Multiscale Anticipatory Behavior by Hierarchical Reinforcement Learning

Matthias Rungger, Hao Ding, and Olaf Stursberg

Institute of Automatic Control Engineering
Technische Universität München
D-80290 Munich, Germany
{matthias.rungger,hao.ding,stursberg}@tum.de

Abstract. In order to establish autonomous behavior for technical systems, the well known trade-off between reactive control and deliberative planning has to be considered. Within this paper, we combine both principles by proposing a two-level hierarchical reinforcement learning scheme to enable the system to autonomously determine suitable solutions to new tasks. The approach is based on a behavior representation specified by hybrid automata, which combines continuous and discrete behavior, to predict (*anticipate*) the outcome of a sequence of actions. On the higher layer of the hierarchical scheme, the behavior is abstracted in the form of finite state automata, on which value function iteration is performed to obtain a goal leading sequence of subtasks. This sequence is realized on the lower layer by applying policy gradient-based reinforcement learning to the hybrid automaton model. The iteration between both layers leads to a consistent and goal-attaining behavior, as shown for a simple robot grasping task.

Keywords: Reinforcement learning, hierarchical model, hybrid automaton, behavioral programming, artificial intelligence, planning.

1 Introduction

A characteristic property of intelligent autonomous systems is the capability to determine goal-attaining behavior for tasks that are posed to the system for the first time. A crucial point in determining such behavior is to anticipate what the outcome of an own action is and how the environment reacts to the action, in order to be able to select the best choice. Several approaches for anticipatory behavior of learning systems have been developed in recent years and are described, e.g. in [5,6]. Reinforcement learning (RL) is one of the main approaches to establish anticipatory behavior [20,3]. RL uses an estimate of the outcome of future actions and selects the actions for which a reward is maximized. The estimate of the outcome is either based on the observation of past behavior (i.e. the system runs iteratively through similar evolutions and assesses which actions lead to an preferable outcome) or on model-based computation. The latter approach, which is chosen in this paper, allows the system to evaluate the effects of a large

G. Pezzulo et al. (Eds.): ABiALS 2008, LNAI 5499, pp. 301–320, 2009.

variety of actions, even those which are potentially harmful for the system or its environment (i.e the system should not encounter corresponding situations in reality). Depending on the type of model used within RL and the period of time over which future behavior is anticipated, one can distinguish between reactive and deliberative planning. Reactive planning (often referred to as bottom-up approach) considers the momentary situation and produces in response a single action or a sequence of actions over a short period of future time. Often this response is suitably determined based on a sophisticated model of the situation or on a learned knowledge-base. Deliberative planning, in contrast, usually does not only consider the momentary situation but, in addition, the future evolution up to the point at which a task is accomplished – this means often that a (longer) sequence of future actions has to be determined. The model must be appropriate for anticipating the outcome of this sequence, i.e. the anticipation typically has to cover a longer time horizon and, as an implication, the underlying model encodes behavior in a more abstract setting to enable real-time computation. For planning of animals or humans, it is natural to combine both types of planning and learning in a hierarchical setting, i.e. deliberative planning leads to a rough plan for accomplishing a task, and this plan is further refined into concrete behavior using repetitively reaction to a changing environment along the envisaged plan.

To employ this principle in technical systems, a number of approaches have been proposed in the last decades, as e.g. multi-modal control, also referred to as behavior-based robotics in [1]): a behavior based architectures consists of a reactive controller and deliberative planner. The reactive controller is designed as a basis behavior with direct access to sensor and actuator signals. The planner acts on the behavior modules and is responsible for the interconnection of the different behaviors, which may be executed in parallel. A crucial point within these approach is the behavior coordination [16], such that emergent behaviors, not designed by the programmer, may arise. In [12] the behavior-based approach is used to speed up RL.

Other relevant published approaches combining learning with hierarchical planning include the following: Parameterized nonlinear differential equations are used in [19] to define motion primitives, which can be concatenated to build complex behavior. Hierarchical reinforcement learning schemes, like MAXQ [7], Options [17], and HAMS [15] use Semi-Markov processes to define subtasks – this, however, in a rather rigid scheme since the exit states for the subtasks are defined in advance. A hierarchical reinforcement learning approach on continuous dynamics is described in [14], where a two-level hierarchy is introduced to speed up learning. The higher level algorithm identifies subgoals in predefined distances within the state space of the dynamic system, which are then used to guide the system faster into the desired goal state.

The method presented in this paper is distinct from previously published ones in the following respects: To represent behavior on two layers of a learning and planning hierarchy, we use two types of formal dynamic models: On the lower layer, behavior is formulated in terms of continuous dynamics (specified by ordi-

nary differential equations) which changes discretely if certain logical conditions become true or false. Hybrid automata, as introduced in [11], are suitable to express such combined continuous-discrete behavior which is either controlled by the discrete mode or in which controllers of the continuous dynamics imply a certain sequence of discrete modes, see. e.g. [4,10,8,13]. For the example of a robotic arm for transporting objects, the mode (or logical condition) may model that different trajectories need to be realized depending on whether an object is currently grasped or not. Starting from the hybrid automaton on the lower layer, a more abstract representation of the behavior is derived for the higher layer to allow deliberative planning. Finite state automata are chosen as the type of the model, where the states represent different subgoals, and the transitions encode the continuous evolution of the system in between two subgoals. The rewards assigned to the transitions on the higher layer are calculated on-line on the lower level. Based on the finite state automaton, value iteration [3] is used to find the sequence of transitions with highest reward. This sequence is refined on the lower layer by applying reinforcement learning to the continuous dynamics of the hybrid automaton for each mode which corresponds to a transition of the higher layer sequence. The result is a goal-attaining sequence of actions which are obtained as a sequence of continuous control trajectories. Compared to the approach proposed before by the authors in [18], we here combine reinforcement learning on two layers, and the subgoals represented on the higher layer are calculated on-line. Using this hierarchical scheme, two scale anticipatory behavior is enforced in the sense that model-based anticipation of the reward of future actions is the basis for a suitable (or even best) choice of actions.

The paper is organized as follows: Section 2 defines the hybrid automaton and the problem of computing the respective action sequence for a given task. The abstraction of the hybrid automaton to the subgoal representation by a finite state automaton, is described in Sec. 3. The learning algorithms on the two layers are introduced in Sec. 4. As the main result, the overall algorithm for combining the two layers is specified in Sec. 5. An illustrating example is introduced in Sec. 7, and Sec. 8 provides conclusions and an outlook on future work.

2 Model and Problem Formulation

2.1 Lower Layer Model: Hybrid Automaton

Hybrid automata, as the type of the model chosen for the lower layer of the learning hierarchy, enable the modeler to formulate distinct continuous behavior for different modes of operation. In a first modeling step, the set of possible modes is identified and a discrete state, referred to as *location*, is assigned to each mode. Next the possible transitions between pairs of locations are identified and are formally defined as the transition structure of the hybrid automaton. For each location, a set of differential equations is identified to suitably describe the change of the relevant continuous state variables over time. This change usually depends on continuous input variables and is expressed by first order differential

equations. Well-known principles of balancing energy, mass, or impulse lead for many systems straightforwardly such rigorous dynamic models; the identification based on measured is a possible alternative if the physical principles of the system to be modeled are not well understood. Finally, the transitions are made dependent on discrete inputs and on the continuous dynamics by specifying conditions for the continuous state variables under which a transition is enabled.

