ELSEVIER

Robotics and Autonomous Systems 36 (2001) 37–51

Robotics and Autonomous Systems

www.elsevier.com/locate/robot

Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning

Jun Morimoto [a,b,*], Kenji Doya [b,c]

[a] *Kawato Dynamic Brain Project, ERATO, JST, 2-2-2 Hikaridai Seika-cho, Soraku-gun, Kyoto 619-0288, Japan*
[b] *Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5 Takayama-cho, Ikoma-shi, Nara 630-0101, Japan*
[c] *ATR International, CREST, JST, 2-2-2 Hikaridai Seika-cho, Soraku-gun, Kyoto 619-0288, Japan*

Received 6 March 2000; received in revised form 2 February 2001; accepted 3 February 2001

Abstract

In this paper, we propose a hierarchical reinforcement learning architecture that realizes practical learning speed in real hardware control tasks. In order to enable learning in a practical number of trials, we introduce a low-dimensional representation of the state of the robot for higher-level planning. The upper level learns a discrete sequence of sub-goals in a low-dimensional state space for achieving the main goal of the task. The lower-level modules learn local trajectories in the original high-dimensional state space to achieve the sub-goal specified by the upper level.

We applied the hierarchical architecture to a three-link, two-joint robot for the task of learning to stand up by trial and error. The upper-level learning was implemented by Q-learning, while the lower-level learning was implemented by a continuous actor–critic method. The robot successfully learned to stand up within 750 trials in simulation and then in an additional 170 trials using real hardware. The effects of the setting of the search steps in the upper level and the use of a supplementary reward for achieving sub-goals are also tested in simulation.

Keywords: Reinforcement learning; Hierarchical; Real robot; Stand-up; Motor control

1. Introduction

Recently, there have been many attempts to apply reinforcement learning (RL) algorithms to the acquisition of goal-directed behaviors in autonomous robots. However, a crucial issue in applying RL to real-world robot control is the curse of dimensionality. For example, control of a humanoid robot easily involves a forty- or higher-dimensional state space. Thus, the usual way of quantizing the state space with grids easily breaks down. We have recently developed RL algorithms for dealing with continuous-time, continuous-state control tasks without explicit quantization of state and time [6]. However, there is still a need to develop methods for high-dimensional function approximation and for global exploration. The speed of learning is crucial in applying RL to real hardware control because, unlike in idealized simulations, such non-stationary effects as sensor drift and mechanical aging are not negligible and learning has to be quick enough to keep track of such changes in the environment.

In this paper, we propose a hierarchical RL architecture that realizes a practical learning speed in high-dimensional control tasks. Hierarchical RL methods have been developed for creating reusable behavioral modules [4,21,25], solving partially observable

* Corresponding author. Fax: +81-774-95-3001.
E-mail address: xmorimo@erato.atr.co.jp (J. Morimoto).

0921-8890/01/$ – see front matter
PII: S0921-8890(01)00113-0

Markov decision problems (POMDPs) [26], and for improving learning speed [3,10].

Many hierarchical RL methods use coarse and fine grain quantization of the state space. However, in a high-dimensional state space, even the coarsest quantization into two bins in each dimension would create a prohibitive number of states. Thus, in designing a hierarchical RL architecture in high-dimensional space, it is essential to reduce the dimensions of the state space [16].

In this study, we propose a hierarchical RL architecture in which the upper-level learner globally explores sequences of sub-goals in a low-dimensional state space, while the lower-level learners optimize local trajectories in the high-dimensional state space.

As a concrete example, we consider a "stand-up" task for a two-joint, three-link robot (see Fig. 1). The goal of the task is to find a path in a high-dimensional state space that links a lying state to an upright state under the constraints of the system dynamics. The robot is a non-holonomic system, as there is no actuator linking the robot to the ground, and thus trajectory planning is non-trivial. The geometry of the robot is such that there is no static solution; the robot has to stand up dynamically by utilizing the momentum of its body.

This paper is organized as follows. In Section 2, we explain the proposed hierarchical RL method. In Section 3, we show simulation results of the stand-up task using the proposed method and compare the performance with non-hierarchical RL. In Section 4, we describe our real robot and system configuration and show results of the stand-up task with a real robot using the proposed method. In Section 5, we discuss the difference between our method and previous methods in terms of hierarchical RL, RL using real robots, and the stand-up task. Finally, we conclude this paper in Section 6.

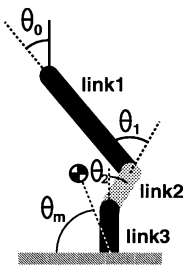

Fig. 1. Robot configuration. θ_0: pitch angle, θ_1: hip joint angle, θ_2: knee joint angle, θ_m: the angle of the line from the center of mass to the center of the foot.

2. Hierarchical reinforcement learning

In this section, we propose a hierarchical RL architecture for non-linear control problems. The basic idea is to decompose a non-linear problem in a high-dimensional state space into two levels: a non-linear problem in a lower-dimensional space and nearly-linear problems in the high-dimensional space (see Fig. 2).

2.1. Task decomposition by sub-goals

In the upper level, the learner deals with the entire task. The reward for the upper-level learner is given by the achievement of the entire task. In the lower level, each learner deals with a sub-task. The reward for the lower-level learner is given by the achievement of a given sub-goal. An action of the upper-level learner is the selection of the next sub-goal for the lower level. An action of the lower-level learner is the command for the actuators. The upper-level learner is activated when the lower-level learner achieves the current sub-goal. Then, the upper-level learner takes a new action, which is given as a new sub-goal for the lower-level learner. The state variables in the lower

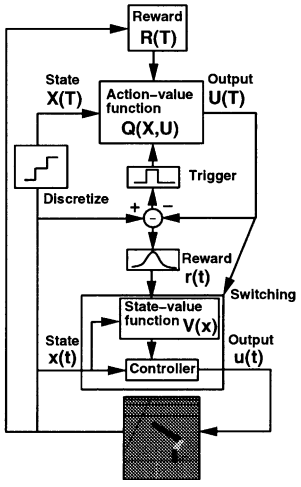

Fig. 2. Hierarchical reinforcement learning architecture.

Table 1
Task decomposition by sub-goals

Level	State	Action	Reward
Upper	Abstract Low dimension	Setting sub-goals	Task achievement
Lower	Physical High dimension	Actuator commands	Sub-goal achievement

level are the physical variables, while those in the upper level are lower-dimensional state variables (see Table 1). The choice of low-dimensional state variables is an important issue in hierarchical RL. In general, the use of task-oriented kinematic variables, such as the positions of the end effector and the center of the body mass, in the upper level would be appropriate. In the stand-up task, we chose the angles of the joints and the center of mass as the state variables. In other words, we chose kinematic variables in the upper level and dynamic variables in the lower level as the input.

2.2. Upper-level learning

In the upper level, the learner explores the entire relevant area of a low-dimensional sub-space of the original high-dimensional state space. In order to facilitate global search, the state space is coarsely discretized and the actions are defined as transitions to nearby states. We then use the $Q(\lambda)$-learning method [20] to learn a sub-goal sequence to achieve the goal of the entire task. Thus, a reward $R(T)$ to the upper level is given depending on the success or failure of the entire task.

