NAIST-IS-DT9861023

Doctoral Thesis

Hierarchical Decomposition and Min-max Strategy for Fast and Robust Reinforcement Learning in the Real Environment

Jun Morimoto

Department of Information Systems
Graduate School of Information Science
Nara Institute of Science and Technology

Doctoral Thesis
submitted to Graduate School of Information Science
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
DOCTOR of ENGINEERING

Jun Morimoto

Thesis committee : Tukasa Ogasawara, Professor
Masatsugu Kidode, Professor
Hirokazu Nishitani, Professor
Mitsuo Kawato, Professor
Kenji Doya, Associate Professor

Hierarchical Decomposition and Min-max Strategy for Fast and Robust Reinforcement Learning in the Real Environment *

Jun Morimoto

Abstract

Humans can learn new behaviors through trial and error without any instruction. If we can understand this learning systems, we can develop robots and softwares which can find the way to achieve their own goal without any control policy designed by humans.

"Reinforcement Learning (RL)" is a framework in which an agent tries to maximize a certain evaluation through trial and error. Recently, there have been many theoretical advances and application studies of RL.

In order to apply reinforcement learning to real world control problems, we consider methods for achieving practical learning speed and robustness against modeling errors or unknown disturbances. Specifically, we propose, a hierarchical reinforcement learning architecture for improving learning speed in a high-dimensional state space, and a robust reinforcement learning algorithm accommodating modeling errors or the unknown disturbances.

In the hierarchical reinforcement learning architecture, 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 a task of learning to stand up by trial and error. The

*Doctoral Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9861023, 0, 0.

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.

In the robust reinforcement learning, we consider input disturbance as well as modeling errors. The use of environmental models in RL is quite popular for both off-line learning by simulations and for on-line action planning. However, the difference between the model and the real environment can lead to unpredictable, often unwanted results. Based on the theory of H-infinity control, we consider a differential game in which a 'disturbing' agent tries to make the worst possible disturbance while a 'control' agent tries to make the best control input. The problem is formulated as finding a min-max solution of a value function that takes into account the changes in the reward due to the disturbance and the amplitude of the disturbance. We derive on-line learning algorithms for estimating the value function and for calculating both the worst disturbance and the best control in reference to the value function. We tested the paradigm, which we call "Robust Reinforcement Learning (RRL)," in the task of inverted pendulum.

Keywords:

reinforcement learning, hierarchical architecture, robust control, real robot, dynamic stand-up movement

Acknowledgments

First, I would like to thank my supervisor, Dr. Kenji Doya for his support, his patient guidance, and constant encouragement throughout this work. His valuable advice and detailed criticism have enabled me to complete this dissertation. I also would like to thank Dr. Mitsuo Kawato for giving me a chance to study at Kawato Dynamic Brain Project, ERATO, JST. I also wish to my gratitude to Professor Tukasa Ogasawara, Professor Masatsugu Kidode, Professor Hirokazu Nishitani, Professor Mitsuo Kawato, and Associate Professor Kenji Doya for their constructive readings of this dissertation and their valuable advice. I would like to thank Professor Minoru Asada and Professor Toyoaki Nishida gave me valuable advice and knowledge when I was bachelor and master course student. I would like to thank Professor Yoh'ichi Tohkura who gave me valuable advice and a chance to study at ATR Human Information Processing laboratories. Finally, I would like to thank all members of the Robotics laboratory, the Kawato Dynamic Brain Project, ERATO, JST, and the ATR Human Information Processing laboratories for their helpful discussion.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently, reinforcement learning (RL) is widely used in machine learning and robotics studies. RL does not require explicit knowledge of the desired trajectories, but only requires a signal that evaluates if the action taken by the learning agent is "good" or "bad". Then, RL is also attractive algorithm for modeling the human brain function, because the human learns several movements or behaviors through trail and error.

However, RL usually needs a huge amount of time for learning tasks in high-dimensional state space because the size of the state space exponentially increase according to the number of dimension. The other problem is that a policy learned by RL is not robust against modeling error, environmental change, or unknown disturbances compare to the control policy of the human.

1.1. Hierarchical Reinforcemtn Learning

Many RL studies focus on theoretical aspect using two-dimensional maze tasks. However, the robot have to learn tasks in the real environment with high-dimensional state space like humans do. In such a case, the humans use abstract representation of the state space for improving learning speed. Then, we first study how the hierarchical architecture of the learning system works for improving learning speed in RL framework.

Hierarchical RL methods have been developed for creating reusable behavioral modules [31, 37, 7], solving partially observable Markov decision problems (POMDPs) [39], and for improving learning speed [6, 16].

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 [23].

In this study, first, 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. 3.1). 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.

1.2. Imitation Learning

Second, we show how imitation learning improve the learning speed in the hierarchical reinforcement learning framework. Imitation learning[30, 21] is the learning method in which learners acquire useful information for achieve tasks from their teacher. However, different from usual supervised learning, the learner can not obtain full information of desired trajectories, but obtain the partial information of desired trajectories. In such a case, the learner have to realize desired trajectories from insufficient information which acquired from their teacher. In this study, we consider to use RL to construct desired trajectories or control policy with using the information from their teacher. We can say that the hierarchical architecture is a suitable representation for imitation learning, because the information from their teacher can be consider as the upper-level state and construct desired control policy by using the lower-level through trial and error.

We implement above ideas in the stand-up task using the 2-joint 3-link robot. As a teacher information, we use the stand-up trajectory of the teacher in joint and pitch angle space, and detect via-points from the trajectory. We find the

subgoal points by using detected via-points according to the distance between subgoals and via-points. Then, we put the action value at the selected subgoal sequence, in which the action value is calculated from reward model and teacher trajectory.

1.3. Robust Reinforcement Learning

Third, we study how we can make the policy learned by RL more robust against modeling error or environmental change. Then, we propose a new reinforcement learning paradigm that we call "Robust Reinforcement Learning (RRL)." Plain, model-free reinforcement learning (RL) is desperately slow to be applied to on-line learning of real-world problems. Thus the use of environmental models have been quite common both for on-line action planning [10] and for off-line learning by simulation [25]. However, no model can be perfect and modeling errors can cause unpredictable results, sometimes worse than with no model at all. In fact, robustness against model uncertainty has been the main subject of research in control community for the last twenty years and the result is formalized as the "H^∞" control theory [43].

In general, a modeling error causes a deviation of the real system state from the state predicted by the model. This can be re-interpreted as a disturbance to the model. However, the problem is that the disturbance due to a modeling error can have a strong correlation and thus standard Gaussian assumption may not be valid. The basic strategy to achieve robustness is to keep the sensitivity γ of the feedback control loop against a disturbance input small enough so that any disturbance due to the modeling error can be suppressed if the gain of mapping from the state error to the disturbance is bounded by $1/\gamma$. In the H^∞paradigm, those 'disturbance-to-error' and 'error-to-disturbance' gains are measured by a max norms of the functional mappings in order to assure stability for any modes of disturbance.

We briefly introduce the H^∞paradigm and show that design of a robust controller can be achieved by finding a min-max solution of a value function, which is formulated as Hamilton-Jacobi-Isaacs (HJI) equation. We then derive on-line algorithms for estimating the value functions and for simultaneously deriving the

worst disturbance and the best control that, respectively, maximizes and minimizes the value function.

We test the validity of the algorithms first in a linear inverted pendulum task. It is verified that the value function as well as the disturbance and control policies derived by the on-line algorithm coincides with the analytical solution given by H^{∞}theory. We then compare the performance of the robust RL algorithm with a standard model-based RL in a nonlinear task of pendulum swing-up [10]. It is shown that robust RL controller can accommodate changes in the weight and friction of the pendulum, which a standard RL controller cannot cope with.

1.4. Outline

This dissertation is organized as follows. In chapter 2, we explain the basic framework of RL. In chapter 3, we explain the proposed hierarchical reinforcement learning method. In chapter 4, we explain the imitation learning method in the hierarchical reinforcement learning framework. In chapter 5, we explain the proposed robust reinforcement learning method. In chapter 6, we conclude this dissertation.

Chapter 2

Reinforcement Learning

Reinforcement learning (RL) does not require explicit knowledge of the desired trajectories, but only requires a signal that evaluates if the action taken by the learning agent is "good" or "bad". Agents acquire the desired trajectories through trail and error by using RL. In concrete, the agents learn to choose action by which the agents maximize accumulated future reward. If an environment is stochastic, the accumulated future reward is represented by expected value. Then, the expected accumulated future reward under policy μ at time t is defined as

$$V^{\mu}(t) = \mathrm{E}[\sum_{n=0}^{\infty} \gamma^n r(t+n)] \tag{2.1}$$

in discrete time system, where $r(t)$ is the reward at time t, and γ is the discount factor for the future reward. The basic strategy of RL is to predict the expected future reward from current state $x(t)$ as the state-value function $V(\mathbf{x}(t))$. Because of this prediction, the agents can evaluate own actions and decide subsequent actions[33].

2.1. Reinforcement Learning in Continuous Time and Space

We can expect that the robot can acquire smooth controller for a control task in continuous time and space by using a learning method defined in continuous time and space. Then, in this study, the robot try to accomplish tasks by using continuous time and space TD-learning method[10, 8]

2.1.1 TD-learning in Continuous Time and State System

Value function

We consider a continuous time and state dynamics,

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), \mathbf{u}(t)), \tag{2.2}$$

where $\mathbf{x} \in X \subset R^n$ is input state and $\mathbf{u} \in U \subset R^m$ is a control output. The reward function is represented as the function of input state and control output as

$$r(t) = r(\mathbf{x}(t), \mathbf{u}(t)). \tag{2.3}$$

Then, the value function is defined as

$$V^{\mu}(\mathbf{x}(t)) = \int_t^{\infty} e^{-\frac{s-t}{\tau}} r(\mathbf{x}(s), \mathbf{u}(s)) ds \tag{2.4}$$

under the control policy $\mathbf{u}(t) = \mu(\mathbf{x}(t))$, where τ denotes a time constant of evaluation. This time constant τ is related to a discount factor γ in discrete time case as $\gamma = 1 - \frac{\Delta t}{\tau}$ (see appendix C).

Temporal difference error

We consider to approximate the value function defined in equation (2.4) by using a function approximator as

$$V^{\mu}(\mathbf{x}(t)) \simeq V(\mathbf{x}(t); \mathbf{w}) \tag{2.5}$$

where $\mathbf{w}$ is parameter vector of the function approximator. The TD-learning update their prediction of the value function in on-line fashion by using a local constraint which the value function should satisfy. We can derive the constraint

$$\frac{dV^{\mu}(\mathbf{x}(t))}{dt} = \frac{1}{\tau} V^{\mu}(\mathbf{x}(t)) - r(t) \tag{2.6}$$

by differentiate equation (2.4). If the prediction is not correct, the learner update their prediction to reduce the prediction error

$$\delta(t) \equiv r(t) - \frac{1}{\tau} V(\mathbf{x}(t)) + \frac{dV(\mathbf{x}(t))}{dt} \tag{2.7}$$

which is the Temporal Difference error (TD-error) in continuous time and space. This TD-error is used for update the value function and the control function in the actor-critic method explained in section 2.1.2.

2.1.2 Actor-critic

In this study, we use the actor-critic architecture, to implement the TD-learning.

In the actor-critic architecture, we use a control network called actor and an evaluation network called critic. The critic predicts the value of the state $V(\mathbf{x}(t))$, and the actor acquires control policy maximize the value of the state $V(\mathbf{x}(t))$(see Fig.5.3).

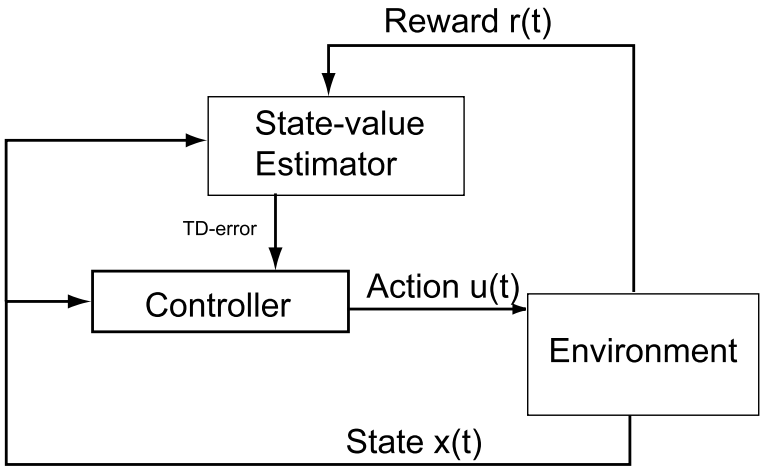


Figure 2.1 Actor-critic architecture

Predicting and learning the value function by the critic

The critic learn to reduce the prediction error $\delta(t)$ (TD-error). Inputs of the critic network are the current states, and output is the prediction of the value function. The predicting and learning the value function are realized as following

1. The critic predicts the value of the states $V(\mathbf{x}(t))$ as a function of current states.
$$V(\mathbf{x}(t)) = \sum_i v_i b_i(\mathbf{x}(t)), \tag{2.8}$$
where $b_i()$ denotes a basis function and v_i denotes a weight of the critic network.

2. The critic calculate the prediction error (TD-error) $\delta(t)$
$$\delta(t) = r(t) - \frac{1}{\tau}V(\mathbf{x}(t)) + \frac{dV(\mathbf{x}(t))}{dt} \tag{2.9}$$

3. The critic updates their network weights
$$\dot{v}_i = \alpha\delta(t)e_i(t), \tag{2.10}$$
where α denotes the learning rate and e_i denotes eligibility trace

4. The critic update their eligibility trace e_i.
$$\dot{e}_i(t) = -\frac{1}{\kappa}e_i(t) + b_i(\mathbf{x}(t)) \tag{2.11}$$

Control output and learning of the actor

The actor learns to achieve the state has high state value $V(\mathbf{x}(t))$, and maximize the accumulated future reward.

1. the actor outputs j-th control output as
$$u_j(t) = u_j^{\max} g\left(\sum_i w_{ij} b_i(\mathbf{x}(t)) + \sigma n_j(t)\right) \tag{2.12}$$
where $b_i()$ denotes a basis function, and w_{ij} denotes network weight. $u_j^{\max}$ is maximum output of j-th control output. $g()$ is used for saturate control output at the maximum control output $u_j^{\max}$ (e.g. we can use sigmoidal function gs $g()$). $\sigma n_j(t)$ is the noise term for exploration. we use the noise sufficiently contain low frequency.

2. the actor updates their network weights as

$$\dot{w}_{ij} = \beta\delta(t)\sigma n_j(t)b_i(\mathbf{x}(t)) \tag{2.13}$$

where β denotes the learning rate.

Although, we showed the learning rule of the continuous time actor-critic architecture, we show the discritized method for the computer simulation in appendix D. We also explained derivation of continuous TD-learning[10] in appendix B and correspondence between discrete and continuous TD-learning in appendix C.

Chapter 3

Hierarchical Reinforcement Learning

3.1. 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 [10]. 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 chapter, 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 [31, 37, 7], solving partially observable Markov decision problems (POMDPs) [39], and for improving learning speed [6, 16].

Many hierarchical RL methods use coarse and fine grain quantization of the state space. However, in a high-dimensional state space, even the coarsest quan-

tization 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 [23].

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. 3.1). 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 chapter is organized as follows. In Section 3.2, we explain the proposed hierarchical reinforcement learning method. In Section 3.3, we show simulation results of the stand-up task using the proposed method and compare the performance with non-hierarchical reinforcement learning. In Section 3.4, 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 3.5, we discuss the difference between our method and previous methods in terms of hierarchical reinforcement learning, reinforcement learning using real robots, and the stand-up task. Finally, we conclude this paper in Section 3.6.

3.2. 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. 3.2).

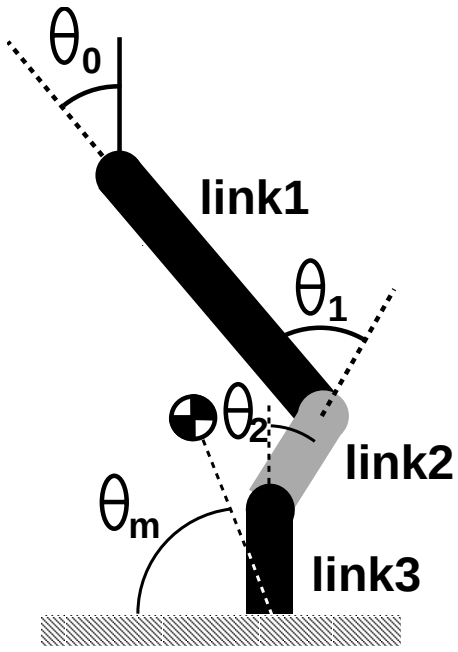


Figure 3.1 Robot configuration. θ_0:pitch angle, θ_1:hip joint angle, θ_2:knee joint angle, θ_m:the angle of the line from the center of mass to the center of the foot.

3.2.1 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 level are the physical variables, while those in the upper level are lower-dimensional state variables (see Table 3.1). The

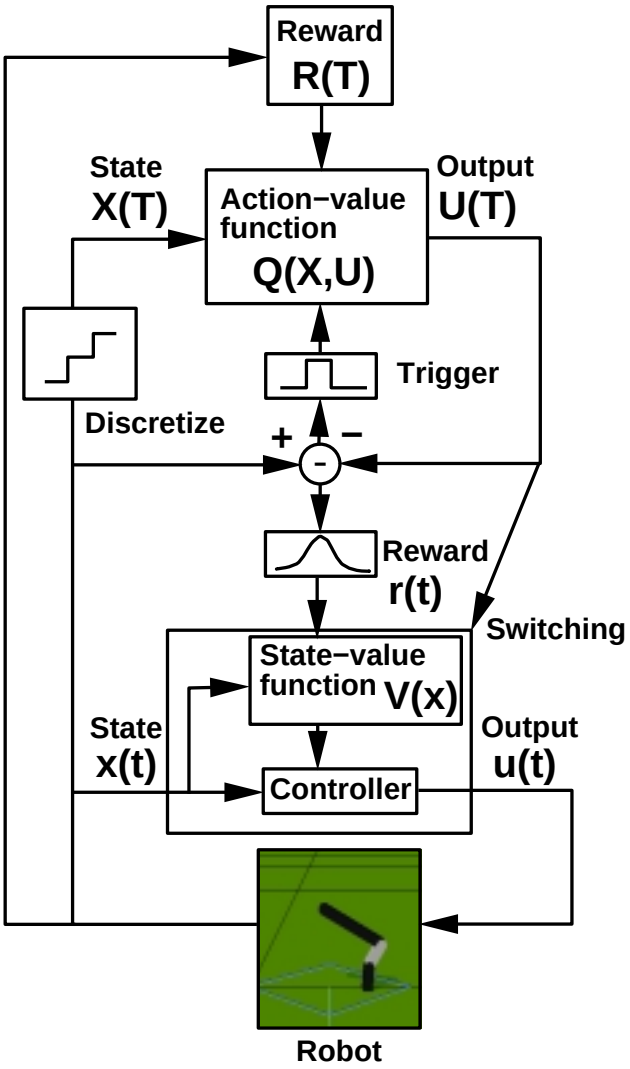


Figure 3.2 Hierarchical reinforcement learning architecture

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.

