Doctor's Thesis

Deep Reinforcement Learning with Smooth Policy Update for Robotic Cloth Manipulation

Yoshihisa Tsurumine

December 15, 2020

Program of Information Science and Engineering
Graduate School of Science and Technology
Nara Institute of Science and Technology

A Doctor's Thesis
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Yoshihisa Tsurumine

Deep Reinforcement Learning with Smooth Policy Update for Robotic Cloth Manipulation[*]

Yoshihisa Tsurumine

Abstract

Deep Reinforcement Learning (DRL), which can learn complex policies with high-dimensional observations as inputs, e.g., images, has been successfully applied to various tasks. Thus it may be suitable to use them for robots to learn and perform daily activities like washing and folding clothes, cooking, and cleaning. However, there are only a few studies that have applied DRL to real robot environments. In this thesis, our objective is to apply DRL to cloth manipulation tasks that are part of daily human tasks. To this end, we consider two main difficulties as follows. (1) generating a massive number of samples in a real robot system is arduous because of the high sampling cost, and (2) learning environments require a reward function to evaluate the tasks; however, designing rewards for cloth with flexible shapes is challenging.

The approach for the first difficulty is to apply sample efficient DRL to real robotic cloth manipulation tasks. Previous value function-based DRL stabilizes the value function's approximation with Deep Neural Networks (DNN) by learning from many samples. In this thesis, we employ a smooth policy update to enable stable learning from a small number of samples. We propose two sample efficient DRL algorithms: Deep P-Network (DPN) and Dueling Deep P-Network (Dueling DPN). Proposed methods are value function-based DRL with smooth policy update and employ the Kullback-Leibler divergence to limit the over policy update. Dueling DPN has a DNN structure suitable for approximating value functions and improves sample efficiency in tasks with large action spaces.

[*]Doctor's Thesis, Department of Information Science, Graduate School of Information Science and Technology, Nara Institute of Science and Technology, December 15, 2020.

The approach for the second difficulty is to learn a cloth manipulation policy without explicit reward function design. This thesis explores an approach of Generative Adversarial Imitation Learning (GAIL) for robotic cloth manipulation tasks, which allows an agent to learn near-optimal behaviors from expert demonstration and self explorations without explicit reward function design. However, the performance in real robot environments may be insufficient because it is difficult for humans to collect appropriate expert samples using a robot's state-action space. In this thesis, we focus on target state labels that can be adequately presented by humans. GAIL with target state labels improves the performance of learning policies by estimating higher target state rewards. We propose P-Generative Adversarial Imitation Learning with Bi-Discriminator (PGAIL-BiD), which learns a policy with two discriminators in reward function and target state. PGAIL-BiD employs DPN to policy updates to enable stable learning from complex reward functions associated with the two discriminators.

Proposed methods are first investigated by a robot-arm reaching task in the simulation and compared to previous performance and sample efficiency methods. In a real robot experiment, we applied the proposed methods in two real robotic cloth manipulation tasks: 1) flipping a handkerchief and 2) folding clothes. In the handkerchief flipping task, proposed methods are compared with conventional methods using a human-designed reward function. We investigate the task success rate and learned feature extraction in the folding clothes task.

Keywords:

Deep Reinforcement Learning, Generative Adversarial Imitation Learning, Robotic Cloth Manipulation, Robotic Learning, Manipulation Planning

方策を滑らかに更新する深層強化学習による実ロボット衣類操作の学習*

鶴峯義久

内容梗概

深層強化学習は高次元情報を入力とする複雑な方策が学習可能な手法である．シミュレーションタスクへの適用では膨大なサンプルから人間と同等以上のパフォーマンスを獲得しており，実ロボットタスクへの適用により洗濯や料理，掃除等の日常的な人間の作業を代替することが期待できる．しかし，深層強化学習を実ロボットタスクに適用した研究は少ない．目標は衣類操作へのDRLの適用とする．そのために必要な次の2つの問題について検討する．(1) ロボットは実際の行動を通してデータを取得するためシミュレーションのように膨大なサンプルを収集できない，(2) 学習には選択した行動を評価する報酬関数が必要だが，形状が柔軟に変化する衣類の報酬設計は困難である．

初めに，サンプル効率の良い深層強化学習を提案し，実ロボットによる衣類操作タスクを学習することを目指す．従来の価値関数ベース深層強化学習は，膨大なサンプルから学習することで深層学習による価値関数の近似を安定化させている．本研究では方策を滑らかに更新することで少数サンプルから安定した学習を実現する．この解決策は，方策の更新量を制約することで方策の過学習を避ける．本研究はサンプル効率の良い深層強化学習であるDeep P-Network (DPN)とDueling Deep P-Network (Dueling DPN)を提案する．提案手法は方策を滑らかに更新する価値関数ベース深層強化学習であり，方策の更新をカルバックライブラー・ダイバージェンスで定量化することで制約する．Dueling DPNは価値関数の近似に適した深層ネットワーク構造を持ち，行動空間が大きいタスクにおいてサンプル効率を改善する．

次に，報酬関数と方策の両方を学習することで，報酬関数の設計なしに衣類操作方策を学習することを目指す．本研究ではエキスパートデモンストレ

*奈良先端科学技術大学院大学 先端科学技術研究科 情報科学領域 博士論文, 2020年 12月 15日.

ーションから報酬関数と方策を同時に学習する敵対的模倣学習フレームワークに注目する．しかし，人間がロボットの状態行動空間を用いて適切なエキスパートサンプルを収集することは困難であり，実ロボットタスクでの高いパフォーマンスは期待できない．本研究では人間が適切に提示できる目標状態ラベルを利用し目標状態の報酬を高く見積もることで，学習方策のパフォーマンスを向上させる．本研究は目標状態を考慮するP-Generative Adversarial Imitation Learning with Bi-Discriminator (PGAIL-BiD)を提案する．PGAIL-BiDの方策はエキスパート判別器と目標状態判別器を含む報酬関数から学習する．PGAIL-BiDでは二つの判別機から生成される複雑な報酬値から安定した学習を実現するために，方策を滑らかに更新するDPNを用いて方策を更新する．

本研究はシミュレーションのn DOFリーチングタスクに提案手法を適用し，学習方策のパフォーマンスやサンプル効率を従来手法と比較した．実機実験においては提案手法を双腕ロボットによるハンカチ裏返しタスクと衣類折り畳みタスクに適用した．ハンカチ裏返しタスクでは人間が設計した報酬関数で提案手法と従来手法の性能を比較した．衣類折り畳みタスクではタスク成功率や学習方策が獲得した特徴抽出を調べた．

キーワード

深層強化学習，敵対的模倣学習，柔軟物操作，ロボット学習，操作計画

Contents

List of Figures

1. Introduction

1.1. Background

With the capability of learning optimal policies by interacting with an unknown environment, Reinforcement Learning (RL) [1] has been applied to a broad range of platforms in robot control, such as an autonomous helicopter/vehicle [2, 3], a robot dog [4], and humanoid robots [5–7]. Most of the RL algorithms in the above studies require either 1) direct access to state variables or 2) well-designed hand-engineered features extracted from sensory inputs. However, they become difficult in general when considering more complex and practical tasks/situations. For example, in household robots, such as humans' daily activities as washing and folding clothes, cooking and cleaning are desirable to be learned and performed by RL, but it is not easy to achieve either 1) or 2) (e.g., [8]).

