Using Stochastic Grammars to Learn Robotic Tasks*

Pedro U. Lima[1] and George N. Saridis[2]

[1] Instituto de Sistemas e Robótica/Instituto Superior Técnico
1096 Lisboa Codex, PORTUGAL
[2] ECSE Dept., Rensselaer Polytechnic Institute
Troy, NY 12180-3590, USA

Abstract. The paper introduces a reinforcement learning-based methodology for performance improvement of Intelligent Controllers. The translation interfaces of a 3-level *Hierarchical Goal-Directed Intelligent Machine* (HGDIM) are modeled by a 2-stage *Hierarchical Learning Stochastic Automaton* (HLSA). The decision probabilities at the two stages are recursively updated from the *success* and *failure* signals received by the bottom stage whenever a primitive action of the HGDIM is applied to the environment where the machine operates.
The top translation stage and the use of regular stochastic grammars to accomplish the translation of commands into tasks are described here. Under this framework, subsets of conflicting grammar productions represent different task strategies to accomplish a command. At this stage, an LSA is associated to each subset of conflicting grammar productions. Results of simulations show the application of the methodology to an Intelligent Robotic System.

1 Introduction

Intelligent Control (IC) techniques [1] particularly qualify for applications to Robotics, due to the need to coordinate a diverse and large number of sensors and actuators. They differ from "conventional" control techniques by aiming to attain higher degrees of *autonomy*, thus dealing with higher *uncertainty*. Among IC techniques, we are especially interested on learning-based methodologies, due to the repetitive nature of some robotic tasks or at least of some of the primitive operations which compose them. Reinforcement Learning is particularly interesting as it involves the exchange of small bandwidth information (failure and success signals only) between robotic subsystems. In typical applications, such as unmanned space and underwater missions, the cost of large bandwith for communications between the central command (earth controller or main vessel

* The first author was supported by the portuguese National Board for Scientific and Technological Research (JNICT), under Grant #BD/1357/91-IA. The second author was supported by the NASA Center for Intelligent Robotic Systems for Space Exploration (CIRSSE) under Grant #NAGW-1333.

controller) is prohibitive and can be reduced by increasing the autonomy of the machine involved in the mission.

This paper summarizes work done towards establishing an analytic design methodology for Intelligent Controllers whose feedback is essentially based on reinforcement learning. An analytic design based on measures of performance recursively improved through feedback assures some degree of certainty about the measurability, repeatability and verifiability of that design. To the best of our knowledge, such a methodology does not currently exist. Here, we focus on learning the optimal tasks of an Intelligent Controller, although this is just a piece of the hierarchical learning methodology we proposed in previous work [2, 3].

Fu [4] was probably the first author to write about Learning Control Systems and to coin as *Intelligent Control Systems* those systems of interdisciplinary nature, in the intersection of Artificial Intelligence and Automatic Control. He also introduced the concepts of *Stochastic Automata* and *Stochastic Grammars.* The former has been developed, although in different directions, by Narendra and his associates [5]. In the last few years, Sutton and his associates explored reinforcement learning solutions which associate these two views of stochastic automata [6].

The paper is organized as follows: in Sect. 2, we summarize basic concepts related to the Theory of Stochastic Grammars. In Sect. 3 we give some insight into the more general problem we are addressing (selection of the optimal tasks and primitive actions from a robotic language hierarchy), focusing later on the particular problem of learning the optimal task selection. Section 4 describes the application of the theory to an intelligent robotic system. Finally, Sect. 5 summarizes the paper and provides suggestions of future research.

2 Stochastic Grammars

Grammars are usually employed to describe the syntax of languages or structural relations defining a pattern. They are useful in the context of Hierarchical Intelligent Controllers to describe the constraints imposed to the ordering of subtasks composing a task. In particular, *Stochastic Grammars* (see a survey by Fu and Booth [7]) allow the assignment of probabilities to conflicting productions or rewrite rules. This turns out to be equivalent to the assignment of probabilities to the different strings of the generated *language*. The probabilities of the productions in each conflicting set can be learned by a *Learning Stochastic Automaton.* Hence, stochastic grammars combined with *Reinforcement Learning* provide the means to learn the ordering of subtasks composing a task.

