Robel : Synthesizing and Controlling Complex Robust Robot Behaviors

Morisset Benoit[1] and **Infantes Guillaume** and **Ghallab Malik** and **Ingrand Felix**[2]

Abstract. We present the Robel supervision system which is able to learn from experience robust ways to perform high level tasks (such as "navigate to"). Each possible way to perform the task is modeled as an Hierarchical Tasks Network (HTN), called *modality* whose primitives are sensory-motor functions. An HTN planning process synthesizes all the consistent modalities to achieve a task. The relationship between supervision states and the appropriate modality is learned through experience as a Markov Decision Process (MDP) which provides a general policy for the task. This MDP is independent of the environment and characterizes the robot abilities for the task.

Introduction

Robust robot navigation is a complex task which involves many different capabilities such as localization, terrain modeling, motion generation adapted to obstacles, and so on. Many sensory-motor (*sm*) functions have been developed and are available to perform navigation into structured (e.g. buildings) and unstructured (e.g. outdoor) environments. Since no single method or sensor has a universal coverage, each *sm* function has its specific weak and strong points. The approach presented here improves the global robustness of complex tasks execution in taking advantage of these *sm* functions complementarity.

To achieve these goals, we propose a two-stepped approach named Robel for *RO*bot *BE*havior *L*earning. First, *sm* functions are aggregated in a collection of Hierarchical Tasks Networks (HTN) [17], that are complex plans called *modalities*. Each modality is a possible way to achieve the desired task. One contribution of this work is to use and to synthesize modalities relying on the HTN formalism. In a second step, the relationship between supervision states and the appropriate modality for pursuing the task is learned through experience as a Markov Decision Process (MDP) which provides a policy for achieving the task. The second contribution of this work is an original approach for learning from the robot experiences an MDP-based supervision graph which enables to choose dynamically a modality appropriate to the current context for pursuing the task. We obtain a system able to efficiently use redundancies of low-level *sm* functions to robustly perform high-level tasks.

In the first Section we describe the *sm* functions and briefly detail their forces and weaknesses. In Section 2 we introduce what are the modalities, the one written by hand or better, how we can synthesize them automatically using a planner. Section 3 describes the *controller* and the learning mechanism which has been deployed to choose at run time the appropriate modality for pursuing a task in the current environment. Finally we present the results obtained both on the modalities planning/synthesizing problem and on learning their best use. We conclude with a discussion, and a prospective on this subject.

[1] SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, USA. email: morisset@ai.sri.com

[2] LAAS-CNRS, 7 avenue du Colonel Roche 31077 Toulouse Cedex 4 FRANCE email: firstname.lastname@laas.fr

1 Sensory-Motor Functions

Thanks to years of research in robotics, a large number of *sm* functions are now available, in particular for navigation tasks. To give an idea of this diversity, we briefly present them, according to the main functionality they provide.

Reliable and precise localization is often hard to obtain, numerous methods have been developed to provide this functionality. *Odometry* is easy to use but, due to drift and slippage, is seldom precise enough to perform long-range navigation. *Segment-based localization* is generally reliable in indoor environment [16], but laser occlusion gives unreliable data. Moreover, in long corridors the laser get little data along the corridor axis, thus the drift increases. *Stereo Vision odometry* [11] is more precise than classic odometry, but requires heavy computations which limits drastically the refresh rate of the current position estimate. Moreover, it is very sensible to any parameters which may impede the stereo correlation. *Global Positioning System* can be used for absolute localization, although one need a differential system to have an accurate measure. *Localization on landmarks* such as wall posters in long corridors can provide an accurate localization [2]. However, landmarks are usually only available and visible in few areas of the environment.

According to the type of mission and environment, different type of path planners can be used: *nav* [22] or *m2d* [21]. The first one is better for exploration of unknown regions, but at a high computational cost. The second one is fairly generic and robust, although it may require further processing to get an executable dynamic trajectory to take into account environment changes that occur during navigation.

For locomotion on rough terrains, we use a motion planner/motion generator named *P3D* [5] which is designed essentially for exploration of static environments. Motion control in dynamic environments has been implemented using *ND* [14]. It offers reactive motion capability that remains efficient in very cluttered space, but may fall in local minima. The *elastic band* [19] gives a very robust method for long range navigation, but can be blocked by a mobile obstacle that traps the band against a static obstacle. Last, the dynamic deformation is computationally intensive and may limit the reactivity in cluttered, dynamic environments and may also limit the band length.