In the upper level, the action-value function $Q(\boldsymbol{X}(T), \boldsymbol{U}(T))$ predicts the accumulated future reward if the learner takes the action $\boldsymbol{U}(T)$ at the state $\boldsymbol{X}(T)$. In $Q(\lambda)$-learning, the action-value function is updated in two steps. First, all of the state-action pairs are updated by

$$\delta(T) = R(T) + \gamma V(\boldsymbol{X}(T+1)) - V(\boldsymbol{X}(T)), \tag{1}$$

$$e(\boldsymbol{X}, \boldsymbol{U}) \leftarrow \gamma\lambda e(\boldsymbol{X}, \boldsymbol{U}), \tag{2}$$

$$Q(\boldsymbol{X}, \boldsymbol{U}) \leftarrow Q(\boldsymbol{X}, \boldsymbol{U}) + \alpha_Q e(\boldsymbol{X}, \boldsymbol{U})\delta(T), \tag{3}$$

where $V(\boldsymbol{X}(T)) = \max_{\boldsymbol{U}} Q(\boldsymbol{X}(T), \boldsymbol{U}(T))$ is the state-value function and $\delta(T)$ is its prediction error, $\gamma = 0.5$ is the discount factor of the action-value function, $e(\boldsymbol{X}, \boldsymbol{U})$ is the eligibility trace, $\lambda = 0.9$ is the decay rate of the eligibility trace, and $\alpha_Q = 0.1$ is the learning rate. Then, the value function for the current state-action pair is updated by

$$\delta'(T) = R(T) + \gamma V(\boldsymbol{X}(T+1)) - Q(\boldsymbol{X}(T), U(T)), \tag{4}$$

$$Q(\boldsymbol{X}(T), \boldsymbol{U}(T)) \leftarrow Q(\boldsymbol{X}(T), \boldsymbol{U}(T)) + \alpha_Q \delta'(T), \tag{5}$$

$$e(\boldsymbol{X}(T), \boldsymbol{U}(T)) \leftarrow e(\boldsymbol{X}(T), \boldsymbol{U}(T)) + 1, \tag{6}$$

where $\delta'(T)$ is the action-value prediction error.

In the stand-up task, we chose the posture of the robot $\boldsymbol{X} = (\theta_m, \theta_1, \theta_2)$ as the state variables (see Fig. 1). The desired posture of the robot $\boldsymbol{U}(T) = \boldsymbol{X}(T-1) + \Delta\boldsymbol{X}$ is given as an action, which is sent to the lower level as the next sub-goal. The upper-level learner chooses an action using Boltzmman distribution [22]. Thus, we have

$$P(\boldsymbol{U}(T) = a) = \frac{\exp[\beta Q(\boldsymbol{X}(T), a)]}{\sum_{b \in \mathcal{A}(\boldsymbol{X})} \exp[\beta Q(\boldsymbol{X}(T), b)]}, \tag{7}$$

where $\mathcal{A}(\boldsymbol{X})$ is the set of possible actions at state $\boldsymbol{X}$ and β is a parameter that controls the randomness in the action selection for exploration. We define the reward for the upper-level learner as follows.

$$R(T) = R_{\text{main}} + R_{\text{sub}}, \tag{8}$$

$$R_{\text{main}} = \begin{cases} 1, & \text{on success of stand-up,} \\ 0, & \text{on failure,} \end{cases} \tag{9}$$

$$R_{\text{sub}} = \begin{cases} 1, & \text{final goal achieved,} \\ 0.25\left(\dfrac{Y}{L} + 1\right), & \text{sub-goal achieved,} \\ 0, & \text{on failure,} \end{cases} \tag{10}$$

where Y is the height of the head of the robot at a sub-goal posture and L is total length of the robot. The final goal is the upright stand-up posture. When the robot achieves a sub-goal, the upper-level learner gets a reward of less than 0.5. Note that reaching the final goal is a necessary but not sufficient condition of successful stand-up because the robot may fall down after passing through the final goal.

When the robot reaches the neighborhood of a sub-goal, the next sub-goal is selected and the action-value function is updated in the upper level.

We consider that the stand-up task is accomplished when the robot stands up and stays upright for more than $2(T + 1)$ seconds. Otherwise (e.g. if the robot falls down, or if a time limit has been reached before the robot successfully stands up), we determine that the robot has failed to stand up.

2.3. Lower-level learning

In the lower level, the learner explores local areas of the high-dimensional state space without discretization. The lower-level learner learns to achieve the sub-goal specified by the upper level from any given initial state. Because each sub-goal is defined in the low-dimensional state space of the upper level, the sub-goal is not a point but a hyper-plane in the high-dimensional state space of the lower level. We use the continuous $TD(\lambda)$-learning with the actor–critic method [6] to learn the control command sequence. In addition, we use an INGnet to implement the actor and the critic [17] (see Appendix A). A reward $r(t)$ is given to the lower level by the achievement of the sub-goal specified by the upper level [3].

In continuous actor–critic learning of the lower-level learner, the critic learns the state-value function $V(\boldsymbol{x}(t))$ that predicts the accumulated future reward at state $\boldsymbol{x}(t)$, while the actor learns the control function $u_j(t) = u_{\max} h(f_j(\boldsymbol{x}(t)) + \sigma n_j(t))$ that specifies a non-linear feedback control law. Here, $h(x) = (\pi/2)\arctan((2/\pi)x)$ is a sigmoid function to saturate output with maximum torque $u_{\max}$, σ is a size of a noise term for exploration of the lower-level learner, and $n_j(t)$ is low-pass filtered noise $\tau_n \dot{n}_j(t) = -n_j(t) + N_j(t)$, where $N_j(t)$ denotes normal Gaussian noise and $\tau_n = 0.1$ [s] is a time constant for the low-pass filter. We use INGnets for the critic and the actor. The output of the critic is given by

$$V(\boldsymbol{x}(t)) = \sum_i v_i b_i(\boldsymbol{x}(t)), \tag{11}$$

where $b_i()$ is a basis function, and v_i is a network weight. The state-value prediction error $\delta(t)$ is calculated by

$$\delta(t) = r(t) - \frac{1}{\tau} V(\boldsymbol{x}(t)) + \frac{\mathrm{d}V(\boldsymbol{x}(t))}{\mathrm{d}t}, \tag{12}$$

where $\tau = 0.5$ [s] is the time constant of the state-value function. The update rule of the critic is

$$\dot{v}_i = \alpha_c \delta(t)\, e_i(t), \tag{13}$$

where $\alpha_c = 0.02$ is the learning rate, and e_i the eligibility trace of each basis function. The update rule of eligibility trace is

$$\dot{e}_i(t) = -\frac{1}{\kappa} e_i(t) + b_i(\boldsymbol{x}(t)), \tag{14}$$

where $\kappa = 0.1$ [s] is the time constant of eligibility.

The output of the actor is given by

$$f_j(\boldsymbol{x}(t)) = \sum_i w_{ij} b_i(\boldsymbol{x}(t)), \tag{15}$$

$$u_j(\boldsymbol{x}(t)) = u_{\max} h(f_j(\boldsymbol{x}(t)) + \sigma n_j(t)), \tag{16}$$

where $b_i()$ is the basis function, and w_{ij} is a network weight. The update rule of the actor is

$$\dot{w}_{ij} = \alpha_a \delta(t) \sigma n_j(t) b_i(\boldsymbol{x}(t)), \tag{17}$$

where $\alpha_a = 0.02$ is learning rate. The state-value prediction error $\delta(t)$ is used as an effective reward that signals the relative goodness of the current action $\boldsymbol{u}(t)$.

In a stand-up task, we chose the pitch and joint angles $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2)$ and the corresponding angular velocities $\dot{\boldsymbol{\theta}} = (\dot{\theta}_0, \dot{\theta}_1, \dot{\theta}_2)$ as the state variables $\boldsymbol{x}(t) = (\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$. We chose torque $\boldsymbol{u}(t) = (\tau_1, \tau_2)$ for the two joints as the action variables. The output torque is the sum of two controllers, a linear servo controller and a non-linear feedback controller $f_i(\boldsymbol{x})$, which is acquired by the lower-level actor:

$$\tau_j = u_{\max} h\left(\frac{1}{u_{\max}}(k(\hat{\theta}_j - \theta_j) - b\dot{\theta}_j) + f_j(\boldsymbol{x}) + \sigma n_j\right), \tag{18}$$

where $k = 0.26$ [N m/deg] and $b = 0.017$ [N m s/deg] are feedback gains, and we set the maximum torque as $u_{\max} = 24$ [N m].