Table 3.1 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

3.2.2 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 [28] 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 (see Fig.3.3). 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 the Q(λ)-learning, the action-value function is updated in two steps. First, all of the state-action pairs are updated by

$$\delta(T) = R(T) + \gamma V_T(\boldsymbol{X}(T+1)) - V_T(\boldsymbol{X}(T)), \tag{3.1}$$

$$e(\boldsymbol{X}, \boldsymbol{U}) = \gamma\lambda e(\boldsymbol{X}, \boldsymbol{U}), \tag{3.2}$$

$$Q_{T+1}(\boldsymbol{X}, \boldsymbol{U}) = Q_T(\boldsymbol{X}, \boldsymbol{U}) + \alpha_Q e(\boldsymbol{X}, \boldsymbol{U})\delta(T), \tag{3.3}$$

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,

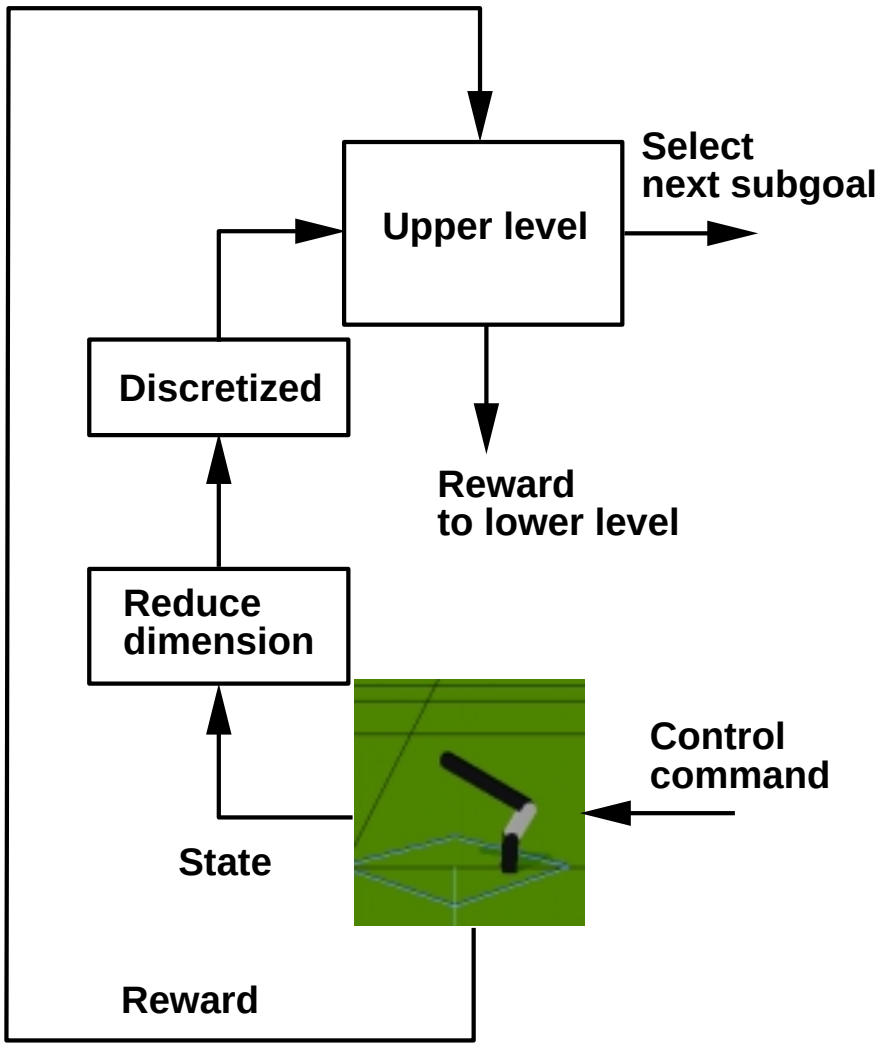


Figure 3.3 Upper level

$e(\boldsymbol{X}, \boldsymbol{U})$ 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

$$\delta'(T) = R(T) + \gamma V_T(\boldsymbol{X}(T+1)) - Q_T(\boldsymbol{X}(T), U(T)), \tag{3.4}$$

$$Q_{T+1}(\boldsymbol{X}(T), \boldsymbol{U}(T)) = Q_{T+1}(\boldsymbol{X}(T), \boldsymbol{U}(T)) + \alpha_Q \delta'(T), \tag{3.5}$$

$$e(\boldsymbol{X}(T), \boldsymbol{U}(T)) = e(\boldsymbol{X}(T), \boldsymbol{U}(T)) + 1, \tag{3.6}$$

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

In the stand-up task, we chose the posture of the robot $\mathbf{X}$=$(\theta_m, \theta_1, \theta_2)$ as the state variables (see Fig. 3.1). The desired posture of the robot $\mathbf{U}(T) = \mathbf{X}(T-1) + \mathbf{\Delta X}$ 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 [33]. Thus we have

$$P(U(T) = a) = \frac{\exp[\beta Q(\mathbf{X}(T), a)]}{\sum_{b \in A(\boldsymbol{X})} \exp[\beta Q(\mathbf{X}(T), b)]}, \tag{3.7}$$

where $A(\boldsymbol{X})$ 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.

$$R(T) = R_{main} + R_{sub}, \tag{3.8}$$

$$R_{main} = \begin{cases} 1 & \text{(on success of stand-up)} \\ 0 & \text{(on failure)} \end{cases}, \tag{3.9}$$

$$R_{sub} = \begin{cases} 1 & \text{(final goal achieved)} \\ 0.25(\frac{Y}{L} + 1) & \text{(subgoal achieved)} \\ 0 & \text{(on failure)} \end{cases}, \tag{3.10}$$

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.

3.2.3 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 [10] to learn the control command sequence. In addition, we use an Incremental Normalized Gaussian Network (INGnet) to implement the actor and the critic [24] (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 [6] (see Fig.3.4). In continuous actor-critic learning of the lower-level learner, the critic learns the state-value function $V(\mathbf{x}(t))$ that predicts the accumulated future reward at state $\mathbf{x}(t)$, while the actor learns the control function $u_j(t) = u_{max} h(f_j(\mathbf{x}(t)) + \sigma n_j(t))$ that specifies a nonlinear feedback control law. Here, $h(x) = \frac{\pi}{2} \arctan(\frac{2}{\pi} x)$ 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 $\tau_n = 0.1$ [sec] 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

$$V(\mathbf{x}(t)) = \sum_i v_i b_i(\mathbf{x}(t)), \tag{3.11}$$

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

$$\delta(t) = r(t) - \frac{1}{\tau} V(\mathbf{x}(t)) + \frac{dV(\mathbf{x}(t))}{dt}, \tag{3.12}$$

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

$$\dot{v}_i = \alpha_c \delta(t) e_i(t), \tag{3.13}$$

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

$$\dot{e}_i(t) = -\frac{1}{\kappa} e_i(t) + b_i(\mathbf{x}(t)), \tag{3.14}$$

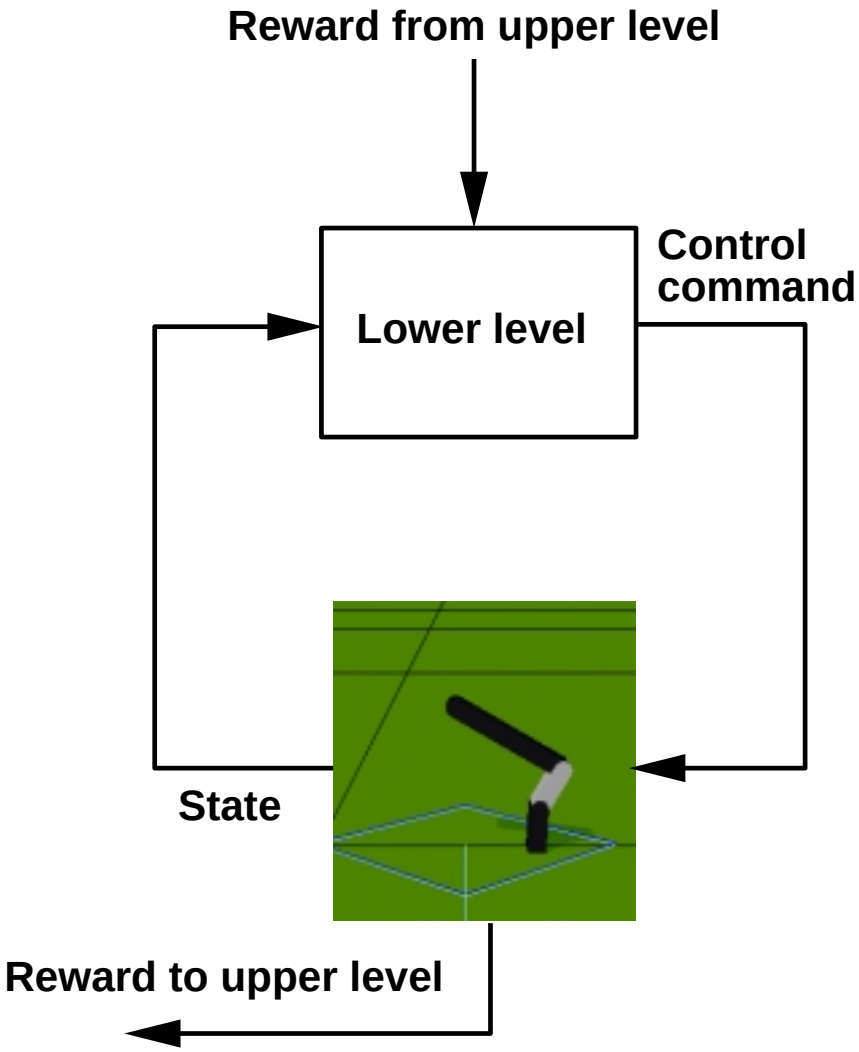


Figure 3.4 Lower level

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

The output of the actor is given by

$$f_j(\mathbf{x}(t)) = \sum_i w_{ij} b_i(\mathbf{x}(t)), \tag{3.15}$$

$$u_j(\mathbf{x}(t)) = u_{\max} h(f_j(\mathbf{x}(t)) + \sigma n_j(t)), \tag{3.16}$$

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

$$\dot{w}_{ij} = \alpha_a \delta(t) \sigma n_j(t) b_i(\mathbf{x}(t)), \tag{3.17}$$

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 $\mathbf{u}(t)$.

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 $\mathbf{x}(t) = (\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$. We chose torque $\mathbf{u}(t) = (\tau_1, \tau_2)$ 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 $f_i(\mathbf{x})$, which is acquired by the lower-level actor:

$$\tau_j = u_{\max} h \left(\frac{1}{u_{\max}} (k(\hat{\theta}_j - \theta_j) - b\dot{\theta}_j) + f_j(\mathbf{x}) + \sigma n_j \right), \tag{3.18}$$

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

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. 3.2). 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

$$r(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \exp\left(-\frac{||\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}||^2}{s_\theta^2} \right) - 1, \tag{3.19}$$

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

$$r(t) = \begin{cases} \exp\left(-\frac{||\dot{\boldsymbol{\theta}}(t) - \hat{\dot{\boldsymbol{\theta}}}||^2}{s_{\dot{\theta}}^2} \right) & \text{(sub-goal achieved)} \\ -1.5 & \text{(The robot falls down)} \end{cases}, \tag{3.20}$$

where $s_{\dot{\theta}} = 60$ [deg/sec] 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 (4.44) before the robot achieves the first stand-up.

3.3. 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.

3.3.1 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. 3.1). We used a low-dimensional state $\mathbf{X} = (\theta_m, \theta_1, \theta_2)$, where $0 \leq \theta_m \leq 90$, $-150 \leq \theta_1 \leq 0$, $0 \leq \theta_2 \leq 25$ [deg] in the upper level and a high-dimensional state $\mathbf{x} = (\theta_0, \theta_1, \theta_2, \dot{\theta}_0, \dot{\theta}_1, \dot{\theta}_2)$, where $-150 \leq \theta_1 \leq 150$, $-150 \leq \theta_2 \leq 150$ [deg] in the lower level. We chose $\Delta\theta_m = 30, \Delta\theta_1 = 50, \Delta\theta_2 = 25$ [deg] as the action step $\boldsymbol{\Delta X}$ in the upper level. Each trial was started with the robot lying on the ground, $\mathbf{x} = (90, 0, 0, 0, 0, 0)$ [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

$$\sigma_s = \begin{cases} \sigma_0 & \text{for final sub-goal} \\ \sigma_1 & \text{otherwise if } m \leq m_1 \\ \frac{(m_2 - m)\sigma_1 + (m - m_1)\sigma_2}{m_2 - m_1} & \text{if } m_1 < m < m_2 \\ \sigma_2 & \text{if } m \geq m_2 \end{cases} . \tag{3.21}$$

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. 3.15 and Table 3.3. The physical system was simulated by a dynamic simulator made by Boston Dynamics Inc., with a time step of 0.001 [sec]. 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.

Figure 3.5(a) shows the time course of learning. The vertical axis shows the performance index given by the integral of the head height $\int_0^{t_e} y(t)dt$, where t_e is terminal time of the trial. Figure 3.5(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. 3.6.

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 Figure 3.7 shows an example of a sub-goal sequence acquired in the upper level. The bottom images of Figure 3.7 shows an example

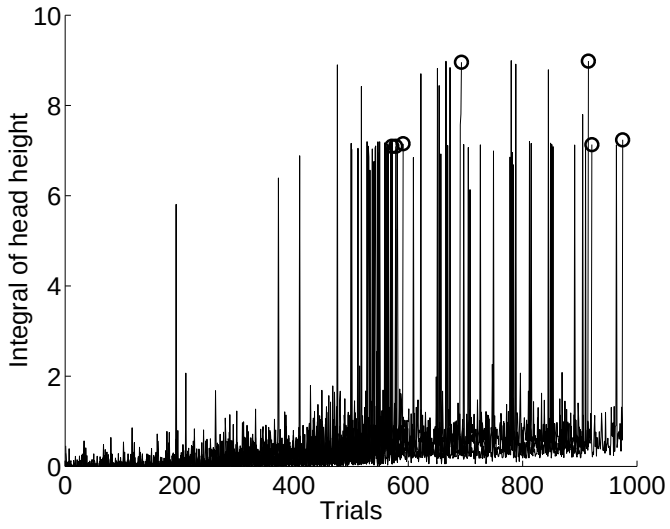
of a stand-up trajectory acquired in the lower level. Each learner successfully learned the appropriate action sequence for the stand-up task.

3.3.2 The effect of step size in the upper level

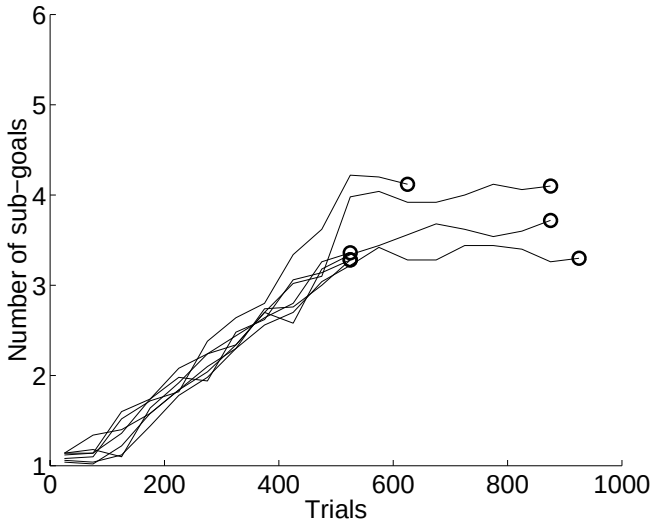
To investigate the effect of the action step size, We compared the upper-level learners with different $\boldsymbol{\Delta X}$. For simplicity, we fixed θ_2 to 0 [deg] by servo control (see Fig. 3.8) and chose action steps as $\Delta\theta_1 = 25, 30, 50$ [deg] and $\Delta\theta_m = 30$ [deg]. [1] Thus, we chose $\mathbf{X} = (\theta_m, \theta_1)$ and $\mathbf{x} = (\theta_0, \theta_1, \dot{\theta}_0, \dot{\theta}_1)$ 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 3.3.1 for each learning algorithm except for $\sigma_1 = 0.3$. Figure 3.11 and Table 3.2 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. Figure 3.12 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. 3.12). 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 reinforcement learning method was not so sensitive to the choice of $\boldsymbol{\Delta X}$ but has a certain range of $\boldsymbol{\Delta X}$ in which the upper-level learner successfully acquired the appropriate sub-goals for stand-up.

[1] Each width of the $\Delta\theta_1$ represent about $17\%, 20\%, 33\%$ of the range of θ_1 in the upper level (150[deg]), respectively, which indicate that $\Delta\theta_1 = 50$[deg] has relatively wide step compare to $\Delta\theta_1 = 25$ and 30[deg].



(a)



(b)

Figure 3.5 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.

