, , 1–36 ()
© Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Module-Based Reinforcement Learning: Experiments with a Real Robot

ZSOLT KALMÁR kalmar@mindmaker.kfkipark.hu
Department of Informatics
"József Attila" University of Szeged
Szeged, Aradi vrt. tere 1, Hungary H-6720

CSABA SZEPESVÁRI szepes@mindmaker.kfkipark.hu
Research Group on Artificial Intelligence
"József Attila" University of Szeged
Szeged, Aradi vrt. tere 1, Hungary H-6720

ANDRÁS LŐRINCZ lorincz@mindmaker.kfkipark.hu
Department of Adaptive Systems
"József Attila" University of Szeged
Szeged, Aradi vrt. tere 1, Hungary H-6720

Present address of all authors:
Associative Computing Ltd.
Budapest 1121, Konkoly Thege M. út 29–33

Editors: Henry Hexmoor and Maja Matarić

Abstract. The behavior of reinforcement learning (RL) algorithms is best understood in completely observable, discrete-time controlled Markov chains with finite state and action spaces. In contrast, robot-learning domains are inherently continuous both in time and space, and moreover are partially observable. Here we suggest a systematic approach to solve such problems in which the available qualitative and quantitative knowledge is used to reduce the complexity of learning task. The steps of the design process are to: *i)* decompose the task into subtasks using the qualitative knowledge at hand; *ii)* design local controllers to solve the subtasks using the available quantitative knowledge and *iii)* learn a coordination of these controllers by means of reinforcement learning. It is argued that the approach enables fast, semi-automatic, but still high quality robot-control as no fine-tuning of the local controllers is needed. The approach was verified on a non-trivial real-life robot task. Several RL algorithms were compared by ANOVA and it was found that the model-based approach worked significantly better than the model-free approach. The learnt switching strategy performed comparably to a handcrafted version. Moreover, the learnt strategy seemed to exploit certain properties of the environment which were not foreseen in advance, thus supporting the view that adaptive algorithms are advantageous to non-adaptive ones in complex environments.

Keywords: reinforcement learning, module-based RL, robot learning, problem decomposition, Markovian Decision Problems, feature space, subgoals, local control, switching control

1. Introduction

Reinforcement learning (RL) is the process of learning the coordination of concurrent behaviors and their timing so as to optimize some performance cost, where the cost is a function of the reinforcement signals communicated to the learner

in each time step. A few years ago Markovian Decision Problems (MDPs) were proposed as the model for the analysis of RL (Werbös, 1977; Sutton, 1984) and since then, a mathematically well-founded theory has been constructed for a large class of RL algorithms. These algorithms are based on modifications of the two basic dynamic-programming algorithms used to solve MDPs, namely the value- and policy-iteration algorithms (Watkins and Dayan, 1992; Jaakkola, Jordan, and Singh, 1994; Littman and Szepesvári, 1996; Tsitsiklis and Van Roy, 1996; Sutton, 1996). The RL algorithms learn via experience, gradually building an estimate of the optimal value function, which is known to encompass all the knowledge needed to behave in an optimal way according to a fixed criterion, usually the expected total discounted-cost criterion. The basic limitations of all of the early theoretical results of these algorithms was that these assumed finite state- and action-spaces, and discrete-time models in which the state was assumed to be available for measurement. In a real-life problem however, the state- and action-spaces are infinite, usually non-discrete, time is continuous and the system's state is not measurable (i.e. with the latter property, the process is only partially observable as opposed to being completely observable) (Kalmár, Szepesvári, and Lőrincz, 1997). Recognizing the serious drawbacks of the simple theoretical case, researchers have begun looking at the more interesting yet theoretically more difficult cases (see e.g. Chrisman, 1992; McCallum, 1993; Singh, Jaakkola, and Jordan, 1995; Tsitsiklis and Van Roy, 1995; Munos, 1997). To date, however, no complete and theoretically sound solution has been found to deal with such involved problems. In fact the above-mentioned learning problem is indeed intractable owing to partial observability. This result follows from a theorem of Littman's (Littman, 1996).

One of the most promising approaches, originally suggested to deal with large, but observable problems, is based on the idea of decomposing the task into smaller subtasks. This very basic idea can be traced back at least to the idea of using abstractions, macro-operators and subgoals (Pólya, 1945) which was studied by Newell and Simon (1972) and Korf (1985a) in planning domains. The attractive property of this decomposition is that if it is done in a hierarchical manner then it can reduce exponential complexity to linear (Korf, 1987). In the framework of planning with MDPs the plan acceleration aspect of using macro-operators has recently received some attention (e.g. Precup, Sutton, and Singh, 1997). On the other hand, the learning of macro-operators, which can be interpreted as learning control modules, has a longer history (Korf, 1985b; Mahadevan and Connell, 1992; Singh, 1992). The third aspect is the learning of the switching of the particular controllers, which has been studied among others by Singh (1992) and more recently by Parr and Russell (1997) in hierarchical models; by Matarić (1997) who used a modified RL algorithm; and by Koza and Rice (1992) and Dorigo and Colombetti (1994) who made use of a genetic algorithms, to mention just a few examples. Inventing subgoals, macro-operators and hierarchies turns to be a more difficult problem. Some results in this direction include those of Thrun and Schwartz (1995) and Tóth, Kovács, and orincz (1995) whose algorithms learn skills useful in multiple tasks, or Wiering and Schmidhuber (1997) who deal with partial observability and follow a divide and conquer approach. Switching controls have also received considerable atten-

tion among traditional control theorists under the name hybrid control (Brockett, 1993; Grossman, Nerode, Ravn, and Rischel, 1993), but they usually focused on more basic properties like existence or stability of solutions (Branicky, Borkar, and Mitter (see 1994) and the references therein), or the existence of optimal controls (Zabczyk, 1973). Recently, the existence of optimal switching strategies was proved by Branicky (1995) for a fairly broad class of systems.

In this article, we propose a systematic approach to solve real-world robotic tasks which builds on the above mentioned works. In order to keep the problems tractable we suggest to incorporate *a priori* knowledge when available and use learning only when it is really needed. Namely, we suggest that high-level, abstract *qualitative* knowledge, which is often available, can and should be used in the planning phase to identify subgoals and macro-operators; *quantitative*, but still rough knowledge can and should be used to design controllers that safely implement the macro-operators under some well defined conditions which should be made measurable from the observation process; and learning should be used to resolve conflicts when the operating condition of more than one controller is met. Since we know in advance that learning will be used to find the appropriate switching function we may bravely incorporate alternative solutions to the same subtask. Another goal to be taken into account in the design phase is that the task to be solved by the learning algorithm should have a finite (small) state & action space, and be a completely observable task.

In the first part of the article (Section 2), in addition to discussing the above design principles in detail, some theoretical tools are put forth which can be used to determine if a given subtask decomposition is proper, i.e., if it results in a solution to the original problem. Also convergence issues of the learning algorithms are touched upon. In the second part (Section 3), the approach is demonstrated via a *real-life* example. We provide a detailed statistical comparison of several RL methods combined with different exploration strategies, such as Adaptive Dynamic Programming (ADP), Adaptive Real-Time Dynamic Programming (ARTDP) and Q-learning, with Boltzmann exploration started from different initial "temperatures". The relationship of our work to that of others is described in more detail in Section 4, and then finally our conclusions and possible directions for further research are given in Section 5.

2. Module-based reinforcement learning

First of all, we will briefly overview Markovian Decision Problems (MDPs), a value-function-approximation-based RL algorithm to learn solutions for MDPs and their associated theory. Next, the concept of recursive features and time-discretization based on these features are elaborated upon. This is then followed by a sensible definition and principles of module-design, together with a brief explanation of why the modular approach can prove successful in practice.

2.1. *Markovian decision problems*

RL is the process by which an agent improves its behavior from observing its own interactions with the environment. One particularly well-studied RL scenario is that of a single agent minimizing the expected discounted total cost in a discrete-time finite-state, finite-action environment, in which the theory of MDPs can be used as the underlying mathematical model. A finite MDP is defined by the 4-tuple $\langle S, A, p, c \rangle$, where S is a finite set of states, A is a finite set of actions, p is a matrix of transition probabilities, and c is the so-called immediate cost function. The ultimate objective of learning is identifying an optimal policy. A policy is some function that tells the agent which set of actions should be chosen under which circumstances. A policy π is optimal under the *expected discounted total-cost criterion* if, with respect to the space of all possible policies, π results in a minimum expected discounted total cost for all states. The optimal policy can be found by identifying the optimal value function, defined recursively by

$$v^*(s) = \min_{a \in U(s)} \left(c(s,a) + \gamma \sum_{s'} p(s,a,s') \, v^*(s') \right)$$

for all states $s \in S$, where $c(s,a)$ is the immediate cost for taking action a from state s, γ is the discount factor, and $p(s,a,s')$ is the probability that state s' is reached from state s when action a is chosen. $U(s)$ is the set of admissible actions in state s. The policy which for each state selects the action that minimizes the right-hand side of the above fixed-point equation constitutes an optimal policy. This yields the result that to identify an optimal policy it is sufficient just to find the optimal value function v^*. The above simultaneous non-linear equations (non-linear because of the presence of the minimization operator), also known as the *Bellman equations* (Bellman, 1957), can be solved by various dynamic-programming methods such as the value- or policy-iteration methods (Ross, 1970).

RL algorithms are generalizations of the DP methods to the case when the transition probabilities and immediate costs are unknown. The class of RL algorithms of interest here can be viewed as variants of the value-iteration method: these algorithms gradually improve an estimate of the optimal value-function via learning from the interactions with the environment. There are two possible ways to learn the optimal value function. One is to estimate the model (i.e., the transition probabilities and immediate costs) while the other is to estimate the optimal action-values directly. The optimal action-value of an action a given a state s is defined as the total expected discounted cost of executing the action from the given state and proceeding in an optimal fashion afterwards:

$$Q^*(s,a) = c(s,a) + \gamma \sum_{s'} p(s,a,s') v^*(s'). \tag{1}$$

The general structure of value-function-approximation based RL algorithms is given in Table 1.

In the RL algorithms, various models are utilized along with an update rule F_t and action-selection rule S_t. In the case of the Adaptive Real-Time Dynamic

Table 1. The structure of value-function-approximation-based RL algorithms. Specific forms for the model update operators F_t and the action selection operator S_t are defined in the examples presented below.

---

1. Let $t = 0$, and initialize the utilized model (M_0) and the Q-function (Q_0)
2. Repeat forever
   (A) Observe the next state s_{t+1} and reinforcement signal c_t.
   (B) Incorporate the new experience (s_t, a_t, s_{t+1}, c_t) into the model and into the estimate of the optimal Q-function:

$$(M_{t+1}, Q_{t+1}) = F_t(M_t, Q_t, (s_t, a_t, s_{t+1}, c_t)),$$

   where F_t is the model update operator.
   (C) Choose the next action to be executed based on (M_{t+1}, Q_{t+1}):

$$a_{t+1} = S_t(M_{t+1}, Q_{t+1}, s_{t+1})$$

   and execute the selected action, where S_t is the action selection operator.
   (D) $t := t + 1$.