The recent advance of Deep Neural Networks (DNN) [9] enables automatic extraction of high-level features to outperform traditional hand-engineered features extracted from high-dimensional observations as input like raw images [10–12] and audio signals [13, 14]. Deep Reinforcement Learning (DRL), e.g., Deep Q-Network (DQN) [15] and Trust Region Policy Optimization (TRPO) [16], have been proposed by exploiting such DNN capabilities for automatic feature extraction in RL. By automatically abstracting good high-level features from raw images, DQN can learn a complex policy with human-level performances on various Atari video games. Therefore, DRL may be suitable to apply them for robots to learn and perform daily activities like washing and folding clothes, cooking, and cleaning since such tasks are difficult for non-DRL methods that often require either 1) direct access to state variables or 2) well-designed hand-engineered features extracted from sensory inputs.

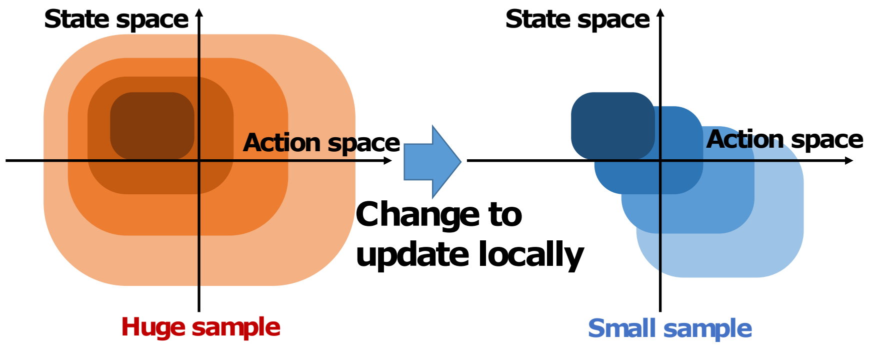
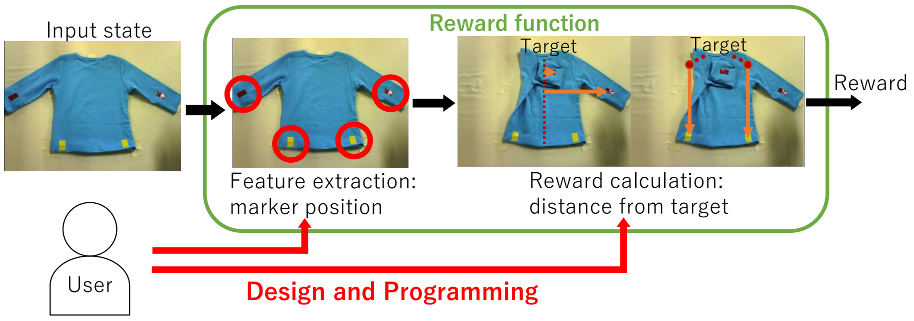


Figure 1.1. Overview of proposed sample efficient DRL.

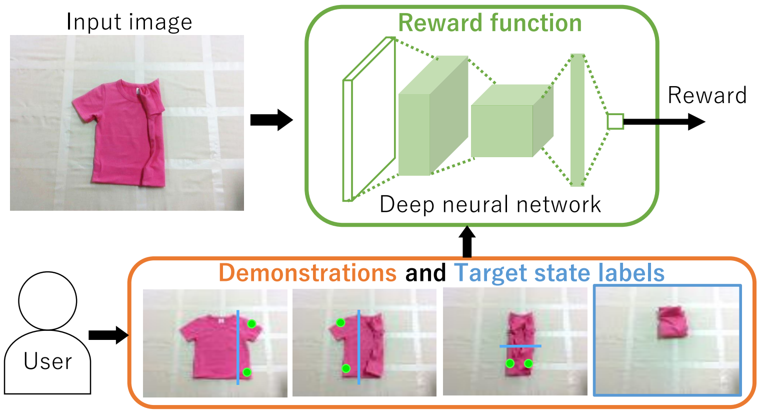
1.2. Motivation

Even though real-world environments and simulation environments are significantly different, applying DRL to a real robotic task is still challenging. This thesis focuses on two problems in a real-world environment: (1) sample efficiency of a real robot, (2) design a reward function for each task that evaluates the selected action. In problem (1), a real robot collects samples by actual robot actions. Thus collecting a huge sample is not feasible. Previous DRL requires a huge sample to be collected using simulation, then applying DRL to a real-world task is difficult. Problem (2) is occurred due to differences in the methods of obtaining the state variables. DRL learns image input policies that do not require state space design, e.g., the object's position and shape. On the other hand, although the state space of the reward function requires design for each task, the reward function design is easier in simulation because the state variables can be obtained directly. However, in a real-world environment, we need to design a task-specific feature extractor to obtain the target state variable. Designing a complex reward function for each task is costly, and the performance of the learning policy depends strongly on the reward function.

In sample efficiency in real robot tasks, previous studies have employed a large sample collection approach. For example, multiple robots sample data in parallel to improve efficiency [17, 18]. This approach can be applied to various methods but requires the preparation, management, and execution of multiple robots [19–21]. Other approaches have employed simulations to learn policies that can be


Input state
Reward function
Target
Target
Reward
Feature extraction: marker position
Reward calculation: distance from target
User
Design and Programming

(a) Conventional: examples of human-designed clothing manipulation rewards



(b) Proposed: learning cloth manipulation rewards from demonstration

Figure 1.2. Conventional and proposed reward function design.

applied to real-world environments [19–21]. This simulation approach allows for the collection of huge samples but can only apply to simulatable tasks.

As a solution to reward function design, we focus on a Generative Adversarial Imitation Learning framework (GAIL) [22], which allows an agent to learn near-optimal behaviors from expert demonstration and self explorations without explicit reward function design. When applying GAIL to a real robot task, humans use the robot's action space to collect demonstrations. However, collecting perfect expert samples is difficult due to the difference in humans and robots' state-action spaces. Therefore, imperfect expert samples reduce the performance of the learning policy. The utilization of the target state labels prevents the performance decline, but adversarial learning is unstable due to complex learning of the reward function. This thesis proposes sample-efficient DRL and GAIL with target state label, based on value function-based RL with smooth policy updates that asymptotic convergence nature to the optimal policy.

This thesis's motivation is to reduce the cost of applying DRL to a real robot task by solving the above two problems as shown in Fig. 1.1 and Fig. 1.2. The solution to both problems is DRL with smooth policy updates. Fig. 1.1 shows that sample efficiency can be improved from a small number of samples with smooth policy updates. The conventional reward function design needs to be designed by a human for each task, as shown in Fig. 1.2a. The proposed method uses incomplete demonstrations and target state labels as shown in Fig. 1.2b to prevent performance decline without designing a task-dependent reward function. We use smooth policy updating to stabilize learning from complex reward values with expert samples and target state labels. We propose sample-efficient DRL and GAIL with target state labels based on value function-based RL with smooth policy updates that can learn globally optimal policies. We show experimentally that the proposed method improves sample efficiency and training stability.

1.3. Contribution

We first present Deep P-Network (DPN) and Dueling Deep P-Network (Dueling DPN) as novel deep reinforcement learning based on Dynamic Policy Programming (DPP) [23]. It automatically abstracts the raw images' features by exploiting the nature of smooth policy update by introducing the Kullback-Leibler divergence between current and new policies as a regularization term into the reward function for better sample efficiency. An extension of DPN with a particularly suitable network structure of DNN, Dueling DPN, is proposed for better generalization capability inspired by the dueling network structure for DQN [24].

Next, we propose P-Generative Adversarial Imitation Learning with Bi-Discriminator (PGAIL-BiD), a GAIL framework with an expert discriminator and a target discriminator. PGAIL-BiD estimates high rewards for target states with the target discriminator and improves learning policies' reaching performance when learning from imperfect expert samples. The reward function with two discriminators in training destabilizes adversarial learning. Therefore, PGAIL-BiD employs a modified value-function based DRL, Entropy maximizing Deep P-Network (EDPN), that can consider both the smoothness and causal entropy in policy update.

