Learning from History for Adaptive Mobile Robot Control

François Michaud

Dept. Elect. & Comp. Engineering
Université de Sherbrooke
Sherbrooke (Québec Canada) J1K 2R1
michaudf@gel.usherb.ca

Maja J Matarić

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
mataric@cs.usc.edu

Abstract

Learning in the mobile robot domain is a very challenging task, especially in non-stationary conditions. This paper presents an approach that allows a robot to learn a model of its interactions with its operating environment in order to manage them according to the experienced dynamics. The robot is initially given a set of "behavior-producing" modules to choose from, and the algorithm provides a means of making that choice intelligently and dynamically. The approach is validated using a vision- and sonar-based Pioneer I robot in non-stationary conditions, in the context of a multi-robot foraging task. Results show the effectiveness of the approach in taking advantage of any regularities experienced in the world, leading to fast and adaptable specialization for the learning robot.

1 Introduction

Learning in the mobile robot domain requires coping with a number of challenges, starting with difficulties intrinsic to the robot. These difficulties are caused by incomplete, noisy, and imprecise perception and action, and by limited processing and memory capabilities. These affect the robot's ability to represent the environment and to evaluate the events that take place. Furthermore, stationary conditions (or slowly-varying non-stationary conditions) are typically required to enable a robot to learn a stable controller or a model of the environment. However, in changing environments such as the multi-robot domain, the dynamics are particularly important, making the learning task even more difficult. Finally, learning must take place while underlying situations and conditions (like obstacle avoidance, navigation, tasks) are being managed in real-time.

Perhaps the most workable approach for learning, from both the engineering and evaluation perspectives, is to initially endow the system with some controller, i.e., some initial policy, and then allow it to further refine and alter that controller over time, as needed. This approach bypasses complete autonomy (by giving up *tabula rasa* learning techniques [3]) in favor of incorporating a bias that accelerates the learning process to achieve worthwhile performance rapidly. To do so, the control policy can be decomposed into "behavior-producing" modules, also called behaviors [1]. This approach has been praised for its robustness and simplicity of construction. In a typical behavior-based system, the constituent behaviors are designed parsimoniously, executed in parallel, and prioritized using some fixed or flexible arbitration mechanism [7]. A fixed and parsimonious behavior set, however, does not allow a robot to easily adapt to changes in the environment, thus resulting in diminished flexibility. The ability to adapt to changing dynamics, however, is especially important for robots operating in unpredictable and non-stationary environments.

To overcome this problem, we present an approach that introduces a modeling and reasoning component into the behavior-based framework. The robot is initially given a set of behaviors, whose subset is used to accomplish the assigned tasks safely. By knowing the purpose of each behavior and by using a representation mechanism based on the history of behavior use over time, the robot can autonomously evaluate and model its interactions with the environment. Using this information, our memory-based approach can then select alternative behaviors that are not normally used, in order to change the way the robot responds to the perceived conditions experienced. We demonstrated and validated our approach on a vision- and sonar-based Pioneer I robot in the context of foraging (object collection) in a multi-robot environment.

2 History-based learning for adaptive behavior selection

Within the behavior-based framework, a robot needs different types of "behavior-producing" modules to operate in its environment. Behaviors for specific operations (like search-for-object, go-to-place, pick-object, drop-object) are required by the robot to accomplish the assigned tasks. Others are required for safe navigation in the environment (like avoid-obstacles). We call the first type "task-behaviors", and the second "maintenance-behaviors". Both types are triggered in response to specific conditions in the environment, and these conditions can be pre-programmed. The robot can also use behaviors that introduce variety into its action repertoire (like wall-following, small-turn, rest, etc.). We call these "alternative-behaviors". In contrast to the other types of behaviors, no *a priori* conditions are given to activate alternative-behaviors. The algorithm we describe learns to activate them according to the robot's past experiences.