Programming (ARTDP) algorithm the model consists (M_t) of the estimates of the transition probabilities and costs, the update-rule F_t being implemented, e.g., through the equations

$$\begin{aligned} p_{t+1}(s_t, a_t, s) &= \left(1 - \frac{1}{n_t(s_t, a_t)}\right) p_t(s_t, a_t, s) + \frac{1}{n_t(s_t, a_t)}\, \delta(s_{t+1}, s), \\ c_{t+1}(s_t, a_t) &= \left(1 - \frac{1}{n_t(s_t, a_t)}\right) c_t(s_t, a_t) + \frac{1}{n_t(s_t, a_t)}\, c_t, \end{aligned}$$

where $\delta(i, j)$ is the Kronecker-function, $n_t(s, a)$ is the number of times the state-action pair (s, a) was visited by the process $\{(s_t, a_t)\}_t$ before time t plus one, and values not shown are left unchanged. Instead of the optimal Q-function, the optimal value function is estimated and stored to spare storage space, and the Q-values are then computed by replacing the true transition probabilities, costs and the optimal value function in Eq. 1 by their estimates. An update of the estimate for the optimal value function is implemented by an asynchronous dynamic-programming algorithm using an inner loop in Step 2 of the algorithm. In each step of this loop, a subset of the states, F_j^t, is selected and the value estimates of the states in F_j^t are updated via

$$v(s) := \min_{a \in U(s)} \left(c_{t+1}(s, a) + \gamma \sum_{s'} p_{t+1}(s, a, s') v(s') \right),$$

v being initialized to v_t at the beginning of the loop and letting $v_{t+1} = v$ at the end of the loop. Algorithms where the value of the actual state is updated are called "real time" (Barto, Bradtke, and Singh, 1995). If, in each step, all the states are updated ($F_j^t = S$), and the inner loop is run until convergence is reached, the resulting algorithm will be called Adaptive Dynamic Programming (ADP). Another popular RL algorithm is Q-learning, which does not employ a model but instead the Q-values are updated directly according to the iteration procedure (Watkins and Dayan, 1992)

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t(s_t, a_t))Q_t(s_t, a_t) + \alpha_t(s_t, a_t)\left(c_t + \gamma \min_a Q_t(s_{t+1}, a)\right),$$

where $\alpha_t(s_t, a_t) \geq 0$,

$$\sum_{t=1}^{\infty} \alpha_t(s, a)\delta(s, s_t)\delta(a, a_t) = \infty$$

and

$$\sum_{t=1}^{\infty} \alpha_t^2(s, a)\delta(s, s_t)\delta(a, a_t) < \infty.$$

For example, one might set $\alpha_t(s, a) = \frac{1}{n_t(s,a)}$ but often in practice $\alpha_t(s, a) = \text{const}$ is employed, which, while yielding increased adaptivity, no longer ensures convergence.

Both algorithms mentioned previously are guaranteed to converge to the optimal value/Q function if each state-action pair is updated infinitely often (Jaakkola, Jordan, and Singh, 1994; Tsitsiklis, 1994). The action-selection procedure S_t should be carefully chosen so that it fits the dynamics of the controlled process in a way that the condition is met. For example, the execution of random actions meets this "sufficient-exploration" condition when the MDP is communicating. However, if on-line performance is important, then more sophisticated exploration is needed, which, in addition to ensuring sufficient exploratory behavior, exploits accumulated knowledge. If the Q-values were already exact, then, according to what was explained above, the optimal action in state s would be $\text{argmin}_a\, Q_t(s, a)$; the choice of such actions corresponding to *pure exploitation*. Unfortunately, pure exploitation applied from the beginning of learning will not work in general (a noteworthy exception being the case of the worst-case cost-criterion (Szepesvári, 1997a)). Typical suggestions to overcome these difficulties include choosing random actions occasionally and exploiting actions at other times, or to select actions that minimize some kind of artificially biased Q-values, where the bias is such that the biased Q-values of less often visited state-action pairs become smaller (for a survey of such methods see, e.g. Kumar (1985)). The most popular of these is randomization when the exploiting action is chosen with a probability smaller than one, while having this probability converge slowly to one with time. Recently, it has been shown that if the probability of selecting non-exploiting actions summed up in time equals infinity (for example, when this probability is proportional to $1/n_t(s)$, where $n_t(s)$ is

the number of times the state s was visited before time t increased by 1), then, on the one hand, sufficient exploration is ensured while on the other, the whole process eventually converges to optimality (Singh, Jaakkola, Littman, and Szepesvári, 1997; Szepesvári, 1997b). The most common form of randomized action selection is called "Boltzmann exploration", where the probability of choosing action a in state s equals

$$\frac{e^{Q_t(s,a)/T(s,t)}}{\sum_{a \in U(s)} e^{Q_t(s,a)/T(s,t)}},$$

where $T(s,t)$ is a "temperature" parameter whose rate of decrease with t should be bounded from below by $o(1/\ln(n_t(s)))$ if one wants to ensure sufficient exploration (Singh, Jaakkola, Littman, and Szepesvári, 1997).

2.2. Recursive features and feature-based time discretization

In the case of a real-life robot-learning task, the dynamics cannot be formulated exactly as a *finite* MDP, nor is the state information available for measurement. This latter restriction is modeled by Partially Observable MDPs (POMDPs) where (in the simplest case) one extends an MDP with an *observation function* h, which maps the set of states S into a set X, called the observation set (which is usually non-countable, just like S). The defining assumption of a POMDP is that the full state s can be observed only through the observation function, i.e. only $h(s)$ is available as input and this information alone is usually insufficient for efficient control since h is usually a non-injection (i.e. h may map different states to the same observations). *Features* which mathematically are just well-designed observation functions, are known to be efficient in dealing with the problem of infinite state spaces. Moreover, when their definitions are extended in a sensible way, they become efficient in dealing with partial observability.

It is well known that in the partial observable case optimal policies can depend on their whole past histories. This leads us to a generalization of features, such that the feature's values can depend on all past observations, i.e. mathematically a feature becomes an infinite sequence of mappings $(f^0, f^1, \ldots f^t, \ldots)$, with $f^t : (X \times A)^t \times X \to F$, where F and X are the feature- and observation-spaces. Since RL is supposed to work on the output of features, and RL requires finite spaces it means that F should be finite. Features that require infinite memory are clearly impossible to implement, so features used in practice should be restricted in such a way that they require a finite "memory". For example, besides stationary features, which take the form $(f^0, f^0, \ldots, f^0, \ldots)$ (i.e. $f^t = f^0$ for all $t \geq 1$) and are called *sensor-based* features, *recursive* features (in control theory these are called *filters* or *state-estimators*) are those that can be implemented using a finite memory.[1] For example, in the case of a one-depth recursive feature the value at the t^{th} step is given by $f_t = R(x_t, a_{t-1}, f_{t-1})$, where $R : X \times A \times F \to F$ defines the recursion and $f_0 = f^0(x_0)$ for some function $f^0 : X \to F$. Features whose values depend on the past observations of a finite window form a special class of recursive filters.[2]

A very simple one-step recursive feature is the switching feature (of Boolean type) whose value depends on two disjoint sets of observation-action pairs, those of "on-pairs" and "off-pairs". The feature's value is one (or `'on'`), if the last observed observation-action pair which was either an on-pair or off-pair is an on-pair, otherwise it is zero (`'off'`). In other words, the feature's value does not change as long as the observation is outside the union of the sets of "on-pairs" and "off-pairs" and the feature's value is reset to the label of these sets when the observation data gets into either of them. In Section 3.2.2 we will give an example of such a feature.

Instead of relying on a single feature, it is usually more convenient to define and employ a set of features, each of which indicates a certain event of interest. Note that a finite set of features can always be replaced by a single feature whose output space is the Cartesian product of the output spaces of the individual features and whose values can be computed componentwise by the individual single features. That is to say, the new feature's values are the 'concatenated' values of the individual features.

Since the feature space is finite, a natural discretization of time can be obtained. The new time counter clicks only when the feature value jumps in the feature space.[3] This makes it useful to think of such features as event-indicators which represent the actuality of certain conditions of interest. This interpretation gives us an idea of how to define features in such a way that the dynamics at the level of the new counter be simplified.

2.3. Modules

So far, we have realized that the new "state space" (the feature space) and "time" can be made discrete. However, the action space may still be infinite. Uniform discretization which lacks a priori knowledge would again be impractical in most of the cases (especially when the action space is unbounded), so we would rather consider an idea motivated by a technique that is often applied in artificial intelligence (AI) to solve large search problems efficiently. The method in question follows a kind of "divide-and-conquer" approach which divides the problem into smaller subproblems that in turn are divided into even smaller subproblems, etc., then at the end routines are provided that deal with the resulting mini-problems. The solution of the entire problem is then obtained by working backwards: the routines that solve mini-problems are combined to get larger routines, then these are combined again to get even larger routines, and this is repeated until the root of the hierarchy is reached. In planning domains the combined routines are called macro-actions. The actual solution of the original problem can be obtained if the macro-action corresponding to the actual state of the search problem is applied (Newell and Simon, 1972; Sacerdoti, 1974). To put it another way, the problem solver defines a set of sub-goals, sub-sub-subgoals, etc. in such a way that if one of the sub-goals is satisfied then the resolution of the main goal will be easier to achieve. It turns out that hierarchical subgoal decomposition can reduce the problem complexity from exponential to linear if the decomposition scheme is optimal

(Korf, 1987). Also in robotic domains sometimes it is convinent to define a subgoals by means of specifying a 'desired beavior' pattern.

In control tasks the same decomposition can usually be done with respect to the main control objective without any difficulty provided that a *qualitatively* correct model of the plant is available. A model is said to be qualitatively correct if it leads to designs which when extended with learning in later stages result in a proper solution of the task, i.e., we do not require that a proper solution of the task could be obtained on the basis of a qualitatively correct model. So qualitatively correct models should be much easier to obtain than ones which are actually used for control. Human are usually good at obtaining a decomposition if they have sufficient knowledge about the controls and sensors. Note that this model should ideally be low level, i.e. it should describe the dynamics of sensors and the plant, but at the same time it should hide details which are unimportant from the point of view of planning. A naive physics description of the problem seems to be suitable if one wanted to automate this step using a planner. Qualitative modelling has a long tradition in artificial intelligence (de Kleer and Seely, 1984; Say and Selahattin, 1996; Brafman and Moshe, 1997).