2 Synthesis of Modalities

A high level task given by a mission planning step requires an integrated use of several *sm* functions among those presented earlier. Each consistent combination of these *sm* functions is a particular plan called a *modality*. A modality is one way of performing the task. A modality has specific characteristics that make it more appropriate for some contexts or environments, and less for others. The choice of the right modality for pursuing a task is far from being obvious. The goal of the *controller* (see Section 3) is to perform such a selection all along the task execution.

We chose to represent modalities as Hierarchical Task Networks. We believe that the HTN formalism is adapted to modalities because of its expressiveness and its flexible control structure [8]. HTNs offer a middle ground between programming and automated planning, allowing the designer to express the control knowledge which is available here. So we can use the same formalism to write a modality by hand or to generate it automatically.

2.1 Model of data flow

Creating a consistent modality may be seen as a planning problem using the correct models. To build a coherent modality, we want the planner to produce a plan which properly connect the available modules implementing the *sm* functions. So we need to model the *sm* functions previously described from a data flow point of view. We give a module a semantic by considering it as a black box with some input and output data. By correctly typing the data, and choosing the right *sm* functions, we are able to build a coherent chain of data processing, for a navigation modality. This chain goes from external data to the goal, through sensors, map building and motion generation of a real movement (see Fig. 1).

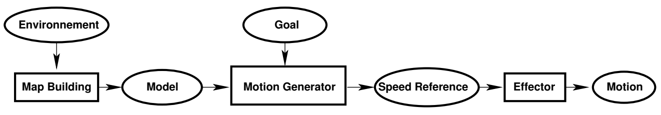

Figure 1. Example of a data flow

This scheme might be generalized easily to other kinds of modality, like exploration, manipulation or pure learning, depending on the available functional modules.

To perform this planning task we use SHOP [18], a hierarchical task planner. It uses a description of the domain, operators, and decomposition methods to go from a high level tasks to terminal actions. To get SHOP to produce the correct connections between modules, we set some planning operators:

```
(!connect_as_prod ?func ?data)
(!connect_as_cons ?func ?data)
```

The first one says that a function produces a type of data, the other that a function consumes a type of data. This operators assert the corresponding predicates, to explicit the current state:

```
(connected_producer ?func ?data)
(connected_consumer ?func ?data)
```

We also have to specify correctly the functions. This is achieved as follows:

```
(function lane.fuse)
  (needs lane.fuse (3D_image video_image))
  (produces lane.fuse 3D_model)
```

In the initial state of the problem, the only available data is input data such as *goal*, and we want to reach a state where the *motion* data is available. We implement a backward-chaining mechanism that starts from the final state and choose an operator (a *sm* function) that will provide this data. The new current state needs the input data of our operator. If they are not available, we choose a new function to produce them and so on. The backtrack points of the search are the choice at this time of the *sm* function. If we need a position (for localization), we may choose as different functions as *localization on landmarks* or *visual odometry* to perform it. We may even choose both of them using some data fusion mechanism available on our robots. This regression/decomposition process stops when all required data are available, typically provided by a sensor.

Thus we have the methods:

```
(:method (find_producer ?d)
         ((connected_producer ?f ?d))
         ()
         ((produces ?f ?d))
         ((!connect_as_prod ?f ?d)
          (find_consumable ?f)))
(:method (find_consumable ?f)
         ((needs ?f ?data)
         ((connect_as_cons ?f ?data)
          (find_producer ?data))))
```

The first one is to find the producers for a type of data. If this data is already produced, we stop the decomposition (this is the meaning of the empty parenthesis), else we find a producer and continue. The second one is straightforward.

However, building a modality is not just linking a network of producers/consumers. If this process can synthesize simple modalities, most complex ones require other attributes and parameters to be taken into account.