Formally a hybrid automaton HA can be defined as a tuple

$$HA = (Z, V, X, U, inv, \Theta, g, \boldsymbol{r}, \boldsymbol{f})$$

consisting of:

- $Z = \{z_1, \ldots, z_{n_z}\}$ as the finite set of discrete *locations* to represent the discrete modes of operation;
- $V = \{v_1, \ldots, v_{n_v}\}$ as the finite set of *discrete inputs*, triggering the transitions by specifying the follow-up location[1];
- the *continuous state* $\boldsymbol{x}$ defined on the continuous state space $X \subseteq \mathbb{R}^{n_x}$;
- the *continuous input* $\boldsymbol{u}$ defined on the continuous input space $U \subseteq \mathbb{R}^{n_u}$;
- $inv : Z \to 2^X$ represents the assignment of *invariants* to locations; these invariants, which are compact subsets of $\mathbb{R}^{n_x}$, represent the permitted values of $\boldsymbol{x}$ as long as HA is in the respective location z;
- the finite set of *discrete transitions* $\Theta \subseteq Z \times Z$;
- a mapping $g : \Theta \to 2^X$ which assigns the so-called *guard sets* to the transitions as the subset of continuous states $g((z_i, z_j)) \subseteq X$ for which a transition $(z_i, z_j) \in \Theta$ is enabled;
- the *reset function* $\boldsymbol{r} : \Theta \times X \to X$ which is evaluated when a transition occurs and which updates the continuous state upon execution of a transition;
- the *continuous state dynamics* $\boldsymbol{f} : Z \times X \times U \to \mathbb{R}^{n_x}$ defining for every location z the evolution of the continuous states over time by a set of ordinary differential equations $\dot{\boldsymbol{x}} = \boldsymbol{f}_z(\boldsymbol{x}, \boldsymbol{u}) := \boldsymbol{f}(z, \boldsymbol{x}, \boldsymbol{u})$.

For these syntactical elements, the evolution of the hybrid automaton can be written formally as follows: Let the ordered set of event times $T = \{t_0, t_1, t_2, \ldots\}$ contain the initial time t_0 and all points of time at which a discrete transition is taken. Let $\overline{z}(t)$ denote the piecewise constant trajectory of the discrete locations with $z_k := \overline{z}(t)$ for $t \in]t_k, t_{k+1}]$. Likewise, $\overline{v}(t)$ is the piecewise constant trajectory of discrete inputs, $\overline{\boldsymbol{u}}(t)$ the continuous input trajectory, and $\overline{\boldsymbol{x}}(t)$ the continuous state trajectory. Define $\boldsymbol{x}_k := \boldsymbol{x}(t_k)$ and $\boldsymbol{x}_k^+ := \boldsymbol{x}(t_k^+)$ with $\boldsymbol{x}(t^+)$ denoting the right hand limit of $\boldsymbol{x}$ at t.

An *admissible hybrid state trajectory* $(\overline{z}(t), \overline{\boldsymbol{x}}(t))$ resulting from a given *control trajectory* $(\overline{\boldsymbol{u}}(t), \overline{v}(t))$ is then obtained as follows: After initialization to $z_0 = z(t_0)$ and $\boldsymbol{x}_0^+ = \boldsymbol{x}(t_0)$, and assuming that no immediate transition occurs at t_0, the progress of HA between two event times t_k and t_{k+1} is given by:

[1] As transitions may occur non-deterministically when the guard sets overlap, the discrete input selects the desired transition.

– the continuous evolution $\overline{\boldsymbol{x}}(t), t \in]t_k, t_{k+1}]$ as existing unique solution of

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(z_k, \boldsymbol{x}(t), \boldsymbol{u}(t)),$$
$$\boldsymbol{x}(t_k) = \boldsymbol{x}_k^+$$

with $\overline{\boldsymbol{x}}(t) \in inv(z_k)\ \forall t \in]t_k, t_{k+1}]$;
– followed by a transition $(z_k, z_{k+1}) \in \Theta$ which is subject to the guard set according to $\boldsymbol{x}_{k+1} \in g((z_k, z_{k+1}))$ and triggered by $z_{k+1} = v(t_{k+1})$. The updated continuous state is then obtained from:

$$\boldsymbol{x}_{k+1}^+ := \boldsymbol{r}((z_k, z_{k+1}), \boldsymbol{x}_{k+1}) \in inv(z_{k+1}).$$

Along a hybrid state trajectory $(\overline{z}(t), \overline{\boldsymbol{x}}(t))$, the guard sets can be interpreted as a sequence of subgoals into which the continuous trajectory is driven within each location. When a guard set is reached, the system can change from one location to another. The combination of the continuous dynamics within an active location and the subgoal is understood as a *subtask* to be accomplished to realize the hybrid state trajectory. A subtask is solved by determining the part of the input trajectory $(\overline{\boldsymbol{u}}(t), \overline{v}(t))$ which refers to the particular location.

2.2 Control Synthesis

A task to be solved can be defined such that a system has to be driven from a current (or initial) state $\boldsymbol{x}_k$ into a given future (or final) state $\boldsymbol{x}_{k+p}$ (p events later) – obviously, accomplishing the task means to solve a sequence of subtasks. If solving the task is based on a behavior representation given by HA, a straightforward interpretation is that the model is used to *anticipate* the behavior of the system under the effect of a chosen input trajectory. We formalize the evolution from an initial state into a goal state by introducing the following sets:

– an initial hybrid state set (z_0, X_0) consisting of hybrid states build from one initial location $z_0 \in Z$ and possible continuous initial states $\boldsymbol{x}_0 \in X_0$ and $X_0 \subset inv(z_0)$,
– a final hybrid state set (z_F, X_F) in which any element is composed of one final discrete location $z_F \in Z$ and a possible continuous final state with $\boldsymbol{x}_F \in X_F$ and $X_F \subset inv(z_F)$.

The *control synthesis task* is to find an input trajectory $(\overline{\boldsymbol{u}}(t), \overline{v}(t))$ for HA such that:

– an admissible trajectory $(\overline{z}(t), \overline{\boldsymbol{x}}(t))$ results for any $\boldsymbol{x}_0 \in X_0$;
– the end state lies within the final set $z(t_e) = z_F$ and $\boldsymbol{x}(t_e) \in X_F$.

In general it is desired to find not only a feasible solution, but one which is optimal with respect to some performance criteria, e.g. the time to accomplish the transfer from the initial state to the goal state. To circumvent the difficulty to solve an optimal control problem for a hybrid automaton as the underlying dynamical

system, local performance criteria are introduced for each location. The continuous control $\overline{\boldsymbol{u}}(t)$ is calculated to maximize the given local performance criteria.