Nevertheless, regardless of what representation and method is used, we end up with a set of macro-actions and their associated subgoals. In the next step the designer should implement the macro-actions as closed-loop local controllers which achieve the associated subgoal. In order to be successful at this task, in general more detailed, *quantitative* knowledge of the plant is needed (see Figure 1). A quantitatively correct model should be suitable for designing local controllers which work as intended under well defined (and observable) conditions. Note that having a quantitatively correct model still does not mean that we can solve the control task without any further refinements. If there is considerable uncertainty in the quantitative model then these local controllers may also have to be learnt by an adaptive method, such as e.g. an adaptive control technique, iterative learning or even by reinforcement learning. Even when using some kind of adaption or robustification, one cannot expect that the resulting controller will work reliably under all possible conditions that can occur. The other reason to restrict the *operating conditions* of the controllers is that, for example, in the case of serializable subgoals (Korf, 1987) in order the application of the macro-action to make sense the previous subgoal must be met. The operating conditions should be given as measurable quantities. The controllers together with their operating conditions, which may also serve as a basic set of features, will be called *modules*. The process of breaking up the problem into small subtasks should be repeated several times recursively before the actual controllers are designed, so that the complexity of the individual controllers can be kept low.

In complex problems, it may happen that a particular controller proves to be useful in accomplishing several subtasks. For example, in a mobile-robot task such a general-purpose controller could be that which 'rescues' the robot when it becomes stuck. For example, in a mobile-robot task such a general-purpose controller could be that which unsticks the robot when it becomes stuck.

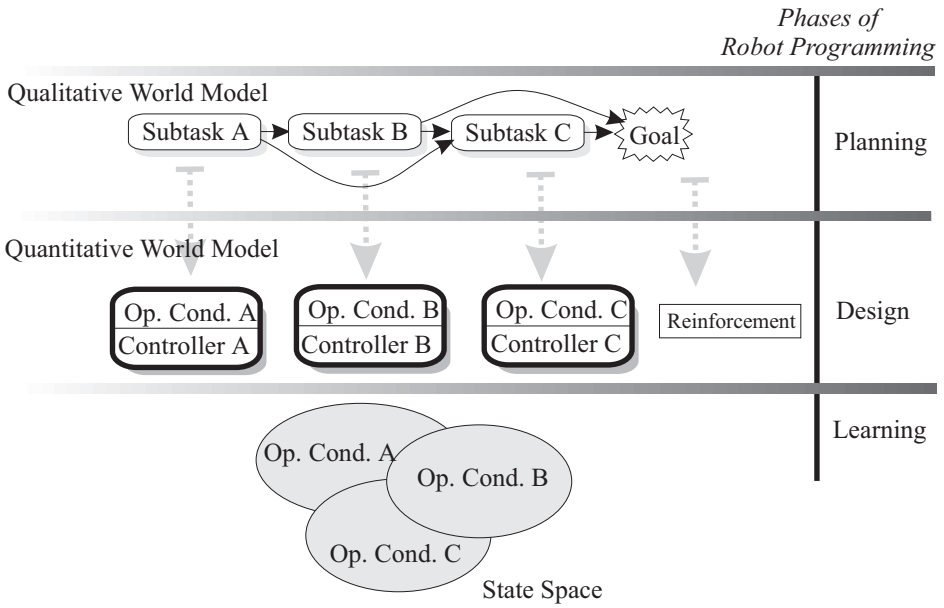

Figure 1. **Illustration of the proposed robot programming method.** The main task is divided into subtasks to reduce complexity of design and learning at lower levels. Each subtask has a controller associated to it and the controllers have operating conditions under which the controller can safely accomplish the given subtask. Controllers and their operating conditions, which are referred together as modules, can be designed by hand on the basis of available quantitative knowledge. Operating conditions of different controllers may overlap in which case more than one controller may be applied at the same time. Reinforcement learning is applied to remove this ambiguity in an optimal way.

In principle, a consistent transfer of the AI decomposition yields the result that the operating conditions of the situation are exclusive and cover every situation. However, such a solution would be very sensitive to perturbations and unmodelled dynamics and is hard to achieve due to the ambiguities in the plant models. A more robust solution can be obtained by considering broader operating conditions or designing alternate modes to solve the problem. This, however, yields that more than one controller can be applied at the same time, under the same conditions. This calls for the introduction of a mechanism, the switching function, that determines which controller has to be activated if there are more than one available. This decision should, however, be made solely on the basis of the state of operating conditions and some some possible additional auxiliary filters. These together compose the feature vector available for the switching mechanism.

So the switching function S maps feature vectors to the index of the module that should be activated when the actual feature vector under consideration is observed. Of course, only those modules can be activated whose operating conditions are satisfied.[4] The operation of the whole mechanism is then the following. A controller remains active until the switching function switches to another controller. Since

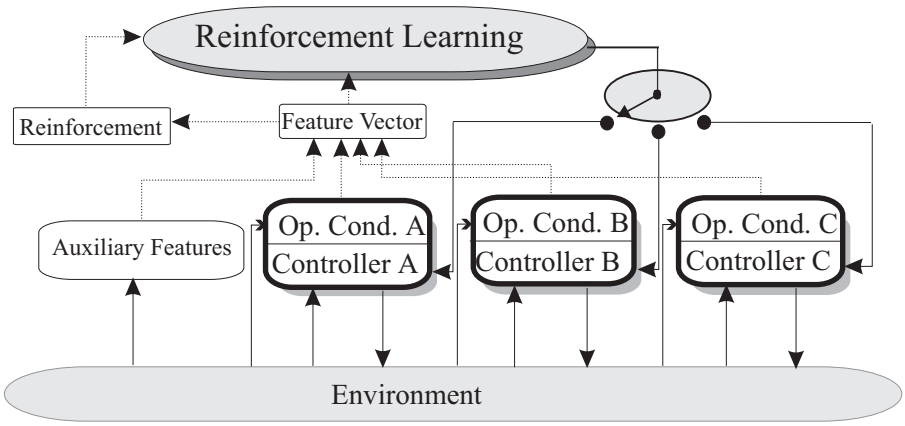

Figure 2. **The control and learning mechanisms.** At any time one and only one controller controls the plant. This controller is selected by the switching function learnt by RL. Switching can occur only when the feature vector changes which is the concatenation of the operating conditions of the controllers and some auxiliary features. The reinforcement signal is also a function of this feature vector. Controllers can only be activated when their associated operating conditions are observed.

the switching function depends on the observed feature values, the controllers will certainly remain active until a change in the feature vector is observed. We further allow the controllers to manipulate the observation process. In this the controllers may inhibit an observation from occurring and thus may hold up their activity for a while, i.e., the controller works then in an open-loop mode. This yields a rougher time-discretization, which reduces the problem complexity again (since less number of decisions is needed). The working mechanism is illustrated in Figure 2.

2.4. Accessibility decision problems

The goal of the design procedure is to set up the modules and additional features in such a way that there exists a switching controller $S : F \to \{1, 2, \ldots, n\}$, which for any given history results in a closed-loop behavior that fulfills the "goal" of control in time. It can be extremely hard to prove even the *existence* of such a valid switching controller. One approach is to use a so-called *accessibility decision problem* for this purpose, which is a 5-tuple $\langle X, A, r, \mathcal{T}, \mathcal{A} \rangle$, where X is the state-space, A is the action space, $r : X \times A \times X \to \mathbb{R}$ gives the immediate reward associated with transitions, $\mathcal{T} : X \times A \to P(X)$ is the transition mapping determining the states which are accessible from any given state-action pair, and $\mathcal{A} : X \to A$ gives us the admissible actions for any state x.

In our case the state space is F (the possible values of the composed feature-vector), the action space is $\{1, 2, \ldots, n\}$, the indices of the controllers, and the transition mapping is defined in the following way. The states that are accessible from feature-state f when using action i are those elements f' of F for which there

exists a history compatible with the feature f, such that when assuming the given history and using the controller indexed by i the next observed feature different from f will be f'. This means that the utilized controllers should be designed such that their operating conditions should not remain satisfied forever.[5] However, this is quite a natural condition since when translated back to the level of design it just means that each subtask should be completed in finite time.[6]

There are two ways of meeting this finite-finishing-time requirement. Firstly, design each controller with special attention to the problem, or secondly employ special features such as operating conditions, which, once "activated", terminate in some finite time. Continuing with the definition of the accessibility decision problem, the set of admissible "actions" corresponding to a feature-vector f consists of the index of those modules whose operating conditions are satisfied in f. Assume that the ultimate goal of control is to reach a certain subset of F (for this, goal-indicator features should be provided). Let us now set up a reward function r as a function of the features such that $r(f) = 1$ only if f is in the goal set, otherwise $r(f) = 0$. In the context of hybrid control, Sastry et al. noted the following: If there exists a controller for the above accessibility decision problem where the cumulated worst-case reward is non-zero (i.e. during evaluations only worst-case transitions are considered (Heger, 1996)) then there exists a switching controller that can solve the original problem (Lygeros, Godbole, and Sastry, 1997). Such a controller will be called proper in the worst-case sense. The reverse of the above implication is not necessarily true and this makes the analysis somewhat limited. Besides this, the exact accessibility decision problem can be very difficult to construct – usually it is easier to construct a slightly broader one, which has additional transitions in it. Another problem is that the exact transitions may well already revoke the existence of a controller that is proper in the above worst-case sense.

A weaker condition to the above, which may seem somewhat artificial at a first glance, is that the total probability of reaching the goal for all possible *accessibility-compatible* transition probabilities should equal one. The idea behind it is that under certain conditions, such as ergodicity, we may assume that transitions can be modelled probabilistically. A transition-probability matrix is called accessibility-compatible if the transition-probability associated with a given transition (f, i, f') is non-zero if and only if f' is accessible from f using i in the sense of the definition given in the previous paragraph. Clearly, if there is a path in the graph underlying the accessibility transitions from each non-goal state f to one of the goal states, then there exists a switching policy under which the probability of reaching the goal states is one for each accessibility-compatible transition-probability matrix. Such a switching strategy will be called an *almost surely proper* switching. Notice that due to the finiteness of the problem the expected number of steps to reach the goal will almost certainly be finite under any proper switching. If we knew that transitions take bounded physical time (this is not immediately obvious since the same transition may happen under an infinite number of different conditions because the process has an infinite state space and also transitions can depend on the history of the process – another infinite set), then the expected total physical time will also be bounded.

Of course, since the definitions of the modules and features depend on the designer, it is reasonable to assume that by clever design a satisfactory decomposition and controllers could be found even if only qualitative properties of the controlled object were known. RL could then be used for two purposes: either to find the best switching function assuming that at least two proper switching functions exist, or to decide empirically whether a valid switching controller exists at all. The first kind of application of RL arises as result of the desire to guarantee the existence of a proper switching function through the introduction of more modules and features than is minimally needed. But then good switching that exploits the capabilities of all the available modules could well become too complicated to find manually.

2.5. RL *and* ε*-stationary decision problems*

If the accessibility decision problem were extendible with transition-probabilities to turn it to an MDP[7] then RL could be rightly applied to find the best switching function. For example if one uses a fixed (maybe stochastic) stationary switching policy and provided that the system dynamics can be formulated as an MDP then there is a theoretically well-founded way of introducing transition-probabilities (see Singh, Jaakkola, and Jordan, 1995). Unfortunately, the resulting probabilities may well depend on the switching policy which can prevent the convergence of the RL algorithms.

