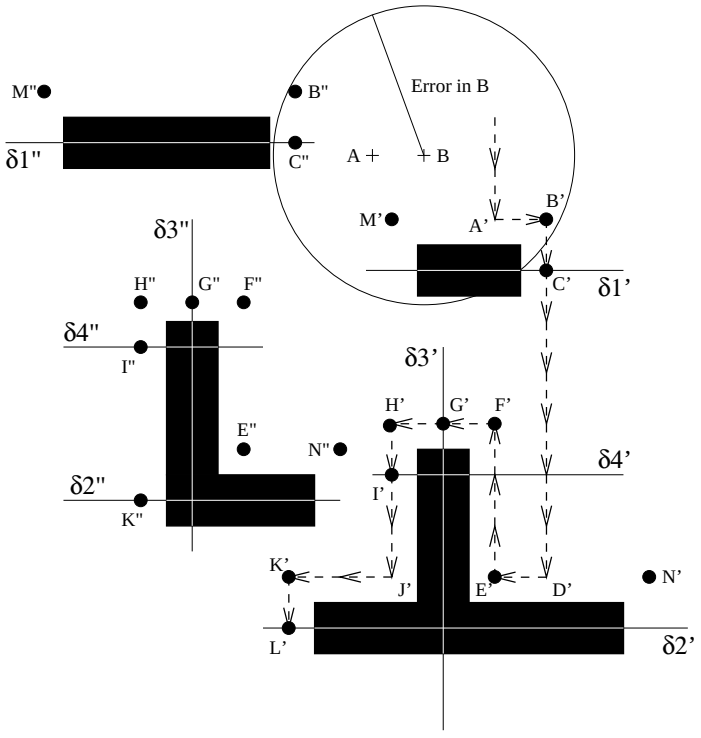

Figure 8: The use of the animat's map for self-positioning. Black dots denote landmarks recorded in the map.

It also assumes that the animat arrives at place A', but erroneously assumes it has reached place A. Thus place A is its current position hypothesis - which is supposedly characterized by an error of 250 and a confidence

level of 100 - and the top of the context-manager pile contains task $\delta 1$. Likewise, the CH_list associated with $\delta 1$ is initialised with the single task $\delta 1$ (Figure 9a).

In A', the animat turns left and reaches place B', which is characterized as a satisfying and marker landmark B. B is thus added to the LC_list associated with $\delta 1$.

Because the coordinates of the animat at places B' and B" fall within the error margin of place B, because the animat's orientation in all these places is the same, and because all these places are characterized as landmarks of the same type, places B' and B" match place B. Likewise, because the origins of the vectors defining lines $\delta 1$' and $\delta 1$" fall within the error margin associated with the origin of the vector defining $\delta 1$, and because the corresponding orientations are the same, tasks $\delta 1$' and $\delta 1$" match task $\delta 1$.

On the contrary, place B doesn't match place M' because, although they are both satisfying landmarks, the animat's orientation in these places differ. Nor does place B match place C", because the latter is a task-erasing landmark and because the animat's orientations in both places differ. Thus, two mapping rules of type 1 that have their condition parts matching the current situation are triggered and post pairs (B', $\delta 1$') and (B", $\delta 1$") to the spatial information processor. These two context hypotheses replace the single task $\delta 1$ on the CH_list of $\delta 1$.

The current position hypothesis becomes B, which is characterized by an error of 251, and two new position hypotheses, B' and B", are posted to the place recognition module with *Max_supports* and *Nb_supports* values set to 1. B' and B" are characterized by an initial confidence level of 25 and a recorded error of 10 and 15 respectively. The confidence level of B, which is not recognized on the map, decreases to 90 (Figure 9b). In other words, as the animat moves away from a landmark recorded in its map, its position error increases and its confidence in the current positioning hypothesis decreases.

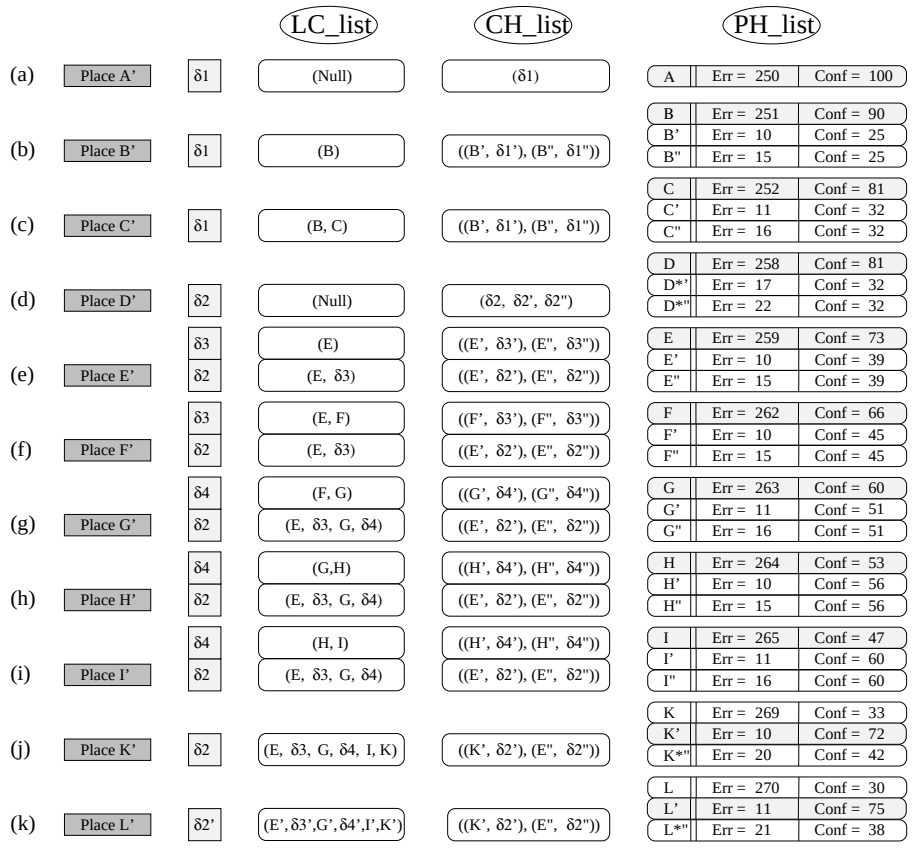

Figure 9: The management of the LC_list, CH_list and PH_list during self-positioning in the environment on Figure 8. Shaded components of the PH_list correspond to the animat's current position hypothesis.

From B' the animat moves to C', where it updates its three position hypotheses, and records that, although

it has probably moved from B to C [2], it might also have moved from B' to C*' or from B" to C*".