An example is illustrated in Fig. 1: It consist of two locations $\{z_1, z_2\}$ representing two different continuous dynamics. The invariants for each location, defining the state space of each continuous subsystem are illustrated by the gray shaded regions. The two continuous dynamics $\{\boldsymbol{f}_1, \boldsymbol{f}_2\}$, defined on $\mathbb{R}^2$, are restricted to the subsets $inv(z_1)$ and $inv(z_2)$ respectively. A trajectory starting within location $z(t_0) = z_1$ and $\boldsymbol{x}(0) \in X_0$, is depicted by the bold black line. When the trajectory reaches the guard set g_{12}, the discrete input is set to $z(t_1) = v(t_1) = z_2$. Thus, the discrete transition is taken and the reset function $\boldsymbol{r}$ is evaluated for the state $\boldsymbol{x}(t_1)$. The evolution of the trajectory starts again in the co-domain of the reset function, governed then by the new dynamics $\boldsymbol{f}_2$, and progresses until the final set X_F is reached. A corresponding technical example

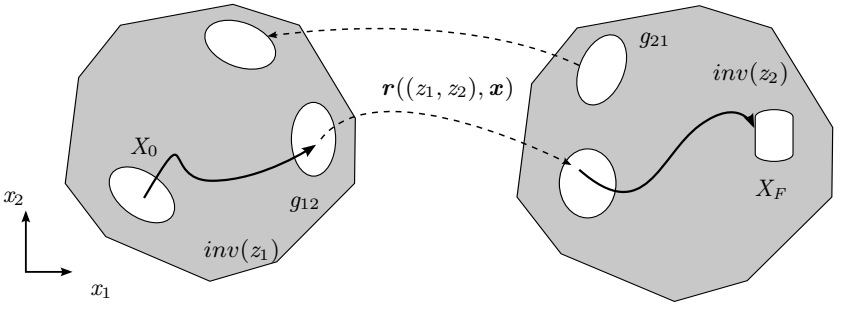


Fig. 1. An illustrating example of an hybrid automaton with two discrete locations z_1 and z_2

from the area of robotics is the transition from a locomotion task to a grasping task. The robot chassis dynamics is active while approaching the object to grasp. When the object is within reach, the grasping dynamics of the robot arm become active.

2.3 Higher Layer Model: Subgoal Automaton

Before introducing the algorithm for solving the control synthesis task, the subgoal automaton for modeling the system behavior in an abstract form on the higher layer is introduced. As mentioned in Sec. 1, the reason for using a second, less detailed behavior representation is that goal attainment by deliberative planning usually requires to cover longer time horizons – searching for control trajectories to solve the aforementioned control problem for HA and for long time horizon often turns out to be too complicated to be accomplished in real time. Thus, we here choose the approach to vertically decompose the problem by temporarily searching for a solution to a task on a simplified behavior representation. This model must still have enough structure to represent the behavior

on a qualitative scale as well as a sufficiently small state space to allow finding an action sequence quickly.

As mentioned above, the trajectory $(\overline{z}(t), \overline{\boldsymbol{x}}(t))$ of a hybrid automaton consists of the alternating sequence between the continuous evolution within one location and the discrete transition to change locations, which represent particular modes of operation or capabilities of the technical system. In this regard, the continuous evolution of the system can be considered as a subtask while the subgoal, associated with the subtask, is given in terms of the guard set. The continuous evolution within one location of the HA is represented by a transition of the *subgoal automaton* SGA, and a state of SGA corresponds to a guard set of HA for representing a particular subgoal. For example, the hybrid trajectory from the previously introduced example (see Fig. 1), is modeled by the sequence (s_0, s_{12}, s_F). For each guard set as well as for the initial set and the final set, a discrete state is introduced in SGA – the corresponding model is shown in Fig. 2.

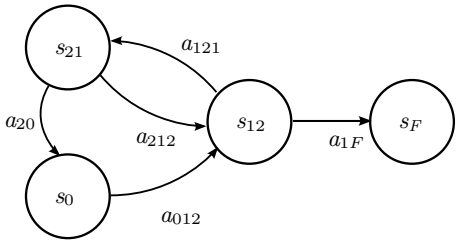


Fig. 2. SGA model for the example in Fig. 1

Formally, the automaton is defined by:

$$SGA = (S, A, h)$$

with:

- the finite set of discrete states $S = \{\ldots, s_{ij}, \ldots\} \cup \{s_0, s_F\}$ with one state s_{ij} for each guard set g_{ij} defined for HA, complemented by the initial state s_0 and the desired final state s_F;
- the set $A = \{\ldots, a_{ss'}, \ldots\}$ of actions $a_{ss'}$ which represent the hybrid evolution of all trajectories originating from the guard set g_s and leading to $g_{s'}$;
- the transition function is defined by $h(s, a_{ss'}) = s'$ for a transition from the state s to the state s' under effect of the action $a_{ss'}$ (introduced for any possible transition of HA). Additionally, transitions for the initial state s_0 and the final state s_F are defined, i.e. s_0 is connected to all states $s \in S$ representing a guard set contained in the invariant $g_s \in inv(z_0)$, and a similar construction is used for the final state s_F.

2.4 Example

The modeling procedure is illustrated for a simple example consisting of a 1-DOF robot arm which can move forward and backward. The task is to grab the ball at position p_0 and move it to p_1.

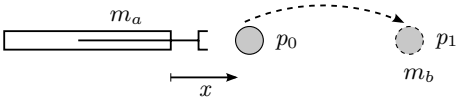


Fig. 3. Robot arm with the aim to move the ball to position p_1

The HA for the robot is modeled by two locations (z_1, z_2), where one represents free arm movement and one represents the movement of the arm with the ball. The dynamics within the locations are given by

$$z_1 : \ddot{x} = \frac{u}{m_a}, \qquad z_2 : \ddot{x} = \frac{u}{m_a + m_b},$$

and the guard sets by:

$$g_{12} = \{x \,|\, x = p_0\}, \quad g_{21} = \{x \,|\, x = p_1\}.$$

The grabbing/releasing of the ball is considered to be triggered by the discrete input signal $v \in \{z_1, z_2\}$.

The states of the corresponding abstract model SGA are given by s_0 (the initial state as shown in Fig. 3), s_{12} (representation of the guard set g_{12}), s_{21} (corresponding to the guard set g_{21}), and s_F : (the final state). A task for this system would be to design a controller which drives the robot arm from $\boldsymbol{x}_0$ to p_0, set $v(t) = z_2$ such that the ball is grabbed and the transition is taken. Then, the ball has to be moved to p_1 and the discrete input is reset to $v(t) = z_1$.

In general, the control synthesis task for the hybrid automaton is solved by: **a)** determining an appropriate sequence of locations, **b)** identifying the appropriate switching times/states for triggering the discrete transitions, and **c)** finding the continuous control laws within each location for driving the system to the identified states within the chosen guard sets.