In this study, we used different lower-level learners for different sub-goals. When the robot reaches the neighborhood of a sub-goal ($\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\| < 10$ [deg]), the upper-level learner switches the current lower-level learner module to the next one according to the choice of next sub-goals (see Fig. 2). Thus, one lower-level actor takes control until either the robot achieves the sub-goal, a time limit is reached, or the robot falls down. We used two types of reward for the lower level. One is given during the control according to the distance from the current posture $\boldsymbol{\theta}$ to the sub-goal posture $\hat{\boldsymbol{\theta}}$ ($= \boldsymbol{U}$) given by the upper level

$$r(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \exp\left(-\frac{\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|^2}{s_\theta^2}\right) - 1, \tag{19}$$

where $s_\theta = 30$ [deg] gives the width of the reward function. Additional reward is given at the end of the control by the distance from the current pitch and joint angular velocity $\dot{\boldsymbol{\theta}}$ to the desired values $\hat{\dot{\boldsymbol{\theta}}}$ that are set by the memory of successful trials

$$r(t) = \begin{cases} \exp\left(-\dfrac{\|\dot{\boldsymbol{\theta}}(t) - \hat{\dot{\boldsymbol{\theta}}}\|^2}{s_{\dot\theta}^2}\right), & \text{sub-goal achieved,} \\ -1.5, & \text{fall down,} \end{cases} \tag{20}$$

where $s_{\dot\theta} = 60$ [deg/s] gives the width of the reward function. If the time limit is reached, the lower-level learner is not updated at the end of control. The desired angular velocity $\hat{\dot{\boldsymbol{\theta}}}$ is initialized at the first successful stand-up as the angular velocity $\dot{\boldsymbol{\theta}}$ when the learner achieves the sub-goal area. It is then updated by $\hat{\dot{\boldsymbol{\theta}}} \leftarrow \eta\hat{\dot{\boldsymbol{\theta}}} + (1-\eta)\dot{\boldsymbol{\theta}}$ with $\eta = 0.9$ in subsequent successful trials. Note that we set reward $r(t) = 0$ in the upper part of (20) before the robot achieves the first stand-up.

3. Simulations

First, we show simulation results of the stand-up task with a two-joint, three-link robot using the hierarchical RL architecture. We then investigate the basic properties of the hierarchical architecture in a simplified stand-up task with one joint. We show how the performance changes with the action step size in the upper level. We also compare the performance between the hierarchical RL architectures and non-hierarchical RL architectures. Finally, we show the role of the upper-level reward R_{sub} for reaching a sub-goal.

3.1. Stand-up task using a two-joint, three-link robot

We tested the performance of the hierarchical RL architectures in the stand-up task by using the two-joint, three-link robot (see Fig. 1). We used a low-dimensional state $\boldsymbol{X} = (\theta_m, \theta_1, \theta_2)$, where $0 \leqslant \theta_m \leqslant 90$, $-150 \leqslant \theta_1 \leqslant 0$, $0 \leqslant \theta_2 \leqslant 25$ [deg] in the upper level and a high-dimensional state $\boldsymbol{x} = (\theta_0, \theta_1, \theta_2, \dot\theta_0, \dot\theta_1, \dot\theta_2)$, where $-150 \leqslant \theta_1 \leqslant 150$, $-150 \leqslant \theta_2 \leqslant 150$ [deg] in the lower level. We chose $\Delta\theta_m = 30$, $\Delta\theta_1 = 50$, $\Delta\theta_2 = 25$ [deg] as the action step $\Delta\boldsymbol{X}$ in the upper level. Each trial was started with the robot lying on the ground, $\boldsymbol{x} = (90, 0, 0, 0, 0, 0)$ [deg], and was continued for $t < 2(T+1)$ seconds in simulated time, where T is the discrete time in the upper level. When the robot fell down and hit its hip or head on the ground, the trial was terminated and restarted again. Each simulation was continued up to 1000 trials.

We set the exploration parameter as $\beta = 0.2M(T)$, where $M(T)$ is the number of trials lasting no fewer than T steps.

The size of the noise term was modulated as $\sigma = \sigma_s \min[1, \max[0, V_1 - V(t)]]$. $V(t)$ is the lower-level state value function and $V_1 = 0.5$ is a exploration parameter for the lower-level learner. The maximal noise level σ_s was also changed according to the sub-goal and the number of trials m as

$$\sigma_s = \begin{cases} \sigma_0, & \text{for final sub-goal,} \\ \sigma_1, & \text{otherwise if } m \leqslant m_1, \\ \dfrac{(m_2 - m)\sigma_1 + (m - m_1)\sigma_2}{m_2 - m_1}, & \\ & \text{if } m_1 < m < m_2, \\ \sigma_2, & \text{if } m \geqslant m_2. \end{cases} \tag{21}$$

The parameters were $m_1 = 300$ [trial], $m_2 = 600$ [trial], $\sigma_1 = 0.5$, $\sigma_2 = 0.1$, and $\sigma_0 = 0.01$.

The physical structure and the parameters of the robot are shown in Fig. 11 and Table 3. The physical system was simulated by a dynamic simulator made

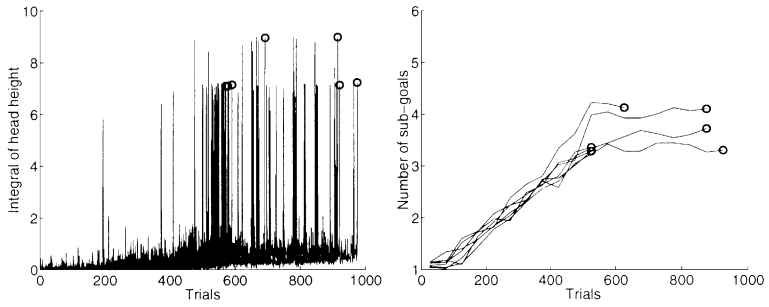

Fig. 3. Time course of learning. Circles show 10th successful stand-up, upon which a simulation run was terminated. (a) Performance index; (b) average number of sub-goals in each set of 50 trials.

by Boston Dynamics with a time step of 0.001 [s]. We used the number of trials made before achieving 10 successful trials as the measure of the learning speed.

The robot successfully learned to stand up in 7 out of 10 simulation runs. The average number of learning trials was 749, which took 30 minutes in simulated time (averaged over 7 successful runs). The upper-level learner used 4.3 sub-goals (averaged over 7 successful runs) for successful stand-up.

Fig. 3(a) shows the time course of learning. The vertical axis shows the performance index given by the integral of the head height $\int_0^{t_e} y(t)\,\mathrm{d}t$, where t_e is terminal time of the trial. Fig. 3(b) shows the number of sub-goals used in each trial. In the first stage of learning, the upper-level learner used only a few sub-goals, but after the middle stage of learning, the number of sub-goals increased because the lower-level learner learned to achieve sub-goals.

After about 300 trials, the upper-level learner learned appropriate sub-goals for the first and second steps, while the lower-level learner successfully learned to achieve each sub-goal, as shown in Fig. 4.