At C', the current position C is detected as a task-erasing and marker landmark, and the LC_list of $\delta 1$ becomes (B,C). The link (B,C) is expanded to three pairs (B,C), (B',C') and (B", C") that correspond to the three position hypotheses C, C' and C" of the PH_list. As there are two context hypotheses (B', $\delta 1$') and (B", $\delta 1$") on the CH_list, two type 3 rules can be triggered. These rules respectively support structures [B'C'] and [B"C"]. Then, structure [BC] is posted on the S_list of $\delta 1$, while the corresponding LC_list shrinks to (C). Three expanded hypotheses, C, C' and C" are sent to the mapping module, where they trigger two type 5 rules, thus confirming that the place the animat is currently in actually belongs to structures [B'C'] or [B"C"]. Because place B matches places B' and B", because place C matches places C' and C", and because the relative displacement the animat made between B and C matches the relative displacement it made from B' to C' and from B" to C", structures [B'C'] and [B"C"] match structure [BC], thus supporting the hypothesis that place C could, in fact, be either place C' or C".

These hypotheses are accordingly posted to the PH_list with *Max_supports* and *Nb_supports* values set to 1. Because positions of C*' and C*" are respectively not farther from places C' and C" than the animat's length, the positions of C*' and C*" can be replaced by those of C' and C", and their error estimates can be reset to 10 and 15 respectively. The confidence estimates of C' and C" thus increase to 32 when the confidence estimate of B decreases to 81 (Figure 9c).

Then the animat travels from C' to D', updating its three position hypotheses, and records that, although it has probably moved from C to D, it might also have moved from C' to D*' or from C" to D*" (Figure 9d). Then it turns right and generates skirting task $\delta 2$. The CH_list of $\delta 2$ is expanded to $\delta 2$, $\delta 2$' and $\delta 2$", which correspond to the three position hypotheses of the PH_list.

At place E', place E is characterized as an unsatisfying and marker landmark that is posted on the LC_list of $\delta 2$. Three expanded marker hypotheses are accordingly sent to the mapping module, where they trigger only two type 1 rules, because context hypothesis $\delta 2$ doesn't match any skirting task on the map. As a consequence, hypotheses $\delta 2$, $\delta 2$' and $\delta 2$" are replaced by (E', $\delta 2$') and (E", $\delta 2$") on the CH_list of $\delta 2$. Then, a new obstacle is detected, and the LC_list of $\delta 2$ becomes (E, $\delta 3$). This internal detection is expanded to (E, $\delta 3$), (E', $\delta 3$') and (E", $\delta 3$"), which trigger two type 4 rules respectively supporting structures [E'K'] within the context of $\delta 2$' and [E"K"] within the context of $\delta 2$". Thus structure [E'K'] is attached to context hypothesis (E', $\delta 2$') on the CH_list of $\delta 2$, together with a *Nb_supports* value of 1. Likewise, structure [E"K"] is attached to context hypothesis (E", $\delta 2$") on the CH_list of $\delta 2$, together with a *Nb_supports* value of 1. Meanwhile, the *Max_supports* value that characterizes the CH_list of $\delta 2$ is set to 1. It is thus acknowledged that the animat actually travels through a given structure in the environment within the context of task $\delta 2$, that this structure has at least one sub-component, and that this sub-component has been recognized on the map in two possible places.

When task $\delta 3$ is posted to the context-manager's pile, the CH_list of $\delta 3$ is initialized to $\delta 3$, $\delta 3$', and $\delta 3$". The triggering of two type 2 rules leads to the replacement of these context hypotheses by two pairs that have been recognized on the map: (E', $\delta 3$') and (E", $\delta 3$"). Two position hypotheses, E' and E" are sent with *Max_supports* and *Nb_supports* values of 1 to the place-recognition module, where they replace the current hypotheses E*' and E*". The corresponding confidence estimates increase to 39, while the confidence estimate of E decreases to 73 (Figure 9e).

At place F', two type 3 rules support structures [E'F'] and [E"F"], and such supports are confirmed by two type 5 rules. Thus, the satisfaying and marker landmark F is matched with F' and F", and structure [EF] is matched with structures [E'F'] and [E"F"]. The confidence estimates of the hypotheses associated with F' and F" continue to increase, while the confidence estimate of the hypothesis associated with F continues to decrease (Figure 9f).

From F' to G', H' and I', the PH_list maintained by MonaLysa is updated according to Figures 9g, 9h and 9i. It must be noted that, in place G', the pair (G, $\delta 4$) is detected within the context of task $\delta 2$ and that two type 4 rules are triggered, thus supporting structures [E'K'] and [E"K"]. Thus structure [E'K'] could be attached to context hypothesis (E', $\delta 2$') on the CH_list of $\delta 2$, but because it has previously already been attached to this context hypothesis, its *Nb_supports* value is incremented by one unit and set to 2. Likewise, the *Nb_supports* value of structure [E"K"] attached to context hypothesis (E", $\delta 2$") on the CH_list of $\delta 2$ is set to 2. Meanwhile, the *Max_supports* value that characterizes the CH_list of $\delta 2$ is also set to 2. It is thus acknowledged that the animat actually travels through a given structure in the environment within the context of task $\delta 2$, that this structure has at least two sub-components, and that each of these sub-components has been recognized on the map in two possible places.

[2] Place C, like several other places that are mentioned in the text, is not shown on Figure 8.

When the animat arrives at K', it supposes itself to be at K and link (I,K) is detected within the context of task $\delta 2$. Because marker K matches marker K', but not marker K", a type 3 rule recognizes this link on the map within context hypothesis (E', $\delta 2$'), but not within (E", $\delta 2$"). The *Nb_supports* value of structure [E'K'] attached to context hypothesis (E', $\delta 2$') is accordingly set to 3, while the *Nb_supports* value of structure [E"K"] attached to context hypothesis (E", $\delta 2$") still equals 2. Meanwhile, the *Max_supports* value that characterizes the CH_list of $\delta 2$ is also set to 3. Again, it is thus acknowledged that the animat actually travels through a given structure in the environment within the context of task $\delta 2$ and that this structure has at least three sub-components. Each of these three sub-components has been recognized in a given region of the map, while only two of them have been recognized in another region.

Then, structure [EK] = ((E, $\delta 3$), (G, $\delta 4$), (I, K)) is posted on the S_list of $\delta 2$. As this structure matches [E'K'] according to a type 5 rule, but does not match [E"K"], only position hypothesis K' is passed to the place recognition module, with *Max_supports* and *Nb_supports* values of 3.

Thus, K*' can be replaced by K' and its error estimate can be reset to 10. Because the confidence in hypothesis K' exceeds a given threshold level (i.e., 50) and because the confidence associated with K and K*" does not, the animat's current hypothesis switches from K to K' (Figure 9j). In other words, the animat has positioned itself accurately in its environment.

It should be understood that structure [E"K"] might have matched the detected structure [EK] in the presence of a high noise level. However, the supports posted during navigation from E' to K' by the three components of the detected structure [EK] would have been more likely to contribute to a greater increase in the confidence estimate of K' than to that of K". Indeed, components (E, $\delta 3$) and (G, $\delta 4$) of [EK] would match the condition parts of type 4 rules that have been created within the contexts of the exploration of [E'K'] and [E"K"] and would receive support from both categories of rules. On the contrary, component (I, K) would be more likely to match the condition part of a type 3 rule created within the context of the exploration of [E'K'] than within the context of [E"K"]. In other words, in the event structure [E"K"] were to be recognized on the map, position hypothesis K" would have been passed to the place recognition module with a *Max_supports* value of 3 and a *Nb_supports* value of 2. Thus, the confidence estimate of position hypothesis K" would have been smaller than that of hypothesis K' because structure [E"K"] led to only partial recognition.