3 Solution Algorithm

In the following, value iteration is used to find the sequence of locations on the higher layer and reinforcement learning is used to calculate the continuous control law on the lower layer. The reward signal on the higher layer, as the underlying driving force of the value iteration algorithm, is first initialized to a guess and then iteratively updated based on computation on the lower layer. The reward signal on both layers is designed such that a positive reward is assigned if a desired state is reached, and a negative reward if not.

3.1 Value Iteration

Given the SGA model on the higher layer, value iteration is applied to find an action sequence which leads the system from the initial state to the desired goal state.

For a policy $\pi : S \rightarrow A$, which provides for each state an appropriate action, the accumulated reward from the initial state s^0 to the goal state s^N is given by:

$$W^{\pi}(s^0) = \sum_{k=0}^{N-1} c(s^k, h(s^k, \pi(s^k))),$$

where $(s^0, s^1, \ldots, s^N)$ is the indexed state sequence of the resulting trajectory. The calculation of the rewards $c_{ss'} := c(s, s')$ associated with every action $a_{ss'}$ will be described in detail in Sec. 4. For realizing the goal oriented behavior, the policy (a state to action mapping) is derived such that the reward-to-come is maximized:

$$W(s^0) = \max_{\pi} \sum_{k=0}^{N-1} c(s^k, h(s^k, \pi(s^k))).$$

Applying the Bellman Principle, the maximal reward-to-come, referred to as the *value function*, is formulated recursively

$$W(s) = \max_{a}\{c(s, h(s, a)) + W(h(s, a))\}, \quad \forall s \in S. \tag{1}$$

A greedy policy is directly derived by maximizing the one step reward plus the expected/anticipated reward-to-come from the resulting state.

The subgoal automaton is a deterministic finite state automaton. Thus, value iteration (see [3]) with a look-up table representation is a viable solution method for the calculation of the value function.

After initialization of all values to zero, $W_0(s) = 0$ for all s, the value function is iteratively updated for all states by:

$$W_i(s) = \max_{a}\{c(s, h(s, a)) + W_{i-1}(h(s, a)\},$$

where i is the iteration index. Since no uncertainty of the outcome of actions need to be considered, the incremental update rule (normally included in value iteration schemes) is omitted. The pseudo code of the iterative procedure is listed in Alg. 1. The state sequence from the initial state to the final state is obtained by using the greedy policy.

3.2 Continuous Valued and Time Reinforcement Learning

In this section, the algorithm for the realization of the control commands on the lower layer is introduced. The objective is to implement a solution which is consistent to the sequence of discrete actions on the higher layer. The same

Algorithm 1. Value iteration

INITIALIZATION: $\forall s : W_0(s) = 0$, $i = 0$, $\delta = \infty$
VALUE ITERATION:
while $\delta > \Delta$ (a small threshold) **do**
  for all $s \in S$ **do**
    $W_{i+1}(s) := \max_{a \in A}\{c(s, h(s, a)) + W_i(h(s, a))\}$
    $\delta = \min(\delta, |W_{i+1}(s) - W_i(s)|)$
    $i := i + 1$
  end for
end while
STATE SEQUENCE:
return $(s^0, \ldots, s^n)$

reward principle as on the higher layer is used. The algorithm for the implementation of a continuous time, continuous-valued version of reinforcement learning was introduced in [9]. It is stated here in condensed form: Since each location is considered separately, the system dynamics is specified by:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)),$$

where the index for the location z is here omitted for simplicity of notation. The reward signal is denoted by $l(\boldsymbol{x}(t), \boldsymbol{u}(t))$ and control inputs are calculated to maximize the cumulative reward – this may model, e.g., the inverse time to reach the next subgoal or the negative quadratic distance to the subgoal.

Similarly as previously introduced for SGA, the reward-to-come for a policy $\boldsymbol{u}(t) = \mu(\boldsymbol{x}(t))$ is given by:

$$V^{\mu}(\boldsymbol{x}(t)) = \int_t^{t_F} e^{-\frac{s-t}{\tau}} l(\boldsymbol{x}(s), \boldsymbol{u}(s))\mathrm{d}s,$$

where t_F is the time at which the trajectory $\boldsymbol{x}(t)$ reaches the guard set, or the subgoal respectively. τ is the time constant for discounting future rewards. The optimal value function maximizing the cumulative future reward is given as:

$$V^*(\boldsymbol{x}(t)) = \max_{\overline{\boldsymbol{u}}(s)} \int_t^{t_F} e^{-\frac{s-t}{\tau}} l(\boldsymbol{x}(s), \boldsymbol{u}(s))\mathrm{d}s,$$

with $\boldsymbol{u}(s)$, $s \in [t, t_F]$. Applying the Bellman principle of optimality leads to a discounted version of the Hamilton-Jacobi-Bellman equation (see [9] for details):

$$\frac{1}{\tau} V^*(\boldsymbol{x}(t)) = \max_{\boldsymbol{u}}\{l(\boldsymbol{x}(t), \boldsymbol{u}(t)) + \frac{V^*(\boldsymbol{x}(t))}{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t))\}, \tag{2}$$

and the policy at a certain time is derived from the right-hand side to:

$$\boldsymbol{u} = \mu(\boldsymbol{x}) = \arg\max_{\boldsymbol{u}}\{l(\boldsymbol{x}, \boldsymbol{u}) + \frac{V^*(\boldsymbol{x})}{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})\} \tag{3}$$

Since a continuous time, continuous-valued dynamical system is considered (within one location), a look-up table representation of the value function is not appropriate. Instead a function approximation architecture is used to represent the value function:

$$V^{\mu}(\boldsymbol{x}(t)) \approx V(\boldsymbol{x}(t), \boldsymbol{w}).$$

Learning of the value function is performed in terms of updating the parameter $\boldsymbol{w}$ of the function approximation. The self-consistency condition, which follows from Eq. (2) to $\dot{V}^{\mu}(\boldsymbol{x}(t)) = \frac{1}{\tau}(V^{\mu}(\boldsymbol{x}(t)) - l(\boldsymbol{x}(t), \mu(\boldsymbol{x}(t)))$, is used to evaluate the current estimate $V(\boldsymbol{x}(t))$ of the value function. The weights of the approximation are adapted such that the error:

$$E(t) = \frac{1}{2}|\dot{V}(\boldsymbol{x}(t)) - V(\boldsymbol{x}(t)) + l(\boldsymbol{x}(t), \mu(t))|^2 \tag{4}$$

is minimized. As described in [9], a potential problem using $\dot{V}$ to update the weights is the symmetry in time. An approach to update the past estimates of $V(\boldsymbol{x}(t))$ without affecting the future estimates is to employ an Euler approximation $\dot{V}(\boldsymbol{x}(t)) = (V(\boldsymbol{x}(t)) - V(\boldsymbol{x}(t - \Delta t)))/\Delta t$. The gradient of the squared error (4) with respect to the parameter w_i results then in:

$$\frac{\partial E(t)}{\partial w_i} = \delta(t)\frac{1}{\Delta t}\left[\left(1 - \frac{\Delta t}{\tau}\right)\frac{\partial V(\boldsymbol{x}(t), \boldsymbol{w})}{\partial w_i} - \frac{\partial V(\boldsymbol{x}(t - \Delta t), \boldsymbol{w})}{\partial w_i}\right]$$

with:

$$\delta(t) = \frac{1}{\Delta t}\left[\left(1 - \frac{\Delta t}{\tau}\right) V(\boldsymbol{x}(t)) - V(\boldsymbol{x}(t - \Delta t))\right] + l(\boldsymbol{x}(t), \mu(\boldsymbol{x})) \tag{5}$$

as the *temporal difference* error. It coincides with the conventional TD error (see [20]). A gradient descent algorithm to search for the w_i, which minimizes the error, uses the rule:

$$\dot{w}_i = -\eta\delta(t)\left[\left(1 - \frac{\Delta t}{\tau}\right)\frac{\partial V(\boldsymbol{x}(t), \boldsymbol{w})}{\partial \boldsymbol{w}_i} - \frac{\partial V(\boldsymbol{x}(t - \Delta t), \boldsymbol{w})}{\partial w_i}\right] \tag{6}$$

with η as the learning rate. This update scheme corresponds to the residual-gradient algorithm (see [2]).

The control law: The proposed procedure is an on-line learning approach, thus the control law stated in Eq. (3) has to be solved in every simulation step. Depending on the complexity of the reward $l(\boldsymbol{x}, \boldsymbol{u})$ and the system dynamics $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$, the solution of this static optimization problem is in general difficult to obtain. A possible approach is to establish an *actor-critic architecture*: the feedback mapping $\mu : X \rightarrow U$ is approximated by a function approximator and is learned online. Under the assumptions that the reward function l is convex in $\boldsymbol{x}$ and the dynamics is linear with respect to $\boldsymbol{u}$, an analytic solution can be derived by differentiating Eq. (3), leading to:

$$0 = \frac{\partial l(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}} + \frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})^T}{\partial \boldsymbol{u}}\frac{\partial V(\boldsymbol{x})^T}{\partial \boldsymbol{x}}. \tag{7}$$

The solution with respect to $\boldsymbol{u}$ results in the control law. In [9], this is referred to the *value-gradient based policy*.

The algorithm which realizes the described steps is specified below as(Alg. 2. For each of a chosen number of N trials, the dynamics and parameter equation is numerically simulated for the time interval $[0\, P]$, where P is an estimated upper bound of time required to reach the corresponding subgoal.

The algorithm (Alg. 2) is embedded into the overall scheme according to Alg. 1 as the step to compute the rewards c, see the next section. A value function V_g for each guard set g is determined when the corresponding continuous evolution into g is requested as part of the action sequence computed on the higher layer.

Algorithm 2. Value-Gradient based value iteration within one location

PARAMETERS: N, P
INITIALIZATION: $\boldsymbol{w}(0) := 0$
PROGRESS:
while $j < N$ **do**
  SET: $\boldsymbol{x}(0) := \boldsymbol{x}_0$, $t := 0$
  while $t \leq P$ **do**
    Simulate $\boldsymbol{x}(t)$ and $\boldsymbol{w}(t)$ with
    $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t))$
    $\dot{w}_i(t) = -\eta\delta(t)\left[\left(1 - \frac{\Delta t}{\tau}\right)\frac{\partial V(\boldsymbol{x}(t),\boldsymbol{w})}{\partial w_i} - \frac{\partial V(\boldsymbol{x}(t-\Delta t),\boldsymbol{w})}{\partial w_i}\right]$
  end while
  $j := j + 1$
end while

4 The Hierarchical Learning Approach

Based on the algorithms 1 and 2, this section describes the overall procedure of hierarchical learning. A crucial point of this procedure is the calculation of the transition rewards for the algorithm 1. The rewards c_{ijk} represent the cumulative reward obtained along the trajectory of the system within one location. Additionally, for each state s of the SGA, a goal state $\boldsymbol{x}_s \in g_s$ for the corresponding subtask is assigned. It is used on the one hand to set the discrete control input v when the continuous state trajectory enters a subgoal, and on the other hand to guide the system within one location, i.e. to determine the low-level reward signal l.

4.1 Subtask Reward Calculation

A transition $a_{ss'}$ of the subgoal automaton SGA represents the transfer of the system from the entry into a location (e.g. by the preceding reset) into the next guard set of the hybrid automaton HA. The transition can be interpreted as the set of all possible trajectories connecting g_s with $g_{s'}$, i.e. the rewards $c_{ss'}$ in the SGA represent the cumulative reward arising from such a transfer.

For the calculation of the transition reward, the value function $V_{s'}$ of the goal location is used in combination with the continuous goal state $\boldsymbol{x}_s$ of the preceding location:

$$c_{ss'} = V_{s'}(\boldsymbol{r}_s(\boldsymbol{x}_s)). \tag{8}$$

As will be described below, the continuous subgoal state $\boldsymbol{x}_s$ is calculated online and may change over the iterations. In the case that the subtask cannot be achieved, i.e. the subgoal state is not reachable, a low reward value is assigned to the corresponding transition of SGA.

4.2 Subgoal State Calculation

Until now, the focus was on the calculation of the sequence of goal leading locations and the corresponding continuous control trajectories $\overline{\boldsymbol{u}}(t)$. To obtain the control for setting the discrete input v, and thus to trigger a particular transition, the *subgoal state* $\boldsymbol{x}_s \in g_s$ is examined further. The subgoal state of a subtask is restricted to lie within the particular guard set g_s which terminates the subtask.

Starting the continuous evolution $\boldsymbol{x}(0) \in inv(z_i)$ of the system within location z_i, the discrete input is set when the trajectory reaches the subgoal state

$$v = z_j \quad \text{if } \boldsymbol{x}(t) = \boldsymbol{x}_{s_{ij}} \in g_{s_{ij}}.$$

By determining the subgoal state within a guard set, the decision where and when to switch is determined autonomously.

The subgoal plays a crucial role for the decision where to change location. The subgoal state is the goal state for the current subtask. It is chosen to complete the subtask (inside the guard set) and to give an initialization for the next subtask. Thus, the subgoal state is calculated within the corresponding guard set and to maximize the reward-to-come for the subsequent subtask s':

$$\boldsymbol{x}_s = \arg\max_{\boldsymbol{x} \in g_s} V_{s'}(\boldsymbol{r}_s(\boldsymbol{x})). \tag{9}$$

Thus, the choice of $\boldsymbol{x}_s$ is based on an anticipation of the reward-to-come of the next subtask. If the optimization result is not unique, the subgoal state is picked randomly from the set of possible states.