To investigate the performance of learned policies and learned reward functions in several conditions, proposed methods are applied to a n DOF simulated manipulator reaching task. In a real-world experiment, proposed methods learned a clothing manipulation task, which is a daily task and challenging to design a reward function. In these tasks, dual-arm humanoid robot NEXTAGE executes manipulation actions, and camera images observe cloth states. We apply proposed methods to two real robot cloth manipulation tasks: (1) handkerchief flipping and (2) clothes folding. The handkerchief flipping task with a simple reward function design evaluated the performance of learning policies. The clothes folding task confirmed its effectiveness in a more realistic task.

In the appendix, we also explain Variationally Autoencoded Dynamic Policy Programming (VAE-DPP), a model-based DRL for discrete state-action space. VAE-DPP learns a discrete state-action space model with input images from pre-collected samples. The learned model enables action planning due to learning DPP policy from a defined reward function. While DPN is model-free and needs to be retrained from the beginning when the reward function changes, VAE-DPP allows for retraining without new samples when the reward function change. When learning multiple policies with different reward functions in the same environment, the model-free RL, DPN, must be retrained from the beginning. In the same case, VAE-DPP allows the relearning of policies without a new sample. We applied VAE-DPP to the handkerchief folding task with NEXTAGE and confirmed its effectiveness.

The remainder of this paper is organized as follows. The preliminaries are introduced in Chapter 2. DPN/Dueling DPN, PGAIL-BiD and the corresponding simulation results, analysis and real robot experiments on cloth manipulation tasks in Chapters 3 and 4. Discussion and conclusion are described in Chapters 5 and 6. Lastly, an appendix gives more details of VAE-DPP.

2. Preliminaries

2.1. Reinforcement Learning

RL [1,25] solves the Markov decision process (MDP) defined by a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. $\mathcal{S} = \{s_1, s_2, ..., s_n\}$ is a finite set of states. $\mathcal{A} = \{a_1, a_2, ..., a_m\}$ is a finite set of actions. $\mathcal{T}^a_{ss'}$ is the probability of transitioning from state s to state s' under action a. The corresponding reward is defined as $r^a_{ss'}$ with reward function $\mathcal{R}$. $\gamma \in (0, 1)$ is the discount parameter. Policy $\pi(a|s)$ represents the probability of action a being taken under state s. The value function is defined as the expected discounted total reward in state s:

$$V(s) = \mathbb{E}_{\pi,\mathcal{T}}\left[\sum_{t=0}^{\infty} \gamma^t r_{s_t} \middle| s_0 = s\right], \tag{2.1}$$

where $r_{s_t} = \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi(a|s_t)\mathcal{T}^a_{s_ts'} r^a_{s_ts'}$ is the expected reward from state s_t.

The objective of RL is to find optimal policy π^* that maximizes the value function to satisfy the following Bellman equation:

$$V^*(s) = \max_{\pi} \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi(a|s)\mathcal{T}^a_{ss'}\Big(r^a_{ss'} + \gamma V^*(s')\Big), \tag{2.2}$$

or a Q function for state-action pairs (s, a):

$$Q^*(s, a) = \max_{\pi} \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'}\Big(r^a_{ss'} + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s')Q^*(s', a')\Big). \tag{2.3}$$

Value function based RL algorithms, e.g., Q-learning [26], SARSA [27], and LSPI [28], approximate the value/Q function using the Temporal Difference (TD) error. For example, the TD update rule in Q-learning follows $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r^{a_t}_{s_ts_{t+1}} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$, where α is the learning rate.

2.2. Dynamic Policy Programming

To exploit the nature of smooth policy update, DPP [23,29] considers the Kullback-Leibler divergence between current policy π and baseline policy $\bar{\pi}$ into the reward function to minimize the difference between the current and baseline policy while maximizing the expected reward:

$$D_{\mathrm{KL}} = \sum_{a\in\mathcal{A}} \pi(a|s) \log \frac{\pi(a|s)}{\bar{\pi}(a|s)}. \tag{2.4}$$

Thus, the Bellman optimality equation Eq. (2.2) is modified as:

$$V^*_{\bar{\pi}}(s) = \max_{\pi} \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi(a|s)\left[\mathcal{T}^a_{ss'}\Big(r^a_{ss'} + \gamma V^*_{\bar{\pi}}(s')\Big) - \frac{1}{\eta}\log\left(\frac{\pi(a|s)}{\bar{\pi}(a|s)}\right)\right]. \tag{2.5}$$

The effect of the Kullback-Leibler divergence is controlled by inverse temperature η. Following [29, 30], we let η be a positive constant. Optimal value function $V^*_{\bar{\pi}}(s)$ for all $s \in \mathcal{S}$ and Optimal policy $\bar{\pi}^*(a|s)$ for all (s, a) satisfy double-loop fixed-point iterations as follows:

$$V^{t+1}_{\bar{\pi}}(s) = \frac{1}{\eta}\log\sum_{a\in\mathcal{A}} \bar{\pi}^t(a|s)\exp\left[\eta\sum_{s'\in\mathcal{S}}\mathcal{T}^a_{ss'}\Big(r^a_{ss'} + \gamma V^t_{\bar{\pi}}(s')\Big)\right] \tag{2.6}$$

$$\bar{\pi}^{t+1}(a|s) = \frac{\bar{\pi}^t(a|s)\exp\left[\eta\sum_{s'\in\mathcal{S}}\mathcal{T}^a_{ss'}\Big(r^a_{ss'} + \gamma V^t_{\bar{\pi}}(s')\Big)\right]}{\exp\Big(\eta V^{t+1}_{\bar{\pi}}(s)\Big)}. \tag{2.7}$$

Action preferences function [1] at the $(t + 1)$-iteration for all state-action pairs (s, a) is defined following [23] to obtain the optimal policy that maximizes the above value function:

$$P_{t+1}(s, a) = \frac{1}{\eta}\log\bar{\pi}^t(a|s) + \sum_{s'\in\mathcal{S}}\mathcal{T}^a_{ss'}\Big(r^a_{ss'} + \gamma V^t_{\bar{\pi}}(s')\Big). \tag{2.8}$$

Combining Eq. (2.8) with Eqs. (2.6) and (2.7), a simple form is obtained:

$$V_{\bar{\pi}}^{t}(s) = \frac{1}{\eta} \log \sum_{a \in \mathcal{A}} \exp\Big(\eta P_t(s,a)\Big) \tag{2.9}$$

$$\bar{\pi}^{t}(a|s) = \frac{\exp\Big(\eta P_t(s,a)\Big)}{\sum_{a' \in \mathcal{A}} \exp\Big(\eta P_t(s,a')\Big)}. \tag{2.10}$$

The update rule of action preference function $P_{t+1}(s,a) = \mathcal{O}P_t(s,a)$ is derived by plugging Eqs. (2.9) and (2.10) into Eq. (2.8):

$$\mathcal{O}P_t(s,a) = P_t(s,a) - \mathcal{L}_\eta P_t(s) + \sum_{s' \in \mathcal{S}} \mathcal{T}_{ss'}^{a}\Big(r_{ss'}^{a} + \gamma \mathcal{L}_\eta P_t(s')\Big), \tag{2.11}$$

where $\mathcal{L}_\eta P(s) \triangleq \frac{1}{\eta}\log\sum_{a\in\mathcal{A}}\exp(\eta P(s,a)) = V_{\bar{\pi}}(s)$. The difference between $P_{t+1}(s,a)$ and $\mathcal{O}P_t(s,a)$ is used to calculate the error signal to train the action preference function.