After about 750 trials, the upper-level learner learned appropriate sub-goals for successful stand-up, while the lower-level learner learned to achieve each sub-goal. The top images of Fig. 5 show an example

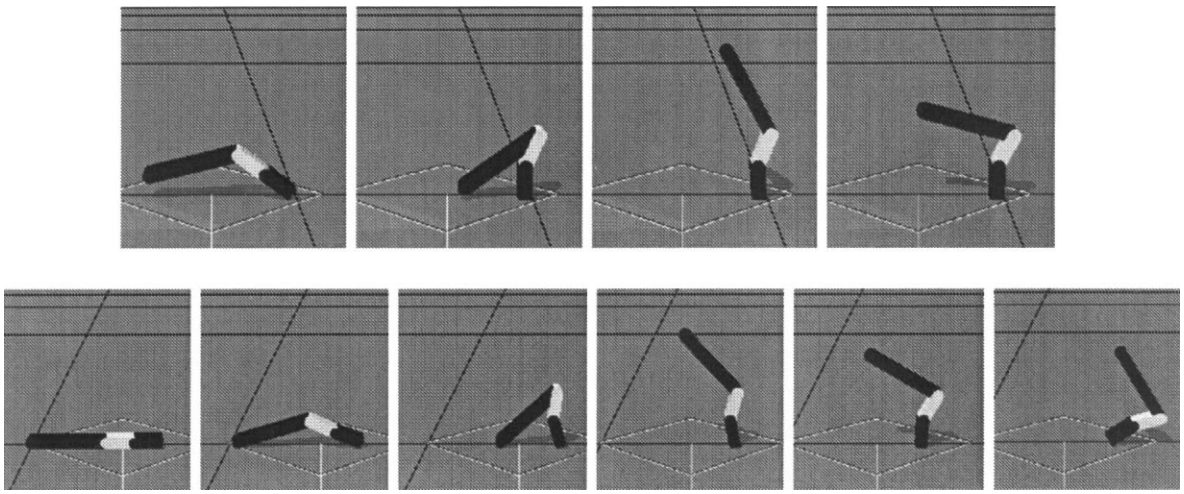

Fig. 4. Example of a sub-goal sequence after 300 trials (top); example of a failed stand-up trajectory after 300 trials (bottom).

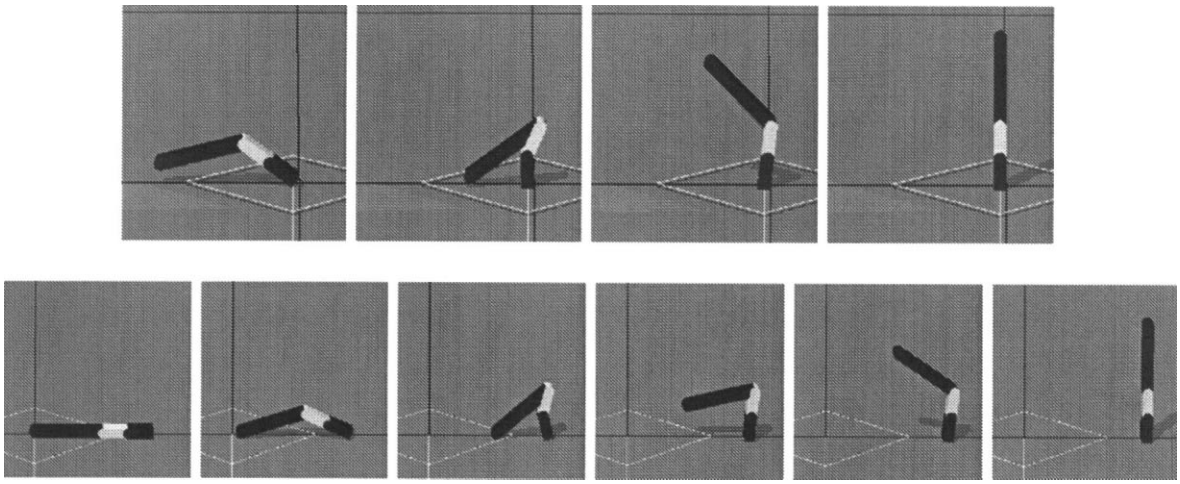

Fig. 5. Example of a successful sub-goal sequence (top); example of a successful stand-up trajectory (bottom).

of a sub-goal sequence acquired in the upper level. The bottom images of Fig. 5 show an example of a stand-up trajectory acquired in the lower level. Each learner successfully learned the appropriate action sequence for the stand-up task.

3.2. The effect of step size in the upper level

To investigate the effect of the action step size, we compared the upper-level learners with different $\Delta \boldsymbol{X}$. For simplicity, we fixed θ_2 to 0 [deg] by servo control (see Fig. 6) and chose action steps as $\Delta\theta_1 = 25, 30, 50$ [deg] and $\Delta\theta_m = 30$ [deg]. Thus, we chose $\boldsymbol{X} = (\theta_m, \theta_1)$ and $\boldsymbol{x} = (\theta_0, \theta_1, \dot{\theta}_0, \dot{\theta}_1)$ as state variables in the upper and the lower levels, respectively. Each simulation was continued up to 1000 trials. We used the same parameters as in Section 3.1 for each learning algorithm except for $\sigma_1 = 0.3$.

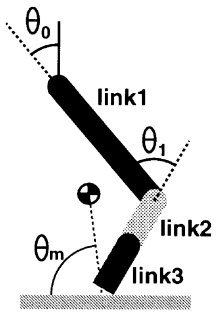

Fig. 6. One-joint, two-link robot configuration.

Fig. 7 and Table 2 show the results of the learning to stand-up with different $\Delta\theta_1$. The robot achieved good performance with $\Delta\theta_1 = 25$ and 30 [deg] but poor performance with $\Delta\theta_1 = 50$ [deg]. The upper level with smaller $\Delta\theta_1$ used more sub-goals for standing up. Fig. 8 shows examples of the stand-up trajectories and the sub-goal points in joint angle space with different $\Delta\theta_1$. The sub-goal locations with $\Delta\theta_1 = 20$ [deg] and $\Delta\theta_1 = 30$ [deg] were different, but both were good via points for generating stand-up trajectories. On the other hand, the sub-goal locations with $\Delta\theta_1 = 50$ [deg] lacked the important via point representing a maximum curvature of the stand-up trajectory (see Fig. 8). Without this via point, the lower-level learner had to learn a difficult sub-task and often failed to acquire a part of the stand-up trajectories.

Thus, we showed that the proposed hierarchical RL method was not so sensitive to the choice of

Table 2
Comparison with different $\Delta\theta_1$

$\Delta\theta_1$ [deg]	Success rate (%)	Average over successful trials		
		Trials	Time [min]	Sub-goals
25	90	408	19	6.3
30	100	375	16	4.5
50	20	463	16	4