2.2 Other Attributes

We list here the other attributes the modality planning system has to take into account, like the **resources**. A physical device may be required, at the same time, by two different modules in the same modality. Similarly, CPU and Memory usage are interesting resources to model too. Such resources usage could be expressed as constraints in the HTN formalism. The **time** is also important: most modules are synchronous and have their own frequency, which on our robots vary from 50 Hz to less than 1 Hz. A realistic model must take this into account to ensure a correct execution. We may also notice the some processing are done **on request**, when instructed by another module or the executive, or can be **started and stopped** and produce some data at a given period. A **synchronous/asynchronous** attribute is necessary: some execution requests may be synchronized with respect to others (waiting for some data to be available) or may be running at their own pace, using whatever data is then available.

As of today, the current implementation of the modalities synthesizing part of Robel uses the data flow model as well as a simple synchronous/asynchronous and cyclic/on request model. Still, we are able to produce interesting modalities (See Section 4.1). Nevertheless, using a complete model taking into account all the attributes, we expect to be able to automatically produce modalities as rich and as robust than the handwritten ones.

The solution to this problem gives an a priori valid modality (with respect to the model), which can then be “tested” on line, thus allowing the *controller* to learn in which situations it is appropriate to use it. To conclude on this part, we may say that we are now able to automatically generate the executable code of a modality from a few information on how the *sm* work. The next challenge is to learn to use them efficiently.

3 The Controller

3.1 Qualitative Model of the Environment

We present in this section an example of a controller adapted to an indoor navigation task. To perform this specific task, the design of the control space and the control process itself require the use of a topological graph. Cells are polygons that partition the metric map. Each cell is characterized by its name and a *color* that corresponds to navigation features such as *Corridor, Corridor with landmarks, Large Door, Narrow Door, Confined Area, Open Area and so on*. Edges of the topological graph are labeled by estimates of the transition length from one cell to the next and by heuristic estimates of how easy such a transition is.

3.2 The Control Space

The controller has to choose a modality that is most appropriate to the current execution state for pursuing the task. In order to do this, a set of *control variables* has to represent control information for the *sm* functions. The choice of these control variables is an important design issue.

For example, in the navigation task in an indoor environment, the control variables are:

- the **cluttering** of the environment which is defined to be a weighted sum of the distances to nearest obstacles perceived by the laser, with a dominant weight along the robot motion axis;
- the **angular variation** of the profile of the laser range data which characterizes the robot area. Close to a wall, the cluttering value is high but the angular variation remains low. But in an open area the cluttering is low while the angular variation may be high;
- the **inaccuracy of the position estimate**, as computed from the co-variance matrix maintained by each localization *sm* function;
- the **confidence** in the position estimate (because the inaccuracy is not sufficient to qualify the localization, each localization *sm* function supplies a confidence estimate about the last processed position);
- the **navigation color** of current area is used when the robot position estimate falls within some labeled cell of the topological graph, the corresponding labels are taken into account;
- the **current modality** is essential to assess the control state and possible transitions between modalities.

A control state is characterized by the discretized values of these control variables. We finally end-up with a discrete control space which allows us to define a *control automaton*.

3.3 The Control Automaton

The control automaton is nondeterministic: unpredictable external events may modify the environment, e.g. someone passing by may change the value of the cluttering variable, or the localization inaccuracy variable. Therefore the execution of the same modality in a given state may lead to different adjacent states. This nondeterministic control automaton is defined as the tuple $\Sigma = \{S, A, P, C\}$:

S is a finite set of control states,

A is a finite set of modalities,

$P : S \times A \times S \rightarrow [0, 1]$ is a probability distribution on the state-transition, $P_a(s'|s)$ is the probability that the execution of modality a in state s leads to state s',

$C : A \times S \times S \rightarrow \Re^+$ is a positive cost function, $c(a, s, s')$ corresponds to the average cost of performing the state transition from s to s' with the modality a.

A and S are given by design from the definition of the set of modalities and of the control variables. P and C are obtained from observed statistics during a learning phase.

The Control automaton Σ is a Markov Decision Process. As an MDP, Σ could be used reactively on the basis of a universal policy π which selects for a given state s the best modality $\pi(s)$ to be executed. However, a universal policy will not take into account the current navigation goal. A more precise approach takes into account explicitly the navigation goal, transposed into Σ as a set S_g of goal states in the control space. This set S_g is given by a look-ahead mechanism based on a search for a path in Σ that reflects a topological route to the navigation goal.