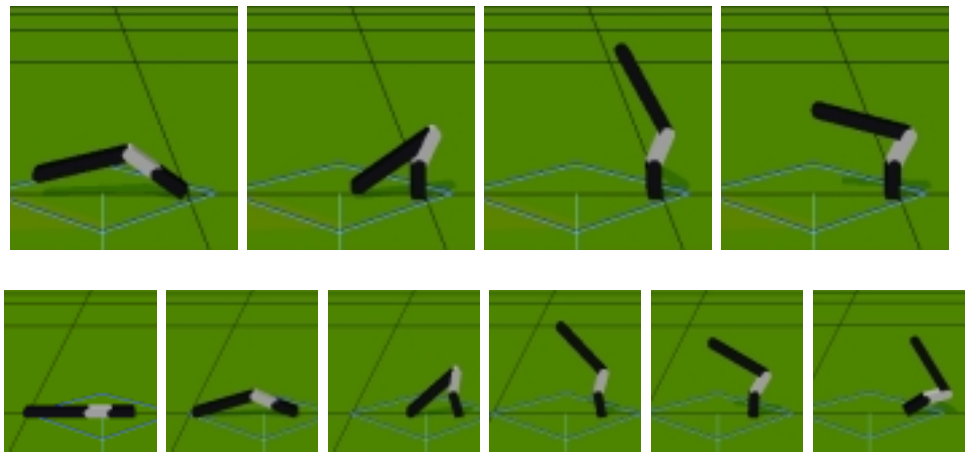


Figure 3.6 (Top): Example of a sub-goal sequence after 300 trials (Bottom): Example of a failed stand-up trajectory after 300 trials

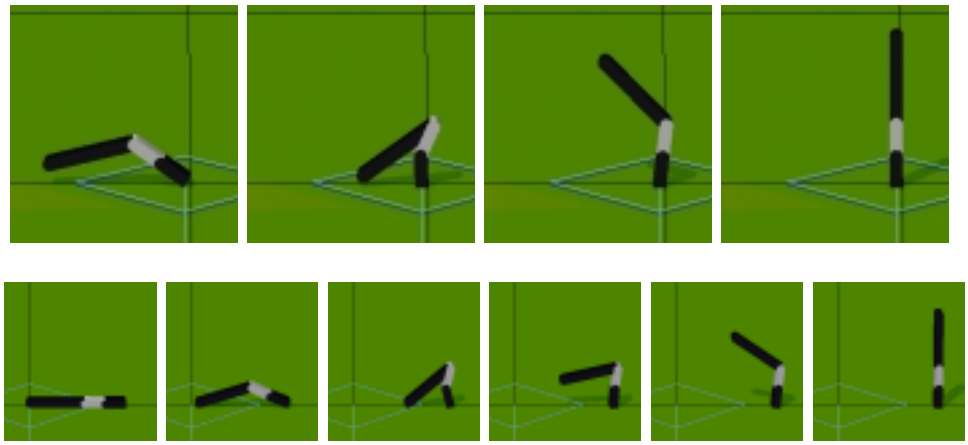


Figure 3.7 (Top): Example of a successful sub-goal sequence (Bottom): Example of a successful stand-up trajectory

3.3.3 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. 3.8), and

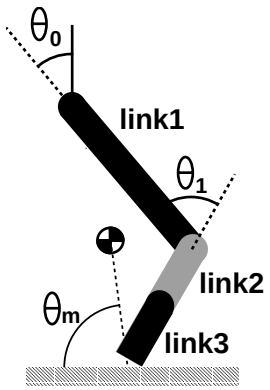


Figure 3.8 One-joint, two-link robot configuration

Table 3.2 Comparison with different $\Delta\theta_1$

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

compared the results in section 3.3.2 with the results of a plain continuous actor-critic architecture [10]. 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

$$r(y) = \begin{cases} 0.3(\frac{y}{L}) + 0.3\sin(\theta_m) + 0.4\exp(-(\frac{\theta_0^2+\theta_1^2}{s_\theta^2} + \frac{\dot{\theta}_0^2+\dot{\theta}_1^2}{s_{\dot{\theta}}^2})) - 1 & \text{(during trial)} \\ -1 \quad \text{(fall down)} \end{cases} \tag{3.22}$$

for the plain architecture, where y is the height of the head of the robot, L is total length of the robot, and $s_\theta = 60$ [deg] and $s_{\dot{\theta}} = 240$ [deg/sec] give the width of the reward function. Each simulation was continued up to 2000 trials. We used

the same parameters in section 3.3.2 for the continuous TD(λ)-learning except for $m_1 = 1000$, $m_2 = 1500$, and $V_1 = 0.0$. We limited the range of joint angle to $-150 \leq \theta_1 \leq 0$ 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). Figure 3.13 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.

We can consider the use of the reward function in equation 3.22 as the use of subgoals like in hierarchical RL. We also used some prior knowledge for introducing hierarchical architecture to RL (see Fig.3.2). However, there are many advantage of using hierarchical RL as

- We do not have to find specific subgoals for achieving tasks successfully while we have to find appropriate reward function in the non-hierarchical RL, because the hierarchical RL learn to find appropriate subgoal sequence from given subgoal set through trial and error. Then, we can say that the hierarchical RL is more general framework than the non-hierarchical RL with carefully designed reward function.

- If we use inappropriate or too simple reward function for the non-hierarchical RL, the learner likely to stuck in a local maximum point. On the other hand, the learner easily avoid a local maximum point in the lower level by using the hierarchical RL with appropriate subgoal set (candidates) in the upper level. Choosing an appropriate reward function is usually more difficult than choosing an appropriate subgoal set in the upper level in the hierarchical RL framework. Then, the hierarchical RL is suitable method for complex learning tasks.

- In the hierarchical RL, the number of dimension are reduced in the upper level. Then, we can expect that the learner can explore in the high-

dimensional state space more efficiently than using the non-hierarchical RL in which the learner explorer simply in original high-dimensional state space. Because of above reason, the robot with the hierarchical RL could learn the stand-up task faster than with the non-hierarchical RL.

3.3.4 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 introduced a supplementary 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 3.3.2 for each learning algorithm except for the size of the noise term, which is always keep to $\sigma = 0.3$ in this section.

Figure 3.14(a) shows the time course of learning without R_{sub}. Figure 3.14(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}.

3.4. 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 3.3.1. We then applied the hierarchical RL to a real robot.

We used a PC/AT with a Pentium 233 MHz CPU and RT-Linux as the operating system for controlling the robot (see Fig. 3.16). The time step of the lower-level learning was $\Delta t = 0.01$ [sec], and that of the servo control was $\Delta t = 0.001$ [sec].

The robot has an inclination sensor to detect the pitch angle and the angular velocity of the link3 (see Fig. 3.1) 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 (θ_0, $\dot{\theta}_0$) by using the above sensor data (see Fig. 3.1). We used the same parameters used in section 3.3.1 except for 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 3.3. We used

Table 3.3 Physical parameters of the real robot

	length	weight	inertia
link1	0.40 m	0.85 kg	0.064 $kg\ m^2$
link2	0.15 m	3.5 kg	0.11 $kg\ m^2$
link3	0.15 m	0.46 kg	0.011 $kg\ m^2$

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). Figure 3.17 shows the time course of learning with the real robot, and Figure 3.18 shows the time course of a successful stand-up trajectory and a sub-goal sequence. These result 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.

3.5. 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.

3.5.1 Hierarchical RL

Hierarchical RL methods have been developed for several different goals, such as solving partially observable Markov decision problems (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) [39].

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 below.

Improving learning speed of a single task

Kimura and Kobayashi [16] 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 [9, 10].

Dayan and Hinton [6] 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 section 3.3.2 and 3.3.1 respectively, it remains to be tested how well it scales with further increase in the dimension of the state space.

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, for example, 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 $\boldsymbol{\Delta X}$ in the upper level, but a method of automatically choosing and adapting step size is also a subject of future work.

Using reusable behavioral modules and abstract action

Singh [31] 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 [14]. 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 [37] 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 [7] 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.

3.5.2 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.

Navigation of wheeled mobile robots

Asada et al. [2, 35] 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. [41] accomplished a ball-pushing task using their mobile robot with RL. Ortiz and Zufiria [27] applied RL to a goal-reaching task using the NOMAD 200 mobile robot. Mataric [20] 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.

Legged locomotion

Maes and Brooks [19] applied RL to the selection of behaviors of a six legged robot by only considering an immediate reward. Kirchner [17] applied RL to learn the

appropriate leg motion of a six legged robot. Yamada et al. [40] 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.

3.5.3 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.

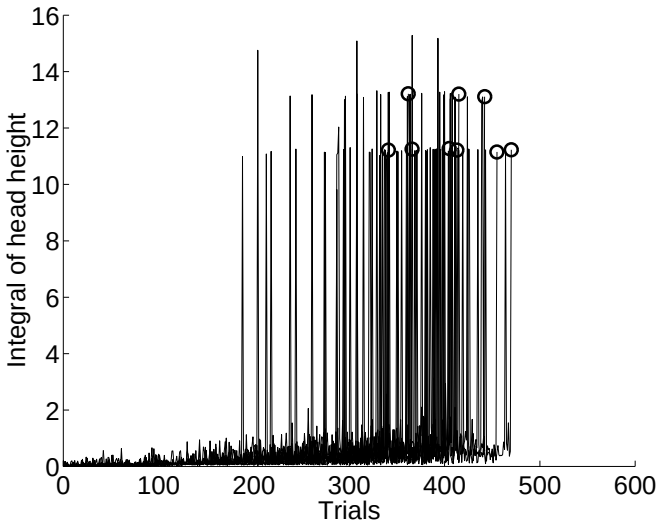
Inaba et al. [13, 15] developed a humanoid robot with 35 degrees of freedom that can stand up statically by using a control scheme pre-programmed by the experimenter. Kuniyoshi and Ngakubo [18] proposed a control strategy called *action oriented control* that does not require a precise dynamical model 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. Thus, the learning of the dynamical stand-up task by using a real robot in our study is a quite new result.

3.6. Conclusions

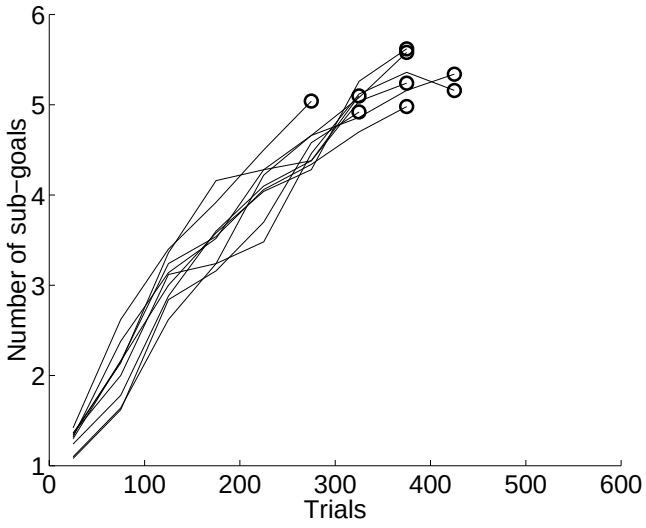
We proposed a hierarchical RL architecture that uses a low-dimensional state representation in the upper level. The stand-up task was accomplished by the hierarchical RL architecture using a real, two-joint, three-link robot. We showed that the hierarchical RL architecture achieved the task much faster 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 fixed an appropriate step size in the upper level, but a method of automatically choosing and adapting the step size remains future work. In addition, the use of a hierarchical architecture with three or more layers, and

reusing lower-level modules in the other tasks, are also interesting topics. We will incorporate these ideas in our hierarchical RL method as future work.

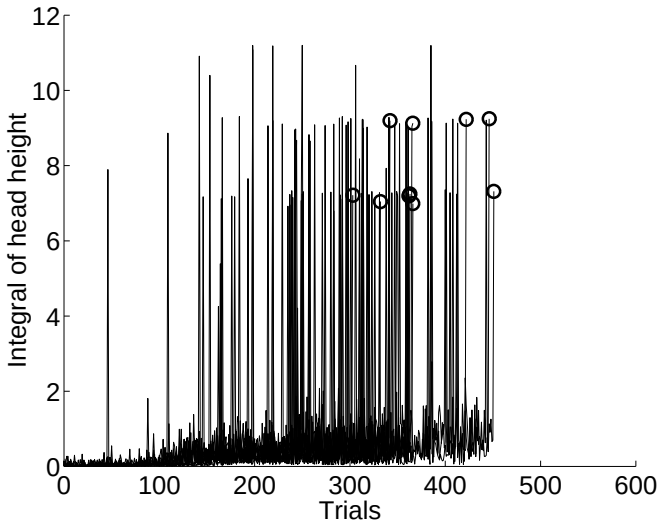


(a)

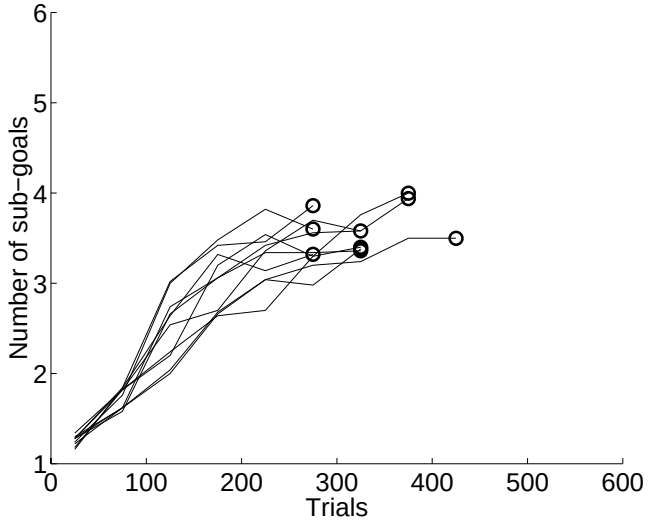


(b)

Figure 3.9 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]

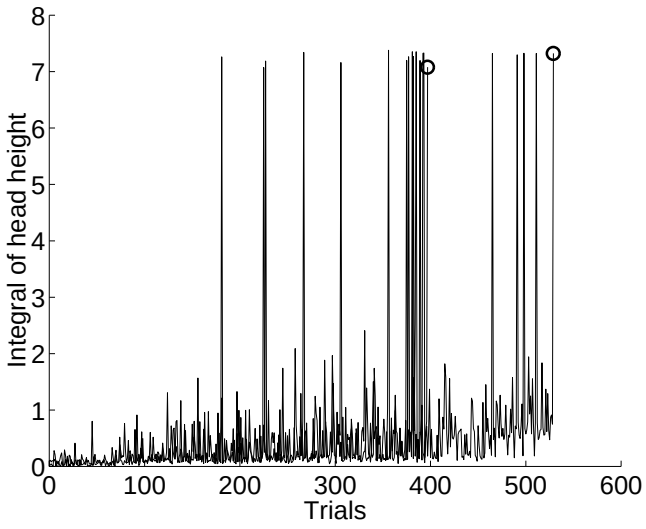


(a)

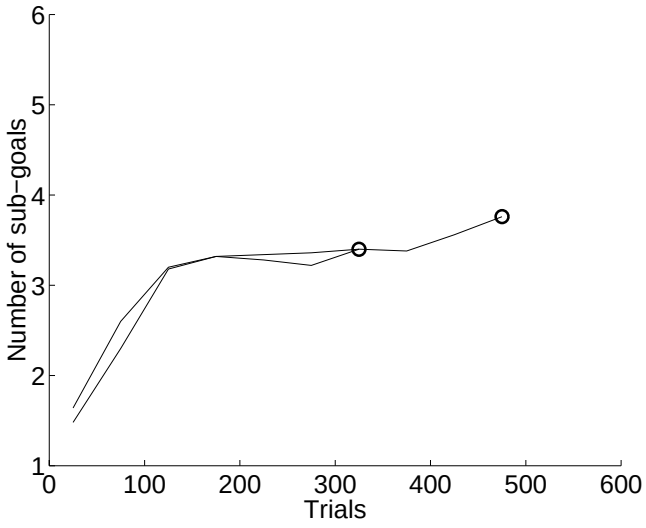


(b)

Figure 3.10 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, $\Delta\theta_1 = 30$ [deg] (b) Average number of sub-goals in each set of 50 trials, $\Delta\theta_1 = 30$ [deg]



(a)



(b)

Figure 3.11 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, $\Delta\theta_1 = 50$ [deg] (b) Average number of sub-goals in each set of 50 trials, $\Delta\theta_1 = 50$ [deg]

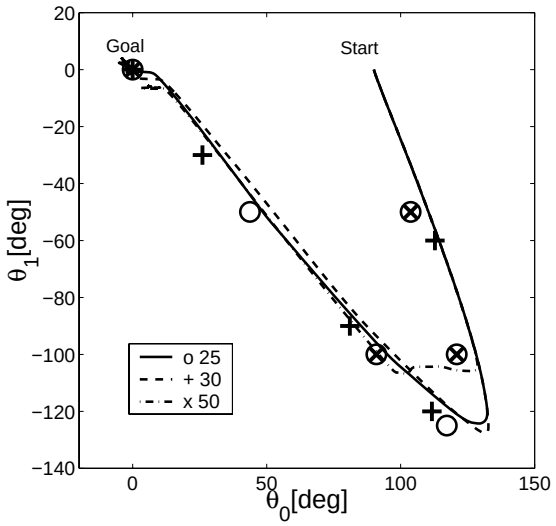


Figure 3.12 Stand-up trajectories and sub-goals using different $\Delta\theta_1$

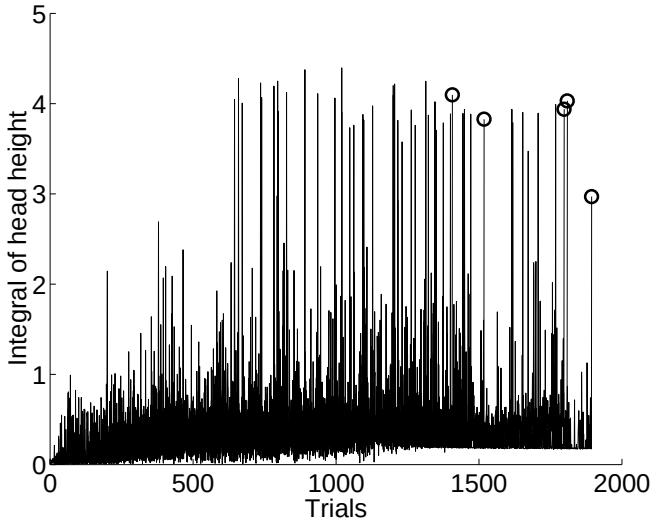
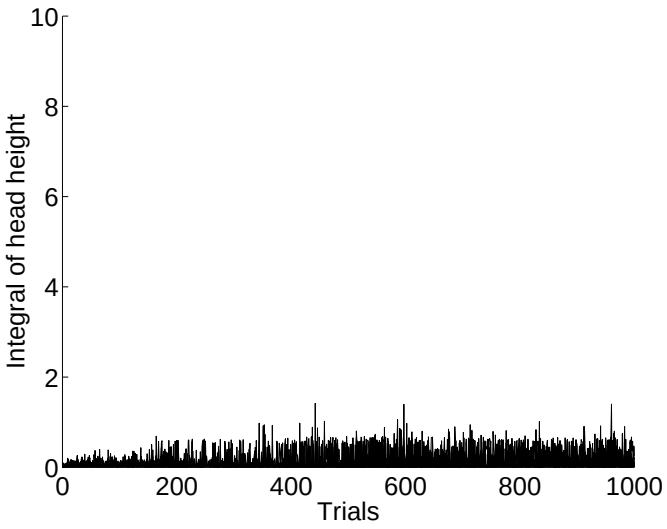
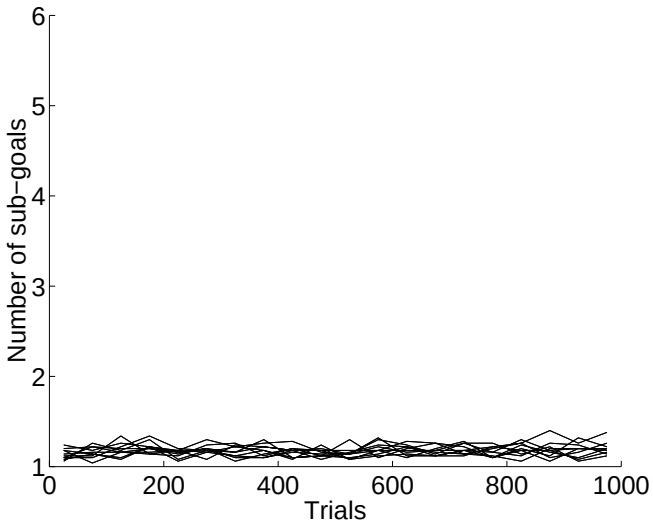


Figure 3.13 Time course of learning with plain architecture. Circles show 10th successful stand-up, upon which a simulation run was terminated.