The original DPP is only applicable to problems with a modest number of discrete states and prior knowledge about the underlying model. Sampling-based Approximate Dynamic Policy Programming (SADPP) [29] extends it to model-free learning with large-scale (continuous) states. For N training samples, $[s_n, a_n]_{n=1:N}$, SADPP approximates $P(s,a)$ by Linear Function Approximation (LFA): $\hat{P}(s_n, a_n; \boldsymbol{\theta}) = \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{\theta}$, where $\boldsymbol{\phi}(\boldsymbol{x}_n)$ denotes the output vector of the basis functions and $\boldsymbol{\theta}$ is the corresponding weight vector. The weight vector is updated by minimizing empirical loss function $J(\boldsymbol{\theta}; \hat{\boldsymbol{P}}) \triangleq \|\boldsymbol{\Phi}\boldsymbol{\theta} - \mathcal{O}\hat{\boldsymbol{P}}\|_2^2$, where $\mathcal{O}\hat{\boldsymbol{P}}$ is an $N \times 1$ matrix with elements $\mathcal{O}\hat{P}(s,a;\boldsymbol{\theta})$ following Eq. (2.11), where $\mathcal{L}_\eta P(s)$ is translated into a Boltzmann softmax operator for more analytically tractable recursion.

Although such an extension to DPP can be applied to toy problems as mountain-car control [29], its scalability is still limited due to the exponentially growing size of the basis functions with increasing input dimensionality and corresponding intractability. More scalable function approximators, such as non-parametric regression, have been employed and successfully applied for real robot control tasks [31, 32]. However, their applications to such very high-dimensional and redundant state like sensor data and raw image data remain infeasible.

2.3. Deep Q-Network

As a combination of Q-learning and DNN, DQN [15] successfully approximates the Q function by DNN. Since the direct approximation of a dynamically changing Q function by DNN is difficult, DQN stabilizes the learning by several tricks, like target network, error clip, and experience replay. When Q function approximated by DNN parameter $\boldsymbol{\theta}$ is represented by $\hat{Q}(s, a; \boldsymbol{\theta})$, a target network is defined as $\hat{Q}(s, a; \boldsymbol{\theta}^-)$. $\boldsymbol{\theta}^-$ is updated every C steps following $\boldsymbol{\theta}^- = \boldsymbol{\theta}$, and $\boldsymbol{\theta}$ is updated every step with sample $(s_j, a_j, r^{a_j}_{s_j s_{j+1}}, s_{j+1})$ from a global memory that stores all the generated samples by performing a gradient descent with the TD error:

$$J(\boldsymbol{\theta}, \boldsymbol{\theta}^-) \triangleq \sum_{(s_j, a_j, r^{a_j}_{s_j s_{j+1}}, s_{j+1}) \in \mathcal{D}} (r^{a_j}_{s_j s_{j+1}} + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \boldsymbol{\theta}^-) - \hat{Q}(s_j, a_j; \boldsymbol{\theta}))^2, \tag{2.12}$$

where $\mathcal{D}$ denotes the experience replay buffer. The gradient descent step on $\boldsymbol{\theta}$ needs to be sufficiently small to make the learning slow and reduce the sample efficiency to avoid excessively changing the target function in the function approximation with DNN. One serious concern of DQN is that the smoothness in the policy update is not explicitly considered during learning. Such a lack of smoothness can drastically deteriorate the learning performance when the new policy is radically different from the previous one. In the subsequent section, we give a short summary of Dynamic Policy Programming [23, 29], which is a value function based RL algorithm that employs a smooth policy update.

2.4. Generative Adversarial Imitation Learning

The Generative Adversarial Imitation Learning (GAIL) framework consists of two parts: a generator to learn the sampling distribution from an expert demonstration, and an expert discriminator to distinguish between generated samples and expert samples. Under the adversarial framework, a sampling distribution indistinguishable to a demonstration is ideally learned, i.e., the expert discriminator learns a better classification ability while the generator learns to confuse the expert discriminator.

Given these ideas, the objective of the GAIL is formulated as follows:

$$\max_{\pi} \min_{D_E} \mathbb{E}_{\pi}[-\log(1 - D_E(s,a))] + \mathbb{E}_{\pi_E}[-\log(D_E(s,a))]. \tag{2.13}$$

Given the current state s and action a as input, the expert discriminator outputs the probability that the input belongs to the expert $D_E(s,a) \in [0,1]$. π is the policy learned by the generator, and π_E is the expert policy. Since DNNs approximate the generator and discriminator, we take a gradient-based numerical simultaneous optimization approach.

The expert discriminator's error function is defined as:

$$\mathbb{E}_{\pi}[-\log(1 - D_E(s,a;\phi))] + \mathbb{E}_{\pi_E}[-\log(D_E(s,a;\phi))] \tag{2.14}$$

where ϕ is defined as the DNNs parameters that approximate the expert discriminator. The expert discriminator is a simpler function than that of policy thus the learning progress is faster than the policy. Hence, the parameters ϕ needs to be updated conservatively to avoid over-fitting.

Given the current discriminator $D_E(s,a;\phi)$, the updated policy π of generator is formulated as:

$$\pi^*(a|s) = \arg\max_{\pi} \mathbb{E}_{\pi}[-\log(1 - D_E(s,a;\phi))]. \tag{2.15}$$

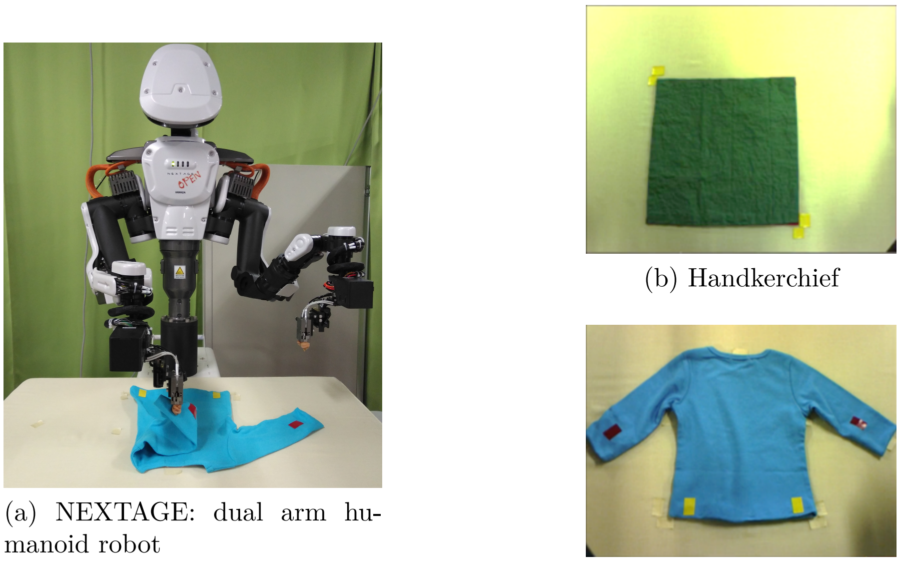
Using $r_s^a = -\log(1 - D_E(s,a;\phi))$ in the reward function, the RL agent learns a policy that maximizes the total reward.

Most of the previously proposed GAIL frameworks employ Trust Region Policy Optimization (TRPO) [16] in the generator, a popular policy search-based RL approach. TRPO is highly suitable for GAIL due to two key features: 1) the smooth policy update for learning stability and 2) the diversity of the policy to sampling in a wide range for training the discriminator. In a benchmark of discrete action spaces [33], value function-based DRL achieved better performance than Proximal Policy Optimization [34], a variant of TRPO. Due to this reason, the existing GAIL frameworks may not be suitable for our purpose, i.e., for robotic cloth manipulation with the discrete action set. Although some value function-based DRL has been proposed [35, 36], these methods may be inappropriate for our purpose because they do not have the two properties simultaneously. Therefore, the proposed GAIL framework employs a modified value-function based deep RL, Entropy-maximizing Deep P-Network (EDPN), that can consider both the smoothness and causal entropy in policy update.