3.3.1 Goal States in the Control Space

Given a navigation task, a search in the topological graph provides an optimal route r to the goal, taking into account estimated cost of edges between topological cells. This route will help finding in the control automaton desirable control states for planning a policy. The route r is characterized by the pair (σ_r, l_r), where $\sigma_r = \langle c_1 c_2 \ldots c_k \rangle$ is the sequence of colors of traversed cells, and l_r is the length of r.

Now, a path between two states in Σ defines also a sequence of colors σ_{path}, those of traversed states; it has a total cost, that is the sum $\sum_{path} C(a, s, s')$ over all traversed arcs. A path in Σ from the current control state s_0 to a state s corresponds to the planned route when the path *matches* the features of the route (σ_r, l_r) in the following way:

- $\sum_{path} c(a, s, s') \geq K l_r$, K being a constant ratio between the cost of a state-transition in the control automaton to corresponding route length,
- σ_{path} corresponds to the same sequence of colors as σ_r with possible repetition factors, i.e., there are factors $i_1 > 0, \ldots, i_k > 0$ such that $\sigma_{path} = \langle c_1^{i_1}, c_2^{i_2}, \ldots, c_k^{i_k} \rangle$ when $\sigma_r = \langle c_1, c_2, \ldots, c_k \rangle$.

This last condition requires that we will be traversing in Σ control states having the same color as the planned route. A repetition factor corresponds to the number of control states, at least one, required for traversing a topological cell. The first condition enables to prune paths in Σ that meet the condition on the sequence of colors but cannot correspond to the planned route. However, paths in Σ that contain a loop (i.e. involving a repeated control sequence) necessarily meet the first condition.

Let $\mathsf{route}(s_0, s)$ be true whenever the optimal path in Σ from s_0 to s meets the two previous conditions, and let $S_g = \{s \in S \mid \mathsf{route}(s_0, s)\}$. A Moore-Dijkstra algorithm starting from s_0 gives optimal paths to all states in Σ in $O(n^2)$. For every such a path, the predicate $\mathsf{route}(s_0, s)$ is checked in a straightforward way, which gives S_g. It is important to notice that this set S_g of control states is a *heuristic projection* of the planned route to the goal. There is no guaranty that following blindly (i.e., in an open-loop control) a path in Σ that meets $\mathsf{route}(s_0, s)$ will lead to the goal, and there is no guarantee that every successful navigation to the goal corresponds to a sequence of control states that meets $\mathsf{route}(s_0, s)$. This is only an efficient and reliable way of focusing the MDP cost function with respect to the navigation goal and to the planned route.

3.3.2 Finding a Control Policy

At this point we have to find the best modality to apply to the current state s_0 in order to reach a state in S_g, given the probability distribution function P and the cost function C. A simple adaptation of the *Value Iteration* algorithm solves this problem. Here we only need to know $\pi(s_0)$. Hence the algorithm can be focused on a subset of states, basically those explored by the Moore-Dijkstra algorithm.

The closed-loop controller uses this policy as follows:

- the computed modality $\pi(s_0)$ is executed;
- the robot observes the state s, it updates its route r and its set S_g of goal states, it finds the new modality to apply to s.

This is repeated until the control reports a success or a failure. Recovery from a failure state consists in trying from the parent state an untried modality. If none is available, a global failure of the task is reported.

3.3.3 Estimating the Parameters of the Control automaton

A sequence of randomly generated navigation goals is given to the robot. During its motion, new control states are met and new transitions are recorded or updated. Each time a transition from s to s' with modality a is performed, the traversed distance and speed are recorded, and the average speed v of this transition is updated. The cost of the transition $C(a, s, s')$ can be defined as a weighted average of the traversal time for this transition taking into account the eventual control steps required during the execution of the modality a in s together with the outcome of that control. The statistics on $a(s)$ are recorded to update the probability distribution function. Several strategies can be defined to learn P and C in Σ. The first one is used initially to expand Σ: a modality is chosen randomly for a given task; this modality is pursued until either it succeeds or a fatal failure is notified. In this case, a new modality is chosen randomly. This strategy is used initially to expand Σ. Σ is used according to the normal control except in a state on which not enough data has been recorded; a modality is randomly applied to this state in order to augment known statistics, e.g, the random choice of an untried modality in that state.