Let us first define formally a stochastic grammar:

Definition 1. A *stochastic grammar* is defined by the quintuple $G = (V_T, V_N, \mathcal{R}, \mathcal{P}, S)$, where

1. V_T is a finite set of *terminal symbols*;
2. V_N is a finite set of *nonterminal symbols*;

3. $\mathcal{R}$ is a finite set of productions or rewrite rules;
4. $\mathcal{P}$ is a finite set of probabilities that are assigned by a one to one mapping to the elements of $\mathcal{R}$;
5. S is the start symbol.

Only *stochastic regular grammars* will be considered here, that is stochastic grammars whose productions have the general syntax

$$A \rightarrow \alpha \quad \text{or} \quad A \rightarrow \alpha B, \; \alpha \in V_T^*, \; A, B \in V_N$$

where the symbol to the left of the arrow is called *premise* while the term to the right of the arrow is the *consequent.* V_T^* denotes the set of all the possible strings composed by elements of V_T, including the null string.

The set of productions $\mathcal{R}$ can be partitioned into m disjoint subsets $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_m\}$, where m is the number of nonterminal symbols, $m = |V_N|$. In particular, $\mathcal{R}_i$ is the subset of productions with the same premise A_i, corresponding to the ith nonterminal symbol.

Correspondingly, the set of probabilities $\mathcal{P}$ can be partitioned into m disjoint subsets, where subset $\mathcal{P}_i$ contains the probabilities of the productions of $\mathcal{R}_i$.

A stochastic grammar is *proper* if

$$\sum_{k=1}^{m_i} p_{ik} = 1, \quad p_{ik} \in \mathcal{P}_i, \quad |\mathcal{P}_i| = m_i, \; i = 1, \ldots, m$$

i. e. if for each subset of productions $\mathcal{R}_i$ the production probabilities in $\mathcal{P}_i$ add to one.

To each string x of the language $L(G)$ generated by G corresponds a word function $f(x)$. If the grammar is unambiguous, that is, if there is only one leftmost derivation for each x,

$$f(x) = \prod_{k=1}^{K(x)} p(k, x), \; \forall x \in L(G)$$

where $K(x)$ represents the number of steps in the derivation of x, and $p(k, x)$ is the probability of the production used in the *kth* step of the derivation of x.

A language $L \subset V_T^*$, where V_T^* represents all strings of finite length composed by elements of V_T, including the null string ϵ, is called a *stochastic language* if there is a function $0 \leq f(x) \leq 1$, $\forall x \in L$, called *probabilistic word function,* $f(x): \; L \rightarrow \mathbb{R} <$, such that $\sum_{x \in L} f(x) = 1$. This is the same as to say that the probabilities of each of the strings of L add up to one.

Not all stochastic grammars generate stochastic languages. Some restrictions must be imposed to the stochastic grammar.

A stochastic grammar G is a *consistent grammar* iff the word function defined over $L(G)$ is a probabilistic word function, that is, iff $L(G)$ is a stochastic language.

If the grammar is proper, then it will be consistent.

3 Hierarchical Learning of Optimal Robotic Tasks

3.1 General Concepts of the Model Proposed

A hierarchical architecture for autonomous intelligent control systems which encompasses most of the concepts described in related work was proposed in the 70s by Saridis (see a recent book by Saridis and Valavanis [8]). This architecture is based on a 3-level hierarchy, where more abstract actions are taken at the top **Organization Level** and more precise actions are taken at the bottom **Execution Level**. Given an *external command* (goal), the Organization Level is responsible for sequencing the pre-defined *primitive tasks* into a *task*. The Execution Level executes a detailed translation of the task, generated by the intermediate **Coordination Level**. This level successively decomposes the *sub-tasks* composing the task into *primitive tasks*, and distributes them by a number of coordinators specialized in specific sub-tasks, such as vision or motion planning for an Intelligent Robotic System. The coordinators invoke the Execution Level *primitive actions* to precisely execute the task.