However, the following "stability" theorem shows that the difference of the cost of optimal policies corresponding to different transition probabilities is proportional to the extent the transition probabilities differ, so we may expect that a slight change in the transition probabilities does not result in completely different optimal switching policies and hence, as will be explained shortly after the theorem, we may expect RL to work properly, after all.

THEOREM 1 *Assume that two MDPs differ only in their transition-probability matrices, and let these two matrices be denoted by* p_1 *and* p_2*. Let the corresponding optimal cost functions be* v_1^* *and* v_2^**. Then*

$$||v_1^* - v_2^*|| \leq \gamma \frac{nC||p_1 - p_2||}{(1-\gamma)^2},$$

where $C = ||c||$ *is the maximum of the immediate costs,* $||\cdot||$ *denotes the supremum norm and* n *is the size of the state space.*

Proof: Let T_i be the optimal cost operator corresponding to the transition-probability matrix p_i, i.e.

$$(T_i v)(s) = \min_{a \in U(x)} \left(c(s,a) + \gamma \sum_{s' \in X} p_i(s,a,s') v(s') \right), \; v : \; S \to \Re, \; i = 1, 2.$$

Proceeding with standard fixed point and contraction arguments (see e.g. Szepesvári and Littman, 1997) we get that $\|v_1^* - v_2^*\| \leq \|T_1 v_1^* - T_1 v_2^*\| + \|T_1 v_2^* - T_2 v_2^*\|$ and

since T_1 is a contraction with index γ, and the inequality $\|T_1 v - T_2 v\| \leq \gamma \|p_1 - p_2\| \sum_{y \in X} |v(y)|$ we obtain $\delta = \|v_1^* - v_2^*\| \leq \gamma\delta + \gamma\|p_1 - p_2\||X|C/(1-\gamma)$, where $\|v_2^*\| \leq C/(1-\gamma)$ has been employed (Ross, 1970). Rearranging the inequality in terms of δ then yields Theorem 1. ■

Motivated by the previous theorem, we define ε-stationary MDPs as the 4-tuple $\langle S, A, p, c \rangle$, where S, A and c are as before but p, the transition probability matrix, may vary in time but with $||p_t - p^*|| \leq \varepsilon$ holding for all $t > 0$. Our expectations are that although the transitions cannot be modelled with a fixed transition probability matrix (i.e. stationary MDP), they can be modelled by an ε-stationary one even if the switching functions are arbitrarily varied.

By still assuming that a stationary MDP corresponds to a fixed switching, we obtain, if a set (maybe stochastic) switching policy is followed during learning, that the values learnt by RL will converge somewhere. However, on-line RL will change the exploration-policy continuously, which may result in oscillating transition-probabilities. We conjecture that the method developed by Szepesvári and Littman (Littman and Szepesvári, 1996) can still be applied, but now with a reduced goal to show that if during learning the transition probabilities were oscillating slightly (say remaining within a set of diameter ε) then RL methods would result in oscillating estimates of the optimal value function, but with the oscillation being asymptotically proportional to ε. Again this is an incomplete result as it leaves open the question of whether the oscillations in the transition probabilities are asymptotically small, which can be hard to answer since the actual policy executed during learning usually depends on the estimated values of the optimal cost function, and the transition probabilities may depend on the learnt values, which all go to close the circle. A possible way out of this vicious circle is to make use of an assumption like, say, that the transition probabilities corresponding to different policies should not differ too much at all, i.e. to assume that the MDP is ε-stationary. This property was clearly observed in our experiments, which we will now describe.

3. Experiments

The validity of the proposed method was checked with actual experiments carried out using a Khepera robot. After the specification of the task the modules were designed and then several RL algorithms were tried and compared. The description of the robot, the experimental setup, general specifications of the modules, and the results are all presented in this section. We would expect similar results for other robots, too.

3.1. The robot and its environment

The mobile robot employed in the experiments is shown in Figure 3.

It is a Khepera[8] robot equipped with eight IR sensors, six in the front and two at the back, the IR sensors measuring the proximity of objects in the range 0-5 cm.

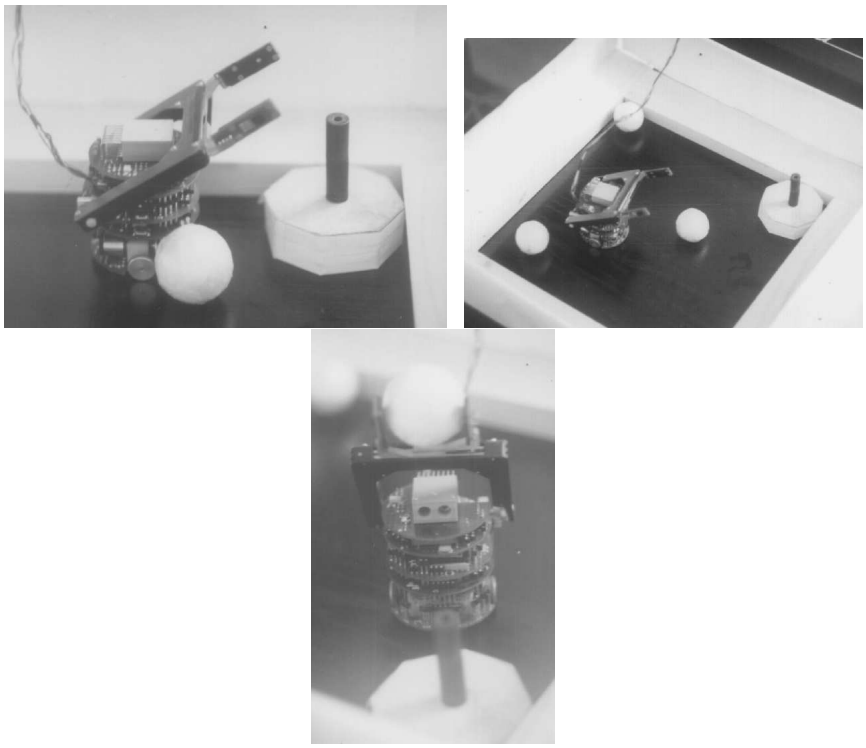

Figure 3. **The Khepera and the experimental environment.** The top-left sub-figure shows a close-up on the Khepera robot. The robot has two independent wheels, a gripper, a vision turret on the top of it (this can be better observed in the third sub-figure) and proximity sensors. The task was to grasp a ball and hit the stick with it. The top-right sub-figure shows a phase when the robot is searching for a ball, while the third sub-figure shows a case when the robot is just about to hit the stick by the ball. The umbilical cord can also be seen in the figures.

The robot has two wheels driven by two independent DC motors and a gripper that has two degrees of freedom and is equipped with a resistivity sensor and an object-presence sensor. The vision turret is mounted on the top of the robot as shown. It is an image sensor giving a linear image of the horizontal view of the environment with a resolution of 64 pixels and 256 levels of grey. The horizontal viewing angle is limited to about 36 degrees. This sensor is designed to detect objects in front of the robot situated at a distance spanning 5 to 50 cm. The image sensor has no tilt angle, so the robot observes only those things whose height is exceeds 5 cm.

The learning task was defined as follows: find a ball in an arena, bring it to one of the corners marked by a stick and hit the stick with the ball. The robot's environment is shown in Figure 3. The size of the arena was 50 cm x 50 cm with a black colored floor and white colored walls. The stick was black and 7 cm long, while three white-colored balls with diameter 3.5 cm were scattered about in the arena. The environment is highly chaotic because the balls move in an unpredictable manner and so the outcome of certain actions is not completely predictable, e.g. a grasped ball may easily slip out from the gripper. Note also that the task is quite complex compared to the tasks considered in the mobile learning literature (see e.g. Birk and Demiris, 1998).

3.2. The modules

3.2.1. Subtask decomposition Firstly, according to the principles laid down in Section 2, the task was decomposed into subtasks. The following subtasks emerged naturally (see Figure 4): (T1) to find a ball, (T2) grasp it, (T3) bring it to the stick, and (T4) hit the stick with the grasped ball. Subtask (T3) was further broken into two subtasks, that of (T3.1) 'safe wandering' and (T3.2) 'go to the stick', since the robot cannot see the stick from every position and direction. Similarly, because of the robot's limited sensing capabilities, subtask (T1) was replaced by safe wandering and subtask (T2) was refined to 'when an object nearby is sensed examine it and grasp it if it is a ball'. Notice that subtask 'safe wandering' is used for two purposes (to find a ball or the stick). The operating conditions of the corresponding controllers arose naturally as (T2) – an object should be nearby, (T3.2) – the stick should be detected, (T4) – the stick should be in front of the robot, and (T1,T3.1) – no condition. Since the behavior of the robot must differ before and after locating a ball, an additional feature indicating when a ball was held was supplied. As the robot's gripper is equipped with an 'object-presence' sensor the 'the ball is held' feature was easy to implement. If there had not been such a sensor then this feature still could have been implemented as a switching feature: the value of the feature would be `'on'` from the time instant when the robot used the grasping behavior until it uses the hitting behavior. An 'rescue' subtask and corresponding controller were also included since the robot sometimes got stuck. Of course yet another feature is included for the detection of "goal states". The corresponding feature indicates when the stick was hit by the ball. This feature's value is `'on'` iff the gripper is half-closed but the object presence sensor does not give a signal. Because of the implementation of the grasping module (the gripper

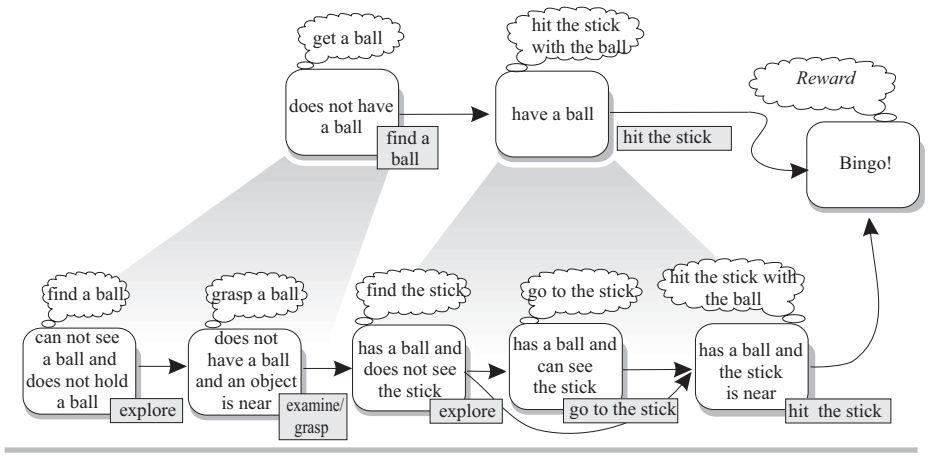

Figure 4. **Subtask decomposition example.**
The main task of hitting the stick with a ball is first broken up into two subproblems: get a ball and then hit the stick with the ball. Then these are again decomposed into smaller subtasks which can already be accomplished by simple controllers. The bubbles show the subtask, the rectangles with rounded corners show the conditions under which the solution of the given subtask is sensible and the shaded rectangles show the macro-actions. The arrows show the ideal flow of working, in practice other transitions are also possible and do exist. Note that the "explore" macro-action is used twice in the ideal flow of working. One particular "maintenance subtask" is not shown to preserve the clarity of the figure. This is the "rescue" subtask.

was closed only after the grasping module was executed) this implementation of the "stick has been hit by the ball" feature was satisfactory for our purposes, although sometimes the ball slipped out from the gripper in which case the feature turned 'on' even though the robot did not actually reach the goal. Fortunately, this situation did not happen too often, and, thus did not affect learning.