It should also be understood that self-positioning success relies upon the fact that MonaLysa has managed simultaneously several hypotheses about the animat's moves and that some of these hypotheses got support every time a given landmark or structure in the environment was matched with a landmark or structure in the map. Thus, the hypothesis that the animat traveled from E' to K' got support not only from the matching of E with E', of F with F', of G with G', of H with H', of I with I', and of K with K', but also from the matching of [EF] with [E'F'], of [FG] with [F'G'], of [GH] with [G'H'], of [HI] with [H'I'], and finally from the matching of the whole structure [EK] with [E'K']. Had the animat failed to recognize a given landmark on the E'K' trajectory, like landmark I' for instance, the supports from the matchings of I with I', of [HI] with [H'I'] and of (I, K) in [EK] with (I', K') in [E'K'] would have been lost. Nevertheless, the other supports would have been retained, thereby preventing the hypothesis that the animat actually moved from E' to K' from being abandoned. Thus the management of hierarchical structures affords the self-positioning procedure of MonaLysa with interesting robustness features.

6 Experimental results

The efficiency of the mapping and localization procedures of MonaLysa have been tested in three different environments, each presenting different challenges to these procedures. Each environment was a square whose side (750 pixels) was 50 times greater than the animat's length (15 pixels). Many experiments were performed, at the beginning of each the animat was placed in a random initial position and was given the mission of exploring its environment during 50000 elementary moves - each probabilistically greater or smaller than *Animat_length*. To fulfill its mission, the animat had to use its noisy dead-reckoning position estimates to build a map of its environment and to use the map to improve its localization. Such improvements could only occur when the animat was able to recognize a place it was currently in as a landmark already recorded on its map and when this landmark was characterized by a smaller position error than the current place. Conversely, when the current place was known with a better accuracy than a recorded landmark, the position and the position error of the landmark were accordingly adjusted in the map. Thus map-building and self-positioning were necessarily held in parallel, with each hopefully serving to improve the other.

Results of specific experiments, that are illustrative of MonaLysa's capacities and limitations, are shown below. General statistics and comparisons are to be found elsewhere (Donnart, 1997).

6.1 Map building

This section describes the three environments that were explored and provides partial graphic representations of the maps built by MonaLysa under two *Noise_level* conditions respectively set to 0 and 0.1. In other words, in the former case the animat was assumed to make no dead-reckoning errors while, in the latter, each elementary move was supposed to be equal to *Animat_length* (i.e., 15 pixels) and was assigned an uncertainty value equal to 20% of the animat's length (i.e., 3 pixels).

6.1.1 The isolated square

The environment shown on Figure 10 contains only an isolated square. It is not ambiguous, in the sense that it includes no sub-parts that may be confused with each other, and that the vectors that define possible skirting tasks are well differentiated. It is nevertheless challenging for self-positioning because a large proportion of its surface is free from any obstacle, thus depriving the animat of landmark encounters, which are its only means of resetting its estimated position.

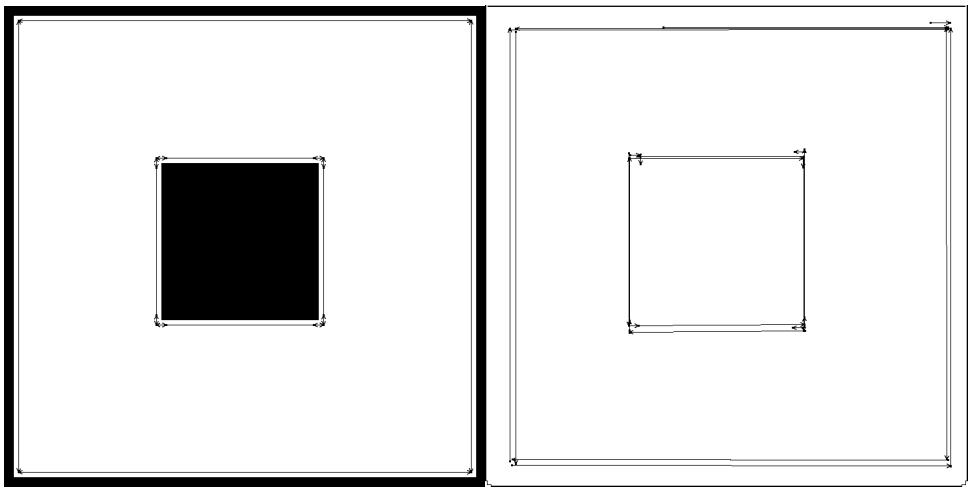

Figure 10: Graphic representation of the maps built in an environment with an isolated square. Left : *Noise_level = 0*: the positions of the square and of the environment's periphery are explicitly shown on the Figure. Right : *Noise_level = 0.1*: the positions of the square and of the periphery are implicitly given by landmarks and links.

The graphic representations of the maps built by MonaLysa during two specific experiments in both *Noise_level* conditions are shown on Figure 10. They are both quite simple and involve landmarks and links that well delineate the environment's periphery and the square's contour. Not surprisingly, more landmarks and links are recorded by MonaLysa when its position estimates are noisy.

6.1.2 The office floor

The environment shown on Figure 11 looks like any environment a real robot could be confronted with. Basically, it is made up of four rooms, each including one or more pieces of furniture. It is quite challenging for self-positioning because obstacles are either simple, but very similar to each other, or more complex and differentiated, but likely to generate many skirting tasks characterized by vectors that may easily be confused.

The graphic representations of the maps built by MonaLysa during two specific experiments in the previously defined *Noise_level* conditions are shown on Figure 11. They are slightly more elaborate than those of the previous environment and, in particular, they involve links between landmarks characterizing separate obstacles. Landmarks and links well delineate the environment's periphery and each obstacle's contour. The map built in noisy conditions contains many redundancies that the animat needed to maintain the accuracy of its position estimates.