4 Experimental results

The justification of the whole system relies on the following principle : the use of the complementarity of several navigation modalities increases the global robustness of the task execution. To validate this principle, 5 handwritten modalities have been integrated inboard one of our robot. In order to characterize the usefulness domain of each modality we measured in a series of navigation tasks, the success rate and other parameters such as the average speed, the distance covered, the number of retries. Various cases of navigation have been considered such as for instance, long corridors or large areas, cluttered or not, occluding the 2D characteristic edges of the area or not. These extensive experiments described in details in [15] required several kilometers of navigation. The result is that for each case of navigation met by the robot there is at least one successful modality. On the other hand, no modality is able to cover all cases. This result clearly supports our approach of a supervision controller switching from one modality to another one according to the context.

A second step of experiments is focused on the automatic modalities synthesis by a planning process. These ongoing results are presented in the next section. Finally, the learning capabilities of the controller are illustrated in section 4.2.

4.1 Modalities Synthesis

Let us give some examples of synthesized modalities. On Fig. 2, we can see the HTN built by SHOP. The corresponding modality is shown on Fig. 3.

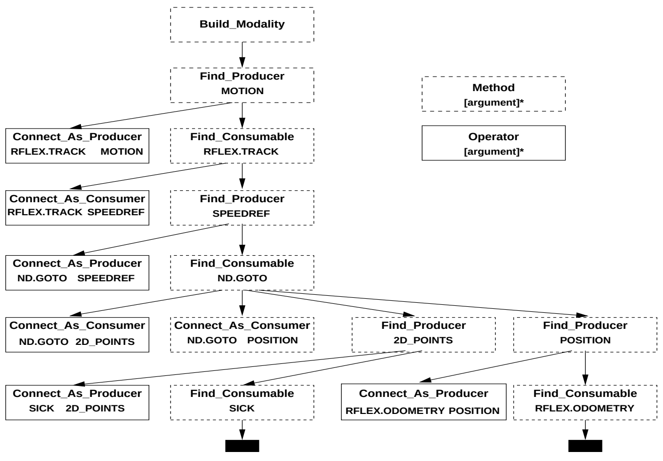
Build_Modality
Find_Producer MOTION
Method [argument]*
Connect_As_Producer RFLEX.TRACK MOTION
Find_Consumable RFLEX.TRACK
Operator [argument]*
Connect_As_Consumer RFLEX.TRACK SPEEDREF
Find_Producer SPEEDREF
Connect_As_Producer ND.GOTO SPEEDREF
Find_Consumable ND.GOTO
Connect_As_Consumer ND.GOTO 2D_POINTS
Connect_As_Consumer ND.GOTO POSITION
Find_Producer 2D_POINTS
Find_Producer POSITION
Connect_As_Producer SICK 2D_POINTS
Find_Consumable SICK
Connect_As_Producer RFLEX.ODOMETRY POSITION
Find_Consumable RFLEX.ODOMETRY

Figure 2. HTN built by SHOP for a simple modality

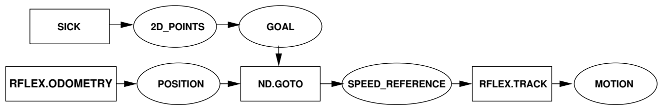

Figure 3. A simple modality

The *odometry sm* function does not need any data at input, nor does the sick (laser range finder). The *ND* motion generator uses points from the sick to find out where are obstacles and gives the speed reference to avoid obstacles and go to the goal.

This modality is better suited for exploration of “slightly” dynamic environments, at low speed. This modality is the most simple of the 42 generated for the outdoor mobile robot, using 16 *sm* functions

Another (more complex) modality is shown on Fig. 4. We can see that the robot uses its cameras to take images, and stereo-correlation to have a 3D image. From here, it uses this image to compute a motion (*Stereo Vision Odometry*) which combined with classic odometry will give the localization. The modality also builds a 3D metric model, the corresponding 3D qualitative model and projects it to obtain a 2D qualitative model. This model is used to give long range path made of way points. This way points are consumed one by one by a motion generator that gives speed references using the 3D metric model (and of course the current position). Then this speed reference is used by the low-level effector to generate effectively the robot motion.