(a)



(b)

Figure 3.14 Time course of learning without R_{sub}. (a)Performance index. (b)Average number of sub-goals in each set of 50 trials.

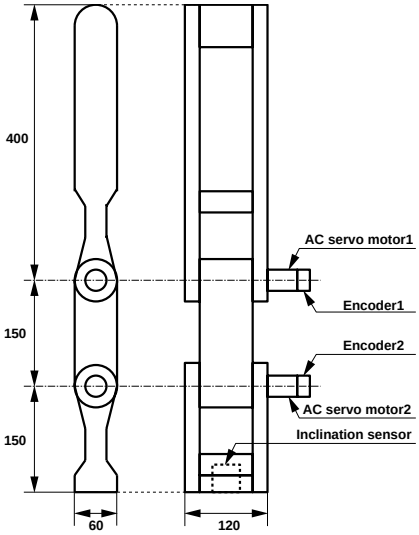


Figure 3.15 Real robot configuration

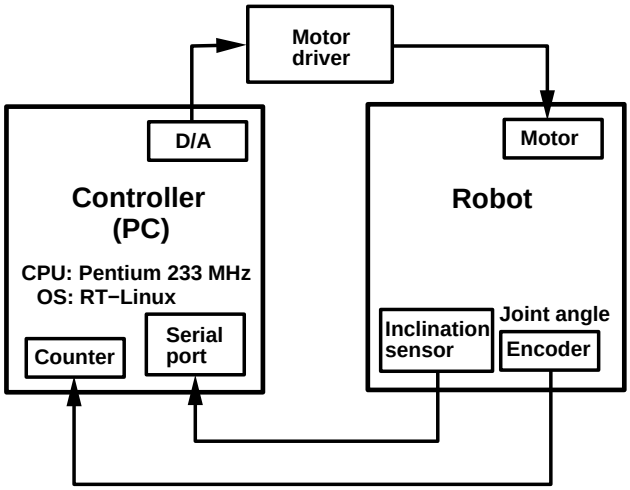


Figure 3.16 System configuration

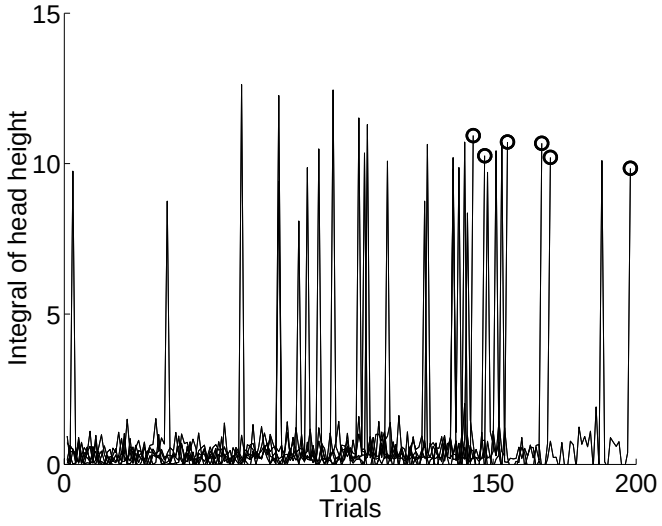


Figure 3.17 Time course of learning with real robot. Circles show 5th successful stand-up, upon which each experiment was terminated.

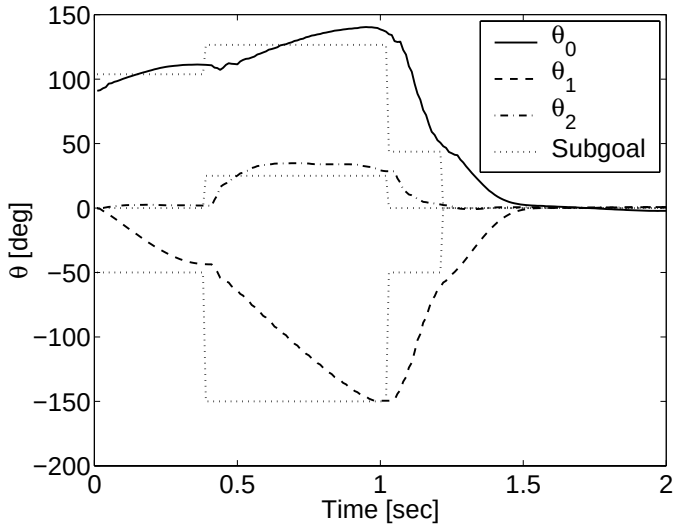
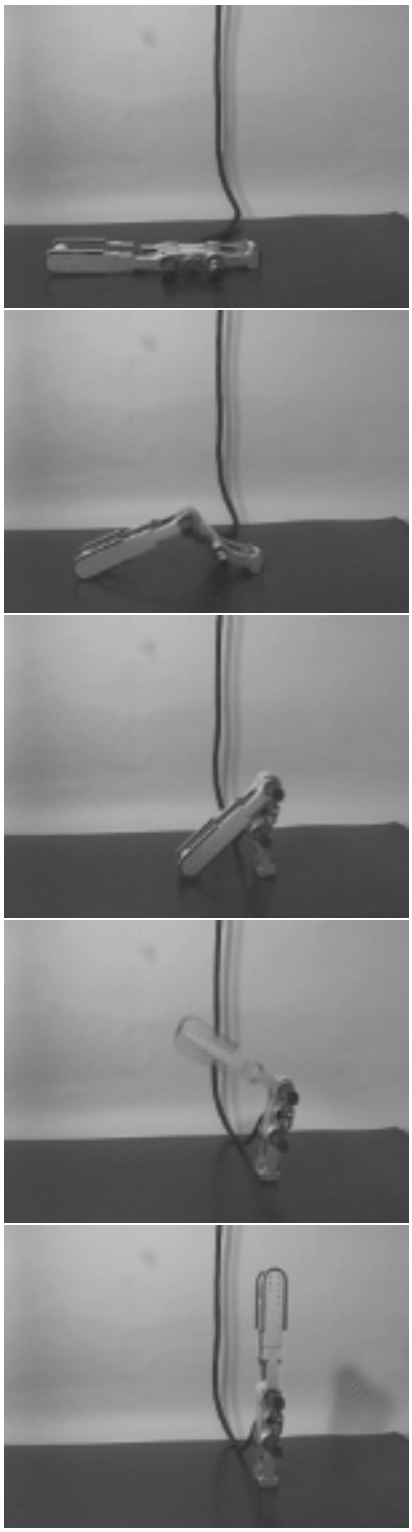


Figure 3.18 Example of a time course of a stand-up trajectory and a sub-goal sequence (θ_0:pitch angle, θ_1, θ_2:joint angle).



44 Figure 3.19 An example of a stand-up trajectory using the real robot

Chapter 4

Imitation Learning

4.1. Introduction

In this chapter, we show how imitation learning improve the learning speed in the hierarchical reinforcement learning framework. Imitation learning[30, 21] is the learning method in which learners acquire useful information for achieve tasks from their teacher. However, different from usual supervised learning, the learner can not obtain full information of desired trajectories, but the learner obtain the partial information of desired trajectories. In such a case, the learner have to construct desired trajectories from insufficient information which acquired from their teacher. In this study, we consider to use RL to realize desired trajectories or control policy with using the information from their teacher. We can say that the hierarchical architecture is a suitable representation for imitation learning, because the information from their teacher can be considered as the upper-level state and construct desired control policy by using the lower-level through trial and error.

We consider stand-up task for verify how imitation learning contribute to improve learning speed. Here, we assume that the learner can obtain the stand-up trajectories in the pitch and joint angle space which are absolutely not enough to construct control policy to achieve stand-up behavior. Then, the robot accomplish stand-up task mainly by using the lower-level learning. In such a case, we can expect that the stand-up task is achieved much faster than without the information from the teacher, the stand-up trajectory.

In our proposed method, the information of the stand-up behavior is converted to the action-value function at the subgoal point in the upper level. For imple-

menting above method, we redefine previously proposed hierarchical RL in Semi Markov Decision [29] process (SMDP) and introduce the idea of *option* is the temporary extended action[?].

4.2. Semi-Markov Decision Process

semi-Markov model consists of four tuples $< S, A, P, Z >$, where S denotes state space, A denotes action space, P denotes a set of transition probabilities, and Z denotes a joint probability distribution of transition times and transition costs dependent on the state and the action.

4.3. Options

The term *options* represent generalization of primitive actions or temporally extended action. Options consist of three tuples $< I, \pi, \beta >$, where π denotes policy, β denotes a termination condition, and I denotes an input set $I \subseteq S$. If the option is selected, then actions are taken according to the policy π until the option terminates stochastically according to β. Sutton [?] prove that options with Markov decision process can consider as SMDP because Z can be defined by policy π and a termination condition β, and input set I is a state space of SMDP.

Here we consider relationship between SMDP (option with MDP) and our hierarchical RL framework explained in chapter 3. In the upper level, the learner acquire their policy in SMDP because the upper level output next subgoal and select policy which realized by the lower-level controller. The termination condition is given as the subgoal area. Especially, we concern the case that the dimension of the upper-level state space is reduced compare to the dimension of the lower-level. However, we need some modification to fit our hierarchical RL to SMDP framework.

First, we modify the reward for the upper level. The upper level learner gets discounted accumulated reward before encounter the termination condition of a option (the upper level action).

In discreate time and space formulation, the reward for the option o is given

as

$$r_s^o = E\{r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k}\}, \tag{4.1}$$

and transition probability of SMDP is represented as

$$p_{ss'}^o = \sum_{i=1}^{\infty} \gamma^i Pr(s_{t+k} = s', k = i) \tag{4.2}$$

Then the value function is defined as

$$\begin{aligned} V^\mu(s) &= E\{r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V^\mu(s_{t+k})\} && (4.3)\\ &= \sum_o \mu(s,o)[r_s^o + \sum_{s'} p_{ss'}^o V^\mu(s')]. && (4.4) \end{aligned}$$

The definition is a Bellman equation for SMDP. Accordingly, the action-value function is defined as

$$\begin{aligned} Q^\mu(s,o) &= E\{r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V^\mu(s_{t+k})\} && (4.5)\\ &= E\{r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k \sum_{o'} \mu(s_{t+k}, o') Q^\mu(s_{t+k}, o')\} && (4.6)\\ &= r_s^o + \sum_{s'} p_{ss'}^o \sum_{o'} \mu(s', o') Q^\mu(s', o'). && (4.7) \end{aligned}$$

Then, the optimal value function is defined as

$$\begin{aligned} V_O^*(s) &= \max_\mu V^\mu(s) && (4.8)\\ &= \max_o E\{r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} && (4.9)\\ &\quad + \gamma^k V^*(s_{t+k})\} && (4.10)\\ &= \max_o [r_s^o + \sum_{s'} p_{ss'}^o V^*(s')] && (4.11)\\ &= \max_o E\{r + \gamma^k V^*(s')\} && (4.12) \end{aligned}$$

Accordingly, the optimal action-value function is defined as

$$\begin{aligned} Q^*(s,o) &= \max_\mu Q^\mu(s,o) && (4.13)\\ &= E\{r_{t+1} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V^*(s_{t+k})\} && (4.14)\\ &= E\{r_{t+1} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k \max_{o'} Q^*(s_{t+k}, o')\} && (4.15)\\ &= r_s^o + \sum_{s'} p_{ss'}^o \max_{o'} Q^*(s', o') && (4.16)\\ &= E\{r + \gamma^k \max_{o'} Q^*(s', o')\} && (4.17) \end{aligned}$$

Then, we can introduce learning rule for action-value function which Sutton[?] named SMDP Q-learning

$$Q(s,o) \leftarrow Q(s,o) + \alpha[r + \gamma^k \max_a Q(s',a) - Q(s,o)], \tag{4.18}$$

where r denotes the cumulative discounted reward.

4.4. SMDP learning in continuous time and space

Our hierarchical RL framework can treat continuous time and space in the lower level. Then, we should redefine SMDP Q-learning in continuous time and space domain.

The definition of the value function in continuous time is given by

$$V(t) = \int_t^\infty e^{-\frac{s-t}{\tau}} r(s) ds. \tag{4.19}$$

The reward for the option o is given as

$$r_X^o = E\{\int_t^{t_T} e^{-\frac{s-t}{\tau}} r(t) ds\}, \tag{4.20}$$

where t_T denotes termination time. Transition probability is represented as

$$p^o_{\mathbf{X}\mathbf{X}'} = \int_t^\infty e^{-\frac{t}{\tau}} Pr(\mathbf{X}_{t_T} = \mathbf{X}', t = t_T). \tag{4.21}$$

Then, the Bellman equation for SMDP is

$$\begin{aligned} V^\mu(\mathbf{X}) &= E\{\int_t^{t_T} e^{-\frac{s-t}{\tau}} r(t) ds + e^{-\frac{t_T}{\tau}} V^\mu(\mathbf{X}')\} & (4.22)\\ &= \sum_o \mu(\mathbf{X}, o)[r^o_{\mathbf{X}} + \sum_{\mathbf{X}'} p^o_{\mathbf{X}\mathbf{X}'} V^\mu(\mathbf{X}')]. & (4.23) \end{aligned}$$

Accordingly, the Bellman equation for action-value function is

$$\begin{aligned} Q^\mu(\mathbf{X}, o) &= E\{\int_t^{t_T} e^{-\frac{s-t}{\tau}} ds + e^{-\frac{t_T}{\tau}} \sum_o \mu(\mathbf{X}', o') Q^\mu(\mathbf{X}', o')\} & (4.24)\\ &= r^o_s + \sum_{\mathbf{X}'} p^o_{\mathbf{X}\mathbf{X}'} \sum_{o'} \mu(\mathbf{X}', o') Q^\mu(\mathbf{X}', o') & (4.25) \end{aligned}$$

Then, the optimal value function is defined as

$$\begin{aligned} V_O^*(s) &= \max_\mu V^\mu(s) && (4.26)\\ &= \max_o E\{\int_t^{t_T} e^{-\frac{s-t}{\tau}} r(t)ds + e^{-\frac{t_T}{\tau}} V^*(\mathbf{X}_{t_T})\} && (4.27)\\ &= \max_o [r_{\mathbf{X}}^o + \sum_{\mathbf{X}'} p_{\mathbf{X}\mathbf{X}'}^o V^*(\mathbf{X}')] && (4.28)\\ &= \max_o E\{r + e^{-\frac{t_T}{\tau}} V^*(\mathbf{X}')\} && (4.29) \end{aligned}$$

Accordingly, the optimal action-value function is defined as

$$\begin{aligned} Q^*(s,o) &= \max_\mu Q^\mu(s,o) && (4.30)\\ &= E\{\int_t^{t_T} e^{-\frac{s-t}{\tau}} ds + e^{-\frac{t_T}{\tau}} V^*(\mathbf{X}_{t_T})\} && (4.31)\\ &= E\{\int_t^{t_T} e^{-\frac{s-t}{\tau}} ds + e^{-\frac{t_T}{\tau}} \max_{o'} Q^*(\mathbf{X}_{t_T}, o')\} && (4.32)\\ &= r_{\mathbf{X}}^o + \sum_{\mathbf{X}'} p_{\mathbf{X}\mathbf{X}'}^o \max_{o'} Q^*(\mathbf{X}', o') && (4.33)\\ &= E\{r + e^{-\frac{t_T}{\tau}} \max_{o'} Q^*(\mathbf{X}', o')\} && (4.34) \end{aligned}$$

Then, corresponding SMDP Q-learning rule is

$$Q_{T+1}(\mathbf{X}, o) \leftarrow Q_T(\mathbf{X}, o) + \alpha[r + e^{-\frac{t}{\tau}} \max_{o'} Q(\mathbf{X}', o') - Q(\mathbf{X}, o)]. \quad (4.35)$$

4.5. Global Reward and Local Reward

SMDP learning framework define the learning method for the upper level learner while the options are consist of pre-designed policies in the lower level. Then, the reward $r(t)$ in previous section represent the reward function for the upper level, global reward. However, without prior knowledge about policies in the lower level, the learner need to acquire policies in the lower level. In other words, the learner need to learn options. For option learning [?], we put the subgoal reward, local reward, at the each state of the upper level. Then, when the lower level learner reach the subgoal which is the termination condition for the upper level SMDP, the lower-level learner gets reward. As a result, options are learned in the lower level.

4.6. Upper level learning

We derive the prediction error of the state-value function as

$$\delta(T) = r^o_{\mathbf{X}} + e^{-\frac{t_T}{\tau}} \max_{\mathbf{U}'} Q(\mathbf{X}', \mathbf{U}') - Q(\mathbf{X}, \mathbf{U}) \tag{4.36}$$

where $\mathbf{X}$ is the state and $\mathbf{U}$ is the action in the upper level. Then, the update rule for the state-value function at the sub-goal point is given as Then, the update rule for the action-value function at the sub-goal point is given as

$$Q_{T+1}(\mathbf{X}(t), \mathbf{U}) = Q_T(\mathbf{X}(t), \mathbf{U}) + \alpha\delta_Q(T) \tag{4.37}$$

4.7. Lower-level learning

The continuous time prediction error in the lower level is given as

$$\delta(t) = r(t) - \frac{1}{\tau}V(\mathbf{x}(t)) + \frac{dV(\mathbf{x}(t))}{dt} \tag{4.38}$$

where V($\mathbf{x}$;$\mathbf{v}$) is a function approximator with parameter $\mathbf{v} = (v_1, v_2, \cdots, v_n)$. The update rule for the lower-level critic is given as

$$\dot{v}_i = \alpha\delta(t)e_i(t) \tag{4.39}$$

where α is a learning rate. e_i is a eligibility trace for the paramenter v_i.