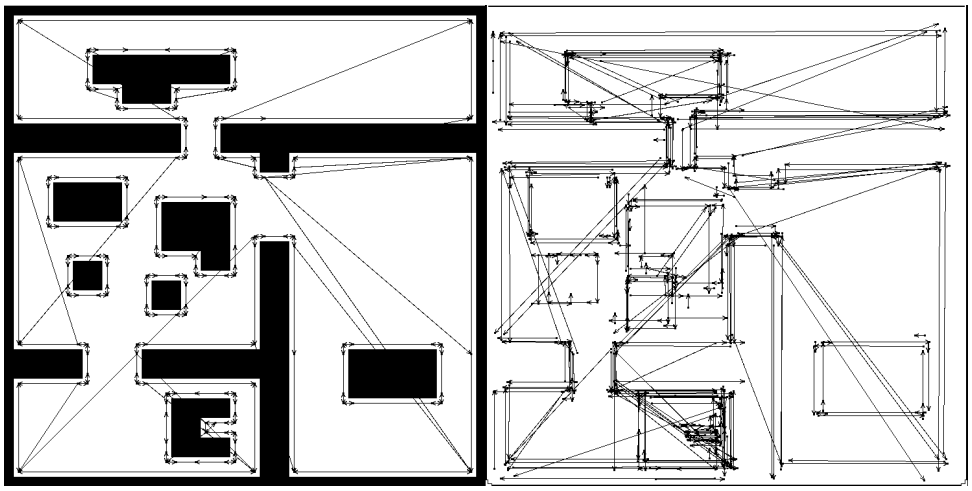

Figure 11: Graphic representation of the maps built of the office floor. Left : *Noise_level = 0*; Right : *Noise_level = 0.1*.

6.1.3 The labyrinth

Of the three environments used here, the labyrinth of Figure 12 is the most challenging for self-positioning. Indeed, local perceptions of the animat are very similar in many places, while the shape of every obstacle necessarily generates many skirting tasks characterized by vectors that are very difficult to disambiguate. Moreover, as was already the case with the previous environments, the size of the corridors doesn't prevent the animat from turning around and traveling in both directions, thus increasing the chances of confusing landmarks or skirting tasks.

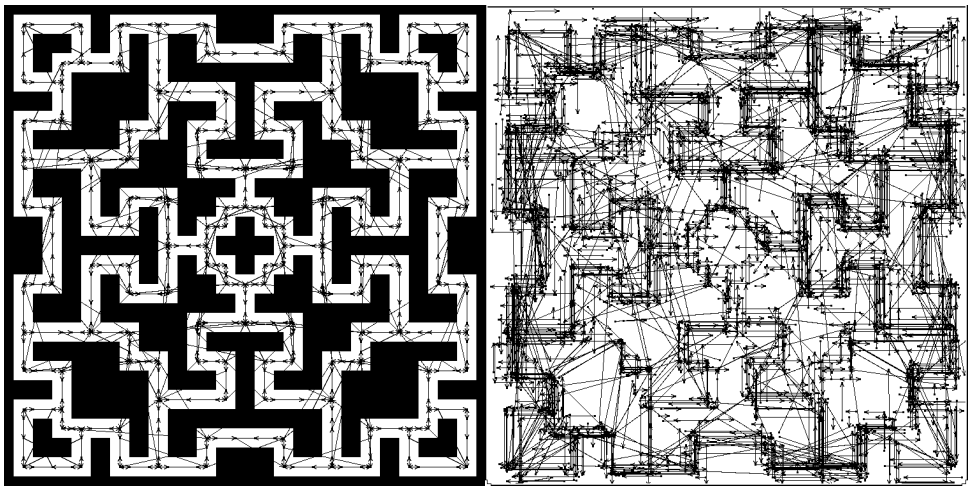

Figure 12: Graphic representation of the maps built of the labyrinth. Left : *Noise_level = 0*; Right : *Noise_level = 0.1*.

The maps that MonaLysa builds in this labyrinth are very complex, as suggested by the partial graphic representations of Figure 12 that correspond to two specific experiments. This is, in particular, the case with the map built in noisy conditions, which exhibits many redundancies. Nevertheless, the overall organization of the labyrinth can be correctly retrieved from such representations.

6.2 Self-positioning

In order to assess the architecture's self-positioning capacities, the distance between the animat's CPH and its actual position - which is called its *estimated position error or EPE* - was monitored over time, when the animat relied upon its dead-reckoning capacities only, and when it also used its map to locate itself. This

section describes results obtained in each of the three preceding environments, under four *Noise_level* conditions, respectively set to 0.10, 0.15, 0.20 and 0.25.

6.2.1 The isolated square

Figure 13 shows results of four specific experiments in the isolated square environment. It appears that the animat's EPE is much lower when MonaLysa uses its map than when it relies on its dead-reckoning capacities only. It also appears that, since the very beginning of each experiment, the use of the map makes it possible to maintain the EPE within a constant interval of variation in each *Noise_level* condition, a capacity that is not afforded by dead-reckoning alone. With a *Noise_level* value of 0.1, the animat's EPE is seldom greater than *3*Animat_length*. When this level increases from 0.1 to 0.25, the EPE's interval also increases, although it doesn't exceed six times the animat's length. Finally, because the peaks of EPE correspond to moves that separate two position resets, it is not surprising that such peaks are higher and larger as the level of noise increases.

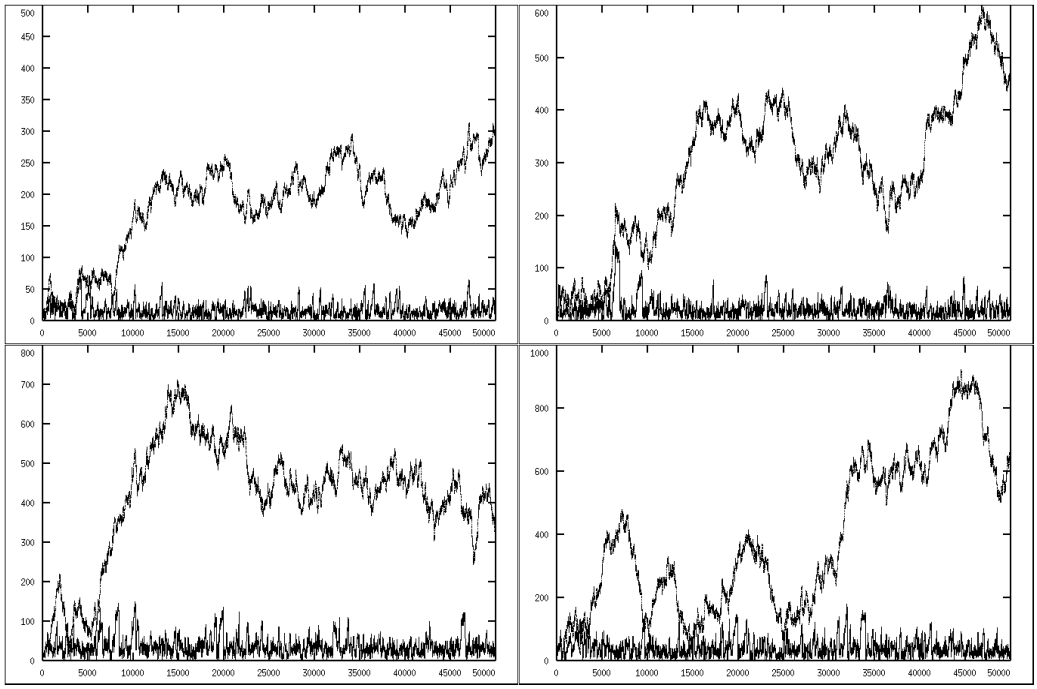

Figure 13: Evolution of the animat's estimated position error (ordinate). On each graph, the upper curve corresponds to results obtained during 50000 elementary moves (abcissa) in the isolated square environment with dead reckoning only, while the lower curve corresponds to results obtained with dead-reckoning and mapping. From left to right and from top to bottom, *Noise_level* values are respectively set to 0.10, 0.15, 0.20 and 0.25.