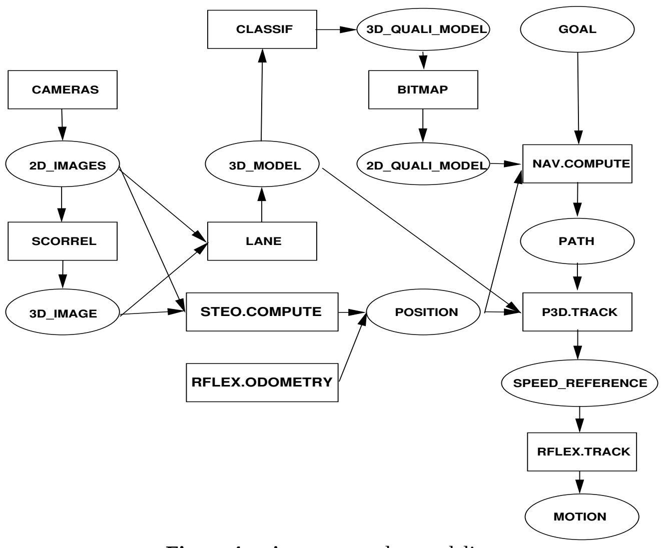

Figure 4. A more complex modality

The problem we are facing now is to define criterion and ways for selecting a *good* set of modalities. A too large set will make the learning of the controller unrealistically long and costly, but we obviously need a sufficiently large set to cover the main ways of combining the *sm* functions. For the moment, this selection is performed interactively by the robot designer. This still provides a significant benefit in robustness, programming and debugging time w.r.t. handwriting the modalities.

4.2 Controller

. We propose here to illustrate the learning capabilities of the controller through an indoor navigation task. To perform this experiment, we start with an empty automaton and 2 complementary modalities: the first one (M_1) is composed by the *elastic band*, *m2d* and the *segment based* localization function, while the second one (M_2) works reactively without any path planner. *ND* performs the obstacle avoidance and the robot is localized with the same function as M_1. The velocity and the path planner make M_1 more efficient in large and open environments. On the other hand, the limited avoidance capabilities of the *elastic band* makes M_2 more adapted in highly cluttered environments.

In this three-stepped experiment, the strategy of learning favors the completion of each transition(3.3.3). If a modality has not been tried for the current state (untried modality), this modality is executed without any computation of π.

Phase 1. The learning starts with a series of 83 navigations in a large open environment. During these navigations, 86 states and 159 transitions are created (Fig. 5). After the 53^{th} navigation (noted $n53$) the number of new transitions met by the system tends to be stable. Between $n53$ and $n83$, the computation of π returns M_1 for any state encountered except for 2 states (in $n60$ and $n70$) whose transitions with M_2 were still untried (Fig. 6). The constant selection of M_1 by π all along the 30 last navigations shows that the controller relevantly learned the superiority of M_1 on M_2 for the open environments.

Phase 2. Some narrow obstacles are added. This new situation generates 14 new states between $n84$ et $n87$ and 10 new transitions are tried until $n93$. During these 9 first navigations, 6 failures are recorded for M_1, each time from a state with a high level for the clutter variable. After $n101$ and until $n114$, each time a state with a high level for the clutter variable is encountered, the execution of M_1 is stopped by the controller and the obstacle avoidance is systematically performed with M_2. No more failures are recorded with M_1. As soon as the clutter level recovers a low level, the computation of π switches back to a selection of M_1. M_1 is then kept as long as the value of the clutter state variable stays low. If in the previous phase, M_1 was more appropriate than M_2, in this second phase, the system is able to learn within 30 navigations, the better efficiency of M_2 in cluttered environments.

Phase 3. The goal of the third step is to check if the learning of the second phase (avoidance) didn't corrupt the learning of the first phase (open navigation). The obstacles are then removed to recover the same environment as the phase 1 and 58 more navigations are performed by the system. This new step shows that the learning of the phase 1 was not complete: 10 new states are created and 23 untried transitions are completed. Despite these untried transitions, between $n114$ and $n152$, 30 navigations are performed with 100 % of selection for M_1 by π. After $n151$, M_1 is constantly selected all along the 21 last navigations. This last step shows that despite an incomplete learning, the efficiency of M_1 in the phase 1 has not been forgotten after the learning of the phase 2.