4.8. Simulation

In the above sections, we define SMDP learning and option learning methods for hierarchical RL. Again, our hierarchical RL which can treat continuous system in the lower level and consider reduced dimension in the upper level has advantage to apply to real world control task which has high-dimensional state space. Then, we apply our hierarchical RL method to the stand-up task with 3-link 2-joint robot (see Fig.3.1).

4.8.1 Learning Stand-up task

In the stand-up task, the upper level learn a appropriate posture sequence for stand-up, and the lower level learn a controller outputs torque at joints to achieve the posture the upper level intended to reach as a next state.

Upper level learning (SMDP Q-learning)

We can implement upper-level learning in the SMDP Q-learning framework. We define the state of SMDP in the upper level as $\mathbf{X} = (\theta_m, \theta_1, \theta_2)$, where θ_m is the angle from the ground to the center of mass, θ_1 is the hip joint angle, and θ_2 is the knee joint angle (see Fig.3.1). Then, the number of the state dimension is reduced because the upper level does not concern angular velocities.

Each action (option) of the upper level is the next target posture.

The reward for the upper level is given according to the height of the robot's head.

$$r = 3.0\frac{y}{h} \tag{4.40}$$

Lower level learning (Option learning)

The lower-level learning can be consider as the option learning for the upper level. For the lower-level learning, we use the subgoal reward with which the lower level gets highest reward at the termination condition of the upper level. Then, the lower level learn to achieve subgoal points. We consider $\mathbf{x} = (\theta_0, \theta_1, \theta_2, \dot{\theta}_0, \dot{\theta}_1, \dot{\theta}_2)$ as the lower-level state.

The action of the lower level is given as the combination of linear and non-linear function approximator $f(\mathbf{x}; \mathbf{a})$ with parameter $\mathbf{a} = (a_1, a_2, \cdots, a_n)$ as

$$u = f(\mathbf{x}) + \mathbf{b}\mathbf{x} + c, \tag{4.41}$$

where $\mathbf{b} = (b_1, b_2, \cdots b_n)$ and c are parameters of linear function approximator. Note that we initialize the linear function approximator as the PD servo controller in which target joint angle is given from the next target posture the upper level intended to achieve.

The reward for the lower level is given according to the distance from the target posture which is the subgoal for the lower level and termination condition for the upper level option as

$$r(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \exp\left(-\frac{||\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}||^2}{s_\theta^2}\right) - 1, \tag{4.42}$$

When the lower-level learner achieve the subgoal area or fall down, the learner

gets extra reward as

$$r(t) = \begin{cases} \exp\left(-\frac{||\dot{\boldsymbol{\theta}}(t) - \hat{\dot{\boldsymbol{\theta}}}||^2}{s_{\dot{\theta}}^2}\right) & \text{(sub-goal achieved)} \\ -1.0 & \text{(The robot falls down)} \end{cases} \tag{4.43}$$

Results

Results show that the robot successfully stand-up with 747 trials by using modified hierarchical RL framework which considering SMDP and option.

4.9. Imitation Learning

RL in the high-dimensional state space usually took long time to accomplish the task. However, when the learner can use prior knowledge, sometimes the task is accomplished very quickly. We construct the imitation learning framework as

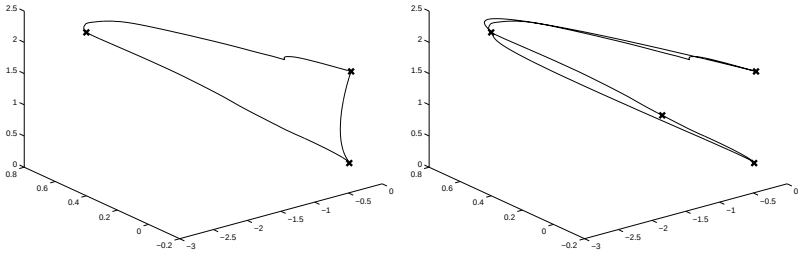
1. Suppose the learner can get target trajectory from teacher's motion in joint angle space.

2. By using the target trajectory, the learner detects via-points by using minimum-jerk criteria.

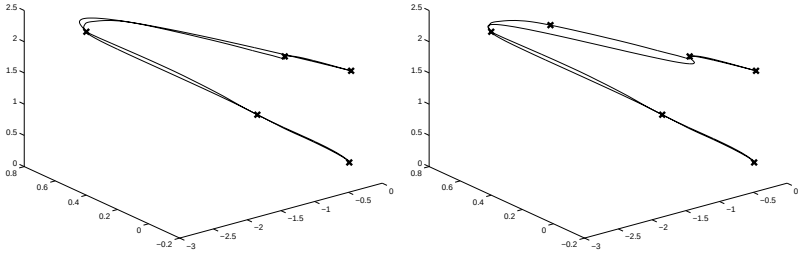
3. The learner calculates action-value at the via-points by using the target trajectory and the reward model.

4. The learner selects the sub-goal points near to the via-points. Then, the learner sets the calculated action-value as initial action-value at the sub-goal points.

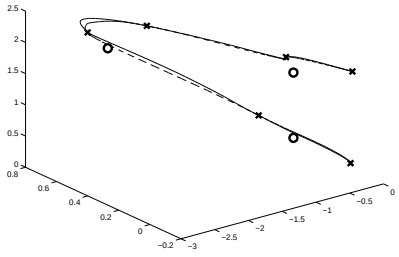
By using this imitation learning framework, the robot successfully learned to stand up in 242 trials. Detected via-points and state-value at each via-points are shown in Figure 4.1, 4.2.



(a) (b)



(c) (d)



(e)

Figure 4.1 Procedure of detecting via-points by using minimum jerk criteria

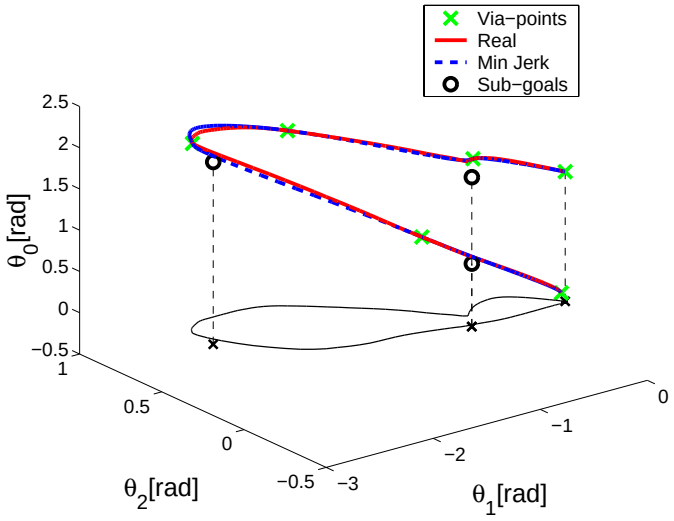


Figure 4.2 Detected via-points, subgoal points and minimum-jerk trajectory

4.10. Conclusions

We proposed a imitation learning method for our hierarchical reinforcement learning. In this study, we used the target trajectory in joint angle space acquired from teacher's motion. Then, we detect the via-points from the trajectory for using via-points as prior knowledge of the subgoal points. The stand-up task was accomplished about four times faster than without using imitation learning.

Chapter 5

Robust Reinforcement Learning

5.1. Introduction

In this chapter, we propose a new reinforcement learning paradigm that we call "Robust Reinforcement Learning (RRL)." Plain, model-free reinforcement learning (RL) is desperately slow to be applied to on-line learning of real-world problems. Thus the use of environmental models have been quite common both for on-line action planning [10] and for off-line learning by simulation [25]. However, no model can be perfect and modeling errors can cause unpredictable results, sometimes worse than with no model at all. In fact, robustness against model uncertainty has been the main subject of research in control community for the last twenty years and the result is formalized as the "H^∞" control theory [43].

In general, a modeling error causes a deviation of the real system state from the state predicted by the model. This can be re-interpreted as a disturbance to the model. However, the problem is that the disturbance due to a modeling error can have a strong correlation and thus standard Gaussian assumption may not be valid. The basic strategy to achieve robustness is to keep the sensitivity γ of the feedback control loop against a disturbance input small enough so that any disturbance due to the modeling error can be suppressed if the gain of mapping from the state error to the disturbance is bounded by $1/\gamma$. In the H^∞paradigm, those 'disturbance-to-error' and 'error-to-disturbance' gains are measured by a max norms of the functional mappings in order to assure stability for any modes of disturbance.

In the following we briefly introduce the H^∞paradigm and show that design of a robust controller can be achieved by finding a min-max solution of a value function, which is formulated as x1Hamilton-Jacobi-Isaacs (HJI) equation. We then derive on-line algorithms for estimating the value functions and for simultaneously deriving the worst disturbance and the best control that, respectively, maximizes and minimizes the accumulated reward.

We test the validity of the algorithms first in a linear inverted pendulum task. It is verified that the value function as well as the disturbance and control policies derived by the on-line algorithm coincides with the analytical solution given by H^∞theory. We then compare the performance of the robust RL algorithm with a standard model-based RL in a nonlinear task of pendulum swing-up [10]. It is shown that robust RL controller can accommodate changes in the weight and the friction of the pendulum, which a standard RL controller cannot cope with.

5.2. H^∞Control

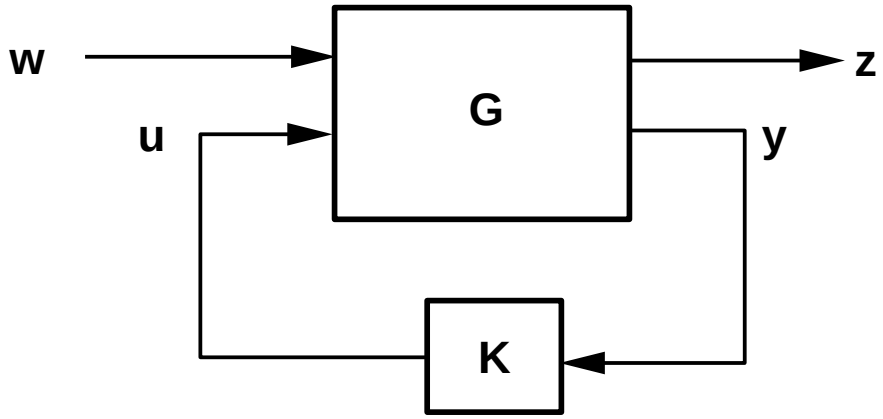


Figure 5.1 Generalized Plant and Controller

The standard H^∞control [43] deals with a system shown in Fig.5.1, where G is the plant, K is the controller, $\mathbf{u}$ is the control input, $\mathbf{y}$ is the measurement available to the controller, $\mathbf{w}$ is unknown disturbance, and $\mathbf{z}$ is the error output that is desired to be kept small. In general, the controller K is designed to

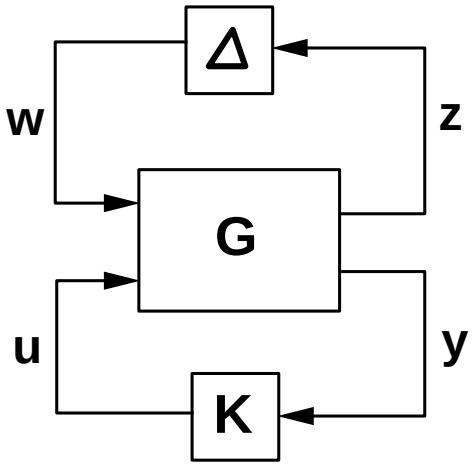


Figure 5.2 Small Gain Theorem

stabilize the closed loop system based on a model of the plant G. However, when there is a discrepancy between the model and the actual plant dynamics, the feedback loop could be unstable. The effect of modeling error can be equivalently represented as a disturbance $\mathbf{w}$ generated by an unknown mapping Δ of the plant output $\mathbf{z}$, as shown in Fig.5.2.

The goal of H^∞control problem is to design a controller K that brings the error $\mathbf{z}$ to zero while minimizing the H^∞norm of the closed loop transfer function from the disturbance $\mathbf{w}$ to the output $\mathbf{z}$

$$||T_{zw}||_\infty = \sup_{\mathbf{w}} \frac{||\mathbf{z}||_2}{||\mathbf{w}||_2} = \sup_{\omega} \bar{\sigma}(T_{zw}(j\omega)). \tag{5.1}$$

Here, $||\bullet||_2$ denotes L_2 norm and $\bar{\sigma}$ denotes maximum singular value. The small gain theorem assures that if $||T_{zw}||_\infty \leq \gamma$, then the system shown in Fig. 5.1(b) will be stable for any stable mapping $\Delta : \mathbf{z} \mapsto \mathbf{w}$ with $||\Delta||_\infty < \frac{1}{\gamma}$.

5.2.1 Min-max Solution to H^∞Problem

We consider a dynamical system

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{w}). \tag{5.2}$$

H^∞control problem is equivalent to finding a control output $\mathbf{u}$ that satisfies a constraint

$$V = \int_0^\infty (\mathbf{z}^T(t)\mathbf{z}(t) - \gamma^2 \mathbf{w}^T(t)\mathbf{w}(t))dt \leq 0 \tag{5.3}$$

against all possible disturbance $\mathbf{w}$ with $\mathbf{x}(0) = \mathbf{0}$, because it implies

$$||T_{zw}||_\infty{}^2 = \sup_{\mathbf{w}} \frac{||\mathbf{z}||_2{}^2}{||\mathbf{w}||_2{}^2} \leq \gamma^2. \tag{5.4}$$

We can consider this problem as differential game[38] in which the best control output $\mathbf{u}$ that minimizes V is sought while the worst disturbance $\mathbf{w}$ that maximizes V is chosen. Thus an optimal value function V^* is defined as

$$V^* = \min_{\mathbf{u}} \max_{\mathbf{w}} \int_0^\infty (\mathbf{z}^T(t)\mathbf{z}(t) - \gamma^2 \mathbf{w}^T(t)\mathbf{w}(t))dt. \tag{5.5}$$

The condition for the optimal value function is given by

$$0 = \min_{\mathbf{u}} \max_{\mathbf{w}} [\mathbf{z}^T\mathbf{z} - \gamma^2 \mathbf{w}^T\mathbf{w} + \frac{\partial V^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}, \mathbf{w})] \tag{5.6}$$

which is known as Hamilton-Jacobi-Isaacs (HJI) equation. From (5.6), we can derive the optimal control output $\mathbf{u}_{op}$ and the worst disturbance $\mathbf{w}_{op}$ by solving

$$\frac{\partial \mathbf{z}^T\mathbf{z}}{\partial \mathbf{u}} + \frac{\partial V}{\partial \mathbf{x}} \frac{\partial f(\mathbf{x}, \mathbf{u}, \mathbf{w})}{\partial \mathbf{u}} = 0 \tag{5.7}$$

$$\frac{\partial \mathbf{z}^T\mathbf{z}}{\partial \mathbf{w}} - 2\gamma^2 \mathbf{w} + \frac{\partial V}{\partial \mathbf{x}} \frac{\partial f(\mathbf{x}, \mathbf{u}, \mathbf{w})}{\partial \mathbf{w}} = 0. \tag{5.8}$$

5.3. Robust Reinforcement Learning

Here we consider a continuous-time formulation of reinforcement learning [10] with the system dynamics

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{5.9}$$

and the reward $r(\mathbf{x}, \mathbf{u})$. The basic goal is to find a policy $\mathbf{u} = g(\mathbf{x})$ that maximizes the cumulative future reward

$$\int_t^\infty e^{-\frac{s-t}{\tau}} r(\mathbf{x}(s), \mathbf{u}(s))ds \tag{5.10}$$

for any given state $\mathbf{x}(t)$, where τ is a time constant of evaluation. However, a particular policy that was optimized for a certain environment may perform badly

when the environmental setting changes. In order to assure robust performance under changing environment or unknown disturbance, we introduce the notion of worst disturbance in H^∞control to the reinforcement learning paradigm.

In this framework, we consider an augmented reward

$$q(t) = r(\mathbf{x}(t), \mathbf{u}(t)) + \omega(\mathbf{w}(t)), \tag{5.11}$$

where $s(\mathbf{w}(t))$ is an additional reward for withstanding a disturbing input, for example,

$$\omega(\mathbf{w}) = \gamma^2 \mathbf{w}^T \mathbf{w}. \tag{5.12}$$

The augmented value function is then defined as

$$V(\mathbf{x}(t)) = \int_t^\infty e^{-\frac{s-t}{\tau}} q(\mathbf{x}(s), \mathbf{u}(s), \mathbf{w}(s)) ds. \tag{5.13}$$

The optimal value function is given by the solution of a variant of HJI equation

$$\frac{1}{\tau} V^*(\mathbf{x}) = \max_{\mathbf{u}} \min_{\mathbf{w}} [r(\mathbf{x}, \mathbf{u}) + \omega(\mathbf{w}) + \frac{\partial V^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}, \mathbf{w})]. \tag{5.14}$$

In the robust reinforcement learning (RRL) paradigm, the value function is update by using the temporal difference (TD) error [10]

$$\delta(t) = q(t) - \frac{1}{\tau} V(t) + \dot{V}(t) \tag{5.15}$$

while the best action and the worst disturbance are generated by maximizing and minimizing, respectively, the right hand side of HJI equation

$$r(\mathbf{x}, \mathbf{u}) + \omega(\mathbf{w}) + \frac{\partial V^*}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}, \mathbf{w}). \tag{5.16}$$

We use a function approximator to implement the value function $V(\mathbf{x}(t); \mathbf{v})$, where $\mathbf{v} = (v_1, \cdots, v_n)$ is a parameter vector. As in the standard continuous-time RL, we define eligibility trace for a parameter v_i[10] as

$$e_i(s) = \int_0^s e^{-\frac{s-t}{\kappa}} \frac{\partial V(t)}{\partial v_i} dt, \tag{5.17}$$

where κ is the time constant of the eligibility trace. We can then derive learning rule for value function approximator [10] as

$$\dot{v}_i = \eta \delta(t) e_i(t), \tag{5.18}$$

where η denotes learning rate. The eligibility trace (5.17) is updated by

$$\dot{e}_i(t) = -\frac{1}{\kappa} e_i(t) + \frac{\partial V(t)}{\partial v_i}. \tag{5.19}$$

Accordingly, we do not assume $f(\mathbf{x} = 0) = 0$. By comparing (5.6) and (5.14), we can see that the output error $\mathbf{z}^T\mathbf{z}$ in H^∞framework is generalized as an arbitrary reward function $r(\mathbf{x}, \mathbf{u})$ in RRL. Furthermore, the sign of the value function is flipped and a discount factor is introduced in RRL framework.