6.2.2 The office floor

Similar results are obtained in the office floor environment, as shown on Figure 14. Because the EPE's peaks are fewer, but higher and often wider, than those that were obtained in the previous environment, these results suggest that this environment affords greater opportunities for position resets, but with a lower precision - a deduction that seems intuitively plausible considering how this environment is organized. Results obtained with a *Noise_level* of 0.25 are significant from this point of view. Although the EPE is generally greater in the isolated square environment than in the office floor environment, the animat gets lost in the latter during approximately 2000 moves, around move number 19000. During this period, its EPE is almost equal to *23*Animat_length*, i.e. to half the size of the environment's side. Nevertheless the animat succeeds in re-locating itself. Such long-lasting positioning errors never happen in the isolated square environment.

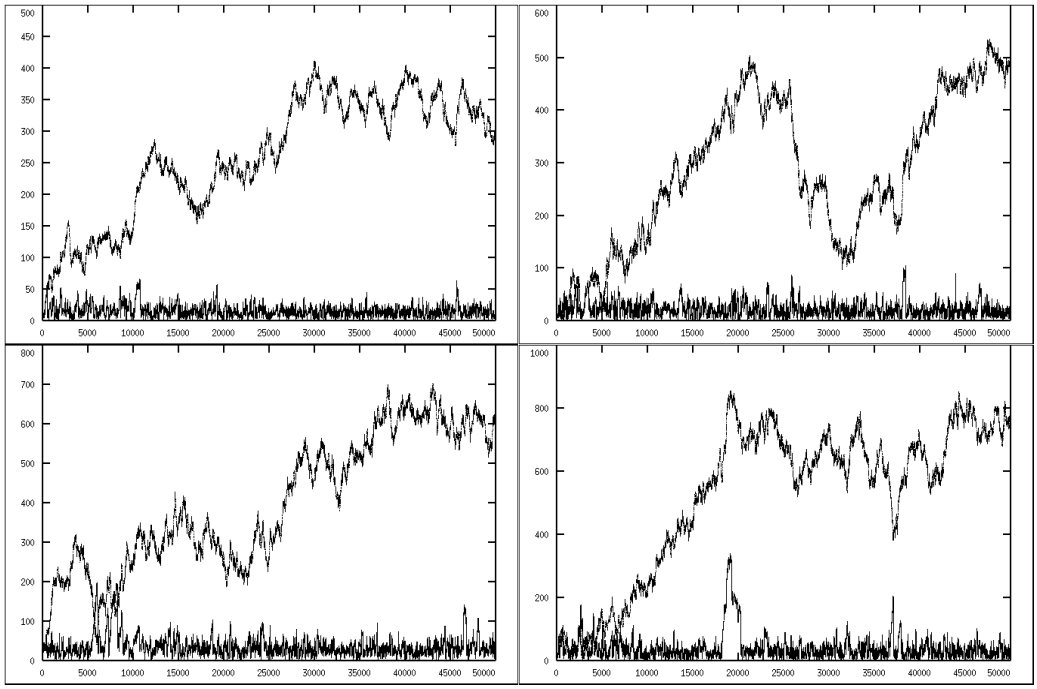

Figure 14: Evolution of the animat's estimated position error (ordinate). On each graph, the upper curve corresponds to results obtained during 50000 elementary moves (abcissa) in the office floor environment with dead reckoning only, while the lower curve corresponds to results obtained with dead-reckoning and mapping. From left to right and from top to bottom, *Noise_level* values are respectively set to 0.10, 0.15, 0.20 and 0.25.

6.2.3 The labyrinth

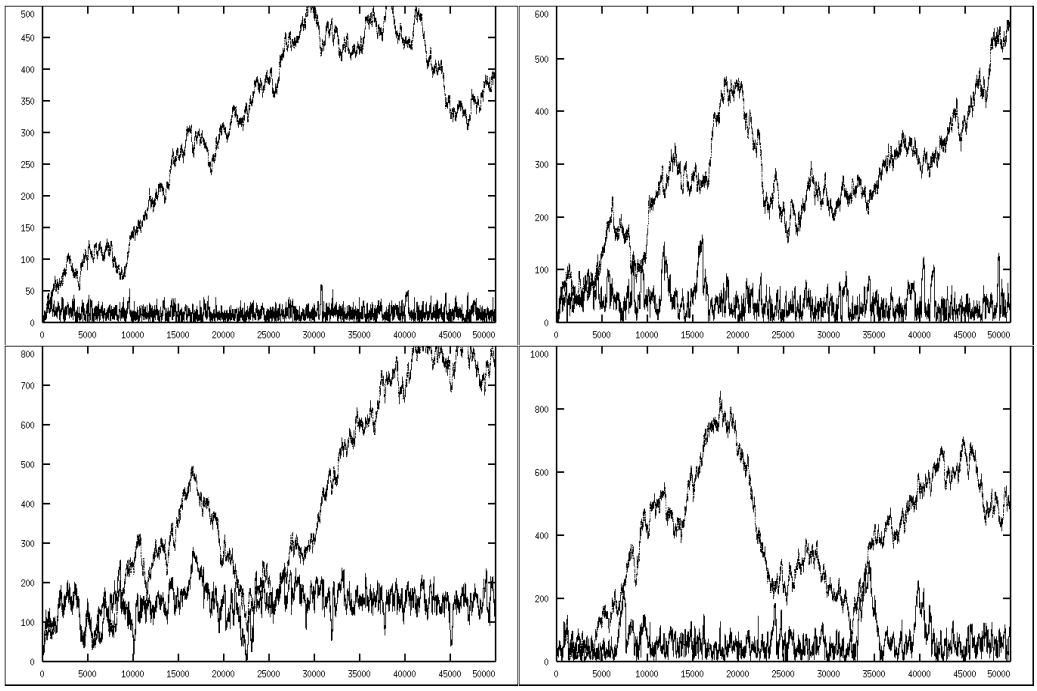

Figure 15: Evolution of the animat's estimated position error (ordinate). On each graph, the upper curve corresponds to results obtained during 50000 elementary moves (abcissa) in the labyrinth environment with dead reckoning only, while the lower curve corresponds to results obtained with dead-reckoning and mapping. From left to right and from top to bottom, *Noise_level* values are respectively set to 0.10, 0.15, 0.20 and 0.25.