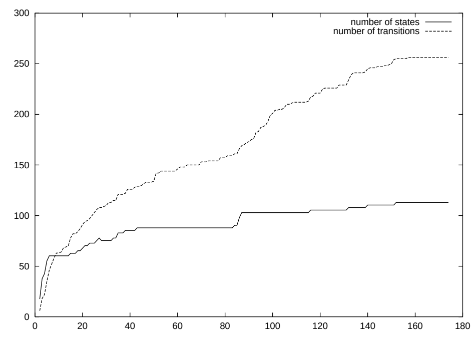

Figure 5. Evolution of the size of the graph

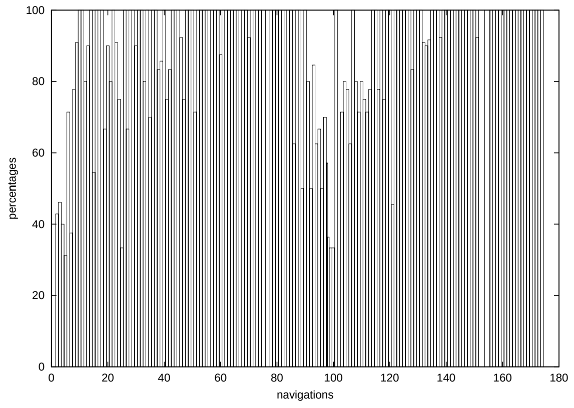

Figure 6. Percentage of choice of M_1

Discussion and Conclusion

This paper addressed the issue of producing complex modalities from sensory motors functions, and how to exploit the complementarity of these modalities to perform a task.

We have shown that it is indeed feasible to synthesize modalities from generic specifications, we still need to improve this planning component to take into account attributes such as resource, time and synchronism.

This is certainly not the first contribution that relies on a planning formalism and on plan-based control in order to program an autonomous robot. For example, the *"Structured Reactive Controllers"* [3] are close to our concerns and have been demonstrated effectively on the Rhino mobile robot. The efforts for extending and adapting the Golog language [12] to programming autonomous robots offer another interesting example from a quite different perspective, that of the Situation Calculus formalism [20]. The *"societal agent theory"* of [13] offers also another interesting approach for specifying and combining sequentially, concurrently or in a cooperative mode several agent-based behaviors; the CDL language used for specifying the agent interactions is similar to our Propice programming environment. Let us mention also the *"Dual dynamics"* approach of [10] that permit the flexible interaction and supervision of several behaviors. These are typical examples of a rich state of the art on possible architectures for designing autonomous robots. (see [1] for a more comprehensive survey).

Here also the use of MDPs for supervision and control of robot navigation tasks is not new. Several authors expressed directly Markov states as cells of a navigation grid and addressed navigation through MDP algorithms, e.g. value iteration [23, 6, 7]. Learning systems have been developed in this framework. For example, XFRMLEARN extends these approaches further with a knowledge-based learning mechanism that adds subplans from experience to improve navigation performances [4]. Other approaches considered learning at very specific levels, e.g., to improve path planning capabilities [9]. Our approach stands at a more abstract and generic level. It addresses another purpose: acquiring autonomously the relationship from the set of supervision states to that of redundant modalities. We have proposed a convenient supervision space. We have also introduced a new and effective search mechanism that projects a topological route into the supervision graph. The learning of this graph relies on simple and effective techniques, whose results provide two particular features:

Portability: Variables of the control state reflect control information for the *sm* functions. No information dedicated to the environment is present in the control state. In this sense we say that the control state is *abstract*. Thanks to this characteristic, a controller learned in an environment can directly be used in another environment.

Adaptativity: In this system, learning and execution are not decoupled : learning of Σ parameters is active all along the robot navigations. If a new situation is encountered, corresponding new states are created in Σ and the new untried transitions are evaluated and taken into account by the next computations of π. This unsupervised learning confers a high level of adaptativity to the controller.

In addition to future work directions mentioned above, an important test of Robel will be the extension of the set of tasks to manipulation tasks such as *"open a door"*. This significant development will require the integration of new manipulation functions, the synthesizing of new modalities for these tasks and the extension of the controller state. Another development which seems rather promising is to learn the control space of the controller instead of relying on one given by hand.