3. Sample Efficient Deep Reinforcement Learning with Smooth Policy Update

This chapter proposes two sample efficient DRL algorithms: Deep P-Network (DPN) and Dueling Deep P-Network (Dueling DPN). These algorithms' core idea is to combine the nature of smooth policy update in value function-based reinforcement learning with the automatic feature extraction from high-dimensional observations in deep neural networks to enhance the sample efficiency and the learning stability with fewer samples. The smoothness of policy update is promoted by limiting the relative entropy or the Kullback-Leibler divergence between the current and new policies in the learning process. Even though several RL algorithms with such smooth policy update have been proposed [37,38], we focus on Dynamic Policy Programming (DPP) [23] for the following reasons: 1) its asymptotic convergence nature to the optimal policy (for discrete state-action cases); 2) a discrete action space that can easily use high-level actions; and 3) success in high-dimensional robot control tasks with direct access to state variables [32].

DPN and Dueling DPN are first applied to a n DOF simulated manipulator reaching task to evaluate their learning performances and compare the effect of different network structures and parameter settings with previous DRL methods. Then Dueling DPN was applied to real robotic cloth manipulation tasks to control a dual-arm humanoid robot NEXTAGE (Fig. 3.1a) to learn 1) the flipping of a handkerchief (Fig. 3.1b) and 2) folding a t-shirt (Fig. 3.1c) with a limited number of samples. We chose robotic cloth manipulation because it requires both a complicated and a high-dimensional state definition and a huge number of training samples to recognize and model the flexible cloth or learn a suitable


(a) NEXTAGE: dual arm humanoid robot

(b) Handkerchief

(c) T-shirt

Figure 3.1. Real robot setting: Our targets are two robotic cloth manipulation tasks with a dual-arm humanoid robot NEXTAGE (a) 1) flipping of a handkerchief (b) and 2) folding a t-shirt (c) with a limited number of samples.

manipulation policy.

3.1. Proposed Method

We present a novel deep reinforcement learning algorithm, Deep P-Network (DPN), that exploits the advantages of both DRL for high-dimensional state space and DPP for smooth policy update. Next we consider a more suitable neural network architecture for DPN as inspired by the Dueling DQN [24] that has a new neural network architecture with two parts to automatically produce separate estimates of value function $V(s)$ and advantage function $A(s,a)$ that fulfills $A(s,a) = Q(s,a) - V(s)$ without any extra supervision. Finally, we present how we can initialize both DPN and Dueling DPN with demonstration data for accelerating learning.

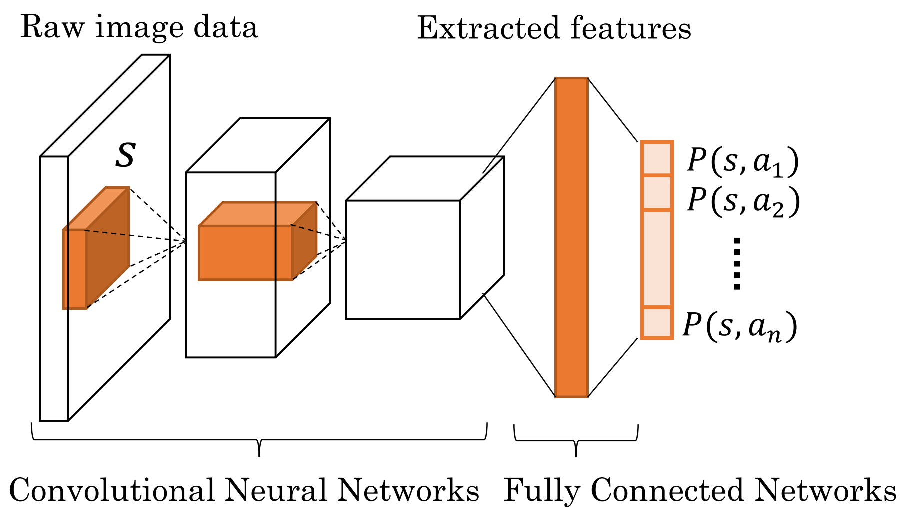


Figure 3.2. Network architectures of Deep P-Network

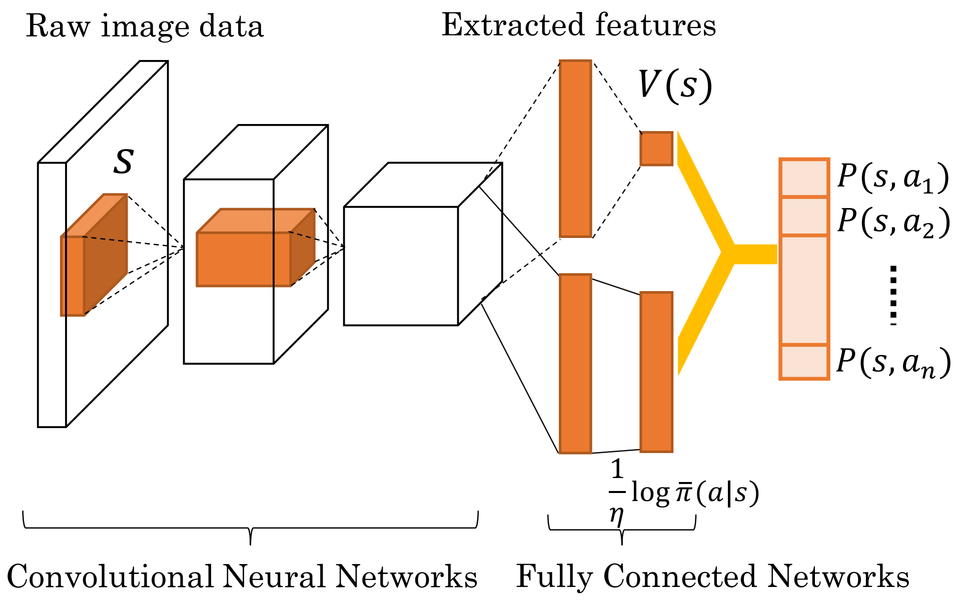


Figure 3.3. Network architecture of Dueling Deep P-Network

3.1.1. Approximation of Action Preference Function by DNNs

In this subsection, we propose DPN, which approximates the action preferences function $P(s, a; \boldsymbol{\theta})$ in Eq. (2.8) by DNN. Its network structure is defined in Fig. 3.2. Initial input state s is a raw RGB/grayscale image that usually has very high dimensionality. A Convolutional Neural Network (CNN) abstracts the raw image to a lower-dimensional high-level feature set. These features are in turn processed by a Fully Connected Network (FCN) and the final layer has m nodes, where m is the number of actions in $\mathcal{A}$ and the i-th node's output is approximated value $\hat{P}(s, a_i; \boldsymbol{\theta})$.