Results obtained with the labyrinth are shown on Figure 15. They prove that MonaLysa's mapping and self-positioning procedures may lead to much better results than dead-reckoning alone, even in very challenging environments. They also suggest that the level of noise has greater influence in this environment than in the previous ones. Indeed, when *Noise_level* is set to 0.1, the EPE's interval of variation is almost the same in the three environments, although more frequent opportunities for position resets seem to be afforded by the labyrinth. Again, this seems intuitively plausible because the corresponding map is much richer. Results obtained with a *Noise_level* of 0.20 indicate that the map that the animat has built is rather precise - because the interval of variation of the EPE is short at the end of the learning process. However the map is shifted by a distance approximately equal to 10 times the animat's length. Should the animat's navigation call upon relative distances only, such a shifted map would not affect its capacities. Finally, results obtained with a *Noise_level* of 0.25 are worse in this environment than in the two others, although the animat's EPE is seldom greater than *8*Animat_length*. The animat gets lost on several occasions, but always succeeds in re-locating itself.

7 Discussion

Results shown above, together with the generalizations provided in Donnart (1997), demonstrate that MonaLysa's map-building and self-positioning procedures, although they rely on a noisy dead-reckoning mechanism, are capable of maintaining the animat's EPE within bounded limits. Such a capacity, which dead-reckoning alone cannot afford, is gracefully degraded when the environment gets more challenging and when the level of noise increases. In particular, it is maintained in a complex labyrinth with a noise level as high as 0.25, which means that the position uncertainty characterizing each dead-reckoning estimate is equal to half the mean size of each elementary move. It should however be stressed that the above results have been obtained while assuming that the animat's orientation estimates were correct and that landmark detection relied on internal satisfaction only. It remains to be verified whether taking sensory information into account for self-positioning would help to further disambiguate landmarks and would compensate for noisy orientation estimates. Should simulations prove this to be the case, an implementation of MonaLysa on a real robot could be attempted in the search for robust navigation capacities and for efficient survey-mapping strategies. Such an endeavor might capitalize on previous successful attempts at implementing MonaLysa's planning procedures on a real robot (Donnart and Meyer, 1996).

Other improvements in the present architecture are planned. In particular, since MonaLysa monitors the confidence estimates of its position hypotheses, several internal mechanisms could depend upon the quality of such estimates. They might affect, for instance, the tendency of further exploring given regions, the way matching conditions are managed, and how much landmark positions are reset on the map. Likewise, since it seems likely that MonaLysa's current mapping and self-positioning capacities would be impaired if the environment contained non-polygonal obstacles, future research will be directed towards the use of more general skirting tasks than mere line crossing, to allow the management of arbitrarily shaped obstacles.

Future research will also address the possibility of recognizing whole obstacles as opposed to only single landmarks or structures. This faculty would introduce an additional hierarchical level in the mapping and self-positioning procedures, and thus enhance the animat's cognitive capacities. In the present implementation, although the animat uses relative distances and orientations to position itself along the external contours of a given obstacle, it resorts to absolute distances and orientations to self-locate when it first encounters a new obstacle. The possibility of characterizing whole objects in the environment will make it possible to evaluate all positions and orientations relatively to each object in the environment, and thus to get rid of any reference to the animat's initial position.

Be as it may, it has been demonstrated here and elsewhere (Donnart and Meyer, 1994; Donnart and Meyer, 1996) that the MonaLysa control architecture enables an animat equipped with a rudimentary sensory-motor apparatus to explore its environment, to extricate itself from dead-ends with arbitrary complicated shapes, to build a cognitive map of its environment, to accurately estimate its current position, to plan trajectories that avoid obstacles and that lead to a given externally-specified or autonomously-generated goal. Such capabilities qualify MonaLysa as a *cognitive* architecture and the animat as a *motivationally autonomous system* (McFarland and Bösser, 1993). They rely upon the use of production rules that take into account not only the animat's current sensory information, but also their " internal context ", which codes the animat's additional knowledge about the current situation. In the case of the reactive rules, this knowledge concerns the direction of the current goal. In the case of the planning rules, it concerns the nature of the current task the animat tries to accomplish. In the case of the mapping rules, it concerns the various tasks associated with markers and structures the animat has detected in its environment. Moreover, the fact that this internal context has a

hierarchical structure confers several advantages on MonaLysa. Besides expediting the learning of the planning rules as described in Donnart and Meyer (1996), it has been shown here that it facilitates the self-positioning procedure and makes it relatively robust with respect to noise.

It has also been shown here that such robustness makes it possible for MonaLysa to detect, memorize and recognize landmarks in the environment, although the corresponding procedures do not rely on the information provided by the animat's proximate sensors. Actually this information is only used by the reactive rules of MonaLysa for the purpose of skirting around obstacles and of escaping from dead-ends, and place recognition depends only upon the proprioceptive position, orientation, and satisfaction estimates. This characteristic is in sharp contrast with many other realizations that implement place-recognition capabilities. For instance, several such realizations draw upon biology (Zipser, 1986; Cartwright and Collett, 1987; Muller et al., 1991; Burgess et al., 1994; Bachelder and Waxman, 1994) and implement a neural network in which the firing of some sensory neurons, tuned to the features of some landmarks sensed in a given place, triggers the firing of a specific *place cell* that codes for this place. Likewise, in Kuipers and Byun (1991), distinctive places are defined as the local maxima of specific functions that are defined over the various sensor readings of the animat, and, in Matarić (1992), landmarks are characterized as features in the world that have physical extensions reliably detectable over time. Few other realizations resort to proprioceptive information for place recognition. A specific example is provided in Nehmzow and Smithers (1991), where landmarks are characterized as convex or concave corners, such that, if the time a robot requires to turn towards a wall exceeds a certain threshold time, a convex corner is detected. Conversely, if the time it takes the robot to get away from a detected obstacle exceeds a certain threshold time, a concave corner is detected.

The mapping and self-positioning capacities of MonaLysa could also be contrasted with other realizations which do not fully exploit the information encoded in their cognitive maps. In such realizations, indeed, a specific place is usually recognized because it has been reached by a specific move from another specific place, but no account is taken of the hierarchical structures that MonaLysa manages. An example of this is to be found in Nehmzow and Smithers (1991), where a specific concave corner can be recognized because it has been reached from a convex corner a given distance away, but not because this specific concave corner belonged to a characteristic structure that linked, for instance, three succeeding convex corners to the concave corner in question. It is certainly because such realizations do not take advantage of the robustness that hierachical contexts afford to mapping and positioning procedures that their exploration capacities are much more limited than those of MonaLysa. The Nehmzow and Smithers's robot (Nehmzow and Smithers, 1991) cannot position itself in the environment if it doesn't keep following its boundary walls. Likewise, Matarić's robot (Matarić, 1992) and Kuipers and Byun's animat (Kuipers and Byun, 1991) rely heavily upon their wall-following and corridor-following exploration strategies to avoid getting lost. In contrast, the animat described in this paper is able to venture in free space and to position itself. However, the present work and that of Matarić have in common the fact that place-recognition not only depends upon topological information, but also upon simultaneously acquired metric information about distances and orientations. Likewise, the present work and that of Kuipers and Byun have in common the fact that place-recognition depends upon topological links that are indexed by the animat's control strategy. With MonaLysa, an actual move between two places can be matched to a possible move on the map provided they occur within the same skirting-task context. In Kuipers and Byun's approach, two moves can be matched if they are actuated by the same *follow-the-midline* or *move-along-object-on-right* control strategies.