5.3.1 Actor-Disturber-Critic

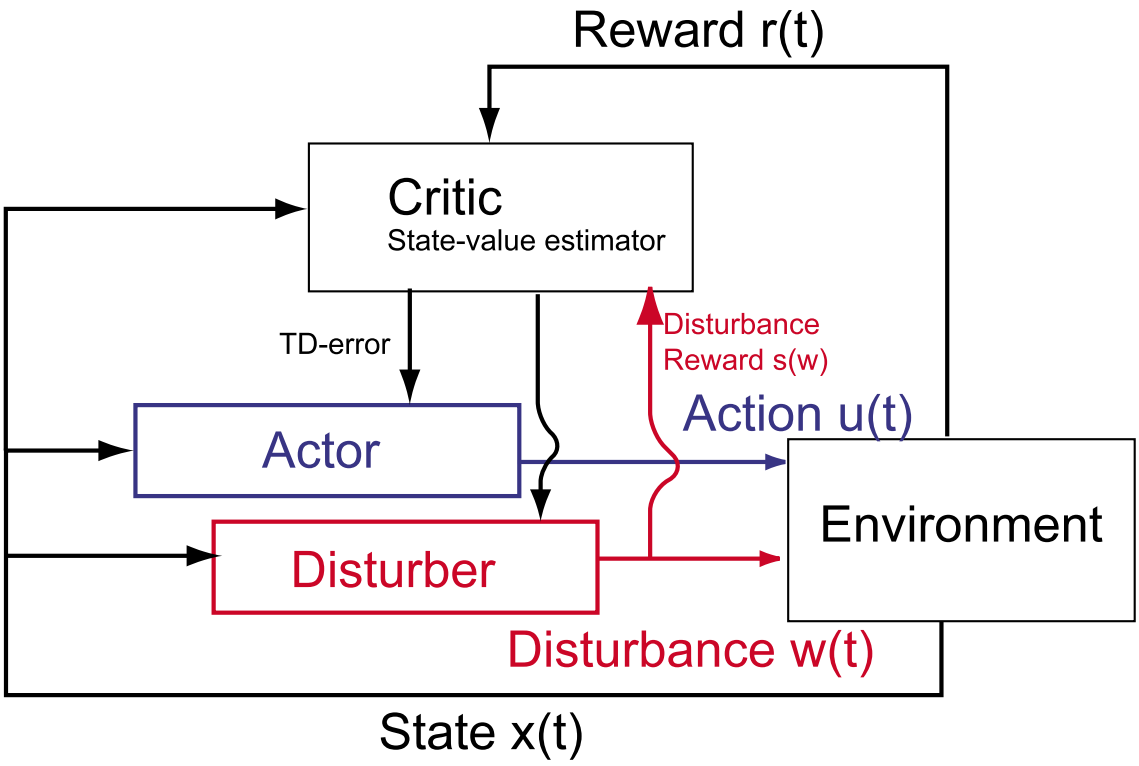


Figure 5.3 Actor-disturber-critic architecture

By extending the actor-critic architecture[3], we propose the actor-disturber-critic architecture, shown in Figure 5.3 to implement robust RL in a model-free fashion. We define the policies of the actor and the disturber as $\mathbf{u}(t) =$

$A_u(\mathbf{x}(t);\mathbf{v}^u)+\mathbf{n}_u(t)$ and $\mathbf{w}(t) = A_w(\mathbf{x}(t);\mathbf{v}^w)+\mathbf{n}_w(t)$, respectively, where $A_u(\mathbf{x}(t);\mathbf{v}^u)$ and $A_w(\mathbf{x}(t);\mathbf{v}^w)$ are function approximators with parameter vectors $\mathbf{v}^u$ and $\mathbf{v}^w$, and $\mathbf{n}_u(t)$ and $\mathbf{n}_w(t)$ are noise terms for exploration. The parameters of the actor and the disturber are updated by

$$\dot{v}_i^u = \eta^u \delta(t)\mathbf{n}_u(t)\frac{\partial A_u(\mathbf{x}(t);\mathbf{v}^u)}{\partial v_i^u} \tag{5.20}$$

$$\dot{v}_i^w = -\eta^w \delta(t)\mathbf{n}_w(t)\frac{\partial A_w(\mathbf{x}(t);\mathbf{v}^w)}{\partial v_i^w}, \tag{5.21}$$

where η^u and η^w denote the learning rates.

5.3.2 Robust Policy by Value Gradient

When the augmented reward function $q(\mathbf{x},\mathbf{u},\mathbf{w})$ is convex with respect to the action $\mathbf{u}$ and disturbance $\mathbf{w}$, the HJI equation has a unique solution, and we can derive a closed-form expression of the greedy policy.

Here, we assume that the augmented reward $q(\mathbf{x},\mathbf{u},\mathbf{w})$ can be separated into three parts: the reward for state $R(\mathbf{x})$, the cost for action $S(\mathbf{x},\mathbf{u})$ and the cost for disturbance $\Omega(\mathbf{x},\mathbf{w})$ while the reward $R(\mathbf{x})$ given by the environment and unknown, the costs $S(\mathbf{x},\mathbf{u})$ and $\Omega(\mathbf{x},\mathbf{w})$ can be chosen as a part of the learning strategy. We specifically consider the case

$$q(\mathbf{x},\mathbf{u},\mathbf{w}) = R(\mathbf{x}) - \sum_{i=1}^{m} S_i(u_i) + \sum_{j=1}^{l} \Omega_j(w_j), \tag{5.22}$$

where $S_i()$ is a cost function for action variable u_i and $\Omega_j()$ is a cost function for disturbance variable w_j. In this case, the condition for the optimal action and the worst disturbance is given by

$$-S_i'(u_i) + \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial u_i} = 0 \qquad (i = 1,\ldots,m) \tag{5.23}$$

$$\Omega_j'(w_j) + \frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial w_j} = 0 \qquad (j = 1,\ldots,l) \tag{5.24}$$

where $\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial u_i}$ and $\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial w_j}$ are the ith and jth column vector of the $n \times m$ input gain matrix $\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial \mathbf{u}}$ and the $n \times l$ disturbance gain matrix $\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial \mathbf{w}}$, respectively.

We now assume that the input gains $\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial u_i}$ and $\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial w_j}$ are not dependent on $\mathbf{u}$ and $\mathbf{w}$; that is, the system is linear with respect to the input and the action cost function $S_i()$ and the disturbance cost function $\Omega_j()$ are convex. Then the above equations have unique solutions,

$$u_i = S_i'^{-1}\left(\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial u_i}\right) \tag{5.25}$$

$$w_j = \Omega_j'^{-1}\left(-\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial w_j}\right) \tag{5.26}$$

where $S_i'()$ and $\Omega_j'()$are monotonic functions. Accordingly, the best action and the worst disturbance are represented in vector notations as

$$\mathbf{u}_{op} = S'^{-1}\left(\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial \mathbf{u}}^T\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}^T\right) \tag{5.27}$$

$$\mathbf{w}_{op} = \Omega'^{-1}\left(-\frac{\partial f(\mathbf{x},\mathbf{u},\mathbf{w})}{\partial \mathbf{w}}^T\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}}^T\right). \tag{5.28}$$

5.3.3 Convex reward function with quadratic cost

Now we assume that an input-Affine model of the system dynamics and quadratic models of the costs for the inputs are available as

$$\begin{aligned}\dot{\mathbf{x}} &= f(\mathbf{x}) + g_1(\mathbf{x})\mathbf{u} + g_2(\mathbf{x})\mathbf{w}\\ q(\mathbf{x},\mathbf{u},\mathbf{w}) &= Q(\mathbf{x}) - \mathbf{u}^T R(\mathbf{x})\mathbf{u} + \gamma^2\mathbf{w}^T\mathbf{w}.\end{aligned} \tag{5.29}$$

In this case, the cost functions $S()$ and $\Omega()$ are given by

$$S(\mathbf{u}) = \mathbf{u}^T R(\mathbf{x})\mathbf{u} \tag{5.30}$$

$$\Omega(\mathbf{w}) = \gamma\mathbf{w}^T\mathbf{w} \tag{5.31}$$

then, we can derive the best action and the worst disturbance from (5.28).

$$\mathbf{u}_{op} = \frac{1}{2}R^{-1}(\mathbf{x})g_1^T(\mathbf{x})(\frac{\partial V}{\partial \mathbf{x}})^T \tag{5.32}$$

$$\mathbf{w}_{op} = -\frac{1}{2\gamma^2}g_2^T(\mathbf{x})(\frac{\partial V}{\partial \mathbf{x}})^T. \tag{5.33}$$

We can use the policies (5.32) and (5.33) using the value gradient $\frac{\partial V}{\partial \mathbf{x}}$ derived from the value function approximator.

5.3.4 Linear Quadratic Case

Here we consider a case in which a linear dynamic model and quadratic reward models are available as

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + B_1\mathbf{u} + B_2\mathbf{w} \\ q(\mathbf{x}, \mathbf{u}, \mathbf{w}) &= -\mathbf{x}^T Q\mathbf{x} - \mathbf{u}^T R\mathbf{u} + \gamma^2 \mathbf{w}^T \mathbf{w}. \end{aligned} \tag{5.34}$$

In this case, the value function is given by a quadratic form $V = -\mathbf{x}^T P\mathbf{x}$, where P is the solution of a Riccati equation

$$A^T P + PA + P(\frac{1}{\gamma^2} B_1 B_1^T - B_2 R^{-1} B_2^T)P + Q = \frac{1}{\tau} P. \tag{5.35}$$

Thus we can derive the best action and the worst disturbance as

$$\mathbf{u}_{op} = R^{-1} B_2^T P\mathbf{x} \tag{5.36}$$

$$\mathbf{w}_{op} = -\frac{1}{\gamma^2} B_1^T P\mathbf{x}. \tag{5.37}$$

5.4. Simulation

We test the robust RL algorithm in a task of swinging up a pendulum[10]. The dynamics of the pendulum is given by $ml^2\ddot{\theta} = -\mu\dot{\theta} + mgl\sin\theta + T$, where θ is the angle from the upright position , T is input torque, $\mu = 0.01$ is the coefficient of friction, $m = 1.0$[kg] is the weight of the pendulum, $l = 1.0$[m] is the length of the pendulum, and $g = 9.8[m/s^2]$ is the gravity acceleration. The state vector is defined as $\mathbf{x} = (x_1, x_2)^T = (\theta, \dot{\theta})^T$.

5.4.1 Linear Case

We first considered a linear problem in order to test if the value function and the policy learned by robust RL coincides with the analytic solution of H^∞control problem. Thus we use a locally linearized dynamics near the unstable equilibrium

point $\mathbf{x} = (0, 0)^T$. The matrices for the linear model are given by

$$A = \begin{pmatrix} 0 & 1 \\ \frac{g}{l} & -\frac{\mu}{ml^2} \end{pmatrix}, B_1 = \begin{pmatrix} 0 \\ \frac{1}{ml^2}, \end{pmatrix}, B_2 = \begin{pmatrix} 0 \\ 1, \end{pmatrix}, Q = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, R = 1. \tag{5.38}$$

The reward function is given by

$$q(t) = -\mathbf{x}^T Q \mathbf{x} - u^2 + \gamma^2 w^2, \tag{5.39}$$

where robustness criteria $\gamma = 2.0$.

The value function, $V = -\mathbf{x}^T P \mathbf{x}$, is parameterized by a symmetric matrix.

$$P = \begin{pmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{pmatrix} \tag{5.40}$$

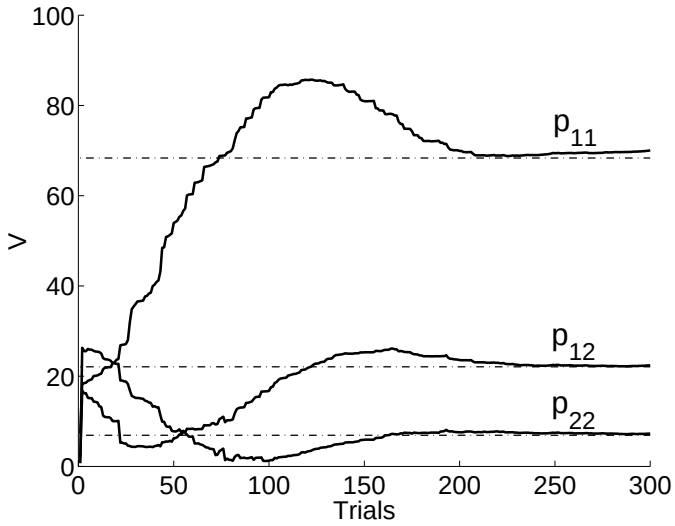
For on-line estimation of P, we define vectors $\tilde{\mathbf{x}} = (x_1^2, 2x_1x_2, x_2^2)^T$, $\mathbf{p} = (p_{11}, p_{12}, p_{22})^T$ and reformulate V as $V = -\mathbf{p}^T \tilde{\mathbf{x}}$. Each element of P is updated using recursive least squares method[4]. Note that we used pre-designed stabilizing controller as the initial setting of RRL controller for stable learning[4].

Learning of the value function

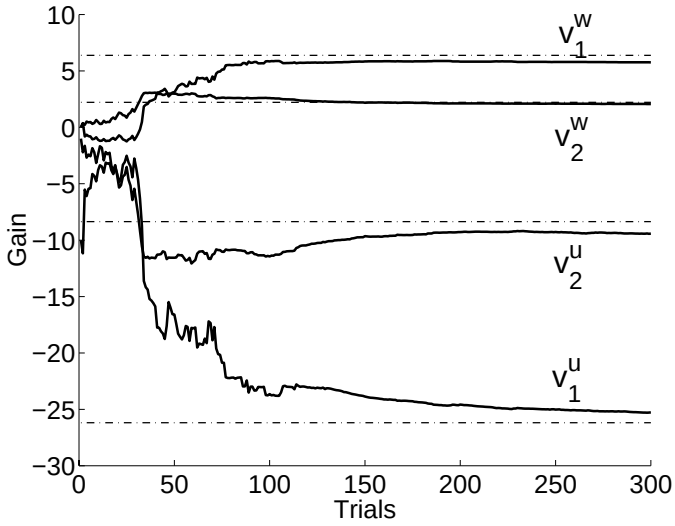
Here, we used the policy by value gradient shown in section 5.3.2. Figure 5.4(a) shows that each element of the matrix P converged to the analytic solution which is derived from the Riccati equation (5.35).

Actor-disturber-critic

Here we used robust RL implemented by the actor-disturber-critic shown in section 5.3.1. In the linear case, the actor and the disturber are represented as the linear controllers, $A_u(\mathbf{x}; \mathbf{v}^u) = \mathbf{v}^u \mathbf{x}$ and $A_w(\mathbf{x}; \mathbf{v}^w) = \mathbf{v}^w \mathbf{x}$, respectively. The parameters of the actor and the disturber converged to the values close to those of the policies in (5.36) and (5.37) derived from the Ricatti equation (5.35) (Fig. 5.4(b)).



(a) Elements of $\mathbf{p}$



(b) Elements of $\mathbf{v}$

Figure 5.4 Time course of (a)elements of vector $\mathbf{p} = (p_{11}, p_{12}, p_{22})$ and (b)elements of gain vector of the actor $\mathbf{v}^u = (v_1^u, v_2^u)$ and the disturber $\mathbf{v}^w = (v_1^w, v_2^w)$. The dash-dotted lines show the solution of the Ricatti equation.

5.4.2 Applying Robust RL to Non-linear Dynamics

We consider a non-linear dynamical system (5.29), where

$$\begin{aligned} f(\mathbf{x}) &= \begin{pmatrix} \dot{\theta} \\ \frac{g}{l} sin(\theta) - \frac{\mu\dot{\theta}}{ml^2} \end{pmatrix}, g_1(\mathbf{x}) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, g_2(\mathbf{x}) = \begin{pmatrix} 0 \\ \frac{1}{ml^2} \end{pmatrix} \\ Q(\mathbf{x}) &= \cos(\theta) - 1, R(\mathbf{x}) = 0.04 \end{aligned} \tag{5.41}$$

From (5.11), the augmented reward function is given by

$$q(t) = \cos(\theta) - 1 - 0.04u^2 + \gamma^2 w^2, \tag{5.42}$$

where robustness criteria $\gamma = 0.22$. For approximating the value function, we used Normalized Gaussian Network (NGnet)[10]. Note that the input gain $g(\mathbf{x})$ was also learned[10].

Fig.5.5 shows the value functions acquired by robust RL and standard model-based RL[10]. The value function acquired by robust RL has a shaper ridge (Fig.5.5(a)) that specify swing up trajectories than that learned with standard RL.

In Fig.5.6, we compare the robustness of the robust RL and the standard RL to the change of physical parameters. Both robust RL controller and the standard RL controller learned to swing up and hold a pendulum with the weight $m = 1.0$[kg] and the coefficient of friction $\mu = 0.01$ (Fig.5.6(a)). The robust RL controller took more swings, indicating its conservative control law.

With different weight $m = 3.0$[kg] and the coefficient of friction $\mu = 0.3$ (Fig.5.6(b)), the robust RL controller could successfully swing up pendulum while the standard RL controller could not. This result shows the robustness of the robust RL controller.

Comparison of the control performance with different robustness parameters

Here, we see how the robustness criteria γ affect the robust performance of the learned controller. We compare the controller learned by RRL with the five different robustness parameters ($\gamma = 0.40, 0.45, 0.50, 0.55, 0.6$) and standard RL (we can consider standard RL as the RRL with $\gamma = \infty$). We trained these

controllers with the pendulum with the mass $m = 1.0$[kg] and compared the maximum weight that each controller can swing up and stabilize at the upright position.

The results of the comparison shows more robust performance with smaller γ (see table 5.1). These results are consistent with H^{∞}control theory[43].

Table 5.1 Comparison with different robustness criteria

γ	0.4	0.45	0.5	0.55	0.6	∞
max m[kg]	2.50	2.18	2.00	1.88	1.76	1.70

Actor-Disturber-Critic

A possible problem in implementing model-free RRL by using the actor-disturber-critic framework for a non-linear system is the credit assignment problem of the results of each behavior for exploration.