In our approach, as behaviors are executed, their sequence is stored within a tree structure representing the history of their use. The nodes of the tree store two items: the name or the type (for task-behaviors) of the behavior (both represented by a letter, e.g., A for *Avoidance*) being executed, and the number of times a transition between the node itself and its successor has been made, as shown in Figure 1. Initially, the tree for a particular task is empty and is incrementally constructed as the robot goes about its task. Leaf nodes are characterized by the letter E (for end-node) and store the total *performance* of the particular tree path. Whenever a path is completely reused, i.e., when the same sequence of behaviors is executed, the average of the stored and current performances for the last 10 trials is computed and stored in E. The history of behavior use and the resulting performance values give the robot the ability to learn to trigger alternative-behaviors according to the dynamics observed of the robot in the environment.

Performance can be evaluated using a variety of different factors. Given a dynamic multi-robot domain in which optimality is difficult to define (and may be non-stationary), we explored time of behavior use as the key evaluation metric. However, other metrics could be used in different domains and for different tasks. Our evaluation function, expressed by Relation 1, is characterized by the amount of time required to accomplish the task and the interference experienced during that process:

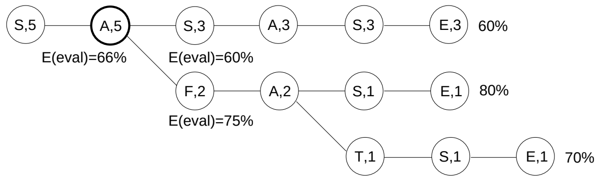

Figure 1: Tree representation of history of behavior use.

$$eval(t) = \max\left(0, \frac{(t_{tb} - t_{mb} + t_c)}{t_{tb}} - \max\left(0, \frac{(t - TT)}{TT}\right)\right) \quad (1)$$

where tb refers to task-behaviors, mb to maintenance-behaviors, and t to the current time-step (corresponding to one cycle of evaluation of all currently active behaviors). TT represents the average total number of time-steps taken to accomplish the task over the last 10 trials; it captures any changes in the dynamics of the environment that affect the time to complete the task. t_c is a correction factor that increases when $eval(t)$ decreases below 0, to boost $eval(t)$ as soon as the robot uses its task-behaviors. The first term of Relation 1 reflects the amount of control the robot has over the task compared to the effect of other events that maintenance-behaviors must handle. The second term penalizes the robot's performance if it takes more time to accomplish the task than it has in the past.

In addition to being used at end-nodes, $eval(t)$ is also used to evaluate the progress being made and to decide whether an alternative-behavior should be tried (each alternative-behavior corresponds to an option; not selecting any alternative-behavior is called the *Observe* option). At a given position in the tree, comparison of $eval(t)$ with the expected performance $E(eval)$ evaluated from the subpaths is used to anticipate the future outcomes of the robot's behavior choices. From a node in the tree, $E(eval)$ is the sum of the stored performances, multiplied by the frequency of use of the subpaths relative to the current position in the tree. For example, using the bold node in Figure 1 as the current node, the expected performance of the S subpath is $60 \cdot 3/3 = 60\%$, while the expected performance of the F subpath is $(80 \cdot 1/2) + (70 \cdot 1/2) = 75\%$: $E(eval)$ for this node is $(60 \cdot 3/5) + (75 \cdot 2/5) = 66\%$.

Using the expected performances at the current position in the tree and those at each of its subpaths, the choice made by the algorithm is based on three criteria (TM to exploit an option, and UO and GOT to explore the usefulness of options), as follows:

IF $(eval(t) \geq E(eval))$ THEN
    Take subpath with $max(E_i(eval))$ (**TM** criterion)
ELSE
    IF there are untried options at this junction THEN
        Select Untried Option (**UO** criterion)
    ELSE
        Select the option with overall best performance
        from the Global Options Table (**GOT** criterion)

Overall, the operation of the learning algorithm consists of updating the tree according to the currently used behavior, and selecting alternative-behaviors based on the criteria above. Note that no search is being performed; the algorithm only looks at the nearest upcoming nodes to decide what to do. When the task is completed, the TT variable is updated and the performance is stored in the current end-node of the tree and in the GOT (by averaging the resulting performance with the ones stored, for all the options used for that trial). The GOT criterion is preferred to using random behavior selection because it favors using knowledge derived from past experiences, instead of blindly exploring all of the options.