8 Conclusion

It has been shown here that the MonaLysa control architecture endows an animat with map-building and self-positioning capacities that are robust with respect to noise. Such capacities rely upon mapping and positioning procedures that take into account specific landmarks and structures in the environment. This approach is original and exhibits several advantages in comparison with other realizations that have similar objectives. In particular it can be used in environments that need not be as carefully designed as usual. In the future, it will be extended so that, in addition to single landmarks and structures, it can characterize and recognize whole objects. Future work will also target the management of general skirting tasks that will hopefully cope with arbitrarily shaped obstacles. It will also be extended to the implementation of MonaLysa on a real robot and to the demonstration that its mapping and planning capacities make robust metric navigation possible.

Appendix

A Giving up skirting tasks

MonaLysa may decide to give up a given skirting task in three possible circumstances: before each elementary move, when the estimated cost of its current trajectory is too high; after the triggering of type 1 and type 2 mapping rules, when its map suggests that the corresponding obstacle is impassable; and after the triggering of a type 6 rule, if it decides it should not continue to skirt around an obstacle.

A.1 cost monitoring

Before deciding which move to make under the control of a given skirting task, MonaLysa probabilistically decides whether or not to give up this task, depending upon the value of a variable *Prob_givup*. This variable is permanently monitored and depends upon the estimated cost of the skirting trajectory:

- If $(Cost(T) < \text{Cmax})$ then
$$Prob\_givup(T) = 0$$
- If $(Cost(T) > 2 * Cmax)$ then
$$Prob\_givup(T) = 1$$
- If $(Cmax \leq Cost(T) \leq 2 * Cmax)$ then
$$Prob\_givup(T) = \frac{Cost(T) - Cmax}{Cmax}$$

where

Cost(T) : is the estimated cost of the trajectory followed since the triggering of task T. After each elementary move, it is incremented by the length of this move;

Cmax : is the cost threshold beyond which the animat decides that the corresponding obstacle is impassable. In an environment of length L and width l, this threshold is set to 4*(L+l). Therefore, in the experiments described herein, its value was equal to 6000 pixels.

In future implementations of MonaLysa, *Prob_givup* will depend upon other factors, like fatigue, boredom, curiosity or relative motivation strengths.

A.2 map information

The likelihood of any obstacle recorded in Monalysa's map being actually impassable is permanently monitored by a variable *Prob_impass* attached to every type 1 and type 2 mapping rules. At rule creation, the corresponding value is set to 0.

When the animat gives up a skirting task, the *Prob_impass* value of every type 1 or type 2 rule that refers to this skirting task is incremented according to the equation:

$$Prob\_impass(R)_{u+1} = Prob\_impass(R)_u + \alpha\_imp * (1 - Prob\_impass(R)_u)$$

where

$Prob\_impass(R)_u$: is the Prob_impass value of rule R after u updates;
α_imp : is a proportionality factor that is set to 0.1 in the present implementation.

When the animat arrives at a task-erasing place, the *Prob_impass* value of every type 1 or type 2 rule that refers to the skirting task to be erased is reset to 0.

MonaLysa is capable of detecting on its map that a given obstacle may be impassable at the moment when type 1 or type 2 rules are triggered and update the CH_list attached to the current skirting task. As many

hypotheses may refer to that obstacle at the same time, MonaLysa probabilistically decides whether or not to give up the current task, depending upon the minimum of the *Prob_impass* values that characterize the set of triggered rules.

A.3 obstacle leaving

In every place where a given skirting task is erased and a new skirting task is about to be triggered, a type 6 skirting rule is created, to which a *Trig_prob* value of 1 is given. Subsequently, every time the condition part of this rule is matched by the current situation, its *Trig_prob* value is decreased according to equation:

$$Trig\_prob(R)_{u+1} = (1 - \alpha\_tri) * Trig\_prob(R)_u$$

where

$Trig\_prob(R)_u$: is the Trig_prob value of rule R after u updates;
α_tri : is a proportionality factor that is set to 0.1 in the present implementation.

To avoid endlessly skirting around the same obstacle, MonaLysa probabilistically decides to trigger the new skirting task specified by a type 6 rule, depending upon the *Trig_prob* value of the rule.

B Position and error resetting

The value of variable β_maj, which is used by MonaLysa to reset the position and error estimates of a memorized landmark L, depends upon the relative uncertainty *Rel_uncert* of the landmark and of the animat's current position hypothesis (CPH) according to equations:

- If (Rel_uncert ≥ 0.5) then
$$\beta\_Maj = \alpha\_Maj * (2 * (1 - Rel\_uncert))$$

- If (Rel_uncert < 0.5) then

$$\beta\_Maj = \alpha\_Maj + (1 - \alpha\_Maj) * (1 - 2 * Rel\_uncert)$$

where

Rel_uncert is given by :

$$Rel\_uncert = \frac{Err(CPH)}{Err(L) + Err(CPH)}$$

and is set to 0.5 when *Err(L)* and *Err(CPH)* are both equal to 0;
α_Maj : is a parameter set to 0.1 in the present application.

C Strengths of new position hypotheses

The strength *Hyp_strength* of each position hypothesis H that is sent to the PH_list depends upon its *Max_supports* and *Nb_supports* values, according to the equation:

$$Hyp\_strength(H) = Match\_success(H) * Match\_hyp(H) * Match\_cph(H)$$

where

Match_success(H) : evaluates how often the corresponding position has been recognized as belonging to the sub-component of a structure memorized on the map. It is given by :

$$Match\_success(H) = 50 * (1 + \frac{Nb\_supports}{Max\_supports})$$

and is set to 100 if Max_supports equals 0, i.e. when H has been posted by the triggering of a type 1, 2 or 6 rule.

Match_hyp(H) : evaluates how well hypothesis H matches the hypotheses (Ei) of the PH_list. If it doesn't match any such hypothesis, *Match_hyp(H)* equals 0. Otherwise, it is given by:

$$Match\_hyp(H) = Max_{Ei}(1 - \frac{Dist(H, Ei)}{Err(H) + Err(Ei)})$$

Match_cph(H) : evaluates how much better hypothesis H matches the CPH than any other hypothesis (Ei) of the PH_list:

$$Match\_cph(H) = (1 - \frac{Dist(CPH, H)}{Max_{Ei}(Dist(CPH, Ei) + Err(Ei))})$$

Dist(A,B) : evaluates the distance between positions A and B;

D PH_list updates

The confidence estimate of every hypothesis of MonaLysa's PH_list is changed every time this list is updated. This occurs under three circumstances :

D.1 hypothesis generation

When a new hypothesis H is added to the PH_list, its confidence estimate $Conf(H)_0$ is initialized according to its *Hyp_strength* value:

$$Conf(H)_0 = Hyp\_strength(H) * \beta\_hyp$$

where β_hyp is a proportionality factor set to 0.5 in the present application.