The training algorithm for DPN is given by Algorithm 1. In DPN, I is the number of iterations of DPN, and each iteration has M episodes with $M \times T$ samples. Local memory $\mathcal{D}$ is maintained to store the current E iteration samples for experience feedback. The updating of networks is operated in every episode. The current parameters as $\boldsymbol{\theta}^-$ are saved to build target network $\hat{P}(s, a; \boldsymbol{\theta}^-)$. The update is divided into N sub-problems. In each one, the agent repeatedly collects mini-batches of samples $(s_j, a_j, r_{s_j s_{j+1}}^{a_j}, s_{j+1})$ from $\mathcal{D}$ and calculates teaching signal y_j following Eq. (2.11):

$$y_j(\boldsymbol{\theta}^-) = \hat{P}(s_j, a_j; \boldsymbol{\theta}^-) - \mathcal{L}_\eta \hat{P}(s_j; \boldsymbol{\theta}^-) + r_{s_j s_{j+1}}^{a_j} + \gamma \mathcal{L}_\eta \hat{P}(s_{j+1}; \boldsymbol{\theta}^-). \tag{3.1}$$

The network parameters are updated by applying gradient descent algorithms to minimize the loss function:

$$J(\boldsymbol{\theta}, \boldsymbol{\theta}^-) \triangleq (y_j(\boldsymbol{\theta}^-) - \hat{P}(s_j, a_j; \boldsymbol{\theta}))^2. \tag{3.2}$$

The loss of the gradient descent is added to the average loss. When the average loss is less than a threshold, the current sub-update is terminated and the target networks are updated after processing N mini-batch by $\boldsymbol{\theta}^- = \boldsymbol{\theta}$. In summary, the following are the main differences between DQN and DPN:

1. Local memory $\mathcal{D}$ with the current E iteration samples is applied to store fewer samples and focuses more on new samples. As the value function is updated using only the data sampled from the new policy of updating, the

improvement of the policy is fast.

2. Update $\boldsymbol{\theta}$ every episode with T steps rather than every step. The DPN updates in the same process as the DPP updating the value function.

3. Parameter $\boldsymbol{\theta}$ of DNN is updated with the number of epochs U using memory $\mathcal{D}$. Switch the target network and update $\boldsymbol{\theta}$ N times. By switching the target network while updating the value function, update the iteration of the value function without new samples.

3.1.2. Dueling Network Architecture for DPN

In this subsection, Dueling DPN (DDPN) is proposed as a natural extension of DPN toward a dueling network [24].

Plugging $V_{\bar{\pi}}(s) = \mathcal{L}_\eta P(s) \triangleq \frac{1}{\eta}\log\sum_{a\in\mathcal{A}}\exp(\eta P(s,a))$ into Eq. (2.11), we obtain:

$$P_{t+1}(s,a) = P_t(s,a) - V^t_{\bar{\pi}}(s) + \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'}\Big(r^a_{ss'} + \gamma V^t_{\bar{\pi}}(s)\Big). \tag{3.3}$$

Combining it with Eq. (2.8), the action preference function can be represented as:

$$P_t(s,a) = \frac{1}{\eta}\log\bar{\pi}^t(a|s) + V^t_{\bar{\pi}}(s), \tag{3.4}$$

You can directly derive Eq. (3.4) from Eqs. (2.9) and (2.10) because the denominator of Eq. (2.10) is given by $\exp(\eta V^t_{\bar{\pi}}(s))$. which can be naturally divided into two parts: value function $V^t_{\bar{\pi}}(s)$ and advantage function $A(s,a) \triangleq \frac{1}{\eta}\log\bar{\pi}^t(a|s)$. Fig. 3.3 shows the architecture of the Dueling DPN that consists of two streams of $V(s)$ and $A(s,a)$ while sharing one convolutional feature abstraction module. One regularization term is added to Eq. (3.4) to fulfill $\sum_{a'\in\mathcal{A}}\exp\Big(\log\bar{\pi}_t(a|s)\Big) = 1$:

$$P_t(s,a) = \frac{1}{\eta}\log\bar{\pi}_t(a|s) + V^t_{\bar{\pi}}(s) - \frac{1}{\|\mathcal{A}\|}\left\{\left(\sum_{a'\in\mathcal{A}}\exp\left(\log\bar{\pi}_t(a|s)\right)\right) - 1\right\}. \tag{3.5}$$

Algorithm 1: Deep P-Network

```
Initialize local memory $\mathcal{D}$ and its size $E$
Initialize network weights $\boldsymbol{\theta}$
Initialize target network weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
Function UpdatePNetwork($\boldsymbol{\theta}$, $\boldsymbol{\theta}^-$, $\mathcal{D}$, $N$):
    Initialize target network update number $N$
    Initialize epoch number $U$
    Initialize number of mini-batches $B = E/(\text{minibatch size})$
    for $n = 1, 2, ..., N$ do
        for $i = 1, 2, ..., U$ do
            Shuffle local memory $\mathcal{D}$ index
            for $j = 1, 2, ..., B$ do
                Sample minibatch of transition $(s_j, a_j, r^{a_j}_{s_j s_{j+1}}, s_{j+1})$ in $\mathcal{D}$
                Calculate the teaching signal:
                $y_j = \hat{P}(s_j, a_j; \boldsymbol{\theta}^-) - \mathcal{L}_\eta \hat{P}(s_j; \boldsymbol{\theta}^-) + r^{a_j}_{s_j s_{j+1}} + \gamma \mathcal{L}_\eta \hat{P}(s_{j+1}; \boldsymbol{\theta}^-)$
                Get *loss* and update $\boldsymbol{\theta}$ by performing a gradient descent
                  step on $(y_j - \hat{P}(s_j, a_j; \boldsymbol{\theta}))^2$
        Update target network $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
Initialize DPP parameters $I, M, T$
for $i = 1, 2, ..., I$ do
    for *episode* $= 1, 2, ..., M$ do
        for $t = 1, 2, ..., T$ do
            Take action $a_t$ with softmax policy based on $\bar{\pi}^t(a_t|s_t)$ following
              $\hat{P}(s_t, a_t; \boldsymbol{\theta})$ and Eq. (2.10)
            Receive new state $s_{t+1}$ and reward $r^{a_t}_{s_t s_{t+1}}$
            Store transition $(s_t, a_t, r^{a_t}_{s_t s_{t+1}}, s_{t+1})$ in $\mathcal{D}$
    UpdatePNetwork($\boldsymbol{\theta}$, $\boldsymbol{\theta}^-$, $\mathcal{D}$, $N$)
    if $i > (E - 1)$ then
        Update $\mathcal{D}$ to store the current $(E - 1)$ iterations' samples
```

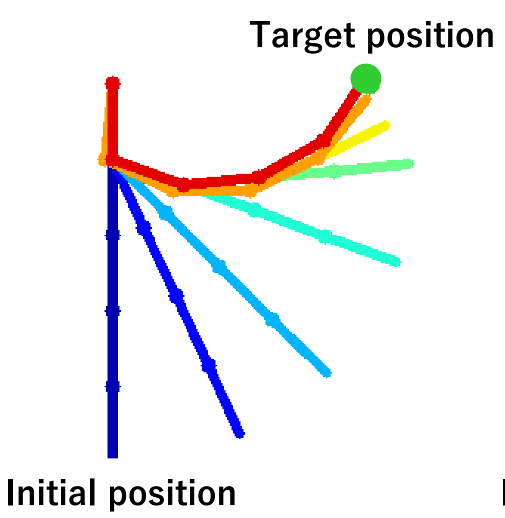
Algorithm 2: Initialization and learning of DPN policy

```
Initialize local memory $\mathcal{D}$ and its size $E$
Initialize network weights $\boldsymbol{\theta}$
Initialize target network weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
Load $L$ points initial sample and store in local memory $\mathcal{D}$
Initialize number of initial updates $K$
UpdatePNetwork($\boldsymbol{\theta}$, $\boldsymbol{\theta}^-$, $\mathcal{D}$, $K$)          # Initialization DPN policy
Initialize DPP parameters $I, M, T$
for $i = 1, 2, ..., I$ do
    for $episode = 1, 2, ..., M$ do
        for $t = 1, 2, ..., T$ do
            Take action $a_t$ with softmax policy based on $\bar{\pi}^t(a_t|s_t)$ following
              $\hat{P}(s_t, a_t; \boldsymbol{\theta})$ and Eq. (2.10)
            Receive new state $s_{t+1}$ and reward $r^{a_t}_{s_t s_{t+1}}$
            Store transition $(s_t, a_t, r^{a_t}_{s_t s_{t+1}}, s_{t+1})$ in $\mathcal{D}$
    UpdatePNetwork($\boldsymbol{\theta}$, $\boldsymbol{\theta}^-$, $\mathcal{D}$, $N$)
    if $i > (E - 1)$ then
        Update $\mathcal{D}$ to store the current $(E - 1)$ iteration samples
```