At the interfaces between IM levels, specific *translations* must be made: the **Organization-to-Coordination Translation Interface** has to choose among different *tasks* capable of attaining a goal. The goal is expressed by a *command* sent to the machine. Any task is composed of *primitive tasks*, usually corresponding to the primitives of a given language. For each primitive task, the **Coordination-to-Execution Translation Interface** has to determine the best *primitive action* from the set of those capable of reaching the elementary subgoal associated to the primitive task.

To compare the different alternatives at each translation interface, we need a *cost function* [3]. In robotic systems, the uncertainty involved in the design of the algorithms implementing *primitive tasks* (move, locate, plan-path) is mostly due to approximate or incomplete modeling and statistical fluctuations around nominal parameters. This uncertainty can be quantified by estimating the reliability of the algorithm, roughly defined as the frequency of successes when attempting to meet its specifications. However, generally the computational cost increases with reliability, in terms of the time it takes to complete, the amount of memory it uses, or the number of resources (e.g. processors) required. Thus, it makes sense to think of a selection technique which includes both reliability and computational cost, but first the two measures must be coherently defined in a context where information is noisy and incomplete [3].

We presented elsewhere [3] equations wich propagate bottom-up the cost function originally defined for *primitive actions*. Hence, the cost function of *primitive actions*, *primitive tasks* and *tasks* is expressed by the balance between cost and complement of reliability of a *task*. *Task* cost and reliability are a direct result of the costs and reliabilities of the *primitive tasks* which compose the task. *Primitive task* cost and reliability are propagated from the same measures for the corresponding *primitive actions*.

The success and failure signals used to estimate the reliability of the algorithms at all levels of the Intelligent Controller are implicitly propagated bottom-up through the hierarchy by the described scheme. The **Execution Level** receives a reinforcement signal as the result of applying a *primitive action*. This occurs each time the *primitive action* is applied by checking if the corresponding specifications were met. The success or failure signal is used to recursively estimate the reliability of the *primitive action*, which is propagated to the reliability of a *task* as explained before.

3.2 Learning Task Selection

When the design is based on the hierarchical model just described, we must define the set of available *primitive tasks* and corresponding *primitive actions* first, and then define the alternative *tasks* for each *command* as strings of a language whose terminal symbols are the *primitive tasks*. If we view the alternative *tasks* for a command as different decompositions of that command into *primitive tasks*, a stochastic grammar is a well suited model of the job performed by the **Organization-to-Coordination Translation Interface**. Each subset of productions of the grammar represents a set of alternatives in the decomposition process.

This use of a stochatic grammar resembles an expert system where the production rules have associated certainty factors. However, the association of a LSA to each subset of productions and the introduction of a performance measure provides a consistent and quantifiable method of learning the best decisions.

We have proposed before a methodology for performance improvement of *Hierarchical Intelligent Controllers* (HICs) based on *Hierarchical Reinforcement Learning*. The two translation interfaces of Saridis' 3-level hierarchy are modeled by a 2-stage *Hierarchical Learning Stochastic Automaton* (HLSA), as sketched in Fig. 1. The HLSA includes a *Hierarchical Reinforcement Learning Scheme* which recursively updates the decision probabilities at the two stages from *success* and *failure* signals received by the bottom level whenever an action of the HIC is applied to the environment. Under this learning scheme, the probability of selecting the best tasks and primitive algorithms converges to 1 with probability one (w.p.1) [2].