Since this algorithm is used in noisy and non-stationary conditions, deleting paths from the tree is necessary to keep the interaction model that the tree represents up to date. Node deletion also serves to regulate memory use. To enable the robot to respond to recent changes in the environment, we opted for deleting the oldest path in the tree when more than 25 different paths were stored. Other criteria can be easily substituted for different applications.

3 Experimental setup and task description

Our experiments were performed on a Real World Interface Pioneer I mobile robot (shown on the right of Figure 2) equipped with seven sonars and a Fast Track Vision System. The robot is programmed using MARS, a language for programming multiple concurrent processes and behaviors [2]. The experiments were conducted in an enclosed 12'×11' rectangular pen containing pink blocks and a home region, marked with a green cylinder. These colored objects were perceivable using the robot's vision system, and suffered from variations in light intensity at different places in the pen and from different angles of approach. Other obstacles and walls were detected using the sonar system; the dark stripe on the wall of the pen is a strip of corrugated cardboard used to improve sonar reflection. IS Robotics R1 robots, equipped with infra-red and contact sensors, were also used in the experiments; they were perceived by the Pioneer's sonars.

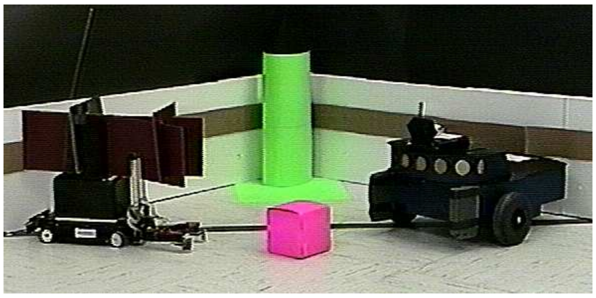

Figure 2: A Pioneer I robot (on the right) near a block, the home region and a R1 robot.

The overall organization of the control system is shown in Figure 3. The behaviors give velocity and rotation actions based on sonar readings and visual inputs. The robot has to accomplish two tasks: search for a block ("Searching Task"), and bring it to the home region ("Homing Task"). Separate learning trees were used for each task, and knowing which one to use was determined by the activated task-behavior. Behaviors that are not activated cannot participate in the control of the robot. There is one specific task-behavior for the Searching Task, *Searching-block*, and two for the Homing Task, *Homing* and *Drop-block*. A task-behavior called *Velocity-control* is used in both of these tasks to make the robot move. *Avoidance* is a maintenance-behavior. Conditions for activating these behaviors were pre-programmed and based on the presence or absence of a block in front of the robot and the proximity of the robot to home. Alternative-behaviors were *Follow-side*, *Rest* and *Turn-randomly* and when selected, they remain activated for pre-set periods of time (5 to 10 seconds). The actual duration of behavior use depends on discrete sensory and temporal events encoded in rules, and on commands issued by other behaviors subsuming it. Note that the organization follows the Subsumption Architecture [1] with the difference that the behaviors that are allowed to issue outputs (i.e., activated behaviors) change dynamically. Whenever a behavior is executed, the symbol associated with it (e.g., F for Follow-side) is added (by still following the subsuming organization of behaviors) to the symbol sequence representing the behavior use at each processing cycle. This is then used to construct the interaction model, i.e., the tree.

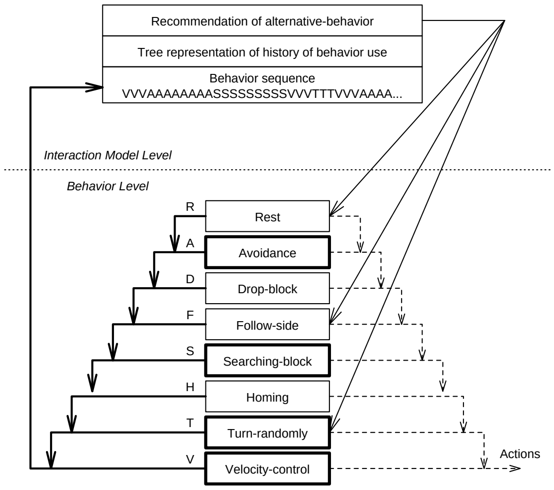

Figure 3: Behaviors and the module responsible for learning from observing behavior use and for selecting alternative-behaviors. Activated behaviors for the Searching Task, with *Turn-randomly* as a chosen alternative-behavior, are depicted in bold.