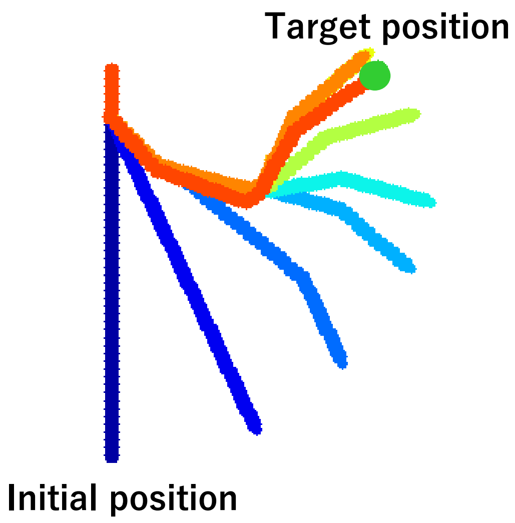
3.1.3. Prior Policy Initialization of DPN/Dueling DPN

Based on previous works [39] [40], DRL can successfully learn tasks with a small number of samples by initializing its policy from demonstrations. DPN and Dueling DPN are expected to fully exploit the successfully initialized policies based on prior knowledge since they employ a smooth policy update in the DRL framework. Even though the given initial policies may not be perfect, they should be beneficial for accelerating reinforcement learning.

We show an initialization procedure for both DPN and Dueling DPN using a small number of demonstration samples (generated by a human operator), summarized in Algorithm 2. By following Algorithm 2, we store these samples in local memory $\mathcal{D}$ and update the P-network before learning the results in the networks including prior knowledge. Parameter K controls the effect of the demonstration data on learning. The nature of the smooth policy update enables the DPN/Dueling DPN to smoothly update the resulting policies from the initialized ones.



(a) 5-DOF manipulater



(b) 50-DOF manipulater

Figure 3.4. Successful behaviors for n (5 and 50) DOF manipulator reaching tasks. Different colored robots represent manipulator states in different steps.

3.2. Simulation Results

3.2.1. Simulation Setting

In this section, we investigated the learning performance of DPN and Dueling DPN in a simulated n DOF manipulator reaching task ($n = 5, 15, 25, 50$) by comparing DQN and Dueling DQN. The state is the entire grayscale 84×84 px image where the n DOF manipulator is drawn, as shown in Fig. 3.4. The length of each limb between the adjoining joints is set to $\frac{1}{n}$. The environment and the DPN parameter settings are respectively shown in Tables 3.1a and 3.1b. For the network architecture, the input layer has $84 \times 84 \times 1$ nodes for each pixel of the state image. The setting of the middle layer and the optimizer follows previous works [15, 24]. The policies of DPN and Dueling DPN are calculated by Eq. (2.10) while DQN uses a ε-greedy policy. All the results are derived in five repetitions. Our hardware platform is a PC with an Intel Core i7-5960 CPU, a Nvidia GTX 1080 GPU, and 64 GB memory. The software platform was built by Tensorflow [41] and Keras [42].

MDP setting	**Description**
State	The entirety of a grayscale 84×84 px image.
Action	Discrete actions $[-0.0875, -0.0175, 0, 0.0175, 0.0875]$ (rad) to increment the joint with the respective angle for each DOF. We define an action at each time step as one move per joint to reduce the number of actions to $(N \times 5)$.
Reward	Reward function is set as $r = -\Big(\lvert X_{\text{target}} - X\rvert + \lvert Y_{\text{target}} - Y\rvert\Big)$ where X, Y is the current position of the manipulator's end-effector, and $X_{\text{target}} = 0.6830, Y_{\text{target}} = 0$ is the target position.
Initial state	The first joint is set to position $[0, 0]$. All angles are initialized to 0 rad at the start of the simulation.

(a) Parameter setting of n DOF manipulator reaching task.

Parameter	**Meaning**	**Value**
η	Parameters controlling the effect of smooth policy update	1
M	Number of episodes for one iteration	5
T	Number of steps for one episode	30
E	Size of memory $\mathcal{D}$ to store sample	450
N	Number of target network updates in one iteration	2
U	Epoch number of DNN updates	80
L	Number of samples for policy initialization	300
K	Number of P-network updates at policy initialization	20

(b) Parameter setting of DPN algorithm.

Table 3.1. Settings and learning parameters of n DOF manipulator reaching task

3.2.2. Learning Results

The learning results with different numbers of DOFs are shown in Fig. 3.5. The left side shows the results without the initialization procedure with demonstration samples. Both DPN and Dueling DPN clearly performed as well as expected in the simulations: they stably improved the performance with only around 2000 samples, but DQN and Dueling DQN could not. By separately learning the action preferences and value functions, Dueling DPN outperformed DPN. The right side of Fig. 3.5 shows the result with prior policy initialization (marked by "init", where the number of demonstration samples is defined as L in Table 3.1b, and in this task, we used 300 samples generated by a non-optimal policy). Compared with other algorithms, Dueling DPN supported the value of initialization more and quickly outperformed the performance of the demonstration samples (purple dashed line). All these results show the sample efficiency of DPN/Dueling DPN. They used fewer samples to achieve higher average reward values than the DQN algorithms under the same DNN setting, and their superiority rose with the DOF increase.

3.2.3. Effect of Parameter η in DPN and Dueling DPN

We investigated the effect of DPN's parameter η in a 5-DOF manipulator reaching task. With an increase of η, the Kullback–Leibler divergence term in Eq. (2.5) limits the policy update less. Since the operator $\mathcal{L}_\eta$ is the log-sum-exp function, it is considered a soft-max operator and it converge to the max operator as $\eta \to \infty$. Therefore, the choice of η determines the smoothness of the operator. In addition, DPP converges to the optimal policy for any η, but it changes the rate of convergence significantly. Both DPN and Dueling DPN were tested in five repetitions where $\eta = [0.01, 0.1, 0.5, 1.5, 3.0, 10, 20]$. Fig. 3.6a shows the results; in a suitable range, i.e., $[0.01, 0.3]$, a larger η resulted in faster learning due to less smoothness in the policy update. On the other hand, an extremely large η caused divergent learning due to the numerical instabilities using the exponential function in the action preferences function [32]. Dueling DPN also has better stability with various η than DPN, maybe because its architecture divides the action preferences function into two parts.