To avoid this problem, we train the actor and the disturber in turns. We show the value function of RRL and stand RL in figure 5.7, and show the control and disturbance functions in figure 5.8. Note that the disturbance function has almost opposite shape of control function. The actor acquired in the actor-disturber-critic framework and the actor acquired in the actor-critic framework both could swing up and stabilize the pendulum with the weight $m = 1.0$[kg] and the friction $\mu = 0.01$ that is the learned environment. However, the actor acquired in the actor-critic framework could not swing up the pendulum with the weight $m = 1.3$[kg] and the friction $\mu = 0.2$ while the actor acquired in the actor-disturber-critic framework could successfully swing it up(see Fig.5.9).

5.5. Implementation of the Robust Reinforcement Learning in the Hierarchical Reinforcement Learning Framework

Here we show how proposed RRL can be implemented in the hierarchical RL framework explained in chapter 3 and add robustness to the learned controllers in the hierarchical RL. As a concrete example, we applied hierarchical RL with RRL to the 2-joint 3-link robot (see Fig.3.1).

5.5.1 Simulation

Here we assume that there is no significant difference between dynamical model used in computer simulation and real environment. Then, we concentrate on adding robustness to the lower-level controller by using RRL while the upper-level outputs the subgoal sequence learned previously in section 3.3.1. We implement RRL by using the actor-disturber-critic architecture (see Fig.5.3). We set the maximum torque of the actor as $u_{\max} = 24$[N.m] and the disturber as $w_{\max} = 10.0$[N.m].

Results

By using the normal hierarchical RL, the robot successfully stand up 27 times in 100 trials. On the other hand, by using the hierarchical RL with RRL, the robot successfully stand up 17 times in 100 trials in the learned environment (coefficient of the coulomb friction at hip joint is 0.5[N.m.sec/rad]). However, when we applied the robot to the new environment (coefficient of the coulomb friction at hip joint is 3.0[N.m.sec/rad]), the robot learned by the normal hierarchical RL only stand up 11 times in 100 trials. On the other hand the robot learned by the hierarchical RL with RRL successfully stand up 17 times in 100 trials. These results shows that the normal hierarchical RL is suitable for the learned environment, however, performance of the hierarchical RL with RRL was not drastically change in the difference environment, which means that RRL gives robust performance to the robot. However, still there is trade off that normal hierarchical RL show better performance for the learned environment.

5.6. Discussion

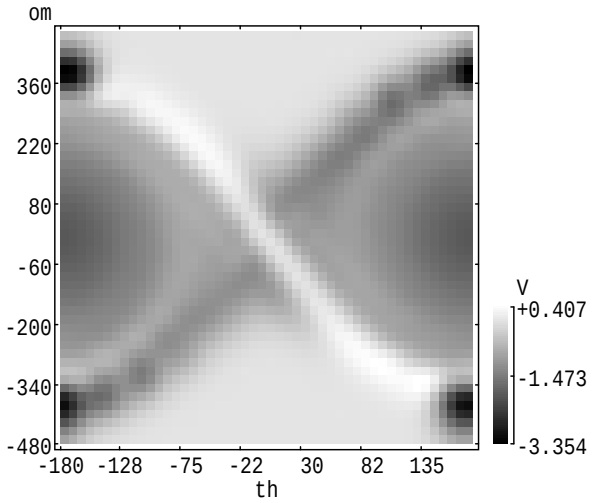
The H^∞control theory gives an analytical solution only for linear systems. For non-linear systems, there is no analytical way of solving the HJI equation. In order to derive a non-linear H^∞controller, the value function is usually derived by iteration by using dynamic programming[12, 5]. However, these methods need off-line calculation and environmental model. On the other hand, the robust RL can derive non-linear H^∞controller by on-line calculation and without environmental model.

The robust RL provides a new way of using min-max solution in RL problems. The min-max RL was applied to games like backgammon[36] and Othello[42]. However, in these studies, each player takes the same role. The min-max RL was also applied to a problem in which an airplane tries to avoid a missile and the missile tries to catch the airplane[11]. However, this study only focuses on linear control problems. We applied min-max RL in which each two players (the control agent and the disturbing agent) has different ability to the non-linear problem of pendulum swing-up.

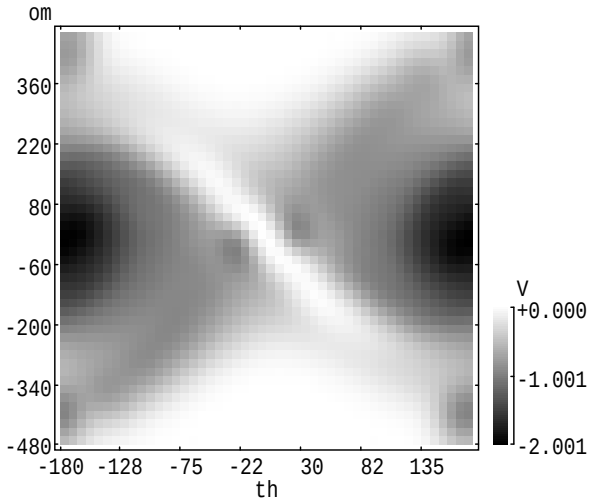
Risk sensitive control studies are also related to our RRL framework. Neuneier and Mihatsch proposed risk sensitive reinforcement learning[26]. In that work, they used an ad hoc update rule of value function and showed the convergence of their proposed method. In the RRL, we used appropriate objective function containing the control cost of disturbance to acquire the robust controller like H^∞control framework.

5.7. Conclusions

In this study, we proposed a new RL paradigm called "Robust Reinforcement Learning (RRL)" We showed that RRL can learn analytic solution of the H^∞controller in the linearized inverted pendulum dynamics and also showed that RRL can deal with modeling error which standard RL can not deal with in the non-linear inverted pendulum swing-up simulation example.

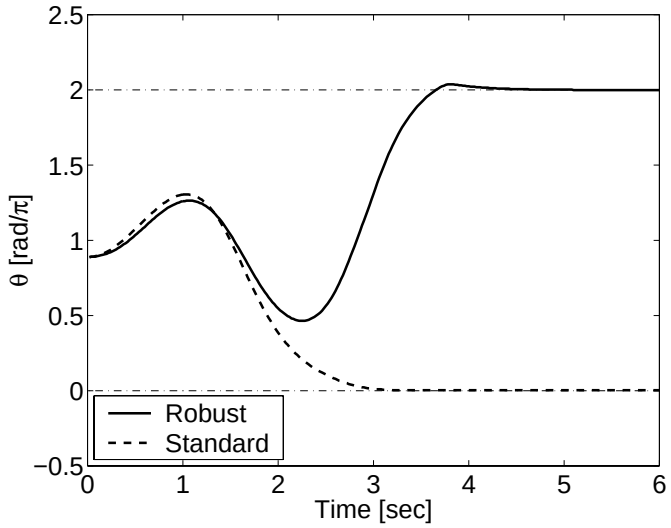


(a) Robust RL

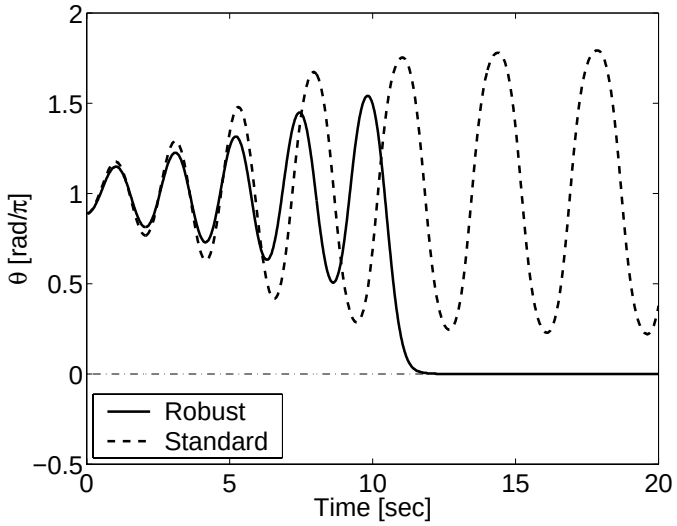


(b) Standard RL

Figure 5.5 Shape of the value function after 500 learning trials with $m = 1.0$[kg] and $l = 1.0$[m]

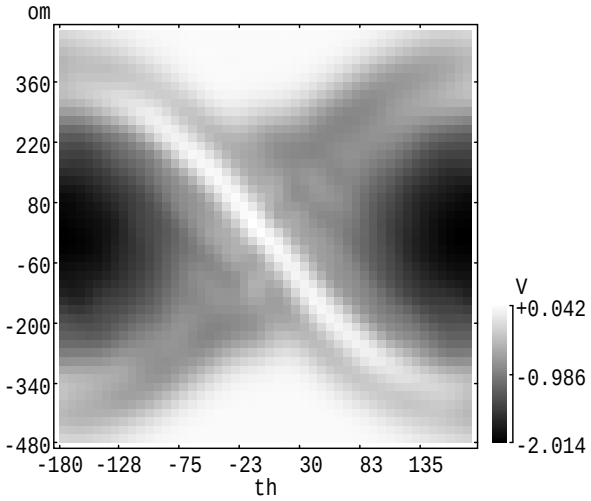


(a) $m = 1.0, \mu = 0.01$

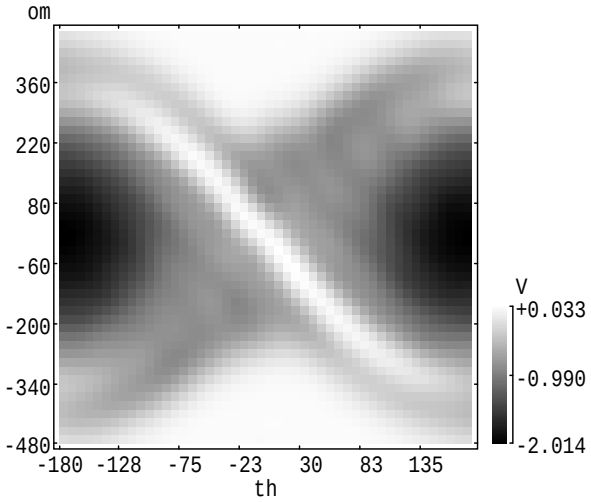


(b) $m = 3.0, \mu = 0.3$

Figure 5.6 Swing up trajectories with pendulum with different weight and friction. The dash-dotted lines show upright position.

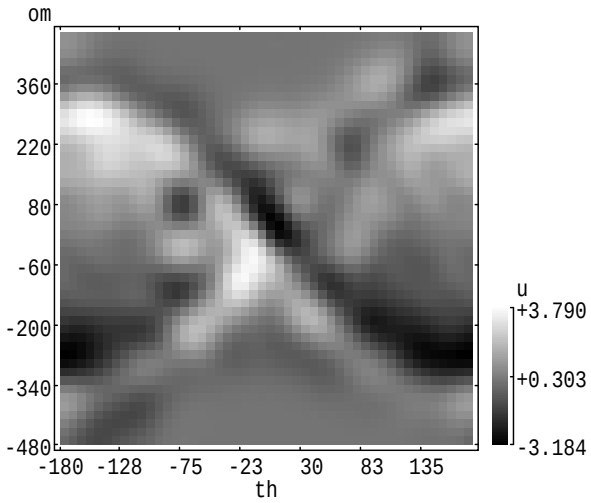


(a) Robust RL

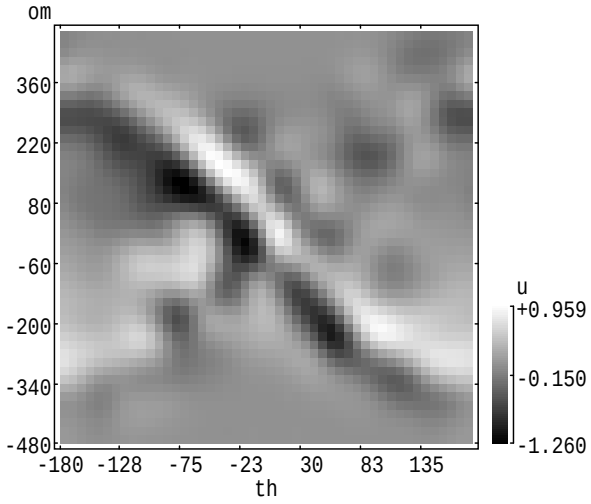


(b) Standard RL

Figure 5.7 Shape of the value function after 1000 learning trials with $m = 1.0$[kg] and $\mu = 0.01$

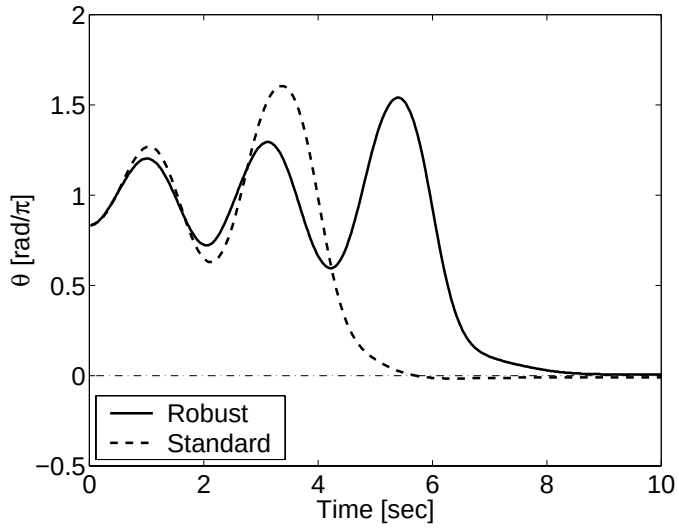


(a) Actor

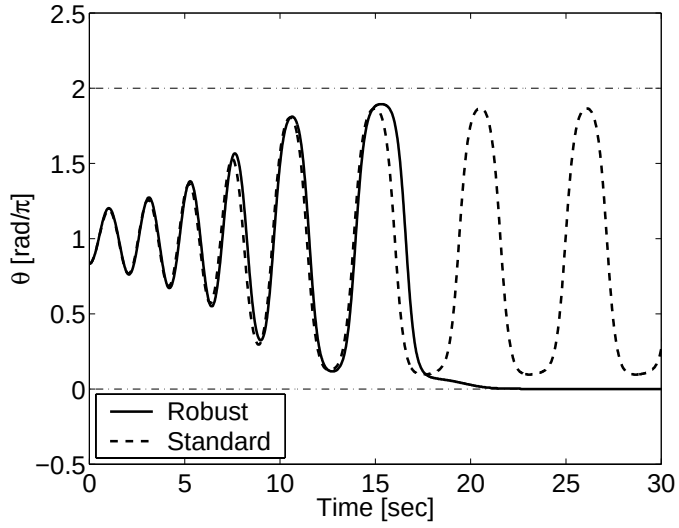


(b) Disturber

Figure 5.8 Shape of the control and disturbance function after 1000 learning trials with $m = 1.0$[kg] and $l = 0.01$[m]



(a) $m = 1.0, \mu = 0.01$



(b) $m = 1.3, \mu = 0.2$

Figure 5.9 Swing up trajectories with pendulum with different weight and friction. The dash-dotted lines show upright position.

Chapter 6

Conclusions

In this dissertation, we proposed the methods which enable the robot to learn the task by using reinforcement learning in high-dimensional state space and in real environment. To realize the reinforcement learning in high-dimensional state space and real environment, we used the following three ideas. 1: introduction of hierarchical architecture in chapter 3, 2: using prior knowledge acquired from a teacher in chapter 4, 3: explicit consideration of modeling error in chapter 5).

In chapter 3, we proposed a hierarchical RL architecture that realizes a practical learning speed in high-dimensional control tasks. In this method, 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 considered a "stand-up" task for a two-joint, three-link robot. The goal of the task was 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 successfully learned to stand up within 750 trials in simulation and then in an additional 170 trials using real hardware.

In chapter 4, we showed that how imitation learning improve the learning speed in the framework of the hierarchical reinforcement learning. The learner acquired stand-up trajectories from teacher's motion and detected via-points. The learner calculated the state value at the via-points and derived action-value of the subgoals. We can consider derived action-value of the subgoals as prior knowledge for the action of the upper-level learner.

We applied our method to stand-up task of 3-link 2-joint robot. The robot learned to stand up about four times faster than without using proposed imitation

learning method.

In chapter 5, we studied how we can make the policy learned by RL more robust against modeling error or environmental change. We proposed a new reinforcement learning paradigm that we call "Robust Reinforcement Learning (RRL)."

As H^∞paradigm suggest, design of a robust controller can be achieved by finding a min-max solution of a value function, which is formulated as Hamilton-Jacobi-Isaacs (HJI) equation. We then derived on-line algorithms for estimating the value functions and for simultaneously deriving the worst disturbance and the best control that, respectively, maximizes and minimizes the value function.

We tested the validity of the algorithms first in a linear inverted pendulum task. It was verified that the value function as well as the disturbance and control policies derived by the on-line algorithm coincides with the analytical solution given by H^∞theory. We then compared the performance of the robust RL algorithm with a standard model-based RL in a nonlinear task of pendulum swing-up. It was shown that robust RL controller could accommodate environmental changes in the mass and the friction of the pendulum, which a standard RL controller could not cope with.

Finally, we combined the hierarchical RL and RRL for acquire the robust policy in the high-dimensional state space. We tested this combined method by applying it to the 2-joint 3-link robot. The results showed that the robot showed robust performance by using the hierarchical RL with RRL.

6.1. Future work

In our hierarchical reinforcement learning method, we fixed an appropriate step size in the upper level, but a method of automatically choosing and adapting the step size remains future work. In addition, the use of a hierarchical architecture with three or more layers, and reusing lower-level modules in the other tasks, are also interesting topics. We will incorporate these ideas in our hierarchical RL method as future work.

In our robust reinforcement learning study, we applied it to the pendulum swing-up task and the stand-up task with using the hierarchical RL. However,

consideration of robustness in the upper level of the hierarchical architecture remains as future work.

Bibliography

[1] M. Asada, H. Kitano, I. Noda, and M. Veloso. Robocup:today and tomorrow – what we have learned. *Artificial Intelligence*, 110:193–214, 1999.

[2] M. Asada, E. Uchibe, and K. Hosoda. Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development. *Artificial Intelligence*, 110:275–292, 1999.

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846, 1983.

[4] S. J. Bradtke. Reinforcement learning Applied to Linear Quadratic Regulation. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 295–302. Morgan Kaufmann, San Mateo, CA, 1993.