The *task* selection process is improved recursively as the cost function estimates improve from the propagation of the cost and reliability learned at the **Coordination-to-Execution Interface**. However, we fill focus here on the top **Organization-to-Coordination Interface**, that is on learning the optimal *task* selection. Nevertheless, it is important to note that this differs from other Hierarchical Reinforcement Learning methods described in the literature (e.g. Lin [9]) aiming to teach some complex task to a robot. Here a primitive action fails or succeeds due to a failure or success to meet its specifications, and the overall task is evaluated from the performance of its composing primitives, while in those works the overall task is directly reinforced from the success or failure of its application, evaluated by an external agent. This evaluation is assumed to be either performed either by a human or by a sensing system which may be itself very unreliable or too complex.

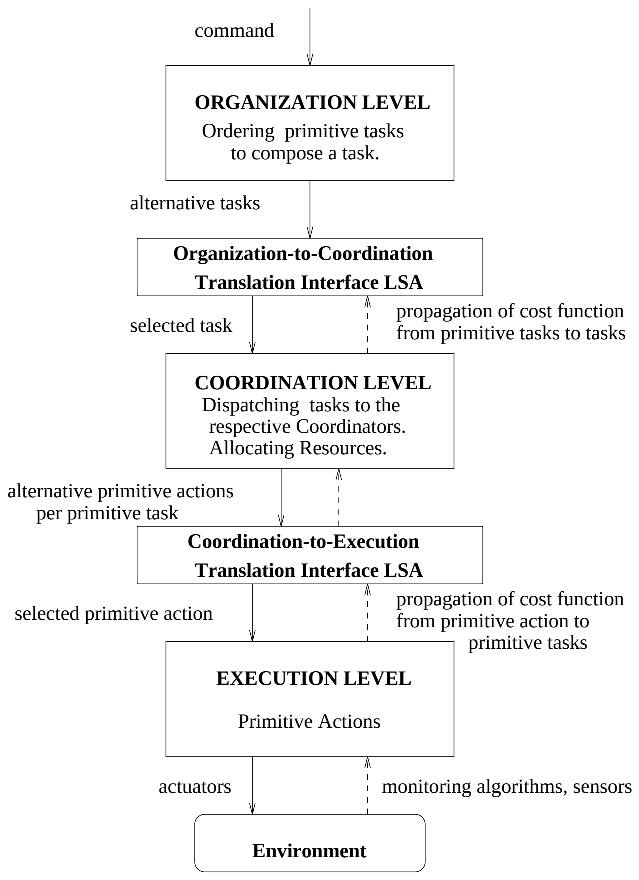
command
ORGANIZATION LEVEL
Ordering primitive tasks to compose a task.
alternative tasks
Organization-to-Coordination Translation Interface LSA
selected task
propagation of cost function from primitive tasks to tasks
COORDINATION LEVEL
Dispatching tasks to the respective Coordinators.
Allocating Resources.
alternative primitive actions per primitive task
Coordination-to-Execution Translation Interface LSA
selected primitive action
propagation of cost function from primitive action to primitive tasks
EXECUTION LEVEL
Primitive Actions
actuators
monitoring algorithms, sensors
Environment

Fig. 1. HLSA and HIC

The solution presented is valid under the general assumptions that the HIC moves inside an environment which can be modeled as a multi-state stochastic process, the HIC can recognize the different states of the model of the environment, and there is a mechanism of error detection and error recovery to detect and recover from situations which could compromise the integrity of the HIC and/or of the environment.

Each *command* interpretable by the HIC is the starting symbol of a regular stochastic grammar. The grammar expresses the constraints imposed by the *command* to the set of *primitive tasks*, the terminal symbols of the grammar. Each string of the language generated by the grammar represents a *task*. Hence, a *task* is generated by the successive application of grammar *productions*.

As we are dealing with regular grammars, the *premise* of each production of the grammar is a non-terminal symbol of the grammar. Each subset of grammar productions with the same premise (refer to Sect. 2) represents one (or more) alternatives on the derivation of the translating *task*. There is a LSA associated to each of these subsets. The productions in each subset are the actions of the LSA, and their probabilities are learned according to the LSA reinforcement scheme explained below. The set of all these LSAs represents the top stage of the hierarchy. The optimal *task* is indirectly learned along time, as a result of learning the optimal production of each production subset. The *task* is optimal in the sense of minimizing the cost function which balances algorithm reliability and cost.