3.2.2. Features and controllers The resulting list of modules and features, each of which we now elaborate upon, is shown in Table 2. The dynamics of the controller associated with `Module 1` was based on the maximization of a function that depended on the proximity of objects and the speed of both motors.[9] If there were no obstacles near the robot, this module made the robot go forward. This controller could thus serve as one for exploring the environment. `Module 2` was applicable only if the stick was in the viewing angle of the robot, which could be detected in an unambiguous way because the only black thing that could get into the view of the robot was the stick. The range of allowed behavior associated with this module was implemented as a proportional controller that drove the robot in such a way

Table 2. **Description of the features and the modules.** 'FNo.' means feature number. Note that features 1-5 are the operating conditions of their associated controllers. In the column labelled by 'on' the conditions under which the respective feature's value is 'on' are listed, while the last column lists the controllers associated with the respective feature (if any).

FNo.	**'on'**	**Behavior**
1	always	explore while avoiding obstacles
2	if the stick is in the viewing angle	go to the stick
3	if an object is near	examine the object grasp it if it is a ball
4	if the stick is near	hit the stick
5	if the robot is stuck	go backward
6	if the ball is grasped	-
7	if the stick is hit with the ball	-

that the angle difference between the direction of motion and line of sight to the stick was reduced. The behavior associated with `Module 3` was applicable only if there was an object next to the robot, which was defined as a function of the immediate values of IR sensors. The associated behavior was the following: the robot turned to a direction that brought it to point directly at the object, then the gripper was lowered. If the object-presence sensor attached to the gripper gave a signal, then the robot judged that the object sensed was a ball and not the fence, so the robot closed the gripper and picked the object up. If the object-presence sensor did not signal the robot lifted up the gripper, then turned to a random direction and started going forward. The observation process was switched off until the whole procedure was finished. `Module 4` was the "hitting" module. Its feature function was `'on'` when the stick was near, i.e., the total activity of the linear-eye sensors exceeded a given constant[10], otherwise it was `'off'`. The associated behavior was to let the gripper down and then raise it anew. This module was independent of whether there was a ball in the gripper or not, which entailed the robot having to learn that this module wasn't needed unless the ball was grasped. `Module 5`, as noted earlier, was created to handle stuck situations. This module makes the robot going backward and is applicable if the robot has not been able to move the wheels into the desired position for a while. This condition is a typical time-window-based feature. The sixth feature indicated the presence of the ball, which was `'on'` if a ball was held, while `Feature 7` was the goal-detection feature described earlier.

The operating conditions of controllers were not exclusive; on the contrary, there were many "states" when more than one behavior was simultaneously applicable. On the other hand, some features were totally independent of each other. For example, if the robot was stuck then there must have been an object nearby, i.e.,

if `Feature 2 = 'on'` then `Feature 4 = 'on'` as well. These dependencies mean that the "actual" state space was much smaller than 2^7 (= 128), the total number of possible states.

Simple case-analysis shows that there is no switching controller that can reach the goal with complete certainty within finite time (in the worst case, the robot could return accidentally to state "10000000" from any state when the goal feature was `'off'`), but this argument also shows that an almost-sure switching strategy, and therefore one which attains the goal in finite expected time, should always exist. This is simply because the goal state can be reached from the state "10000000" with positive probability under a simple action sequence.

3.3. The cost structure

In order to promote fast learning using RL one must design the immediate costs in a careful manner. A dense cost structure was applied: the cost of using each behavior was one except when the goal was reached, which had a cost of zero. Costs were discounted at a rate of $\gamma = 0.99$. Note that from time to time the robot by chance became stuck (the robot's 'stuck feature' was `'on'`), and the robot tried to execute a module which could not change the value of the feature vector. This meant that the robot did not have a second option to try another module since by definition the robot could only make decisions if the feature-representation changed. As a result the robot could sometimes get stuck in a "perpetual" or so-called "jammed" state. To prevent this from happening, we built in an additional rule which was to stop and reinitialize the robot when it got stuck and could not unjam itself after 50 sensory measurements. A cost equivalent to the cost of never reaching the goal, i.e. a cost of $\frac{1}{1-\gamma}$ (= 100) was then communicated to the robot, which mimicked in effect that such actions virtually last forever.

3.4. The details of learning

Experiments were fully automated and organized in trials. Each trial run lasted until the robot reached the goal or the number of decisions exceeded 150 (a number that was determined experimentally), or until the robot became jammed. The 'stick was hit' event was registered by checking the state of the gripper (see also the description of `Feature 7`).

During learning, the Boltzmann-exploration strategy was employed where the temperature was reduced by $T_{t+1} = 0.999\,T_t$ uniformly for all states (Barto et al., 1995).[11] During the experiments, the cumulative number of successful trials were measured and compared to the total number of trials done so far, together with the average number of decisions made in a trial.

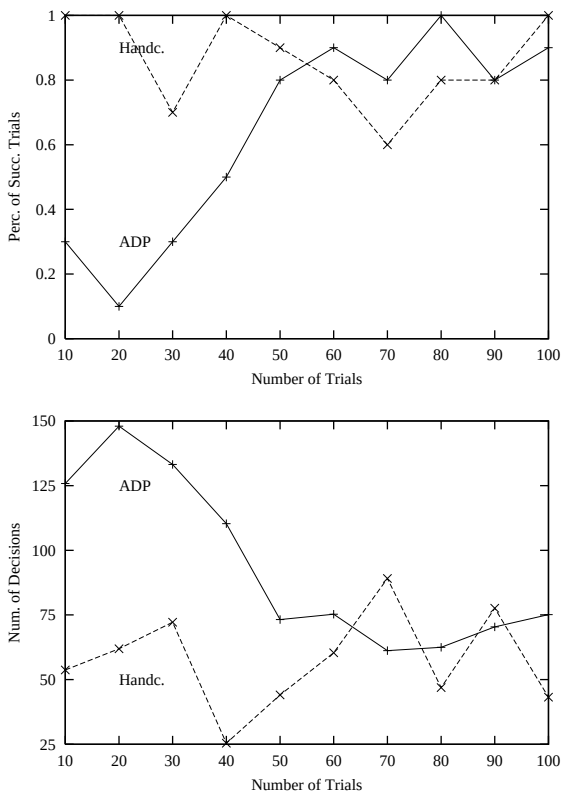

Figure 5. **Learning curves.** In the first graph, the percentage of successful trials out of ten are shown as a function of the number of trials. In the second graph, the number of decisions taken by the robot and averaged over ten trials is shown, also as a function of the number of learning trials. Results are shown for both the rules obtained by ADP and handcraft.

3.5. Results

Two sets of experiments were conducted. The first set was performed to check the validity of the module-based approach, while the second was carried out to compare different RL algorithms. In the first set, the starting exploration parameter T_0 was set to 100 and the experiment lasted for 100 trials. These values were chosen in such a way that the robot could learn a good switching policy, the results of these experiments being shown in Figure 5. One might conclude from the left subgraph, which shows the percentage of task completions in different stages of learning, that the robot could solve the task after 50 trials fairly well. Late fluctuations were attributable to unsuccessful ball searches: as the robot could not see the balls if they were far from it, the robot had to explore to find one and the exploration sometimes took more than 150 decisions, yielding trials which were categorized as

Table 3. **One part of the learned policy.** The six middle entries of the rows are the feature values corresponding to those features listed and numbered in Table 2. (The values of `Feature` 7 are not shown as these are always zero here). The column marked by the label 'Mid' denotes the number of the module that was chosen by a pure exploitation policy under the conditions described by the respective feature values. A handcrafted policy is shown in the column labeled by 'Hand'. For example, in State 3 the robot would use the module "examine object" (Controller 3) under the pure exploitation strategy and the controller "go backward" under the handcrafted policy.

No.	1	2	3	4	5	6	Mid	Hand
1	1	0	1	0	0	0	3	3
2	1	1	0	0	0	0	2	**1**
3	1	1	1	1	1	0	3	**5**
4	1	0	1	0	1	0	5	5
5	1	1	0	1	0	0	2	**1**
6	1	0	1	0	0	1	1	1
7	1	1	0	0	0	1	2	2
8	1	1	1	1	1	1	4	4
9	1	0	1	0	1	1	5	5
10	1	1	0	1	0	1	4	4

being failures. At the very beginning of learning, the robot tried out the appropriate types of behavior almost completely randomly, resulting in a large number of decisions per trial. The evaluation of behavior coordination is also observed in the second subgraph, which shows the number of decisions per trial as a function of time. The reason for later fluctuations is again due to a ticklish ball search. The performance of a handcrafted switching policy is shown on the graphs as well. As can be seen the differences between the respective performances of the handcrafted and learnt switching functions are negligible. In order to get a more precise evaluation of the differences the average number of steps to reach the goal was computed for both switchings over 300 trials, together with their standard deviations. The averages were 46.61 and 48.37 for the learnt and the handcrafted switching functions respectively, with nearly equal standard deviations of 34.78 and 34.82, respectively.

One part of the learned policy is shown in Table 3, where 10 states were selected from the 25 explored ones together with their learned associated behaviors. Theoretically, the total number of states is $2^7 = 128$, but as learning concentrates

on feature configurations that really occur, this number happened to be just 25 here. The expected difference between the robot's behavior before and after finding a ball can readily be seen. For example, in State 5 the robot moves towards the stick (compare Tables 2 and 3), because it has realized that this behavior leads to an object with high confidence that usually happens to be a ball. In the dual state, State 10, which differs from State 5 only in that ball is now held, the robot properly chose the hitting action. Note that the learned policy was always consistent with the handcrafted rules, but in certain cases the learned rules are more refined than their handcrafted counterparts. One part of the handcrafted rules are shown in the Table 3. For example, as the above example shows the robot learned to exploit the fact that the arena was not completely level and as a result balls were biased towards the stick. The learned actions in States 3 and 4 reveal another unexpected result: when the robot was stuck, the presence of the stick made a difference. If the stick was in the viewing angle the robot chose the object-examination behavior, otherwise it went backwards. This difference is again due to the fact that the balls are frequently situated around the stick and consequently the object-examination behavior in a stuck state, and when the robot was close to the stick, would often lead to the gripping of a ball while simultaneously freeing the robot with high probability – so it was worth trying the object-examination behavior in states like State 3.