REFERENCES

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, 'An Architecture for Autonomy', *IJRR*, **17**(4), 315–337, (April 1998).

[2] V. Ayala, J.B. Hayet, F. Lerasle, and M. Devy, 'Visual localization of a mobile robot in indoor environments using planar landmarks', in *IEEE IROS'2000, Takamatsu, Japan*, pp. 275–280, (November 2000).

[3] M. Beetz, 'Structured reactive controllers - a computational model of everyday activity.', in *3rd Int. Conf. on Autonomous Agents*, (1999).

[4] M. Beetz and T. Belker, 'Environment and task adaptation for robotics agents', in *ECAI*, (2000).

[5] D. Bonnafous, S. Lacroix, and T. Simon, 'Motion generation for a rover on rough terrains', in *International Conference on Intelligent Robotics and Systems*, Maui, HI (USA), (October 2001). IEEE.

[6] Cassandra, Kaelbling, and Kurien, 'Acting under uncertainty: Discrete bayesian models for mobile robot navigation', in *Proceedings of IEEE/RSJ IROS*, (1996).

[7] T. Dean and M. Wellman, 'Planning and control', in *Morgan Kaufmann*, (1991).

[8] K. Erol, J. Hendler, and D.S. Nau, 'HTN planning: Complexity and expressivity.', in *AAAI*, (1994).

[9] K. Z. Haigh and M. Veloso, 'Learning situation-dependent costs: Improving planning from probabilistic robot execution', in *In 2nd Int. Conference on Autonomous Agents*, (1998).

[10] J. Hertzberg, H. Jaeger, P. Morignot, and U. R. Zimmer, 'A framework for plan execution in behavior-based robots', in *ISIC-98 Gaithersburg MD*, pp. 8–13, (1998).

[11] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila, 'Autonomous rover navigation on unknown terrains, functions and integration', *International Journal of Robotics Research*, (2003).

[12] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl, 'GOLOG: A logic programming language for dynamic domains', *Journal of Logic Programming*, **31**(1-3), 59–83, (1997).

[13] D. C. MacKenzie, R. C. Arkin, and J. M. Cameron, 'Multiagent mission specification and execution', in *Autonomous Robots, 4(1):29 V52*, (1997).

[14] J. Minguez, L. Montano, T.Simeon, and R. Alami, 'Global nearness diagram navigation (GND)', in *ICRA2001, Korea*.

[15] B. Morisset, *Vers un robot au comportement robuste. Apprendre combiner des modalits sensori-motrices complmentaires.*, Ph.D. dissertation, Université Paul Sabatier, Toulouse, novembre 2002.

[16] P. Moutarlier and R. G. Chatila, 'Stochastic Multisensory Data Fusion for Mobile Robot Location and Environment Modelling', in *Proc. International Symposium on Robotics Research, Tokyo*, (1989).

[17] D. Nau, Y. Caoand, A. Lotem, and H. Munoz-Avila., 'Shop: Simple hierarchical ordered planner', in *IJCAI*, (1999).

[18] D. Nau, H. Munoz-Avila, Y. Cao, A. Lotem, and S. Mitchell, 'Total-order planning with partially ordered subtasks', in *IJCAI*, Seatle, (2001).

[19] S. Quinlan and O. Khatib, 'Towards real-time execution of motion tasks', in *Experimental Robotics 2*, eds., R. Chatila and G. Hirzinger, Springer Verlag, (1992).

[20] R. Reiter, 'Natural actions, concurrency and continuous time in the situation calculus.', in *KR*, pp. 2–13, (1996).

[21] T. Simeon and B. Dacre Wright, 'A practical motion planner for all-terrain mobile robots', in *IEEE/RSJ IROS*, (1993).

[22] S.Lacroix, I.K.Jung, J.Gancet, and J.Gonzalez, 'Towards long range autonomous navigation', in *7th ESA Workshop on Advanced Space Technologies for Robotics and Automation*, Noordwijk (The Netherlands), (November 2002).

[23] S. Thrun, A. Buecken, W. Burgard, D. Fox, T. Froehlinghaus, D. Henning, T. Hofmann, M. Krell, and T. Schmidt, 'Map learning and high-speed navigation in rhino', in *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, eds., D. Kortenkamp, R.P. Bonasso, and R. Murphy. MIT Press, (1998).