4.3 Algorithm

The formulae Eq. (8) and Eq. (9) complete the set of components required to formulate the overall learning algorithm. After the initialization of the value function V_s for each state s of the SGA, the iteration is started. The first value iteration results in a shortest path sequence from the initial state s_0 to the final state s_F, since no value function on the lower layer is trained and no subgoal $\boldsymbol{x}_s$ is reached yet (equal rewards for all transitions). After the sequence of guard sets $(s^0, \ldots, s^n)$ is determined by Alg. 1, the following is carried out for each

transition $a_{ss'}$ referring to this sequence: first, the continuous subgoal state $\boldsymbol{x}'_s$ is calculated, and then Alg. 2 is executed to learn $V_{s'}$. If the system trajectory reaches the subgoal state $\boldsymbol{x}(t) = \boldsymbol{x}_{s'}$, the transition $a_{ss'}$ proposed by the higher layer is realized and the learning of V_s continues for the next location. If the subgoal state was found to be not reachable, the learning for the particular sequence $(s^0, \ldots, s^n)$ stops, the value function results in a low reward, and thus the corresponding transition of the SGA is avoided subsequently. Then, the value iteration on the higher layer resumes, this time with updated value functions V_s to adapt the transition rewards. In this manner, the higher layer value iteration is steered towards a state sequence for which the the guard sets on the lower layer are reachable, and accordingly the *control synthesis task* in Sec. 3.1 is solved. The algorithm, which is listed in Alg. 3, stops when the final state is reached. Of course, the iteration may be repeated to enhance the performance.

Algorithm 3. Algorithm for hierarchical reinforcement learning

```
INITIAL/FINAL STATE: x_{s_0} := x_0, x_{s_F} := x_F
INITIALIZATION: render SGA, w_s(0) := 0,
PROGRESS:
while ||x(t) − x_F|| < ε do
  for all a_{ss'} ∈ A do
    c_{ss'} := V_{s'}(r_s(x_s))
  end for
  determine (s^0, ..., s^n) by Alg. 1
  for i = 1 : n do
    x_{s^{i+1}} := arg max_{x ∈ g_{s^{i+1}}} V_{s^{i+2}}(r(x))
    use Alg. 2 with x_0 := x_{s^i}, x_G := x_{s^{i+1}}
    if x(t) ≠ x_{s^{i+1}} then
      break and resume with outer while loop
    end if
  end for
end while
```

5 Simulation Results

In this section, the procedure is illustrated by means of a 2-DOF robot arm. Similar to the previously introduced example, the task is specified as a transportation problem, in which the robot arm has to move to position p_0, grab the ball, move it to position p_1, and release it there (see Fig. 4). To illustrate the proposed approach, the behavior of the robot arm is fixed to separated linear and rotational motion. The resulting hybrid automaton consists of 4 locations, representing *linear motion without ball*, *linear motion with ball*, *rotational motion*

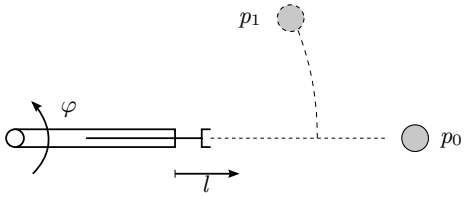


Fig. 4. The robot arm aims to move the ball from position p_0 to position p_1 by using its revolute and prismatic joints

without ball, and *rotational motion with ball*. The dynamics for the locations is given as:

$$z_1 : \dot{\boldsymbol{x}} = \begin{pmatrix} u_1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad z_2 : \dot{\boldsymbol{x}} = \begin{pmatrix} u_1 \\ 0 \\ u_1 \\ 0 \end{pmatrix}$$

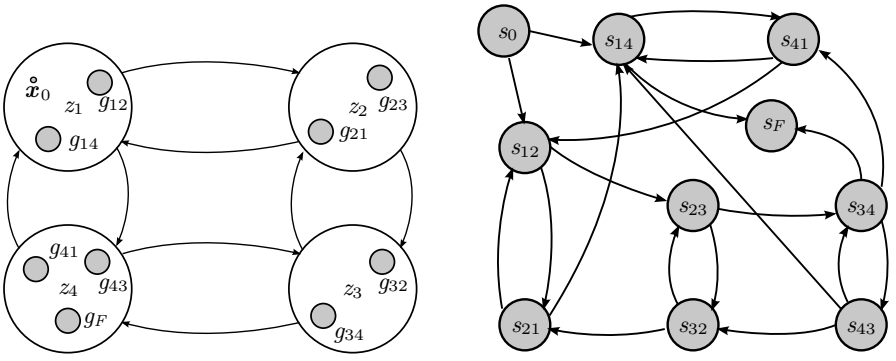
$$z_3 : \dot{\boldsymbol{x}} = \begin{pmatrix} 0 \\ u_2 \\ 0 \\ 0 \end{pmatrix} \qquad z_4 : \dot{\boldsymbol{x}} = \begin{pmatrix} 0 \\ u_2 \\ 0 \\ u_2 \end{pmatrix}$$

with $\boldsymbol{x} = (l, \varphi, l_B, \varphi_B)^T$, denoting the translational positions (l, l_B) and the angles (φ, φ_B) of the robot's end-effector, and the ball respectively (index $_B$). Neglecting the inertial forces, it is assumed that the translational control u_1 and the rotational control u_2 are commanded directly. The invariants of all locations are given by $inv(z) = [0.5,\, 1.2] \times [-\pi,\, \pi] \times [0.5,\, 1.2] \times [-\pi,\, \pi]$.

Fig. 5(a) displays the hybrid automation with the four locations and the corresponding guard sets and transitions. The transition from location z_1 to z_2, i.e. from linear motion without ball to linear motion with ball is bound to the guard set $g_{12} = \{\boldsymbol{x} \,|\, |x_1 - x_3| < 0.01\}$ indicating the state space, where the end-effector approaches the ball (grabbing the ball is neglected). Since it is everywhere allowed to release the ball, the guard set g_{21} for the reverse transition coincides with the invariant of the location. The guard sets corresponding to the transitions of the system from linear motion to rotational motion (z_1 to z_4 and z_2 to z_3) also coincide with the invariant, and such that it is always possible to take the transition. The reverse transition (from rotational to linear motion) is restricted to the part of the state space in which the angular displacement is zero: $g_{32} = g_{41} = \{\boldsymbol{x} \,|\, x_2 = 0\}$. The task initial state $\boldsymbol{x}_0 = (0.5\,0\,1.1\,0)^T \in inv(z_1)$ and final position $\boldsymbol{x}_F = (0.8\,0\,0.8\,\pi/3)^T \in inv(z_4)$ are marked in the figure.

The generated SGA is shown in Fig. 5(b). Its state set consists of one state each for representing the guard sets s_{ij}, the initial state s_0 and the final state s_F.

According to the given task, the algorithm 3 is able to determine the following solution, which intuitively is the correct one: The end-effector is first moved from $\boldsymbol{x}_0 \in inv(z_1)$ to the ball position $x_1 = x_B$, thus entering the guard set g_{12} and triggering the transition to z_2. Then the ball is moved to $x_1 = 0.8$ and



(a) The hybrid automation with its guard sets for the considered robot arm.