At every step, a (sub)optimal *task* is selected by random decision, based on the current subsets of production probabilities. Each *task* is a string of *primitive tasks*. The *primitive tasks* composing *task* t will also be in the *consequent* of some production involved in the generation of t.

The **cost function**, valid at all levels of a Hierarchical Intelligent Controller, is defined by:

$$J = 1 - R + \rho C \tag{1}$$

where R is the reliability, C the cost and ρ a weighting factor such that $\rho C \in [0, 1]$. In general ρ will be such that the cost does not overwhelm the reliability when directing the search for the optimal action.

Equation (1) applies to the three levels of the HIC, i.e., the performance of a *primitive action*, *primitive task* or *production* can be evaluated by (1), if the cost and reliability are appropriately propagated bottom-up through the hierarchy. The propagation equations can be found in Lima and Saridis [3].

To update the probability density function over the set of *tasks* for a *command*, Fu's stochastic approximation reinforcement learning scheme used in this work is based on the generalized LSA proposed by Nikoliç and Fu [10].:

$$p_{ij}(n_i + 1) = p_{ij}(n_i) + \gamma(n_i + 1)(\lambda_{ij}(n_i) - p_{ij}(n_i + 1)) \tag{2}$$

where $n_i = \sum_j n_{ij}$, $0 \leq \lambda_{ij}(n_i) \leq 1$, $\sum_j \lambda_{ij}(n_i) = 1$, $i = 1, \ldots, d$ denotes environment states, $j = 1, \ldots, r$ denotes *tasks*. As the environment is assumed to be multi-state stochastic, we must create a LSA per environment state.

Given the estimates of the performance function for the top stage, $\lambda_{ij}(n_i)$ is given at each step by

$$\lambda_{ij}(n_i) = \begin{cases} 1 \text{ if } \hat{J}_{ij}(n_{ij}) = \min_k \hat{J}_{ik}(n_{ik}) \\ 0 \text{ if } \hat{J}_{ij}(n_{ij}) \neq \min_k \hat{J}_{ik}(n_{ik}) \end{cases} \tag{3}$$

We use $\gamma(n_i + 1) = \frac{1}{n_i+1}$, which satisfies the condition of Theorem 2 in [10]. Given this, it has been proven that the probability density function of the action probabilities for state i converges w.p.1 to zeros for all the actions (*tasks*) except the optimal action m at each stage. The optimal action is the one which minimizes J. If m is the index of the optimal action, $J_{im} = \min_{k=1,\ldots,r}\{J_{ik}\}$, where J_{ij} is the cost function at the decision stage under consideration. Hence, the theorem implies that $\Pr\{\lim_{n_i \to \infty} p_{im}(n_i) = 1\} = 1$, for state i, $i = 1, \ldots, d$ and action j, $j = 1, \ldots, r$.

4 Application to an Intelligent Robotic System

The coordination of vision and motion algorithms is one of the typical problems in Intelligent Robotic Systems. This is the subject of the case study described in this section.

A manipulator PUMA 560 has to grasp a cylindrical strut whose 3D pose (position + orientation) is roughly known. There is a pair of cameras in the ceiling, overviewing the working space of the manipulator and used by a stereo vision system to determine more accurately the 3D pose of the object. The manipulator has position, velocity and force sensors. The scene is well illuminated but from time to time lights go off, deteriorating the accuracy of vision algorithms. The environment has 2 states, one corresponding to lights on (state 0) and the other to lights off (state 1).

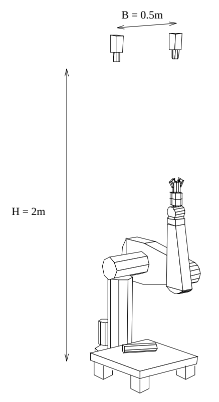

Fig. 2. Workspace setup for case study 2

A realistic simulation of this setup has been performed, including PUMA dynamics, realistic vision system parameters and dimensions, and compliance. Detailed description of the *primitive actions* used for each *primitive task* can be found in [2].