4 Experiments

Experiments with multiple robot systems are very difficult to analyze. To systematically address this problem, we first used static environment conditions with increasing complexity to verify that the algorithm could learn stationary conditions from its interactions with the environment. Next, we performed experiments with multiple robots. The same behavior repertoire was used in both sets of experiments, without optimization or retraining of the vision system parameters for color detection. The objective was to create situations where the designer could not know or adjust the behaviors according to *a priori* knowledge of the environment. All the computations were done on-board with a limited amount of memory (the entire robot program used 66 Kbytes of memory, with 27 Kbytes used by the learning algorithm, leaving 43 Kbytes for storing the trees; a tree of 200 nodes takes approximately 7 Kbytes of memory).

For the multi-robot experiments, the block to be brought home was put in the center of the pen, as shown in Figure 4. Using this configuration, 15 tests of 60 to 80 trials were run, lasting between 1.5 and 2 hours and using two or three R1s. In some of these tests, two learning Pioneer robots were used, allowing us to compare their learned strategies.

During these experiments, the non-stationary conditions arose from the presence and movement of the R1 robots, and varied greatly during and between tests. The R1s were programmed to move about, avoid all obstacles, and stay out of the dark regions of the floor (shown in Figure 4). In some cases, one or two R1s stayed close to the home region, while in others all moved around the region near the block. As the learning is primarily dependent on the sequence of past experiences, which change in each test run, optimality cannot be determined from the task alone (unless a very large amount of data is available for statistical analysis). To counter this, we used the run-time data and studied the experiment video tapes in order to analyze what, why, and how strategies were learned, if regularities were found from the interactions dynamics, and how the algorithm adapted to changes detected from these interactions.

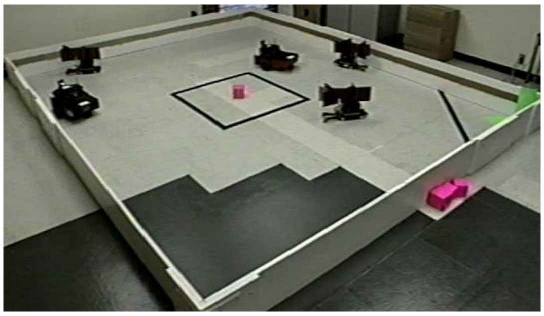

Figure 4: Environmental configuration for the multi-robot experiments, with two Pioneers and three R1s. The R1s are programmed to avoid obstacles and the areas marked with dark tiles or tape.

For conciseness, we describe only the results obtained for the Searching Task because they best illustrate the properties of the learning algorithm. In this task, the learned strategies largely involved the use of the *Observe*, *Rest* and *Turn-randomly* options. The sequences of options learned were: *Observe* – *Rest*; *Observe* – *Rest* – *Observe*; *Rest*; *Rest* – *Turn-randomly*; *Rest* – *Observe*; and *Turn-randomly*. The strategies changed over time according to past experience: when performance for one strategy decreased, another was used based on what was learned previously. One or two stable strategies "won out" in most experiments.