In the rest of the experiments, we compared two versions of ARTDP and three versions of *real-time Q-learning* (RTQL). The two variants of ARTDP were ADP, and ARTDP with the single case when $F_t := \{s_t\}$ and only one iteration in the inner cycle of the algorithm was performed. Note that due to the small number of states and module-based time, discretization even ADP could be run in real time. But variants of RTQL differ in the choice of the learning rate's time dependence. RTQL:SC refers to the choice of the so-called *search-then-converge* method, where $\alpha_k(s,a) = \frac{50}{100+n_k(s,a)}$, $n_k(s,a)$ being the number of times the event $(s,a) = (s_t, a_t)$ happened before time k plus one (the parameters 50 and 100 were determined experimentally as being the best choices). In the other two cases (the corresponding algorithms were denoted by RTQL:0,1 and RTQL:0,25 respectively), constant learning rates (0.1 and 0.25, respectively) were utilized.

The online performances of the algorithms were measured as the cumulative number of successful trials. An example of the time-dependence of these values are depicted in Figure 6 during the learning procedure, and for the learning cases with ADP and RTQL:0,25. The starting exploration constant was set to $T_0 = 50$, a slope of 45° meaning that all of the trials were successful. Again, late drops in the graphs can be ascribed to unlucky ball searches. The bigger the curve slope, the faster was the rate of learning, i.e. the smaller is the regret of learning. By definition, the regret R_t at time t is the difference between the performance of an optimal agent (robot) and that of the learning agent accumulated up to trial t, i.e. it is the price of learning up to time t. If $s(t)$ denotes the number of successful trials out of the first t trials and the robot learns to behave "optimally" after trial number t_0 (i.e., it is able to hit the stick with the ball in every trial after time t_0) then $R = R_t = t - s(t) = t_0 - s(t_0)$ is the total regret assuming that the optimal agent can

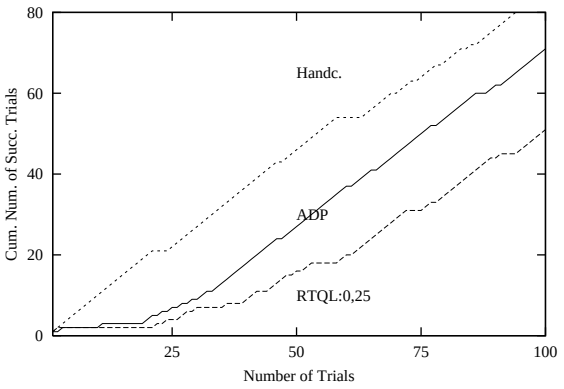

Figure 6. **The cumulative number of successful trials** in the case of learning with ADP and RTQL:0,25, and Boltzmann exploration with initial temperature $T_0 = 50$. The results for the handcrafted rule are also shown.

Table 4. **The table shows the statistics used in** ANOVA**.** In each cell the sample mean and the sums of squares (SS) are shown for a given initial temperature and an algorithm type. The initial temperatures are shown in the first column, while the acronyms for algorithms are shown in the first row. The numbers in the last column give the sample mean and the sums of squares for a given initial temperature and, similarly, the numbers in the last row give the same statistics for a given algorithm type.

	ADP	ARTDP	RTQL:SC	RTQL:0,25	RTQL:0,1	Total Rows
$T_0 = 100$	60.4;449.3	67;376.5	76.8;245.2	84.2;62.2	87;116.5	75.08;314.33
$T_0 = 50$	53.8;306.7	64;502.5	67.4;62.3	83.8;391.7	80.2;194.7	69.84;367.89
$T_0 = 25$	44.8;154.7	68.6;191.3	71.8;368.7	68.6;460.8	83.2;150.2	67.4;384
$T_0 = 0$	54.8; 702.7	45.6;277.3	68.8;152.7	82.8;383.7	79;270	66.2;506.17
Total Columns	53.45;372.58	61.3;373.06	71.2;188.17	79.85;318.03	82.35;164.03	

reach the goal in every trial (this assumption is relevant since the handcrafted agent could almost achieve this performance). All algorithms were examined with all the four different exploration parameters ($T_0 = 100$, $T_0 = 50$, $T_0 = 25$, $T_0 = 0$) since the same exploration rate may well result in different regrets for different algorithms, as was also confirmed in the experiments. For each algorithm and temperature 5 independent experiments were performed. (Altogether $5 \times 4 \times 5$ experiments were conducted which took a total of 40 days and nights.) The results are evaluated by the analysis of variance (ANOVA). Since ANOVA requires the normality of the data and that the within group variances are equal we performed the following tests: Denote the data obtained from the kth experiment ($1 \leq k \leq 5 = n$) for algorithm index i and temperature index j by ξ_{ijk}, denote the sample average of

Table 5. **The table shows the results of** ANOVA **at the confidence level of 90%.** The rows give the same statistics for different groups of data. SS is the sum of squares, df is the degrees of freedom for the F-statistics, MS is the empirical variance, F is the obtained F-statistics and $F_{\text{crit.}}$ is the critical F-value at the confidence level 90%. If $F \geq F_{\text{crit.}}$ then the null-hypothesis that the given factor does *not* influence the variances must be *rejected* at the given level.

Factors\Statistics	SS	df	MS	F	p-value	$F_{\text{crit.}}$
Temperature	1162.11	3	387.37	1.331237	0.270082	2.718785
Algorithm	11997.86	4	2999.465	10.30797	0	2.485883
Between groups	2460.54	12	205.045	0.704658	0.742507	1.875261
Within group	23278.8	80	290.985			
Total	38899.31	99				

ξ_{ijk} for fixed i, j and variable k by $\overline{\xi}_{ij\cdot}$, the empirical variance of the same data by $s^2_{ij\cdot}$. If ξ_{ijk} are independent and for fixed (i, j) and variable k they are normal from the same distribution, then $\sqrt{n-1}\left(\xi_{ijk} - \overline{\xi}_{ij\cdot}\right)/s_{ij\cdot}$, i, j, $1 \leq k \leq 4$ are independent and t-distributed with parameter 4. The Kolmogorov test was performed to check this. The obtained statistics was 0.1067, which corresponds to a p-level of 0.32 since $n = 5 \times 4 \times 4 = 80$, which means that only if we allow a larger than 32% error-probability on the test can the null-hypothesis (i.e. that the data is normal) be rejected. Second, we computed the Bartlett-statistics to check the equality of within group variances and obtained $K^2 = 13,2786$ which corresponds to a probability of $0,82$ with $f = 19$ being the degrees of freedom, thus the hypothesis on the equality of variances can be accepted. After this ANOVA was performed. The variance table is shown in Table 4. The results of the ANOVA are shown in Table 5. The main conclusions of this analysis are that with a 90% confidence i) there is no interaction in between the temperature and the algorithms, i.e., the effects of these are can be decoupled ($p = 0.74$); ii) the regret is not effected by the temperature ($p = 0.27$), but iii) the choice of algorithms influences the regret ($p = 0$). Considering the average regrets for the different algorithms one gets the ordering (ADP,ARTDP,RTQL:SC,RTQL:0,25,RTQL:0,1) in terms of increasing regret. Further analysis showed that the p-levels for the differences between the means are ADP-ARTDP:0,44, ARTDP-RTQL:SC: 0,37, RTQL:SC-RTQL:0,1: 0,41, RTQL:0,1-RTQL:0,25: 0,77, i.e., the model-based algorithms have a significant advantage over the model-free ones, among which the best algorithm which uses the search-and-then-converge learning rate schedule performs significantly better than the others with constant learning rates.

We have also tested another exploration strategy which Thrun found the best among several undirected methods[12] (Thrun, 1992). These runs reinforced our previous findings that estimating a model (i.e. running ADP or ARTDP instead of Q-learning) could reduce the regret rate by as much as 50%.

4. Related work

There are two main research tracks that influenced our work. The first was the introduction of features in RL. Learning while using features was studied by Tsitsiklis and Van Roy to deal with large finite state spaces, and also to deal with infinite state spaces (Tsitsiklis and Van Roy, 1995). Working on the output of features can well make the problem partially observable, so one should not expect that RL algorithms that involve optimization will work in general (the theoretical results of Tsitsiklis and Van Roy concern only estimation types of algorithms, such as TD(λ), or non-adaptive algorithms (Tsitsiklis and Van Roy, 1996)). Issues of learning in partially observable environments have been discussed by Singh, Jaakkola, and Jordan (1995).

The second track is related to the use of local controllers together with a switching function that selects the controller to be activated at any arbitrary time. In connection with this topic, very recently and independently of us, Sastry proposed the use of a hybrid-control approach to solve complex problems like highway-traffic control (Sastry, 1997). He proposed the design of several controllers, such as a car-following controller and an overtaking-controller, among others, which are imagined to work under different and well-specified conditions. He assumed that the system dynamics were known and so the main concern of his approach was to find a switching controller that switched between the different controllers and that met certain requirements such as safety, maximal comfort (of the passengers) and maximal throughput (of the highway), the criteria importance having been ordered in this way. He used analytical tools to derive the accessibility decision problem and suggested worst-case analysis to prove feasibility. Clearly, in contrast with his, we assume here only a qualitative knowledge of the system-dynamics, which nevertheless enables us to design the modules and perform a preliminary feasibility analysis. Further, we let let the system itself find a good switching strategy by adapting to the actual environment.

A different approach was taken by Connell and Mahadevan whose work complements ours in that they set up subtasks to be learned by RL and fixed the switching controller (Mahadevan and Connell, 1992). The main aim of their work was to prove that RL could be applied to learn good controllers at the noisy and unreliable sensor-actuator level, where also the state and action spaces were infinite. They found that statistics-based clustering combined with RL could solve this problem. Dorigo and Colombetti (1994) also considered the learning controllers using genetic algorithms. In future, we plan to extend our work in this direction, i.e. only the subtask decomposition and the features will be designed by hand, and RL will then be employed to learn both the low-level controllers and the switching controller, possibly simultaneously (for other hierarchical RL studies see the survey (Kaebling, Littman, and Moore, 1996) and the references therein).

Asada et al. considered many aspects of mobile-robot learning. They applied a vision-based state-estimation approach and defined "macro-actions" similar to our controllers (Asada, Noda, Tawaratsumida, and Hosoda, 1996). In one of their papers, they describe a goal-shooting problem in which a mobile robot shot a goal

while avoiding another robot (Uchibe, Asada, and Hosoda, 1996). First, the robot learned two behaviors separately: the "shoot" and "avoid" behaviors. Then, the two behaviors were synthesized by a handcrafted rule and later this rule was refined via RL. The learned action values of the two behaviors were reused in the learning process while the combination of rules took place at the level of state variables.