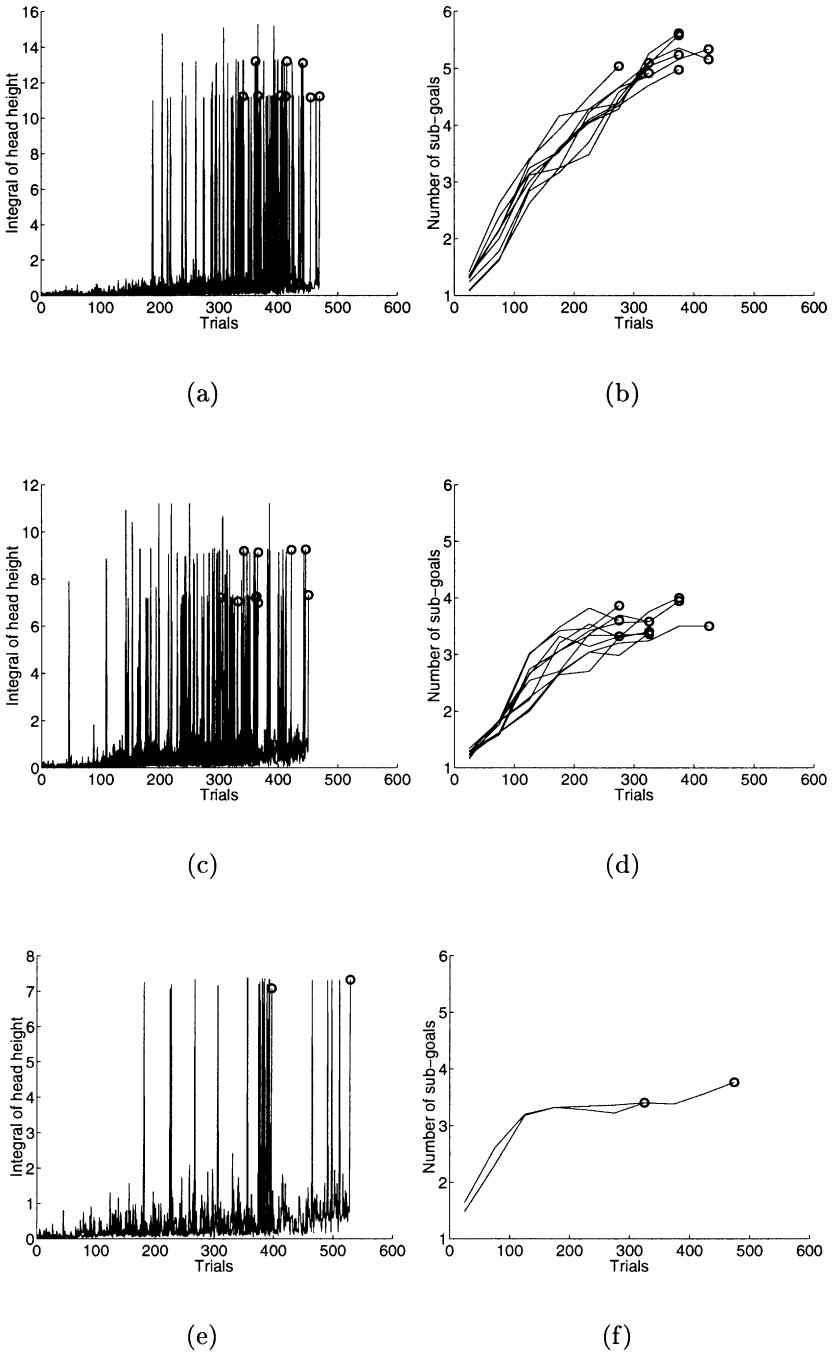

Fig. 7. Comparison of the time course of learning with different $\Delta\theta_1$. Circles show 10th successful stand-up, upon which a simulation run was terminated. (a) Performance index, $\Delta\theta_1 = 25$ [deg]; (b) average number of sub-goals in each set of 50 trials, $\Delta\theta_1 = 25$ [deg]; (c) performance index, $\Delta\theta_1 = 30$ [deg]; (d) average number of sub-goals in each set of 50 trials, $\Delta\theta_1 = 30$ [deg]; (e) performance index, $\Delta\theta_1 = 50$ [deg]; (f) average number of sub-goals in each set of 50 trials, $\Delta\theta_1 = 30$ [deg].

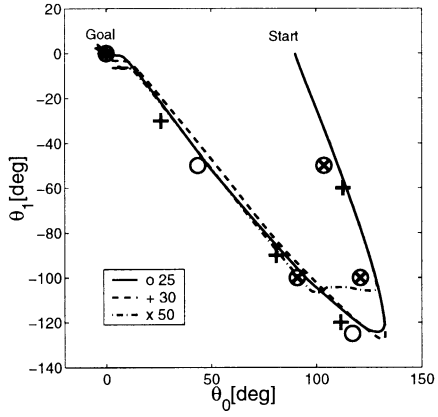

Fig. 8. Stand-up trajectories and sub-goals using different $\Delta\theta_1$.

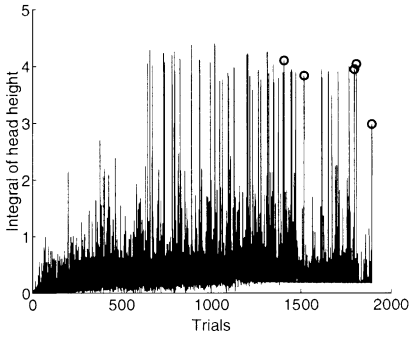

Fig. 9. Time course of learning with plain architecture. Circles show 10th successful stand-up, upon which a simulation run was terminated.

$\Delta \boldsymbol{X}$ but has a certain range of $\Delta \boldsymbol{X}$ in which the upper-level learner successfully acquired the appropriate sub-goals for stand-up.

3.3. Comparison between hierarchical and plain architectures

We then compared the hierarchical RL architecture with a non-hierarchical, plain RL architecture. We again used a one-joint, two-link robot (see Fig. 6), and compared the results in Section 3.2 with the results of a plain continuous actor–critic architecture [6]. In the plain architecture, the actor and the critic have to learn a highly non-linear control function and value function, respectively. In preliminary experiments, we used a simple reward function such as the height of the head for the plain architecture without success. Thus, we prepared a hand-crafted reward function

$$r(y) = \begin{cases} 0.3\left(\dfrac{y}{L}\right) + 0.3\sin(\theta_m) \\ +0.4\exp\left(-\left(\dfrac{\theta_0^2+\theta_1^2}{s_\theta^2} + \dfrac{\dot\theta_0^2+\dot\theta_1^2}{s_{\dot\theta}^2}\right)\right) - 1, \\ \qquad \text{during trial}, \\ -1, \qquad \text{the robot falls down} \end{cases} \tag{22}$$

for the plain architecture, where y is the height of the head of the robot, L the total length of the robot, and $s_\theta = 60$ [deg] and $s_{\dot\theta} = 240$ [deg/s] give the width of the reward function. Each simulation was continued up to 2000 trials. We used the same parameters in Section 3.2 for the continuous $TD(\lambda)$-learning except for $m_1 = 1000$, $m_2 = 1500$, and $V_1 = 0.0$. We limited the range of joint angle to $-150 \leqslant \theta_1 \leqslant 0$ for the plain architecture to make the stand-up task easy.

The robot successfully learned to stand-up within 1685 trials, which took 56 minutes in simulated time (averaged over 5 successful runs out of 10 simulation runs). Fig. 9 shows the time course of learning with the plain architecture. Compared to the robot with the hierarchical RL architecture, about four times as many learning trials were necessary with the plain RL architecture. Moreover, the robot with the hierarchical RL architecture (with $\Delta\theta_1 = 25$ and 30 [deg]) learned to stand up in a more robust way than the one with the plain architecture because the robot with the hierarchical architecture achieved about twice as many successful runs as the robot with the plain architecture.

3.4. The role of sub-goal reward R_{sub}

The sub-goals chosen as the actions of the upper level must satisfy two conditions:

1. they should be helpful for accomplishing a successful stand-up;
2. each of them must be achievable by the lower-level learner.

We used the rewards R_{main} and R_{sub} to satisfy both demands. R_{main} is given only when the robot successfully stands up. This means that the robot cannot get any reward in the early stage of learning if there is only reward R_{main} for the upper level. In such a case, the robot needs many trials to learn the task. Thus, we

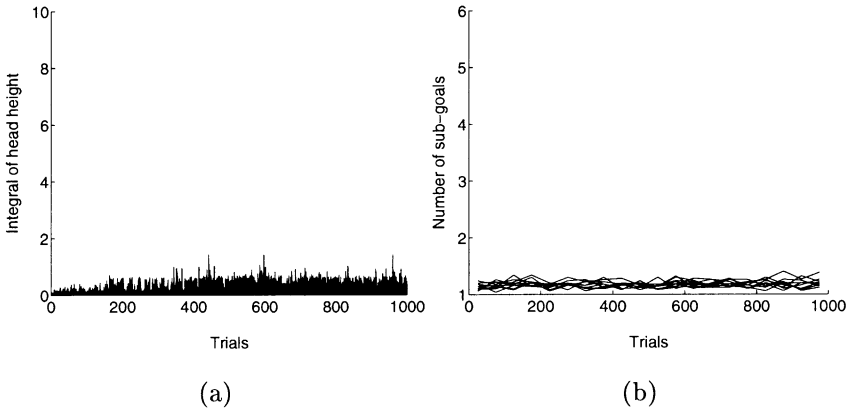

Fig. 10. Time course of learning without R_{sub}. (a) Performance index; (b) average number of sub-goals in each set of 50 trials.

introduced a supplementary reward R_{sub} in the upper level. Because of this reward, the upper-level learner is encouraged to use sub-goals that can be achieved by the lower-level learner, which avoids irrealizable relevant sub-goals in the early stage of learning. To verify the effect of R_{sub}, we applied the hierarchical RL method to the one-joint, two-link robot without R_{sub}. We used the same parameters as those in Section 3.2 for each learning algorithm except for the size of the noise term, which is always kept to $\sigma = 0.3$, in this section.