Figure 5 shows the trace of the choices made for the initial 16 trials and the last 11 trials of an experiment. Data associated with a choice are delimited by parentheses and consist of the trial number, the name of the chosen option, the name of the selection criterion

```
1:(O TM 0 0)
2:(S TM 98 17)
3:(F UO 0 40)
4:(R UO 0 60) (O TM 98 0)
5:(TR UO 0 58) (O TM 96 0)
6:(F TM 75 57)
7:(F GOT 0 62) (S TM 96 91) (F UO 89 91)
8:(F TM 90 63)
9:(F TM 75 65)
10:(F TM 95 66) (TR UO 70 76) (O TM 42 0)
11:(F GOT 0 61) (F UO 87 91)
12:(F TM 96 62) (O TM 64 0)
13:(F TM 96 57) (R UO 45 64) (O TM 43 0)
14:(F TM 92 53) (F UO 45 82) (O TM 43 0)
15:(S GOT 0 50) (S TM 98 17)
16:(TR TM 96 49)
...
58:(S GOT 0 49) (R TM 92 37) (F TM 96 32) (O TM 53 0)
59:(TR TM 92 49)
60:(S GOT 0 50)
61:(TR TM 94 51) (S TM 77 55) (F UO 54 55)
62:(S TM 98 52) (S GOT 51 52) (R TM 58 37) (S TM 57 29)
   (R UO 26 32) (O TM 8 0)
63:(S TM 97 50) (R TM 89 35) (F TM 88 27) (O TM 13 0)
   (O TM 13 0)
64:(S TM 87 50) (64 S GOT 37 50)
65:(S GOT 0 50) (R TM 88 34) (E TM 96 27) (O TM 92 0)
   (O TM 0 0)
66:(S TM 96 50) (S GOT 48 50) (R TM 56 32)
67:(S GOT 0 51) (R TM 99 34) (S TM 97 25) (S TM 83 27)
   (R TM 57 31) (S TM 65 35) (F UO 34 35) (O TM 3 0)
68:(S TM 98 51) (R TM 75 33) (TR TM 74 23) (E TM 66 3)
   (R UO 2 3) (O TM 0 0)
```

Figure 5: Trace of choices made for the Searching Task tree, multi-robot experiment.

used, $eval(t)$ and $E(eval)$ at the current position in the tree. In the shown example, *Follow-side* was the first favored option in the initial trials. However, this strategy did not remain stable for very long: from trials 28 to the end of the experiment, the $Observe - Rest$ strategy was preferred (e.g., in trials 58, 62, 63, 65 to 68). Between these trials, different options were explored. Shorter paths are a sign that the robot was able to find the block without experiencing much interference from the R1s. Long sequences of choices are generated when longer paths are reused, characterizing what to do when greater interactions are experienced when accomplishing the task. These choices were made using the TM and GOT selection criteria. Finally, Figure 6 shows the decreasing number of nodes used over consecutive trials, demonstrating reuse of stored paths.

5 Discussion

The different components of the learning algorithm try to establish a compromise between *exploration* (learning to adapt to noise and changes in the environment) and *exploitation* of a stable behavior selection

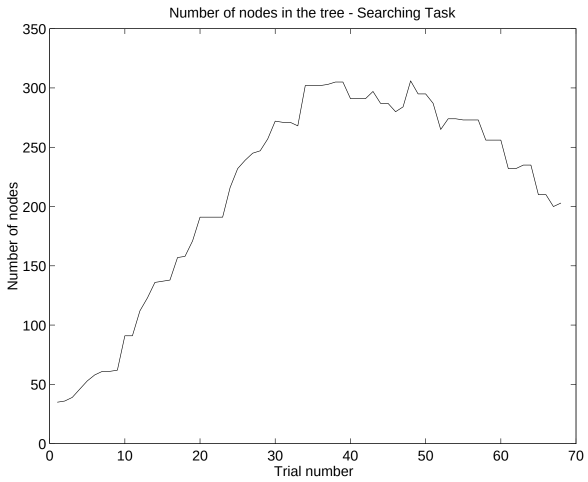

Figure 6: Graph of the number of nodes for a Searching Task tree from a multi-robot experiment.

strategy. The evaluation function characterizes the current situation and the past experiences. The tree representation captures sequences of behavior use in a compact fashion to make a decision based on past experiences. The selection criteria exploit both of these components to determine when and what choice to make. Finally, path deletion influences the choices made by affecting $E(eval)$ in the tree. The deletion factor defines the storage capability of the algorithm, which can also be viewed as a type of principled bias set up by the designer.