Matarić considered a multi-robot learning task where each robot had the same set of behaviors and features (Matarić, 1997). Although the features of Matarić could clearly be interpreted as operating conditions of behaviors she did not restrict the applicability of the behaviors to the appropriate subspaces, which increased the complexity of the decision problem unnecessarily. Just as in our case, her goal was to learn a good switching function by RL. She considered the case when each of the robots learned separately and the ultimate goal was that learning should lead to a good collective behavior, i.e. she concentrated mainly on the more involved multi-agent perspective of learning. She found that Q-learning worked badly compared to her "shaped reinforcement" approach, which she found to be comparable to a handcrafted rule. In her approach, time averages of summed and appropriately defined immediate reinforcements served as the basis of decisions, i.e the handcrafted immediate reinforcements balanced the different aspects of the task and helped encode the structure of the switching function. This design required a lot a priori knowledge and experience, and seemed to be costly compared to the design of a good switching function. In her experiments with Q-learning, Matarić used a sparse reward structure, a unit reward was communicated to the learner when it reached the goal, otherwise a reward of zero was given. However, it is well known that dense rewards facilitate clever exploration rather than sparse ones. (If dense rewards are used, the agent will be able to differentiate between tried and untried actions independently of whether the goal is reached or not, which can reduce the search complexity in the trials considerably (Koenig and Simmons, 1997)). In contrast to her work, we followed a more engineering-oriented approach when we suggested designing the modules based on well-articulated and simple principles. Contrary to her findings, we discovered that RL (with a dense-reward structure) can indeed work well at the modular level.

In the AI community, there is an interesting approach to mobile-robot control called Behavior-Based Artificial Intelligence in which "competence" modules or behaviors have been proposed as the building blocks of "creatures" (Maes, 1991b; Brooks, 1991b). Each of these modules has a list of preconditions similar to our own operating conditions cited here. The decision-making procedure is, on the other hand, usually quite different from ours. Maes, for example, proposed what she called a local-computation scheme. The modules were linked together through different channels and may inhibit or excite each other. Activation was spread along the channels and accumulated at the most relevant behavior nodes. The behavior whose activation first went above threshold was selected and executed. After finishing the behavior, the activation level of the behavior was reset to zero and the whole process repeated. Like every ad hoc method, this method required careful tuning. Tyrrell found that, in a complex decision task aimed to simulate the task faced by a zebra living in the African Savannah, the decision-making mechanism by Maes could not

work well compared to other action-selection mechanisms. Moreover, he added that there might be theoretical reasons behind this failure (Tyrrell, 1993). Maes later pointed out that Tyrrell's findings could be debated (Maes, 1991a). In another work of hers, she also proposed the learning of links between the modules (Maes, 1992) and she also tried out this on a real-robot (Maes and Brooks, 1990).

Yet another main direction of automatic robot programming research uses genetic algorithms to find good robotic programs (see e.g. Brooks, 1991a; Koza and Rice, 1992) in a space of possible programs. Alternatively basic behaviors, including their coordination, can be learned by using a classifier systems' approach (Dorigo, 1995) and genetic algorithm. Like us, Dorigo also emphasized that design and learning should be well balanced and outlined a general "methodology for behavior engineering" (Colombetti, M.Dorigo, and Borghi, 1996). Here we have gone further as we suggested specific tools which link the design issues to theoretically well based disciplines such as planning in AI systems, classical control designs and reinforcement learning. In this way a consistent view of the design issues has been developed and so the role of different components (models, planning, subgoals, behaviors, operating conditions, features, filters, modules, reinforcement, learning, etc.) becomes clear. Nevertheless, Dorigo touched some issues which are outside the scope of our work. For example, he considered some further complex relationships between behaviors, such as the combination, 'independent sum' and sequences of controllers, i.e. respectively: the superposition of control signals coming from different local controllers; different local controllers operating simultaneously but affecting a disjoint set of actuators; and the operation where controllers are only used sequentially, each controller waiting for the preceding controller to 'finish' before acting. Sequencing can be viewed as a tool to resolve problems related to partial observability. It is a form of implicit memory usage and could be easily incorporated in our framework by adding a feature which becomes activated only when a controller finishes working. The other two relationships can be viewed as tools to exploit different properties of the task, such as a superposition property of control in the case of combination and subgoal independence (Korf, 1987) in the case of 'independent sum'.

A more involved problem related to subgoal independence has recently been explored by Singh and Cohn (1997) who considered the problem of finding an optimal policy in the direct product of a finite set of MDPs with a *single* action set (the state space of the product MDP is the direct product of the state spaces of the individual MDPs, the transition probabilities are also multiplied, but the rewards are summed up). True subgoal independence could be modelled in this way if the action sets were independent.[13] As noted in (Korf, 1987), the importance of independent subgoals should not be underestimated: *i)* independent subgoals reduce the branching factor by allowing the problem solver to focus on only a subset of actions at any given time and *ii)* most goals that we try to satisfy in our everyday life are almost independent. In the case of ii), think of the independence of actions needed to accomplish chores, job-related tasks, and recreational or social objectives related tasks. The only dependance between these tasks is the limited resources (time or money) available. Note that this dependence is quite weak until we reach

the limit of these resources. As we do not have any restrictions on the kind of subgoals to be used, independent subgoals can indeed be utilized in our design, however the notion of composite actions are not currently supported.

Sometimes dependence of subgoals is explicit and clear. For example, Dorigo and Colombetti (1994) implicitly use dependant subgoals to define useful behaviors, e.g. in their definition of Chase/Feed/Escape behavior when they declare that in this behavior the subgoal of escaping from a predator has precedence over the subgoal of feeding which again has precedence over chasing moving objects. In MDPs such precedence relations can be captured by certain vector-valued evaluation functions (Henig, 1983) and also RL algorithms can be derived which take into account the predefined precedences (Gábor, Kalmár, and Szepesvári, 1998).

Our module concept (operating conditions together with controllers) fits well with the skill concept of Thrun and Schwartz (1995) who derived an algorithm that learns "skills" useful to complete a set of tasks. The algorithm minimizes the sum of the loss due to the use of skills (instead of the low level actions) and the storage size needed to represent the policies using the acquired skills. They note that skills are also needed in certain single tasks. Here we would like to note that multiple task problems usually correspond to a subgoal decomposition of a single task (with e.g. independent, or serializable, or block serializable subgoals). We find this view fascinating since this is why it becomes meaningful to speak about the interplay of the subtasks! The algorithm of Thrun and Schwartz (1995) saves memory since a skill has a single associated value for each (sub)task independently of the actual state of the system. Skills also have domains which can be identified with our operating conditions – only those skills can be activated whose domain indicator are triggered by the actual state. In (Thrun and Schwartz, 1995) the probability of choosing an available skill is proportional to its squared value. The learned domains of skills could provide a very useful way of doing state-space abstraction when a true RL algorithm would work on the top of the feature space induced by these domains, as in our case where the RL algorithm worked on the feature space induced by the operating conditions. Restricting the decisions to be only dependant on the features' values introduces deviations from optimality but enables the introduction of hierarchies of modules (more precisely, a lattice structure over the modules): every module can activate any other module at a lower level in the hierarchy (the hierarchy prevents infinite cycles) or a low level action. A method similar to that of Thrun and Schwartz (1995) could then be used to invent operating conditions and/or the relationships among the modules, i.e., their coordination. The operating conditions of a controller could also be learned by relying on their definitions. Another application could be to directly use the algorithm of (Thrun and Schwartz, 1995) in the planning phase provided that the qualitative plant model is given as a MDP. This is the subject of our future research. Note that, in contrast to the macro-action utilization of (Precup et al., 1997), where the macro actions are used in a transparent manner (i.e., the computed policy can eventually be given purely by simple actions) to facilitate planning, we suggest to transfer information about subgoal decompositions to the actual control level also. This serves the purpose of task-oriented abstracting space, time and action. The idea of (Precup et al., 1997)

(and also that of Thrun and Schwartz (1995)) could be utilized in the planning phase of our method to initialize the coordination of behavior modules.

It is very important to note that the qualitative knowledge of the plant can be represented by any method, such as dynamical equation, symbolic rules, and is not restricted to MDPs. This may mean a very compact representation and may enable different kind of algorithms to work at the planning phase. Earlier, we suggested a method to learn a rule based representation on the top of an MDP representation (Kalmár, Szepesvári, and Lőrincz, 1994, 1995). This algorithms relies on a 'triplet' representation of MDPs (see Szepesvári and Lőrincz, 1994; Szepesvári, 1994) when transitions are represented and evaluated instead of state-action pairs or states. Transitions are then interpreted as rules that apply to specific situations and are combined to get new, more general rules which apply to a larger set of situations. We argued that the algorithm works well in deterministic problems (Kalmár et al., 1995). Such an algorithm, when only the most probable transitions are kept, could well be used to derive the qualitative representation needed in the planning phase of the method presented here.

5. Summary and conclusions

Following the traditions of RL based robot programming, an approach to module-based reinforcement learning was proposed to solve the coordination of multiple "behaviors" or controllers. Extended features (filters) served as the basis of time and space discretization as well as specifying the operating conditions of the modules. The construction principles of the modules were to: *i)* decompose the problem into subtasks using a qualitative model of the plant; *ii)* for each subtask, design controllers and specify the controllers' operating conditions using a more detailed model of the plant; *iii)* check if the problem could be solved by the controllers under the operating and observability conditions, add additional features or modules if necessary; *iv)* set up the reinforcement function and learn a switching function from experience. Although individual elements of our methods existed previously in the literature, we have combined them into a single, coherent, framework.

One particularly important motivation behind our approach was that a partially observable decision problem can usually be transformed into a completely observable one if appropriate features and local controllers are employed. Of course, some *a priori* knowledge of the task and robot is required to find those features and controllers. However, it is important to note that because of the adaptive part, the controllers and the interactions among them need not to be fine-tuned which allows quick and easy development of robot programming. It was argued that RL could work well even if the resulting problem was only almost stationary.

The design principles were applied to a fairly complex real-life robot learning problem and several RL-algorithms were compared in practice using the Analysis of Variance. We found that estimating the model and solving the optimality equation at each step (which could be done owing to the economic, feature-based time-discretization) yielded significantly better results than other approaches. The robot learned the task after 700 decisions, which usually took less than 15 minutes in real-

time. We conjecture that using a rough initial model good initial solutions could be computed off-line that could further decrease the time required to learn the optimal solution for the task.

The main difference between earlier works and our approach here is that we have established principles for the design modules and found that our subsequent design and simple RL worked splendidly. Plans for future research include extending the method via automatic subtask decomposition mechanisms, the learning of modules and operating conditions, and even by the learning of qualitatively correct world models which can be used in the planning phase to invent the subtask decomposition. These would reduce the amount of *a-priori* human knowledge which is important when human knowledge is unavailable such as in industrial process control. Also the analysis of almost stationarity in decision problems would be important to consider since this notation may provide a bridge between the theory, when we consider probabilistic models and practice, which corresponds to a deterministic, but chaotic world.

Acknowledgments

The authors would like to thank Zoltán Gábor for his efforts of building the experimental environment. This work was supported by the CSEM Centre Suisse d'Electronique et de Microtechnique, Switzerland, OTKA Grants No. F20132 and T017110, and the Hungarian Ministry of Education Grant No. FKFP 1354/1997.