Fig. 10(a) shows the time course of learning without R_{sub}. Fig. 10(b) shows the time course of the sub-goals used in each trial. The robot never learned to stand up within 1000 trials in 10 simulation runs. This result shows the usefulness of R_{sub}.

4. Real robot experiments

Next, we applied the hierarchical RL to a real robot. As the initial condition for the real robot learning, we used the sub-goal sequence and non-linear controllers acquired by the simulation in Section 3.1. We then applied the hierarchical RL to a real robot (see configuration in Fig. 11).

We used a PC/AT with a Pentium 233 MHz CPU and RT-Linux as the operating system for controlling the robot (see Fig. 12). The time step of the lower-level learning was $\Delta t = 0.01$ [s], and that of the servo control was $\Delta t = 0.001$ [s].

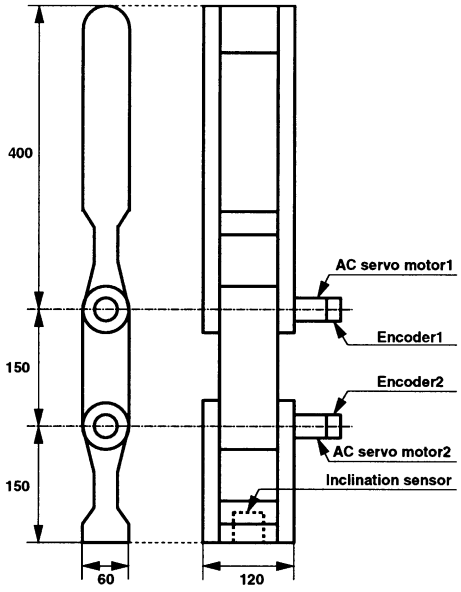

Fig. 11. Real robot configuration.

The robot has an inclination sensor to detect the pitch angle and the angular velocity of the link3 (see Fig. 1) and two rotary encoders to detect joint angles (θ_1, θ_2). We derived joint angular velocity $(\dot{\theta}_1, \dot{\theta}_2)$ by numerically differentiating the joint angles. We calculated the pitch angle and angular velocity $(\theta_0, \dot{\theta}_0)$ by using the above sensor data (see Fig. 1). We used the same parameters used in Section 3.1 except for

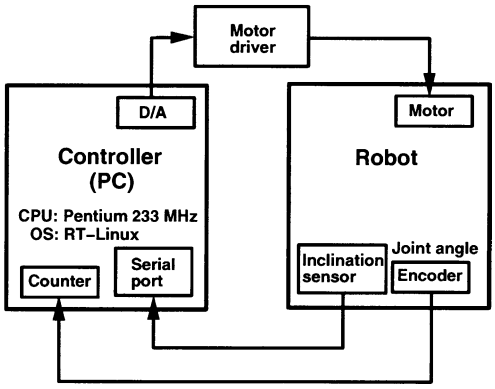

Fig. 12. System configuration.

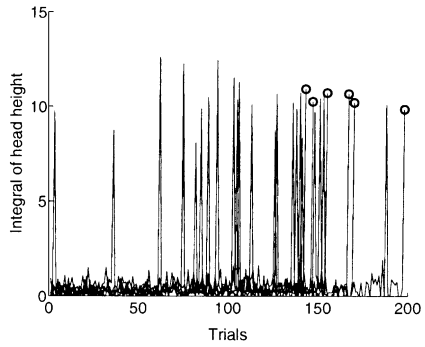

Fig. 13. Time course of learning with real robot. Circles show 10th successful stand-up, upon which a simulation run was terminated.

Table 3
Physical parameters of the real robot

	Length [m]	Weight [kg]	Inertia [kg m^2]
Link1	0.40	0.85	0.064
Link2	0.15	3.5	0.11
Link3	0.15	0.46	0.011

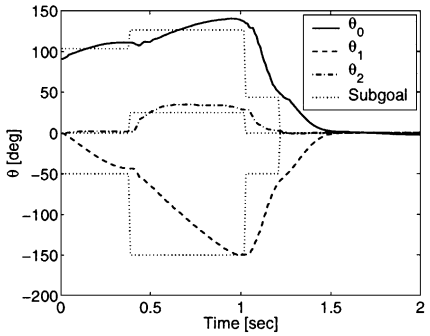

Fig. 14. Example of a time course of a stand-up trajectory and a sub-goal sequence (θ_0: pitch angle, θ_1, θ_2: joint angle).

the following parameters: learning rate of the critic $\alpha_c = 0.05$, learning rate of the actor $\alpha_a = 0.05$, initial amplitude of perturbation $\sigma_1 = 0.3$, final amplitude of perturbation $\sigma_2 = 0.05$, initial time for scheduling perturbation $m_1 = 0$, and final time for scheduling perturbation $m_2 = 150$. The physical parameters of the real robot are shown in Table 3.

We used the sub-goal sequence and non-linear controllers acquired by the learning with 7 successful simulation runs as the initial setting for the real robot experiments. Each experiment was continued up to 200 trials. The robot successfully learned to stand up in 6 out of 7 experiments within 164 trials (averaged over 6 successful runs). Fig. 13 shows the time course of learning with the real robot, and Fig. 14 shows the time course of a successful stand-up trajectory and a sub-goal sequence. These results show that the proposed hierarchical RL method enabled the real robot to accomplish the stand-up task and that the sub-goal sequence and non-linear controllers acquired by the simulation is useful for the learning by the real robot (see Fig. 15).

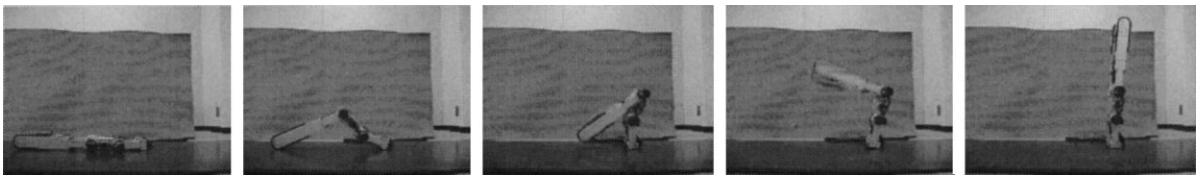

Fig. 15. Example of a stand-up trajectory using the real robot.

5. Discussion

In this section, we summarize the achievement of this study in relation to the previous studies of the hierarchical RL, RL using real robots, and the stand-up task for robots.

5.1. Hierarchical RL

Hierarchical RL methods have been developed for several different goals, such as solving POMDPs, improving learning speed, and creating reusable behavioral modules. For example, hierarchical Q-learning methods have been used for solving POMDPs by dividing the state space into several regions in which each task is reduced to a Markov decision problem (MDP) [26].

Our main interest is in improving the learning speed of a single task, in selection of reduced variables in the upper level, and in creating reusable behavioral modules for multiple tasks. We focus on these topics in the following sections.

5.1.1. Improving learning speed

Kimura and Kobayashi [10] used Q-learning in the upper level and local linear actor–critic controllers in the lower level. They applied their method to a cart-pole swing-up task, but not faster than non-hierarchical RL [5,6].