(b) The generated transition automaton.

Fig. 5. Dynamic models for the robot example

the transition to rotational motion occurs by which ϕ is changed to $x_2 = \pi/3$. The point ($\boldsymbol{x}_{s_{34}} = (0.8\,0\,0.8\,0)^T$) is determined iteratively by Eq. (9) within the algorithm. The ball is then released and the robot arm moved to $x_2 = 0$. The corresponding state sequence for the SGA is $(s_0, s_{12}, s_{23}, s_{34}, s_F)$.

To obtain this result, the continuous control on the lower layer is achieved by formulating the reward for guiding the end-effector to the desired subgoals $\boldsymbol{x}_s$ within the different locations as:

$$r(\boldsymbol{x}, \boldsymbol{u}) = -|\boldsymbol{x}(t) - \boldsymbol{x}_s|^2 - \int_0^{u_i} \nu \tan\left(\frac{\pi}{2}\frac{u}{u_i^{\max}}\right) \mathrm{d}u,$$

such that the control law using Eq. (7) results in:

$$\mu_z(\boldsymbol{x}) = \frac{2}{\pi} u^{\max} \arctan\left(\frac{1}{\nu}\frac{\partial \boldsymbol{f}_z(\boldsymbol{x}, \boldsymbol{u})^T}{\partial \boldsymbol{u}}\frac{\partial V(\boldsymbol{x}, \boldsymbol{w})^T}{\partial \boldsymbol{x}}\right).$$

The constant ν is chosen to 0.01, and the underlying approximating function consists of linearly weighted Gaussian bell-shaped functions, also known as radial basis functions (RBF).

For the iterative computation on the higher layer, the transition rewards for SGA are initialized with -0.1, and the weights of the RBF-network are initialized to 0. The first value iteration results in the shortest path sequence since all transitions are initialized with the same reward, i.e. the sequence is:

$$(s_0, s_{14}, s_F).$$

As a result, the guard set g_{14} on the lower layer is the subgoal in the first step. After the calculation of the particular state $\boldsymbol{x}_{s_{14}}$ in g_{14} by evaluating $\arg\max_{\boldsymbol{x}} V_{s_F}$, (see Eq. (9)) algorithm 2 is evoked with $N = 10$ trials for $P = 10\,\mathrm{sec}$. The trajectory within the last trial reaches the guard set g_{14}, thus the transition is

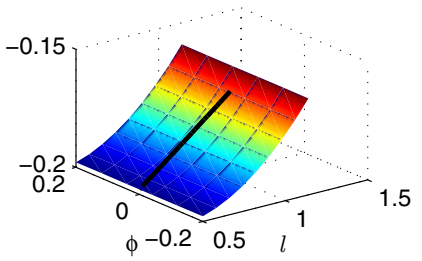
taken and the algorithm continues within location z_4. Alg. 2 is started again, now with $\boldsymbol{x}_{s_F} = (0.8\,0\,0.8\,\pi/3)$ targeting the final state. Since the last trajectory within the iteration does not reach the final state (the ball is still at the initial position), the iteration is interrupted, and the value iteration (Alg. 1) for SGA is started again. This time, the rewards for the transition (s_0, s_{14}) and (s_{14}, s_F) are updated by the values from $V_{g_{14}}$ and V_{g_F} trained in the previous iteration. While learning V_{g_F}, the goal is never reached, thus the reward diminished to -0.30 (see Tab. 1). Thereafter, the value iteration results in the desired sequence $(s_0, s_{12}, s_{23}, s_{34}, s_F)$, but it can not be realized on the lower level since the corresponding value functions for the low level are not yet trained well enough. Different sequences on the higher level are evaluated until again the desired sequence $(s_0, s_{12}, s_{23}, s_{34}, s_F)$ is selected in the 5^{th} iteration, which eventually can be realized on the lower layer.

Table 1. Value function for SGA. Next to each value the index within the state sequence computed for SGA is listed.

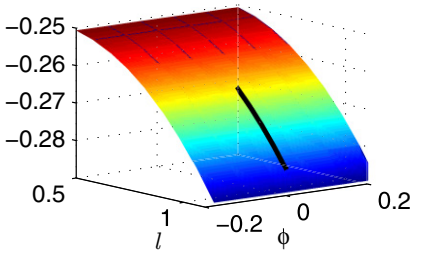
	W^0		W^1		W^2		W^4		W^5	
s_0	-0.20	1	-0.40	1	-2.48	1	-2.50	1	-3.16	1
s_{12}	-0.30	–	-0.30	2	-0.50	2	-1.64	–	-1.64	2
s_{23}	-0.20	–	-0.20	3	-0.20	–	-0.20	–	-0.20	3
s_{34}	-0.10	–	-0.10	4	-0.10	6	-0.10	4	-0.10	4
s_{41}	-0.20	–	-0.40	–	-0.40	–	-0.40	–	-1.74	–
s_{14}	-0.10	2	-0.30	–	-0.30	4	-0.30	2	-1.84	–
s_{21}	-0.20	–	-0.40	–	-0.40	3	-0.40	–	-1.74	–
s_{32}	-0.30	–	-0.30	–	-0.30	–	-0.30	–	-0.30	–
s_{43}	-0.20	–	-0.20	–	-0.20	5	-0.20	3	-0.20	–
s_F	0	3	0	5	0	7	0	5	0	5

The value functions $V_{g_{12}}, V_{g_{23}}, V_{g_{34}}, V_{g_F}$ used for the calculation of the continuous control law for the final sequence are plotted in Fig. 6(a-d). The black solid lines show the trajectory of the end-effector within the last iteration. The value functions have their maxima where the corresponding highest rewards are observed. For example, $V_{s_{12}}$ is the value function driving the system from the linear motion without ball to linear motion with ball. The transition occurs when the end effector reaches the ball position at $x_1 = 1.1$. It can be seen that $V_{g_{12}}$ has its maximum at this value, and thus the end effector is driven to the ball position.

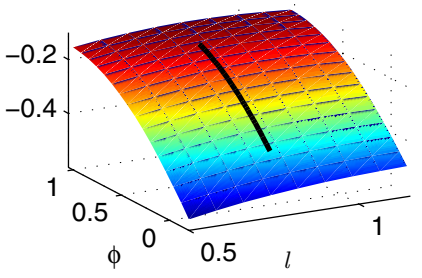
The subgoal state $\boldsymbol{x}_{s_{23}} \in [0.5\,1.2] \times 0 \times [0.5\,1.2] \times 0$ triggering the transition from linear motion to rotational motion with ball is determined by evaluating $\arg\max_{\boldsymbol{x}} V_{s_{34}}$. The guard set g_{34} represents the transition from the rotational motion with ball to the rotational motion without ball. It is triggered at $\boldsymbol{x}_{s_{34}} = (0.8\,\pi/3\,0.8\,\pi/3)^T$. The value function $V_{s_{34}}$ is plotted over the guard set g_{23} in Fig. 6(e). It can be observed that the maximum is at $x_1 = 0.8$, hence $\boldsymbol{x}_{s_{23}} = (0.8\,0\,0.8\,0)^T$. The trajectory of the end-effector for completing the task is plotted over time in Fig. 6(f).