The only *command* available is c=Grab-Strut. The *primitive task* set is composed by 5 *primitive tasks*, $E = \{e_1, e_2, e_3, e_4, e_5\}$, where $e_1 \overset{\Delta}{=}$move manipulator, $e_2 \overset{\Delta}{=}$grasp object with compliance, $e_3 \overset{\Delta}{=}$locate object, $e_4 \overset{\Delta}{=}$plan trajectory, and $e_5 \overset{\Delta}{=}$grasp object hard.

Primitive task e_1 represents the motion of the manipulator tip along a preplanned trajectory in joint space. Uncertainty is due to unmodeled dynamics and noise.

The goal of *primitive tasks* e_2 and e_5 is to slowly move the tip (tool) of the manipulator to the object and grasp it. The *Position Accommodation Control* method is used by all algorithms translating e_2 to accomplish compliance control of the manipulator. Different parameterizations of the required impedance (similar to a mass, spring and damper system) result in different algorithms with different costs and reliabilities. Motion driven by some desired force moves the tool tip to the object. e_5 is position controlled and has only one translating algorithm. Only passive compliance exists in this case.

Primitive task e_3 determines the pose of an object using stereo vision algorithms. The uncertainty on pose determination by stereo vision is mainly due to matcher errors when determining which image pixels in the two cameras correspond to the same point in the observed scene. This may be due both to spot noise and pixel resolution, and leads to disparity errors which affect the 3D pose estimation.

Changing the state of the environment deeply affects the two algorithms. Switching the lights off increases spot noise which affects the estimation (by any of the algorithms) of the strut end-points in each of the images. This will increase disparity errors and consequently pose estimation errors.

Primitive task e_4 plans a trajectory in joint space whose end-points are the joint-space vectors corresponding to the initial and final pose required for the manipulator tip when moving from a standby position to the grasping position. A *minimum-jerk trajectory generator* is the only algorithm used. It generates the desired joints acceleration, velocity and position at a number of pre-specified points along the trajectory by a method which minimizes the jerk, or third derivative of joint position.

Table 1 shows the costs of the *primitive actions* for the different *primitive tasks*. Cost does not change with the state of the environment. During the simulation, the reliabilities of the *primitive actions* were estimated based on the rewards resulting from the successes of their application over the simulated environment.

Table 1. Primitive tasks, primitive actions and computed costs.

pr. task	e_1		e_2				e_3		e_4	e_5
pr. action	a_1^1	a_2^1	a_1^2	a_2^2	a_3^2	a_4^2	a_1^3	a_2^3	a_1^4	a_1^5
cost	0.55	0.44	0.18	0.14	0.16	0.22	0.35	0.35	0.00	0.3

To grab the strut, the system must first estimate the strut pose. Then, it plans a path from the current pose of the manipulator tip to the neighborhood of the strut pose, and moves along that path. Finally, the manipulator slowly approaches the strut and tries to grasp it. Alternative tasks differ by the inclusion or not of compliance in the final move, and by using or not using stereo vision to refine the *a priori* knowledge of the strut pose. The stochastic regular grammar for the command is:

$$\begin{array}{lcl}
G & = & (V_T,\ V_N,\mathrm{R},\mathrm{P},\ \mathrm{S}\ ) \\
V_N & = & \{\ \mathrm{S},\ \mathrm{A},\ \mathrm{B}\ \} \\
V_T & = & \{e_1, e_2, e_3, e_4, e_5\} \\
\mathrm{R} & = & \{ \\
\multicolumn{3}{l}{0.5\ \mathrm{S} \rightarrow e_3\ \mathrm{A}} \\
\multicolumn{3}{l}{0.5\ \mathrm{S} \rightarrow \mathrm{A}} \\
\multicolumn{3}{l}{1.0\ \mathrm{A} \rightarrow e_4\ e_1\ \mathrm{B}} \\
\multicolumn{3}{l}{0.5\ \mathrm{B} \rightarrow e_2} \\
\multicolumn{3}{l}{0.5\ \mathrm{B} \rightarrow e_5\ \}}
\end{array}$$

The numbers to the left of the productions are the initial production probabilities. Productions such as S $\rightarrow$ A, with no terminal symbols in its right-hand side, are assigned zero cost and 0.5 reliability.