Dayan and Hinton [3] proposed the feudal RL method which used multiple resolutions in space and time. They showed that the method could accomplish the two-dimensional maze task faster than non-hierarchical RL.

On the other hand, by using our proposed hierarchical RL, the robot successfully learned to stand-up in the high-dimensional state space. Here, we summarize the reasons for the successful learning of the stand-up task by the hierarchical architecture, which can be helpful in other tasks as well. First, the upper level decomposed the original task into simplified sub-tasks. Furthermore, the upper level reward of the success of a sub-task (R_{sub}) encouraged the upper level to set realizable sub-goals. Second, the dimension reduction in the upper-level dramatically reduced the number of state in high-dimensional state space. Third, the coarse exploration in the upper level enabled the robot to explore efficiently in the entire state space and prevented it from getting in stuck local optimum. Fourth, in the hierarchical architecture, prior knowledge can be easily included. We set the appropriate size and direction of action steps in the upper level and provided linear feedback component in the lower level.

Although the proposed hierarchical RL method was successfully applied to a robot with four- and six-dimensional state space in Sections 3.1 and 3.2, respectively, it remains to be tested how well it scales with further increase in the dimension of the state space.

5.1.2. Selection of reduced variables

In this study, we chose the angles of the joints and the center of mass as the low-dimensional state variables for the upper level. However, this strategy of neglecting the velocity components has a limitation that the dimension can be reduced at most to the half of the original dimension. For systems with much higher-dimensional state space, e.g., arms or legs with excess degrees of freedom, we should consider the use of task-oriented kinematic variables in the upper level. For example, in manipulation task with a multi-joint arm, the position of the end effector can be a good state vector in the upper level. For another example, position of the center of mass or the zero-moment point (ZMP) can be a good higher-level representation in locomotion or posture control task with multiple legs. How to select such essential variables by learning remains as a subject of future work.

In addition, we chose an appropriate step size $\Delta \boldsymbol{X}$ in the upper level, but a method of automatically choosing and adapting step size is also a subject of future work.

5.1.3. Using reusable behavioral modules and abstract action

Singh [21] proposed compositional Q-learning (CQ-L) in which each lower-level module was automatically adapted to manage each subtask. The architecture of this learning method was similar to the mixtures of experts [8]. A gating module stochastically switches the lower level module, and a bias module estimates the state-value for compositional tasks. Each lower-level module can be reused in several compositional tasks. Tham [25] proposed an extended version of the CQ-L in which rewards can

be defined not only at goal states but also at non-goal states and each lower-level module can have more than two Q-networks for learning behaviors of more than two actuators. He applied the extended CQ-L to non-linear control tasks using a simulated two-linked manipulator.

Digney [4] proposed nested Q-learning in which a hierarchical structure was also learned. In his method, a state is detected as a sub-goal according to the experience: non-typical reinforcement is given in the state or the learner visits the state many times. Each sub-task then becomes one of the actions that the learner can choose as a primitive action. As a result, sub-tasks were nested in original tasks; in other words, the hierarchical structure was learned. However, the CQ-L and the nested Q-learning are suitable when the lower-level modules are reused in several tasks and were not developed for improving learning speed by focusing on a single task as in our study.

Sutton et al. [23] developed a concept called *option* which is a temporary abstract action. The option gives an interface between MDP and semi-MDP, and a theoretical formulation for the hierarchical RL. We will consider adapting this formulation to our method for theoretical soundness in future work.

5.2. RL in real robots

Many studies of RL focus on theoretical aspects and apply their method in a typical two-dimensional maze. However, in order to apply RL to real world problems, we should try to apply RL to realistic tasks such as real robot control. Recently, there have been several attempts to apply RL to real robots.

5.2.1. Navigation of wheeled mobile robots

Asada et al. [2,24] applied RL to soccer-playing robots entered in the RoboCup [1] middle-size class. Their robots successfully learned shooting and passing behaviors. Yamaguchi et al. [28] accomplished a ball-pushing task using their mobile robot with RL. Ortiz and Zufiria [19] applied RL to a goal-reaching task using the NOMAD 200 mobile robot. Mataric [14] investigated social behaviors of robots. A group of four mobile robots successfully learned a foraging task. However, these tasks did not have critical dynamic constraints as in our work.

5.2.2. Legged locomotion

Maes and Brooks [13] applied RL to the selection of behaviors of a six legged robot by only considering an immediate reward. Kirchner [11] applied RL to learn the appropriate leg motion of a six legged robot. Yamada et al. [27] developed hybrid controller composed of a linear control module, an RL module, and a selection module for controlling a stilt-type biped robot. Most of these studies dealt with stable limit cycle behaviors. The novelty of our work is that the task involves transient behavior under critical dynamic constraints.

5.3. The stand-up task

Although the stand-up behavior is necessary for any practical biped robot, there have not been so many studies focused on the stand-up behavior due to the difficulty in designing an appropriate controller.

Inaba et al. [7,9] developed a humanoid robot with 35 degrees of freedom that can stand up statically by using a control scheme pre-programmed by the experimenter. Kuniyoshi and Nagakubo [12] proposed a control strategy called *action oriented control* that does not require a precise dynamical model of robots. They applied their method to a dynamic stand-up task for a humanoid robot by specifying several postures and arranging them at appropriate time intervals chosen by the experimenter. However, successful results have only been obtained in simulation. Nakakuki and Yamafuji [18] developed a two-joint, three-link robot with curved contour. This robot can stand up dynamically through oscillatory movement by using its curved contour. However, the oscillatory movement is previously designed and not acquired by the learning. Furthermore, the robot cannot stay at the upright position because of its curved contour. Thus, the learning of the dynamic stand-up task by using a real robot in our study is a quite new result.

6. Conclusions

We proposed a hierarchical RL architecture that uses a low-dimensional state representation in the upper level. The stand-up task was accomplished by the hierarchical RL architecture using a real, two-joint, three-link robot. We showed that the hierarchical RL

architecture achieved the task much faster and more robustly than a plain RL architecture. We also showed that successful stand-up was not so sensitive to the choice of the upper-level step size and that upper-level reward R_{sub} was helpful for efficient exploration.

In this study, We used two-layered hierarchical architecture. As an extension, the use of a hierarchical architecture with three or more layers, and reusing lower-level modules in the other tasks, are interesting topics. We will incorporate these ideas in our hierarchical RL method as future work.

Acknowledgements

We would like to thank Mitsuo Kawato, Stefan Schaal, Christopher G. Atkeson, Tsukasa Ogasawara, Kazuyuki Samejima, Andrew G. Barto, and the anonymous reviewers for their helpful comments.

Appendix A. Normalized Gaussian network (NGnet)

The normalized Gaussian basis function is represented by

$$b_k(\boldsymbol{x}) = \frac{a_k(\boldsymbol{x})}{\sum_{l=1}^{K} a_l(\boldsymbol{x})}, \tag{A.1}$$

where

$$a_k(\boldsymbol{x}) = \mathrm{e}^{-\|\boldsymbol{s}_k^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{c}_k)\|^2} \tag{A.2}$$

is a Gaussian activation function [15]. The vectors $\boldsymbol{c}_k$ and $\boldsymbol{s}_k$ define the center and the size of the kth basis function, respectively. Note that if there is no neighboring basis function, the shape of the basis functions extend like sigmoid functions by the effect of normalization.