[5] S. P. Coraluppi and S. I. Marcus. Risk-Sensitive and Minmax Control of Discrete-Time Finite-State Markov Decision Processes. *Automatica*, 35:301–309, 1999.

[6] P. Dayan and G. E. Hinton. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems 5*, pages 271–278, San Francisco, CA, 1993. Morgan Kaufmann.

[7] Bruce L. Digney. Learning Hierarchical Control Structures for Multiple Tasks and Changing Environments. In *Proceedings of the Fifth Conference on the Simulation of Adaptive Behavior*, pages 321–330, Cambridge, MA, 1998. The MIT Press.

[8] K. Doya. Temporal difference learning in continuous time and space. In

D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1073–1079. MIT Press, Cambridge, MA, 1996.

[9] K. Doya. Efficient nonlinear control with actor-tutor architecture. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 1012–1018. MIT Press, Cambridge, MA, 1997.

[10] K. Doya. Reinforcement Learning in Continuous Time and Space. *Neural Computation*, 12(1):219–245, 2000.

[11] M. E. Harmon, L. C. Baird III, and A. H. Klopf. Advantage updating applied to a differential game. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 353–360. MIT Press, Cambridge, MA, USA, 1995.

[12] K. Imafuku. Singularities of nonlinear control systems designed by Hamilton-Jacobi equations. *Doctoral thesis, Nara Institute of Science and Technology*, 1999.

[13] M. Inaba, I. Igarashi, K. Kagami, and 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*, volume 1, pages 29–36, 1996.

[14] R. A. Jacobs and M. I. Jordan. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[15] F. Kanehiro, M. Inaba, and 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*, volume 1, pages 23–28, 1996.

[16] H. Kimura and S. Kobayashi. Efficient Non-linear Control by Combining Q-learning with Local Linear Controllers. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 210–219, San Francisco, CA, 1999. Morgan Kaufmann.

[17] F. Kirchner. Q-Learning of Complex Behaviours on a Six-Legged Walking Machine. In *Proceedings Second EUROMICRO Workshop on Advanced Mobile Robots*, pages 51–58, 1997.

[18] Y. Kuniyoshi and A. Nagakubo. Humanoid As a Research Vehicle Into Flexible Complex Interaction. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1997.

[19] P. Maes and R. A. Brooks. Learning to Coordinate Behaviors. In *Proceedings of AAAI-90*, pages 796–802, 1990.

[20] M. Mataric. Learning Social Behaviors. *Robotics and Autonomous Systems*, 20:191–204, 1997.

[21] H. Miyamoto, S. Schaal, F. Gandolfo, H. Gomi, Y. Koike, R. Osu, E. Nakano, Y. Wada, and M. Kawato. A Kendama Learning Robot Based on Bi-directional Theory. *Neural Networks*, 9:1281–1302, 1996.

[22] J. Moody and C. J. Darken. Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation*, 1:281–294, 1989.

[23] J. Morimoto and K. Doya. Hierarchical Reinforcement Learning of Low-dimensional Subgoals and High-dimensional Trajectories. In *Proceedings of the Fifth International Conference on Neural Information Processing*, volume 2, pages 850–853, Burke, VA, 1998. IOS Press.

[24] J. Morimoto and K. Doya. Reinforcement Learning of Dynamic Motor Sequence: Learning to Stand Up. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1721–1726. OMNIPRESS, 1998.

[25] J. Morimoto and K. Doya. Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. In *Proceedings of Seventeenth International Conference on Machine Learning*, pages 623–630, San Francisco, CA, 2000. Morgan Kaufmann.

[26] R. Neuneier and O. Mihatsch. Risk Sensitive Reinforcement Learning. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 1031–1037. MIT Press, Cambridge, MA, USA, 1998.

[27] M. Ortiz and P. Zufiria. Evaluation of reinforcement learning autonomous navigation systems for a nomad 200 mobile robot. In *Intelligent Autonomous Vehicles 1998 (IAV'98). Proceedings volume from 3rd IFAC Symposium*, pages 309–314, 1998.

[28] J. Peng and R. Williams. Incremental Multi-step Q-learning. *Machine Learning*, 22:283–290, 1996.

[29] M. L. Puterman. *Markov Decision Problems.* Wiley, New York, 1994.

[30] S. Schaal. Learning From Demonstration. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 1040–1046. MIT Press, Cambridge, MA, 1997.

[31] S. Singh. Transfer of Learning by Composing Solutions of Elemental Sequential Tasks. *Machine Learning*, 8:323–339, 1992.

[32] R. S. Sutton. Learning to predict by the methods of temporal difference. *Machine Learning*, 3:9–44, 1988.

[33] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, MA, 1998.

[34] R. S. Sutton, D. Precup, and S. Singh. Intra-option learning about temporary abstract actions. In *Proceedings of the 15th International Conference on Machine Learning*, pages 556–564, 1998.

[35] Y. Takahashi, M. Asada, and 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*, volume 3, pages 1518–1524, 1996.

[36] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[37] C. K. Tham. Reinforcement Learning of Multiple Tasks using a Hierarchical CMAC Architecture. *Robotics and Autonomous Systems*, 15:247–274, 1995.

[38] S. Weiland. Linear Quadratic Games, H_∞, and the Riccati Equation. In *Proceedings of the Workshop on the Riccati Equation in Control,*

Systems, and Signals, pages 156–159. 1989.

[39] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1997.

[40] S. Yamada, A. Watanabe, and M. Nakashima. Hybrid reinforcement learning and its application to biped robott control. In *Advances in Neural Information Processing Systems 10*, pages 1071–1077, 1998.

[41] T. Yamaguchi, M. Masubuchi, K. Fujihara, and 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*, volume 3, pages 1321–1327, 1996.

[42] T. Yoshioka and S. Ishii. Strategy acquisition for the game "othello" based on reinforcement learning. In S. Usui and T. Omori, editors, *International Conference on Neural Information Processing*, pages 841–844. ISO Press, 1998.

[43] K. Zhou, J. C. Doyle, and K. Glover. *Robust Optimal Control*. PRENTICE HALL, New Jersey, 1996.

Achievements

List of Publications

Journal Papers

1. Morimoto, J. and Doya, K., "Learning Dynamic Motor Sequence in High-Dimensional State Space by Reinforcement Learning –Learning to Stand Up–, The Transactions of the Institute of Electronics, Information and Communication Engineers D-II, Vol. J82-D-II No.11, pp.2118-2131, 1999.
2. Morimoto, J. and Doya, K.,"Acquisition of Stand-up Behavior by a Real Robot using Hierarchical Reinforcement Learning", Robotics and Autonomous Systems (accepted).
3. Morimoto, J. and Doya, K.,"Hierarchical reinforcement learning for motion learning: learning "stand-up" trajectories", Advanced Robotics, vol.13, No.3, pp.267-268, 1999.

International Conference Papers

1. Morimoto, J. and Doya, K., "Reinforcement learning of dynamic motor sequence: Learning to stand up", Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, vol.3, pp.1721-1726, 1998.
2. Morimoto, J. and Doya, K., "Hierarchical reinforcement learning of low-dimensional subgoals and high-dimensional trajectories", The 5th International Conference on Neural Information Processing, vol.2, pp.850-853, 1998.
3. Morimoto, J. and Doya, K., "Acquisition of Stand-up Behavior by a Real Robot using Hierarchical Reinforcement Learning", Proceedings of International Conference on Machine Learning, pp.623-630, 2000.

4. Morimoto, J. and Doya, K., "Robust Reinforcement Learning", Advances in Neural Information Processing Systems, 2000.

Domestic Conference Papers

1. Morimoto, J. and Doya, K., "Control of nonholonomic systems using reinforcement learning: Learning "stand-up" trajectories" Proceedings of the 15th Annual Conference of the Robotics Society of Japan, vol.1, pp.17-18, 1997.
2. Morimoto, J. and Doya, K., "Hierarchical reinforcement learning for motion learning: Learning "stand-up" trajectories", Proceedings of the 16th Annual Conference of the Robotics Society of Japan, vol.1, pp.431-432, 1998.
3. Morimoto, J. and Doya, K., "Acquisition of Stand-up Behavior by a Real Robot using Hierarchical Reinforcement Learning", 1999 Annual Conference of Japanese Neural Network Society, pp.147-148, 1999.
4. Morimoto, J. and Doya, K., "Acquisition of Stand-up Behavior by a Real Robot using Hierarchical Reinforcement Learning", Proceedings of the 5th Robotics Symposia, pp.397-402, 2000.
5. Morimoto, J. and Doya, K., "Robust Reinforcement Learning", Proceedings of the 18th Annual Conference of the Robotics Society of Japan, vol.3, pp.1263-1264, 2000.

Domestic Workshop Papers

1. Morimoto, J. and Doya, K., "Learning "stand-up" trajectories using reinforcement learning", Technical report of IEICE, Vol.97, No.201, pp.25-32, 1997.
2. Morimoto, J. and Doya, K., "Robust Reinforcement Learning", Technical report of IEICE, Vol.100, No.191, pp.59-66, 2000.

Award

1. Young Investigator Award from Japanese Neural Network Society, 2000

Appendix

A. Normalized Gaussian Network (NGnet)

The normalized Gaussian basis function is represented by

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

where

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

is a Gaussian activation function [22]. The vectors $\mathbf{c}_k$ and $\mathbf{s}_k$ 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 Incremental Normalized Gaussian Network (INGnet) [24] 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 criterion $e_{\max}$ and the activation of all existing units is smaller than a threshold $a_{\min}$, that is,

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

The new unit is initialized with $w_k = \hat{y}(\mathbf{x})$, $\mathbf{c}_k = \mathbf{x}$, and $\mathbf{s}_k = \mathrm{diag}(\mu_i)$, where $\hat{y}(\mathbf{x})$ is a desired output, and μ_i is the inverse of the radius of the basis function. In Sections 3.3 and 3.4, we set these parameters as $\mu_\theta = 0.035$ [1/deg] for pitch and joint angle dimension, $\mu_{\dot{\theta}} = 0.0087$ [sec/deg] for pitch and joint angular velocity dimension, $e_{\max} = 0.0$, and $a_{\min} = 0.4$. 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.

B. Continuous-time TD(λ)-learning

In this section, we explain TD(λ)-learning[10, 8] defined in continuous time.

Here, we fix a trajectory $\mathbf{x}(t), \mathbf{u}(t)$ in a trial. and represent $V(\mathbf{x}(t)), V^{\mu}(\mathbf{x}(t)), r(\mathbf{x}(t), \mathbf{u}(t))$ as $V(t), V^{\mu}(t), r(t)$.

We derive an equation below from the definition of TD-error (equation 2.7).

$$\frac{dV(t)}{dt} = \frac{1}{\tau}V(t) - (r(t) - \delta(t)) \tag{B.4}$$

Then, we have the equations.

$$\begin{aligned} V(t) &= \int_t^{\infty} e^{-\frac{s-t}{\tau}}(r(s) - \delta(s))ds \\ V^{\mu}(t) - V(t) &= \int_t^{\infty} e^{-\frac{s-t}{\tau}}\delta(s)ds \end{aligned} \tag{B.5}$$

The learning rule for estimated value function $V(t)$ to achieve real value V^{μ} is given as

$$\begin{aligned} \Delta w_i &\propto -\int_0^{\infty} \frac{\partial \frac{1}{2}(V^{\mu}(t) - V(t))^2}{\partial w_i}dt \\ &= -\int_0^{\infty} -(V^{\mu}(t) - V(t))\frac{\partial V(t)}{\partial w_i}dt \\ &= \int_0^{\infty}\int_t^{\infty} e^{-\frac{s-t}{\tau}}\delta(s)ds\frac{\partial V(t)}{\partial w_i}dt \end{aligned} \tag{B.6}$$

where w_i is the parameter of the value function $V(t)$ and we defined Δw_i as $\Delta w_i = \int_0^{\infty} \dot{w}_i \, dt$. We change the order of the integral as,

$$\Delta w_i \propto \int_0^{\infty}\int_0^{s} e^{-\frac{s-t}{\tau}}\frac{\partial V(t)}{\partial w_i}dt \, \delta(s)ds \tag{B.7}$$

We define the eligibility trace as

$$e_i(s) = \int_0^{s} e^{-\frac{s-t}{\tau}}\frac{\partial V(t)}{\partial w_i}dt \tag{B.8}$$

On-line update rule of the parameter w_i become

$$\dot{w}_i = \eta\delta(t)e_i(t) \tag{B.9}$$

General update rule is given by using time constant $\kappa \leq \tau$. We represent equation (B.8) as

$$e_i(t) = \int_0^t e^{-\frac{t-s}{\kappa}} \frac{\partial V(s)}{\partial w_i} ds \tag{B.10}$$

By differentiating above equation, we get the update rule of $e_i(t)$,

$$\dot{e}_i(t) = -\frac{1}{\kappa} e_i(t) + \frac{\partial V(t)}{\partial w_i} \tag{B.11}$$

We can say that $e_i(t)$ is the eligibility trace[3] in continuous time TD-learning. Furthermore, when we define λ as $\lambda = \frac{\tau(\kappa-\Delta t)}{\kappa(\tau-\Delta t)}$ and discritize by time step Δt, it become same learning rule to discrete time TD(λ)-learning[32](see appendix C).

C. Discrete and continuous time TD-learning

C.1 Definition of TD-learning

Continuous time TD-error is given by equation (2.7). We approximate time differentiation of the value function $\dot{V}(t)$ as $\dot{V}(t) = (V(t) - V(t-\Delta t))/\Delta t$. Then, we apply it to equation (2.7), we get

$$\delta(t) = r(t) + \frac{1}{\Delta t}\left[(1 - \frac{\Delta t}{\tau})V(t) - V(t-\Delta t)\right]. \tag{C.12}$$

On the other hand, discrete time TD-error[32] is given as

$$\delta(t) = r(t) + \gamma V(t) - V(t-\Delta t) \tag{C.13}$$

By comparing equation (C.12) and (C.13) with discount factor $\gamma = 1 - \frac{\Delta t}{\tau}$, these equations become same definition except for scaling factor $\frac{1}{\Delta t}$.

C.2 Update rule for the value function

Update rule for the continuous time TD-learning is given by equation (B.9) and (B.11). Then, when we approximate $\dot{e}_i(t) = \frac{e_i(t+\Delta t) - e_i(t)}{\Delta t}$, $\dot{w}_i = \frac{\Delta w_i}{\Delta t}$, we get

$$e_i(t+\Delta t) = (1 - \frac{\Delta t}{\kappa})e_i(t) + \frac{\partial V(t)}{\partial w_i}\Delta t \tag{C.14}$$

$$\Delta w_i = \eta\delta(t)e_i(t)\Delta t \tag{C.15}$$

On the other hand, the update rule for the discrete time TD-learning[33] is given as

$$e_i(t+\Delta t) = \gamma\lambda e_i(t) + \frac{\partial V(t)}{\partial w_i} \tag{C.16}$$

$$\Delta w_i = \alpha\delta(t)e_i(t) \tag{C.17}$$

Then, when we compare equations (C.14),(C.15) and (C.16), (C.17), we can find same definition of the update rule in discrete and continuous domain if $\gamma = 1 - \frac{\Delta t}{\tau}$ and $\alpha = \eta$ except for scaling factor Δt.

D. Implementation of continuous time TD-learning

When we implement the continuous time TD-learning in the computer simulation, the way of time differentiation of the value function should be considered. In equation (B.11), e_i can be considered as weighted sum of $\frac{\partial V}{\partial w_i}$ with time constant κ. In other words, e_i is considered as $\frac{\partial \bar{V}}{\partial w_i}$ where $\bar{V}$ is the weighted sum of the value function V with the time constant κ. Then, from equation (2.7), we have $\delta(t) = r(t) - \frac{1}{\tau}V(t) + \frac{V(t)-\bar{V}(t)}{\kappa}$ where $\dot{V} = \frac{V-\bar{V}}{\kappa}$. Then, we have the update rule of the network weight to minimize TD-error as

$$\begin{aligned} \dot{w}_i &= -\nu\frac{1}{2}\frac{\partial\delta^2(t)}{\partial w_i} \\ &= -\nu\delta(t)\frac{1}{\kappa}((1-\frac{\kappa}{\tau})\frac{\partial V(t)}{\partial w_i} - \frac{\partial\bar{V}(t)}{\partial w_i}) \end{aligned} \tag{D.18}$$

We consider that we should derive TD-error only according to the smooth value function $\bar{V}(t)$ which contain past information of the value. Then, the update rule of the network weight is given as

$$\dot{w}_i = \eta\delta(t)\frac{\partial\bar{V}(t)}{\partial w_i} \tag{D.19}$$

where $\eta = \nu\frac{1}{\kappa}$. The equation (D.19) become same definition to equation (B.9). Then, we can say that $\dot{V} = \frac{V-\bar{V}}{\kappa}$ is appropriate way to derive $\dot{V}$.

We show the learning algorithm in the computer simulation

1. initialize the value function by $\bar{V}(0) = V(0)$

2. repeat (a) $\sim$ (g)

   (a) derive control output $u(t)$ at the state $\mathbf{x}(t)$ by equation (3.16).

   (b) update the eligibility trace of the basis functions as in equation (C.14).

$$e_i(t+\Delta t) = (1 - \frac{\Delta t}{\kappa})e_i(t) + {b^C}_i(t)\Delta t \tag{D.20}$$

   (c) calculate dynamics of the robot by using $u(t)$
   time: $t \leftarrow t + \Delta t$
   state: $\mathbf{x}(t + \Delta t)$
   reward: $r(t + \Delta t)$

   (d) calculate TD-error $\delta(t + \Delta t)$ as

$$\begin{aligned} \delta(t+\Delta t) &= r(t+\Delta t) - \frac{1}{\tau}V(\mathbf{x}(t+\Delta t)) \\ &+ \frac{V(\mathbf{x}(t+\Delta t)) - \bar{V}(t)}{\kappa} \end{aligned} \tag{D.21}$$

   (e) update the critic as in equation (4.40)

$$v_i(t+\Delta t) = v_i(t) + \alpha\delta(t+\Delta t)e_i(t+\Delta t) \tag{D.22}$$

   (f) update the actor as in equation (3.17)

$$w_i(t+\Delta t) = w_i(t) + \beta\delta(t+\Delta t)\sigma n_j(t){b^A}_i(t) \tag{D.23}$$

   (g) update the value $\bar{V}$ as

$$\bar{V}(t+\Delta t) = (1 - \frac{\Delta t}{\kappa})\bar{V}(t) + \frac{\Delta t}{\kappa}V(\mathbf{x}(t+\Delta t)) \tag{D.24}$$