The simulation described in the previous section was run for several different situations in order to point out the main concepts and tradeoffs of the proposed execution and design methodologies, and also to suggest future research avenues.

All results shown are the average over 50 sample functions of some stochastic process, such as the evolution of action probabilities or cost function, except when noted. Each sample function consists of 150 *task* runs (iterations in the figures), except when noted. The production and algorithm probabilities are updated after a task is applied. The initial state of the environment is always state 0 (lights on).

The plots of action probabilities and cost functions for "conflicting" productions 0 and 1 are displayed at the top of Fig. 3. Their evolution is slow. The bottom of the same figure shows the result of using a convergence acceleration method which combines a modified version of the algorithm proposed by Nikoliç and Fu [10], with the use of initial estimates for the reliabilities, based on models or past experience of the designer. initial estimates $R(0)$ are weighted by a confidence factor.

Figure 4 shows the results of testing the adaptiveness of the learning scheme to an unacknowledged change in the state of the environment. Only one sample of the stochastic processes is shown. The state of the environment switches from state 0 (lights on) to state 1 (lights off) at iteration 150, but state 0 remains the estimated state for the HLSA, which does not recognize the change. The simulation consisted of 400 task runs. A recursive sample mean algorithm with *forgetting factor* was used to improve the adaptiveness of cost function and probability estimates at the bottom stage. No proof of convergence exists for this algorithm, which resembles the stochastic approximation algorithm described before, but gives more weight to recent samples than to old samples.

After the state switching, production 0 is no longer the best production, since the performance of the stereo algorithms deteriorates under poor lighting conditions, and the initial rough estimate of the strut pose is preferred. In this run, the HLSA learned the change.

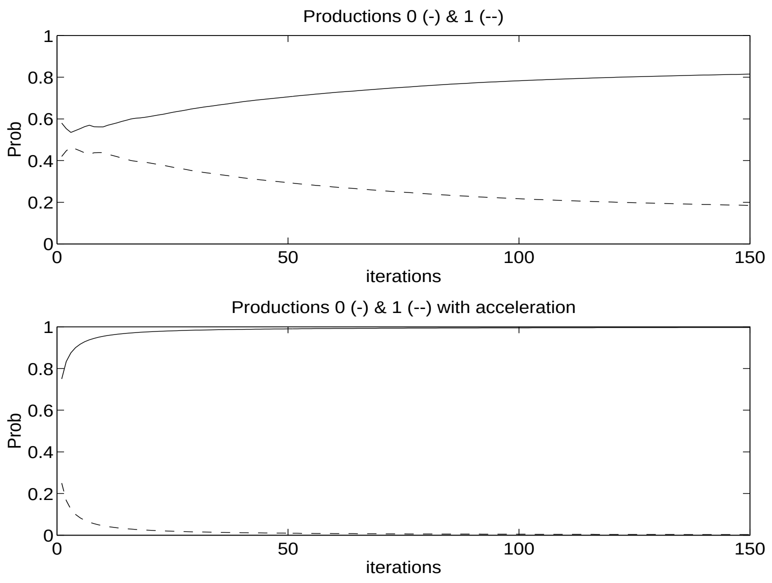

Fig. 3. Learning the best between productions 0 and 1. Top: without acceleration; Bottom: with acceleration