In effect, these influences allow the algorithm to establish *critical points* for behavior selection, i.e., decision factors for activating behaviors, based on interactions with the environment. These critical points are self-determined and modified in response to continuing changes in the dynamics, which could be caused by alterations of the environment, the behavior of other robots, or the variability of the robot's abilities (e.g., due to noise, battery power, etc.). Even in a non-stationary environment, these critical points provide the robot with some ability to generate expectations based on previous experiences, and thus evaluate its current performance.

Our experiments revealed some additional interesting properties of our approach. First, an initial strategy can serve as a bias toward the elaboration of a specialized controller, but does not prevent it from changing dynamically. Second, we also experimented with two robots learning simultaneously in the same test environment. We found that they specialized toward different strategies based on their specific, individual experiences. Having two Pioneers or more R1s in the pen did not influence the ability of the learning robot to find stable strategies; the dynamics of the interactions had a more significant effect.

6 Related work

Our algorithm learns at the behavior selection level (i.e., decides which behavior's output should be executed [3]). Past approaches to learning behavior selection [4, 5, 6] used sensory inputs as the selection criterion. In contrast, our approach uses the stored history of past behavior use. Our history-based approach was inspired by [8], whose algorithm partitions the state space from raw sensory experiences and learns a variable-width window that serves simultaneously as a model of the environment and a finite-memory policy [3]. In our case, the algorithm learns a finite-memory strategy of behaviors use, since our tree representation uses behaviors as an abstraction. The development of our algorithm was derived from a general control architecture for intelligent agent [9], which is based on dynamic selection of behaviors.

7 Summary and conclusions

The goal of our approach is to enable a robot to learn and utilize the interaction dynamics with its environment. The learning algorithm uses history of behavior use to derive an incrementally-updated tree representation as a model of these interactions. Learning the model and using it to derive behavior selection strategies are done relative to the exploitation of the controlling resources (i.e., the behaviors) of the robot. This decreases the burden of behavior design by not having to fine-tune the behaviors or acquire knowledge about the specific and complex dynamics of the system. The algorithm is focused on continuous life-time adaptation rather than on learning a one-time, static controller. The approach is computationally efficient and the demonstrated learning is performed on-board the robot in real-time.

Results show the effectiveness of the approach in taking advantage of any regularities experienced in the world, leading to fast and adaptable robot specialization. Since interaction with the environment is affected by the dynamics and the robot's own limitations in perception, action and processing, our approach advocates internal evaluation of the interaction dynamics, and taking advantage of them while they are still valid. By doing so, the approach leads to learning unanticipated behavior strategies from the perspective of the robot instead of the designer, thus increasing robot autonomy.

Acknowledgments

Support for F. Michaud's Postdoctoral Fellowship was provided by the NSERC of Canada. This research is funded by ONR Grant N00014-95-1-0759 and NSF Grant CDA-9512448 to M. J. Matarić. We thank P. Melville and the Interaction Lab for their support.

References

[1] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[2] R. A. Brooks. MARS: Multiple Agency Reactivity System. Tech. Report, IS Robotics, 1996.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[4] P. Maes and R. A. Brooks. Learning to coordinate behaviors. In *Proc. National Conf. on Artificial Intelligence (AAAI)*, Vol. 2, pp. 796–802, 1990.

[5] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[6] M. J. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73-83, 1997.

[7] M. J. Matarić. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2-3), 1997.

[8] A. K. McCallum. Learning to use selective attention and short-term memory in sequential tasks. In *Proc. 4th Int'l Conf. on Simulation of Adaptive Behavior*, pp. 315–324, Cape Cod, 1996.

[9] F. Michaud, G. Lachiver, and C. T. Le Dinh. A new control architecture combining reactivity, deliberation and motivation for situated autonomous agent. In *Proc. 4th Int'l Conf. on Simulation of Adaptive Behavior*, pp. 245–254, Cape Cod, 1996.