In particular, we use an INGnet [17] to represent a value function in the critic and a non-linear control function in the actor. In INGnet, a new unit is allocated if the error is larger than a threshold $e_{\max}$ and the activation of all existing units is smaller than a threshold $a_{\min}$, i.e.,

$$|y(\boldsymbol{x}) - \hat{y}(\boldsymbol{x})| > e_{\max} \quad \text{and} \quad \max_k a_k(\boldsymbol{x}) < a_{\min}. \tag{A.3}$$

The new unit is initialized with the weight $w_k = \hat{y}(\boldsymbol{x})$, the center $\boldsymbol{c}_k = \boldsymbol{x}$, and the size $\boldsymbol{s}_k = \mathrm{diag}(\mu_i)$, where $\hat{y}(\boldsymbol{x})$ is a desired output, and μ_i is the inverse of the radius of the basis function. In Sections 3 and 4, we set the inverse of the radius of the basis function as $\mu_\theta = 0.035$ [1/deg] for pitch and joint angle, and $\mu_{\dot{\theta}} = 0.0087$ [s/deg] for pitch and joint angular velocity. We also set the error threshold and the activation threshold as $e_{\max} = 0.0$ and $a_{\min} = 0.4$, respectively. Note that when a new basis function is allocated, the shapes of neighboring basis functions also change because of the nature of normalized Gaussian basis functions.

References

[1] M. Asada, H. Kitano, I. Noda, M. Veloso, RoboCup: Today and tomorrow — What we have learned, Artificial Intelligence 110 (1999) 193–214.
[2] M. Asada, E. Uchibe, K. Hosoda, Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development, Artificial Intelligence 110 (1999) 275–292.
[3] P. Dayan, G.E. Hinton, Feudal reinforcement learning, in: Advances in Neural Information Processing Systems, Vol. 5, Morgan Kaufmann, San Francisco, CA, 1993, pp. 271–278.
[4] B.L. Digney, Learning hierarchical control structures for multiple tasks and changing environments, in: Proceedings of the Fifth Conference on the Simulation of Adaptive Behavior, MIT Press, Cambridge, MA, 1998, pp. 321–330.
[5] K. Doya, Efficient nonlinear control with actor–tutor architecture, in: M.C. Mozer, M.I. Jordan, T. Petsche (Eds.), Advances in Neural Information Processing Systems, Vol. 9, MIT Press, Cambridge, MA, 1997, pp. 1012–1018.
[6] K. Doya, Reinforcement learning in continuous time and space, Neural Computation 12 (1) (2000) 219–245.
[7] M. Inaba, I. Igarashi, K. Kagami, I. Hirochika, A 35 DOF humanoid that can coordinate arms and legs in standing up, reaching and grasping an object, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'96), Osaka, Japan, Vol. 1, 1996, pp. 29–36.
[8] R.A. Jacobs, M.I. Jordan, Hierarchical mixtures of experts and the EM algorithm, Neural Computation 6 (1994) 181–214.
[9] F. Kanehiro, M. Inaba, H. Inoue, Development of a two-armed bipedal robot that can walk and carry objects, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'96), Osaka, Japan, Vol. 1, 1996, pp. 23–28.
[10] H. Kimura, S. Kobayashi, Efficient non-linear control by combining Q-learning with local linear controllers, in: Proceedings of the 16th International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA, 1999, pp. 210–219.
[11] F. Kirchner, Q-learning of complex behaviours on a six-legged walking machine, in: Proceedings of the Second

EUROMICRO Workshop on Advanced Mobile Robots, 1997, pp. 51–58.
[12] Y. Kuniyoshi, A. Nagakubo, Humanoid as a research vehicle into flexible complex interaction, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'97), Grenoble, France, 1997.
[13] P. Maes, R.A. Brooks, Learning to coordinate behaviors, in: Proceedings of AAAI'90, Boston, MA, 1990, pp. 796–802.
[14] M. Mataric, Learning social behaviors, Robotics and Autonomous Systems 20 (1997) 191–204.
[15] J. Moody, C.J. Darken, Fast learning in networks of locally-tuned processing units, Neural Computation 1 (1989) 281–294.
[16] J. Morimoto, K. Doya, Hierarchical reinforcement learning of low-dimensional sub-goals and high-dimensional trajectories, in: Proceedings of the Fifth International Conference on Neural Information Processing, Burke, VA, Vol. 2, IOS Press, Amsterdam, 1998, pp. 850–853.
[17] J. Morimoto, K. Doya, Reinforcement learning of dynamic motor sequence: learning to stand up, in: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98), Victoria, BC, Vol. 3, Omni Press, 1998, pp. 1721–1726.
[18] K. Nakakuki, K. Yamafuji, Motion control of a robot composed of three serial-links with curved contour (2nd report, several motions of the robot), Transactions of the Japan Society of Mechanical Engineers, Part C 59 (559) (1993) 850–854.
[19] M. Ortiz, P. Zufiria, Evaluation of reinforcement learning autonomous navigation systems for a NOMAD 200 mobile robot, in: Proceedings of the Third IFAC Symposium on Intelligent Autonomous Vehicles 1998 (IAV'98), 1998, pp. 309–314.
[20] J. Peng, R. Williams, Incremental multi-step Q-learning, Machine Learning 22 (1996) 283–290.
[21] S. Singh, Transfer of learning by composing solutions of elemental sequential tasks, Machine Learning 8 (1992) 323–339.
[22] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
[23] R.S. Sutton, D. Precup, S. Singh, Intra-option learning about temporary abstract actions, in: Proceedings of the 15th International Conference on Machine Learning, Madison, WI, 1998, pp. 556–564.
[24] Y. Takahashi, M. Asada, K. Hosoda, Reasonable performance in less learning time by real robot based on incremental state space segmentation, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'96), Osaka, Japan, Vol. 3, 1996, pp. 1518–1524.
[25] C.K. Tham, Reinforcement learning of multiple tasks using a hierarchical CMAC architecture, Robotics and Autonomous Systems 15 (1995) 247–274.
[26] M. Wiering, J. Schmidhuber, HQ-learning, Adaptive Behavior 6 (2) (1997) 219–246.
[27] S. Yamada, A. Watanabe, M. Nakashima, Hybrid reinforcement learning and its application to biped robot control, in: Advances in Neural Information Processing Systems, Vol. 10, 1998, pp. 1071–1077.
[28] T. Yamaguchi, M. Masubuchi, K. Fujihara, M. Yachica, Realtime reinforcement learning for a real robot in the real environment, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'96), Osaka, Japan, Vol. 3, 1996, pp. 1321–1327.

Jun Morimoto received his B.E. in Computer-Controlled Mechanical Systems from Osaka University in 1996, M.E. in Information Science from Nara Institute of Science and Technology in 1998, and Ph.D. in Information Science from Nara Institute of Science and Technology in 2001. He was a Research Assistant at Kawato Dynamic Brain Project, ERATO, JST in 1999. He is now a postdoctoral fellow at the Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania. He is a member of Japanese Neural Network Society, and Robotics Society of Japan. He received Young Investigator Award from Japanese Neural Network Society in 2000. His research interests include reinforcement learning and robotics.

Kenji Doya received his B.S., M.S., and Ph.D. in Mathematical Engineering from University of Tokyo in 1984, 1986, and 1991, respectively. He was a Research Associate at University of Tokyo in 1986, a post-graduate researcher at the Department of Biology, UCSD in 1991, and a Research Associate of Howard Hughes Medical Institute at Computational Neurobiology Laboratory, Salk Institute in 1993. He took the positions of a Senior Researcher at ATR Human Information Processing Research Laboratories in 1994, the leader of Computational Neurobiology Group at Kawato Dynamic Brain Project, ERATO, JST in 1996, and the leader of Neuroinformatics Project at Information Sciences Division, ATR International in 2000. He has been appointed as a visiting Associated Professor at Nara Institute of Science and Technology since 1995, and the Director of Metalearning, Neuromodulation, and Emotion Research, CREST, JST since 1999. He is an Action Editor of Neural Networks and Neural Computation, a board member of Japanese Neural Network Society, and a member of Society for Neuroscience and International Neural Network Society. His research interests include non-linear dynamics, reinforcement learning, the functions of the basal ganglia and the cerebellum, and the roles of neuromodulators in metalearning.