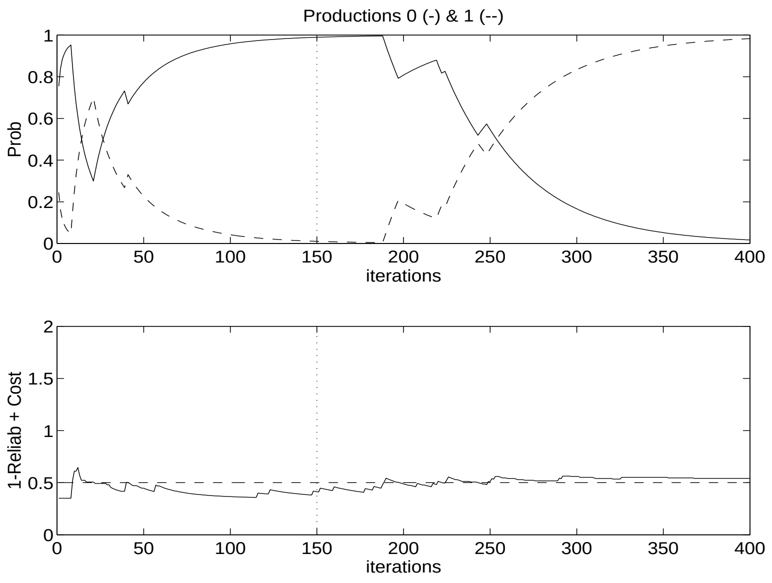

Fig. 4. Adaptiveness of learning scheme when seeking the best between productions 0 and 1

5 Conclusions and Future Research

Measuring its performance and improving that performance using feedback from the environment are desirable features of any Intelligent Controller. We have described part of a model of a Hierarchical Intelligent Controller which uses a Hierarchical Reinforcement Learning Scheme to converge w.p.1 to the actions which minimize a proposed cost function. The cost function balances the complement of reliability and cost at all levels of the Intelligent Controller.

The paper focused on the translation from commands to tasks, performed by a regular stochastic grammar. The probabilities of production subsets of the grammar are learned along time from estimates of the tasks cost function. These are obtained by bottom-up propagation of the cost function of the primitive actions of the Intelligent Controller, estimated from successes and failures of these actions to meet the specifications for the primitive tasks they translate.

Future research should contemplate the extension from regular to more complex grammars (such as context-free) in order to accommodate a broader language for task generation from commands. The behavior of Hierarchical Learning Stochatic Automata (used to model the whole Hierarchical Intelligent Controller) in the presence of non-stationary environments is a subject of current research.

References

1. Technical Committee on Intelligent Control, "Report of task force on Intelligent Control, IEEE Control Systems Society," *IEEE Control Systems Magazine*, vol. 14, June 1994. P. Antsaklis, editor.
2. P. U. Lima and G. N. Saridis, "Hierarchical reinforcement learning and decision making for Intelligent Machines," in *Proceedings of 1994 IEEE Int. Conf. Robotics and Automation*, May 1994.
3. P. U. Lima and G. N. Saridis, "A performance measure for Intelligent Machines based on complexity and reliability," in *Proceedings of SY.RO.CO 94*, September 1994.
4. K. S. Fu, "Learning Control Systems - review and outlook," *IEEE Transactions on Automatic Control*, vol. AC-15, no. 2, 1970.
5. K. S. Narendra and M. A. L. Thathachar, *Learning Automata - an Introduction.* Prentice Hall, 1989.
6. R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning in direct adaptive optimal control," *IEEE Control Systems Magazine*, vol. 12, no. 2, pp. 19–22, 1992.
7. K. S. Fu and T. L. Booth, "Grammatical inference: Introduction and survey – part II," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-5, no. 4, 1975.
8. K. P. Valavanis and G. N. Saridis, *Intelligent Robotic Systems.* Kluwier Publishers, 1992.
9. L.-J. Lin, "Scaling up reinforcement learning for robot control," in *Proceedings of the Tenth International Conference on Machine Learning*, 1994.
10. K. S. Fu and Z. J. Nikoliç, "On some reinforcement techniques and their relation to the stochastic approximation," *IEEE Transactions on Automatic Control*, vol. AC-11, no. 2, pp. 756–758, 1966.