Notes

1. If the state space is infinite then not all sensor-based features can be realized in practice.
2. If time is continuous, then recursive features should be replaced by features that admit continuous-time dynamics such as $\dot{f} = R(x, a, f)$. In order to have a finite-space output, such features should be used together with discretization mappings, i.e. the output of the feature is given by $h_R(f)$ instead of f, where $h_R : F_0 \to F$ is a discretization mapping, F_0 being a suitable subset of a vector space (such as a connected subset of $\Re$) and F the finite feature space.
3. For continuous-time systems some additional care is needed since arbitrarily fast transitions cannot be observed in practice. This places restrictions on the dynamics of the system and its features, but we do not concern ourselves with such structural questions here. In digital control, time is already discretized, but the introduction of feature-jump based clocks is still worth the effort since the complexity of the feature-level "dynamics" might be a great deal simpler than the original one.
4. This is a very important restriction: It reduces the problem complexity hugely, by a factor of $1/2^{2^n(1+o(1))}$, where n is the number of operating conditions.
5. One exception may be a controller whose purpose is just to maintain the "goal state".
6. Again, the goal state may be retained for an infinitely long period of time.
7. Note that as the original control problem is deterministic it is not immediate when the introduction of probabilities can be justified. One idea is to refer to the ergodicity of the control problem.
8. The Khepera was designed and built at Laboratory of Microcomputing, Swiss Federal Institute of Technology, Lausanne, Switzerland.
9. Modules are numbered by the identification number of their features.

10. All the feature functions of the modules could be implemented as a threshold sum of the measurements of one sensor modality. The thresholds were chosen in such a way that the task remained solvable, but no additional fine-tuning of these parameters was performed. Nevertheless, the values of these parameters could influence the resulting performance.
11. The theoretically-funded inverse logarithmic decrease was tried as well, but it was found to yield worse on-line performances than the faster geometric decrease, which, in this particular specific case, seemed to guarantee the sufficiency of exploration.
12. An exploration strategy is called *undirected* when the exploration does not depend on the number of visits to the state-action pairs.
13. Korf's definition would also require that only one action is chosen at each time step from the union of the disjoint action sets (Korf, 1987), but this is unnecessarily restricting in our case.

References

M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.

A.G. Barto, S. J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 1(72):81–138, 1995.

R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

A. Birk and J. Demiris. *Sixth European Workshop on Learning Robots.* Lecture Notes in Artificial Intelligence. Springer, Berlin, 1998.

R.I. Brafman and T. Moshe. Modeling agents as qualitative decision makers. *Artificial Intelligence*, 94(1):217–268, 1997.

M.S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control.* PhD thesis, Laboratory of Information and Decision, MIT, 77 Massachusetts Avenue, Cambridge, MA 02139-4307 USA, 1995.

M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framework for hybrid control: Background, model, and theory. Technical report lids-p-2239, Laboratory for Information and Decision Systems, MIT, 77 Massachusetts Avenue, Cambridge, MA 02139-4307 USA, 1994.

R.W. Brockett. Hybrid models for motion control systems. In *Essays in Control: Perspectives in the Theory and its Applications*, pages 29–53. Birkhäuser, Boston, 1993.

R. Brooks. Artificial life and real robots. In *Proceedings of the First European Conference on Artificial Life (ECAL)*, pages 3–10. MIT Press, 1991a.

R.A. Brooks. Elephants don't play chess. In *Designing Autonomous Agents.* Bradford-MIT Press, 1991b.

L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188, San Jose, CA, 1992. AAAI Press.

M. Colombetti, M.Dorigo, and G. Borghi. Behavior analysis and training: A methodology for behavior engineering. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(3):365–380, 1996.

J. de Kleer and B.J. Seely. A qualitative physics based on confluences. *Artificial Intelligence*, 24(1–3):7–83, 1984.

M. Dorigo. Alecsys and the autonomouse: Learning to control a real robot by distributed classifier systems. *Machine Learning*, 19(3):209–240, 1995.

M. Dorigo and M. Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71:321–370, 1994.

Z. Gábor, Zs. Kalmár, and Cs. Szepesvári. Multi-criteria reinforcement learning. Technical report 98-115, Research Group on Artificial Intelligence, JATE-MTA, 1998.

R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, 1993.

M. Heger. The loss from imperfect value functions in expectation-based and minimax-based tasks. *Machine Learning*, 22:197–225, 1996.

M.I. Henig. Vector-valued dynamic programming. *SIAM J. Control and Optimization*, 21(3):490–499, 1983.

T. Jaakkola, M.I. Jordan, and S.P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

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

Zs. Kalmár, Cs. Szepesvári, and A. Lőrincz. Generalization in an autonomous agent. In *Proc. of IEEE WCCI ICNN'94*, volume 3, pages 1815–1817, Orlando, Florida, 1994. IEEE Inc.

Zs. Kalmár, Cs. Szepesvári, and A. Lőrincz. Generalized dynamic concept model as a route to construct adaptive autonomous agents. *Neural Network World*, 5: 353–360, 1995.

Zs. Kalmár, Cs. Szepesvári, and A. Lőrincz. Module based reinforcement learning for a real robot. In *Proc. of the 6th European Workshop on Learning Robots*, pages 22–32, 1997.

S. Koenig and R.G. Simmons. Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains. *Machine Learning: A Special Issue on Reinforcement Learning*, 12:234–345, 1997.

R.E. Korf. *Learning to solve problems by searching for macro-operators.* Pitman Publisher, Massachusetts, 1985a.

R.E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985b.

R.E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33: 65–88, 1987.

J.R. Koza and J.P. Rice. Automatic programming of robots using genetic programming. In *Proceedings of Tenth National Conference on Artificial Intelligence*, pages 194–201, Menlo Park, CA, 1992. AAAI Press/The MIT Press.

P.R. Kumar. A survey of some results in stochastic adaptive controls. *SIAM Journal of Control and Optimization*, 23:329–380, 1985.

M.L. Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Department of Computer Science, Brown University, 1996. Also Technical Report CS-96-09.

M.L. Littman and Cs. Szepesvári. A Generalized Reinforcement Learning Model: Convergence and applications. In *Int. Conf. on Machine Learning*, pages 310–318, 1996.

J. Lygeros, D.N. Godbole, and S.S. Sastry. A design framework for hierarchical, hybrid control. *IEEE Transactions on Automatic Control, special issue on Hybrid Systems*, 1997. (submitted).

P. Maes. Adaptive action selection. In *Proc. of the Thirteenth Annual Conf. of the Cognitive Science Society.* Lawrence Erlbaum Associates, 1991a.

P. Maes. A bottom-up mechanism for behavior selection in an artificial creature. In J.A. Meyer and S. Wilson, editors, *Proc. of the First International Conference on Simulation of Adaptive Behavior.* MIT Press, 1991b.

P. Maes. Learning behavior networks from experience. In *Toward a Practice of Autonomous Systems (Proc. First European Conference on Artificial Life)*, pages 48–57. MIT Press, Cambridge, Massachusetts, 1992.

P. Maes and R.A. Brooks. Learning to coordinate behaviors. In *Proc. of AAAI-90*, pages 796–802, Boston, MA, 1990.

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

M. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4, 1997.

R. A. McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 190–196, Amherst, Massachusetts, 1993. Morgan Kaufmann.

R. Munos. Finite-element methods with local triangulation refinement for continuous reinforcement learning problems. In M.van Someren and G. Widmer, editors, *Machine Learning: ECML'97 (9th European Conf. on Machine Learning, Proceedings)*, volume 1224 of *Lecture Notes in Artificial Intelligence*, pages 170–183. Springer, Berlin, 1997.

A. Newell and H.A. Simon. *Human Problem Solving.* Prentice-Hall, Englewood Cliffs, NJ, 1972.

R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1997. MIT Press. in press.

Gy. Pólya. *How to solve it?* Princeton University Press, Princeton, NJ, 1945.

D. Precup, R.S. Sutton, and S.P. Singh. Planning with closed-loop macro actions. In *Working notes of the 1997 AAAI Fall Symposium on Model-directed Autonomous Systems.* AAAI Press/The MIT Press, 1997. in press.

S.M. Ross. *Applied Probability Models with Optimization Applications.* Holden Day, San Francisco, California, 1970.

E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

S. Sastry. Algorithms for design of hybrid systems. In *Proc. of Int. Conf. of Information Sciences*, 1997.

A.C.C. Say and K. Selahattin. Qualitative system identification: deriving structure from behavior. *Artificial Intelligence*, 83(1):75–141, 1996.

S. Singh and D. Cohn. How to dynamically merge markov decision processes. In *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1997. MIT Press. in press.

S. Singh, T. Jaakkola, M.L. Littman, and Cs. Szepesvári. On the convergence of single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 1997. accepted.

S.P. Singh. Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 202–207, San Jose, CA, 1992. AAAI Press.

S.P. Singh, T. Jaakkola, and M.I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *Proc. of the Eleventh Machine Learning Conference*, pages pp. 284–292, 1995.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, 8, 1996.

R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 1984.

Cs. Szepesvári. Dynamic Concept Model learns optimal policies. In *Proc. of IEEE WCCI ICNN'94*, volume 3, pages 1738–1742, Orlando, Florida, 1994. IEEE Inc.

Cs. Szepesvári. Learning and exploitation do not conflict under minimax optimality. In M.van Someren and G. Widmer, editors, *Machine Learning: ECML'97 (9th European Conf. on Machine Learning, Proceedings)*, volume 1224 of *Lecture Notes in Artificial Intelligence*, pages 242–249. Springer, Berlin, 1997a.

Cs. Szepesvári. *Static and Dynamic Aspects of Optimal Sequential Decision Making.* PhD thesis, Bolyai Institute of Mathematics, University of Szeged, Szeged, Aradi vrt. tere 1, HUNGARY, 6720, 1997b.

Cs. Szepesvári and M.L. Littman. Generalized Markov Decision Processes: Dynamic programming and reinforcement learning algorithms. *Neural Computation*, 1997. in preparation.

Cs. Szepesvári and A. Lőrincz. Behavior of an adaptive self-organizing autonomous agent working with cues and competing concepts. *Adaptive Behavior*, 2(2):131–160, 1994.

S.B. Thrun. *The role of exploration in learning control.* Van Nostrand Rheinhold, Florence KY, 1992.

Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. The MIT Press, Cambridge, 1995.

G.J. Tóth, Sz. Kovács, and A. Lőrincz. Genetic algorithm with alphabet optimization. *Biological Cybernetics*, 73:61–68, 1995.

J. N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 8(3–4):257–277, 1994.

J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

J.N. Tsitsiklis and Ben Van Roy. An analysis of temporal difference learning with function approximation. Technical Report LIDS-P-2322, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1995.

T. Tyrrell. *Computational Mechanisms for Action Selection.* PhD thesis, University of Edinburgh, 1993.

E. Uchibe, M. Asada, and K. Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robot and Sytems*, pages 1329–1336, 1996.

C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 3(8):279–292, 1992.

P. J. Werbös. Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook*, 22:25–38, 1977.

M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2), 1997.

J. Zabczyk. Optimal control by means of switching. *Studia Mathematica*, 65:161–171, 1973.