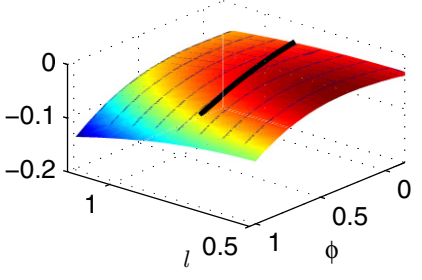
(a) Value function $V_{g_{12}}(\boldsymbol{x})$



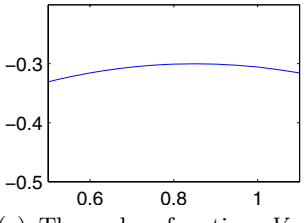
(b) Value function $V_{g_{23}}(\boldsymbol{x})$



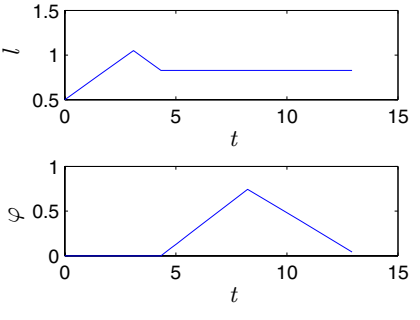
(c) Value function $V_{g_{34}}(\boldsymbol{x})$



(d) Value function $V_{g_F}(\boldsymbol{x})$



(e) The value function $V_{s_{34}}$ evaluated over g_{23}.



(f) Goal leading trajectory of the end-effector over time.

Fig. 6. Numerical results for the hierarchical reinforcement learning algorithm

6 Summary and Conclusion

A hierarchical algorithm is proposed by which a technical system can establish autonomous goal-attaining behavior for new tasks. To select between (sequences of) possible actions to accomplish the task, model-based anticipation of the outcome of actions is used. The starting point, a hybrid automaton model, represents the different capabilities of the system, but is often to complex for finding control trajectories that solve the given task. Thus, the suggestion is to generate the subgoal automaton (SGA), for which value iteration leads to a

coarse and potentially goal-attaining sequence of subtasks. The rewards of the transitions of SGA are updated iteratively from the lower level execution. The vertical decomposition of the task solution together with the model-based anticipation contribute to finding plans an control actions for rather complicated task without the necessity of exploring the complete hybrid state space. On both layers of the hierarchy an anticipated estimate of the future reward outcome of possible action are computed – since the complete computation is model-based, the system does not need to experiences behavior which is not successful (or possibly harmful) in reality.

The introduced example demonstrates the viability of the proposed approach for task solving, when different dynamics need to be activated in sequential manner. The benefits of the approach is that the complicated tasks is split into a (deliberative) planning of abstract action sequences on the higher layer and the realization (reactive planning) on the low layer. Even if the solution is completely unclear to the system when the task is posed, the integrated solution scheme achieves to find a feasible solution after a relatively low number of iterations without exploring large parts of the state search space of the original problem (defined for HA). Thus, the hierarchical approach seems promising to render reinforcement learning applicable to relative complex problems, by enforcing motion constraints and defining simple basic motion primitives, as in the example where the robot arm is restricted to activate linear motion or rotational motion sequentially. It is a matter of current work to investigate in detail what complexity of tasks can be accounted for by the proposed approach.

Future work will focus on a formal convergence proof of the approach as well as on a reduction of the number of hand tuned parameters, like the duration and number of trials for the continuous time reinforcement learning.

References

1. Arkin, R.C.: An Behavior-based Robotics. MIT Press, Cambridge (1998)
2. Baird, L.: Residual algorithms: Reinforcement learning with function approximation. In: Proceedings of the Twelfth International Conference on Machine Learning, pp. 30–37 (1995)
3. Bertsekas, D.P., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific, Belmont (1996)
4. Branicky, M.S.: Behavioral Programming. In: Working Notes AAAI Spring Symp. on Hybrid Systems and AI (1999)
5. Butz, M.V., Sigaud, O., Gérard, P.: Anticipatory Behavior: Exploiting Knowledge About the Future to Improve Current Behavior. In: Butz, M.V., Sigaud, O., Gérard, P. (eds.) Anticipatory Behavior in Adaptive Learning Systems. LNCS, vol. 2684, pp. 1–10. Springer, Heidelberg (2003)
6. Butz, M.V., Sigaud, O., Gérard, P.: Internal Models and Anticipations in Adaptive Learning Systems. In: Butz, M.V., Sigaud, O., Gérard, P. (eds.) Anticipatory Behavior in Adaptive Learning Systems. LNCS, vol. 2684, pp. 86–109. Springer, Heidelberg (2003)
7. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Artificial Intelligence Research 13, 227–303 (2000)

8. Ding, H., Rungger, M., Stursberg, O.: Intelligent Planning of Manufacturing Systems with Hybrid Dynamics. In: IFAC Conf. on Manufacturing Modeling, Management, and Control, pp. 181–186 (2007)
9. Doya, K.: Reinforcement learning in continuous time and space. Neural Comput. 12(1), 219–245 (2000)
10. Egerstedt, M.: Behavior Based Robotics Using Hybrid Automata. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 103–116. Springer, Heidelberg (2000)
11. Henzinger, T.: The Theory of Hybrid Automata. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996), pp. 278–292 (1996)
12. Mataric, M.J.: Reward functions for accelerated learning. In: Proc. of the 11th Int. Conf. on Machine Learning, pp. 181–189. Morgan Kaufmann, San Francisco (1994)
13. Tejas, R.: Mehta and Magnus Egerstedt. Multi-modal control using adaptive motion description languages. Automatica 44, 1912–1917 (2008)
14. Morimoto, J., Doya, K.: Acquisition of stand-up behavior by a real robot using hierarchical RL. Robotics and Autonomous Systems 36(1), 37–51 (2001)
15. Parr, R., Russell, S.: Russell Reinforcement learning with hierarchies of machines. In: Advances in Neural Information Processing Systems, vol. 10, pp. 1043–1049. The MIT Press, Cambridge (1997)
16. Pirjanian, P.: Multiple objective behavior-based control 31, 53–60 (2000)
17. Precup, D., Sutton, R.S., Singh, S.P.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112(1-2), 181–211 (1999)
18. Rungger, M., Stursberg, O., Spanfelner, B., Leuxner, C., Sitou, W.: Efficient Planning of Autonomous Robots using Hierarchical Composition. In: 5th Int. Conf. on Informatics, Control, Automation, Robotics, pp. 262–267 (2008)
19. Mohajerian, P., Schaal, S., Ijspeert, A.: Dynamics Systems vs. Optimal Control – A Unifying View, ch. 27, pp. 425–445. Elsevier, Amsterdam (2007)
20. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)