D.2 hypothesis strengthening

When a hypothesis H that is sent to the place-recognition module matches a hypothesis Ei of the PH_list, the confidence estimate of Ei is reset according to equation:

$$Conf(Ei)_{u+1} = Conf(Ei)_u * (1 - \alpha\_struct) + Hyp\_strength(H) * \alpha\_struct$$

where

$Conf(Ei)_u$: is the confidence estimate of hypothesis Ei after u updates;
α_struct : is a proportionality factor that assesses the weight of hypothesis H. It gives greater emphasis to hypotheses that have been recognized as belonging to bigger structures :

$$\alpha\_struct = \alpha\_const + (1 - \alpha\_const) * \frac{Min(Max\_supports, Max\_struct)}{Max\_struct}$$

where

α_const : is a parameter set to 0.5 in the present application;
Max_supports (H) : is the size of the structure to which H has been recognized as belonging;
Max_struct (H) : is a parameter set to 5 in the present application.

D.3 hypothesis weakening

The confidence estimate of every hypothesis Ei that doesn't match any hypothesis H sent by the spatial-information processor is decreased according to equation:

$$Conf(Ei)_{u+1} = Conf(Ei)_u * (1 - \alpha\_strength) + \alpha\_close * \alpha\_strength$$

where

$Conf(Ei)_u$: is the confidence estimate of hypothesis Ei after u updates;
$\alpha\_strength$: is a proportionality factor that depends on the strengths of the new hypotheses and on the constant parameter α_const :

$$\alpha\_strength = \alpha\_const * Max_{Hj}(Hyp\_strength(Hj))$$

alpha_close : is a proportionality factor that depends on how distant Ei is from Hprox, the closest of the new hypotheses. This distance is compared to the maximum distance between Hprox and the other hypotheses of the PH_list:

$$\alpha\_close = Hyp\_strength(Hprox) * (1 - \frac{Dist(Hprox, Ei)}{MAX_{Ej}(Dist(Hprox, Ej))})$$

References

Bachelder, I.A. and Waxman, A.M. (1994). A neural system for qualitative mapping and navigation in visual environments. In Gaussier, P. and Nicoud, J. (Eds.), *From Perception to Action. Proceedings of the PerAc'94 Conference.* IEEE Computer Society Press.

Brooks, R.A. (1985). Visual map making for a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation.* St Louis, Missouri.

Burgess, N., Recce, M. and O'Keefe, J. (1994). A model of hippocampal function. *Neural Networks, 7*(6/7), 1065-1081.

Cartwright, B.A. and Collett, T.S. (1987). Landmark learning in bees. Experiments and models. *Journal of Comparative Physiology A, 151*, 521-543.

Chatila, R. and Laumond, J. (1985). Position referencing and consistent world modeling for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation.* St.Louis, Missouri.

Donnart, J.Y. (1997). *Architecture cognitive et propriétés adaptatives d'un animat autonome du point de vue motivationnel.* Thèse de Doctorat de l'Université de Paris 6 - Spécialité Biomathématiques - In press.

Donnart, J.Y. and Meyer, J.A. (1994). A hierarchical classifier system implementing a motivationally autonomous animat. In Cliff, D., Husband, P., Meyer, J.A. and Wilson, S.W. (Eds.), *From Animals to Animats 3. Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: MIT Press.

Donnart, J.Y. and Meyer, J.A. (1996). Learning reactive and planning rules in a motivationally autonomous animat. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, 26*(3), 381-395.

Gallistel, C.R. (1990). *The organization of learning.* Cambridge, MA: MIT Press.

Holland, J.H., Holyoak, K., Nisbett, R. and Thagard, P. (1986). *Induction. Processes of inference, learning and discovery.* Cambridge, MA: MIT Press.

Holland, J.H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R.S., Carbonell J.G. and Mitchell T.M. (Eds.), *Machine learning: An Artificial Intelligence approach.* Volume 2. San Mateo, CA: Morgan Kaufmann.

Kuipers, B.J. (1982). The ' map in the head ' metaphor. *Environment and Behavior, 14*(2), 202-220.

Kuipers, B.J. and Byun, Y.T. (1988). A robust, qualitative method for robot spatial learning. In *Proceedings of the AAAI conference.* St.Paul/minneapolis, MN.

Kuipers, B.J. and Byun, Y.T. (1991). A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems, 8*, 47-63.

Latombe, J.C. (1991). *Robot motion planning.* Kluwer Academic Publishers.

Levitt, T.S. and Lawton, D.T. (1990). Qualitative navigation for mobile robot. *Artificial Intelligence, 44*, 305-360.

Matarić, M.J. (1992). Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation, 8*(3), 304-312.

McFarland, D. and Bösser, T. (1993). *Intelligent behavior in animals and robots.* Cambridge, MA: MIT Press.

Muller, R.U., Kubie, J.L. and Saypoff, R. (1991). The hippocampus as a cognitive graph. *Hippocampus, 1*(3), 243-246.

Nehmzow, U. and Smithers, T (1991). Mapbuilding using self-organising networks in 'really useful robots'. In Meyer, J.A. and Wilson, S. (Eds.), *From Animals to Animats : Proceedings of the 1st International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: MIT Press.

Nehmzow, U. (1995). Animal and robot navigation. *Robotics and Autonomous Systems, 15*, 71-81.

O'Keefe, J. A. and Nadel, L. (1978). *The hippocampus as a cognitive map.* Oxford: Oxford University Press.

Penna, M.A. and Wu, J. (1993). Models for map building and navigation. *IEEE Transactions on Systems, Man, and Cybernetics, 23*(5), 1276-1301.

Poucet, B. (1993). Spatial cognitive maps in animals: New hypotheses on their structure and neural mechanisms. *Psychological Review, 100*, 163-182.

Riolo, R. L. (1991). Lookahead planning and latent learning in a classifier system. In Meyer, J.A. and Wilson, S. (Eds.), *From Animals to Animats: Proceedings of the 1st International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: MIT Press.

Roitblat, H. L. (1994). Mechanism and process in animal behavior: Models of animals, animals as models. In Cliff, D., Husband, P., Meyer, J.A. and Wilson, S.W. (Eds.), *From Animals to Animats 3. Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: MIT Press.

Schmajuk, N.A. and Thieme, A.D. (1992). Purposive behavior and cognitive mapping: A neural network model. *Biological Cybernetics, 67*, 165-174.

Schölkopf, B. and Mallot, H.A. (1995). View-based cognitive mapping and path planning. *Adaptive Behavior 3*(3), 311-348.

Thinus-Blanc, C. (1988). Animal spatial cognition. In Weiskrantz, L. (Ed.), *Thought without language.* Clarendon.

Tolman, E.C. (1948). Cognitive maps in rats and men. *The Psychological Review, 55*, 189-208.

Trullier, O., Wiener, S., Berthoz, A. and Meyer, J.-A. (1997). Biologically-based Artificial Navigation Systems: Review and Prospects. *Progress in Neurobiology.* In press.

Zipser, D. (1986). Biologically plausible models of place recognition and goal location. In Rumelhart, D.E. and McClelland, J.L. (Eds.), *Parallel Distributed Processing.* Cambridge, MA: MIT Press.