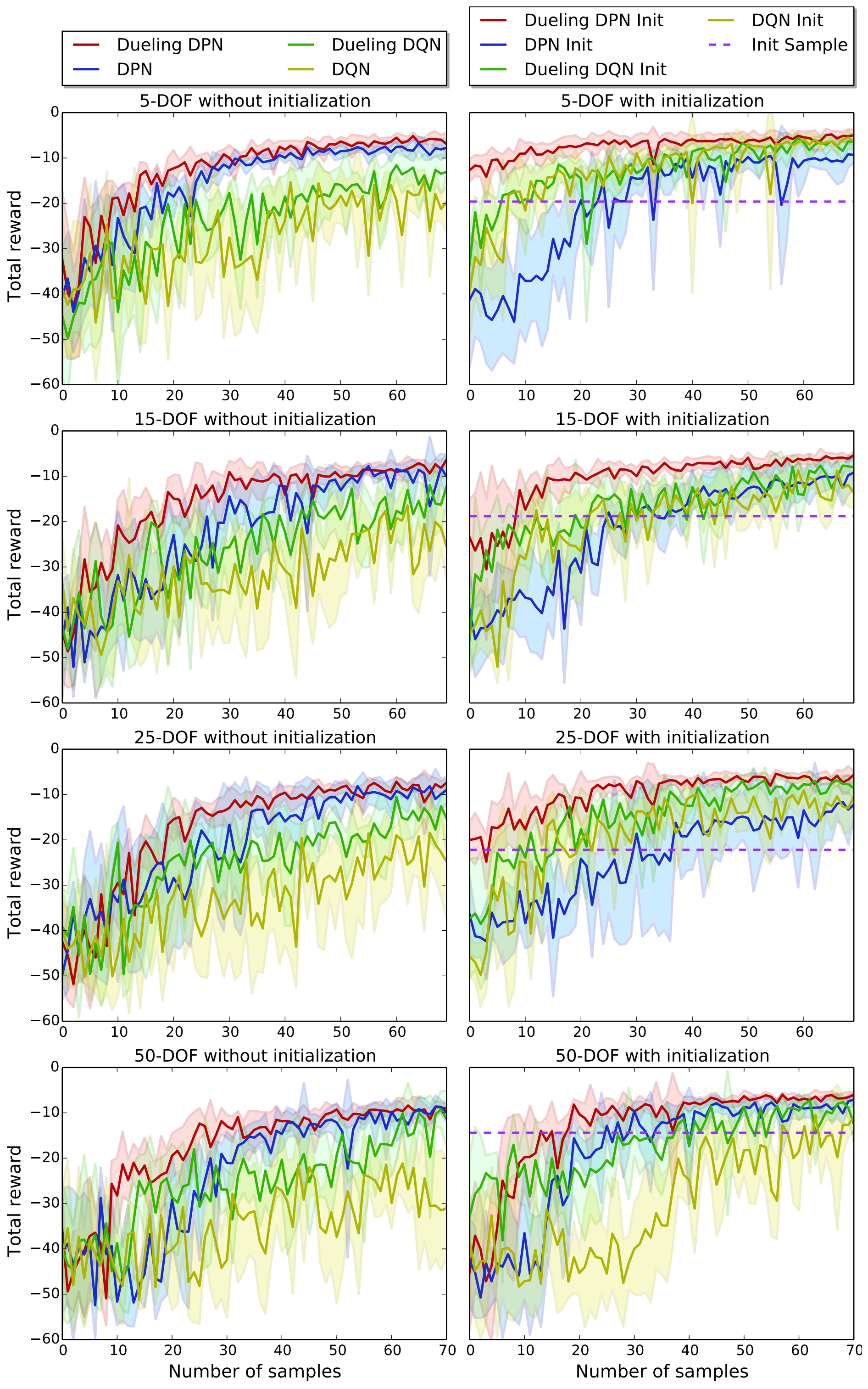
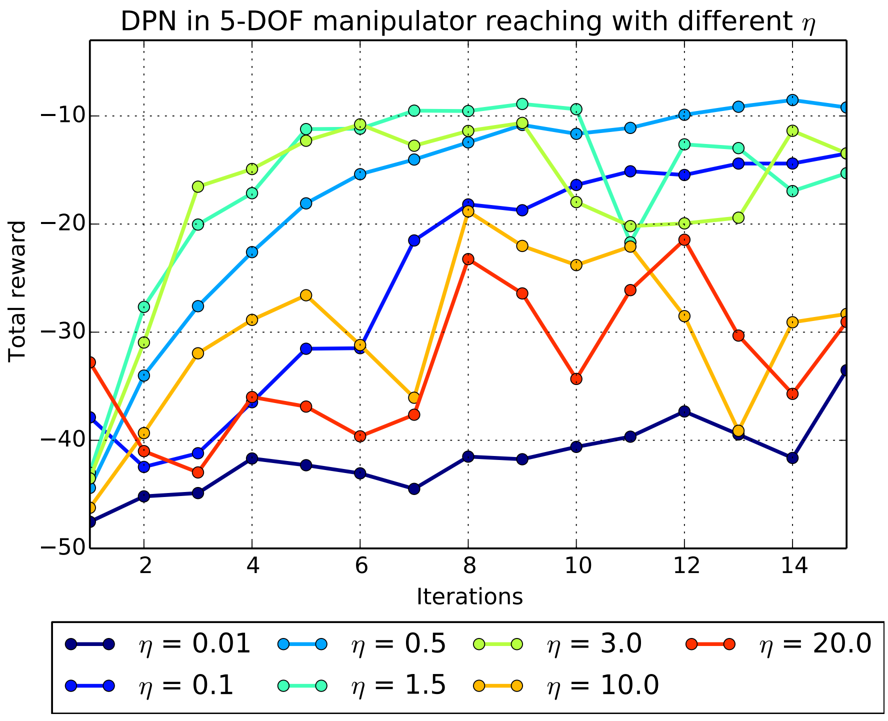
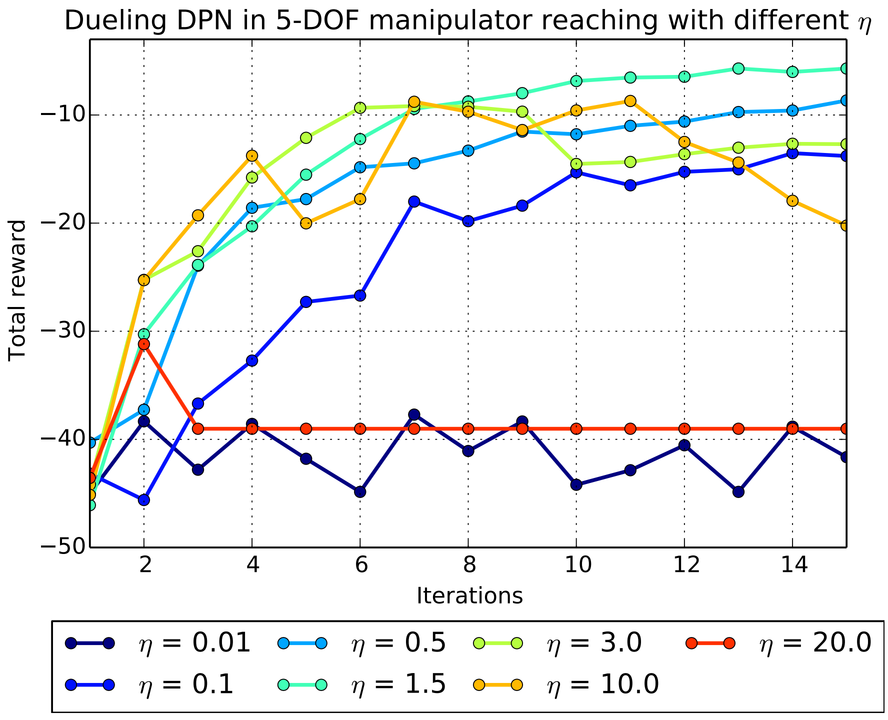


Figure 3.5. Learning curve of n DOF manipulator reaching task



(a) DPN



(b) Dueling DPN

Figure 3.6. Average learning results of DPN/Dueling DPN with different values of η in the 5-DOF manipulator reaching task over ten experiments

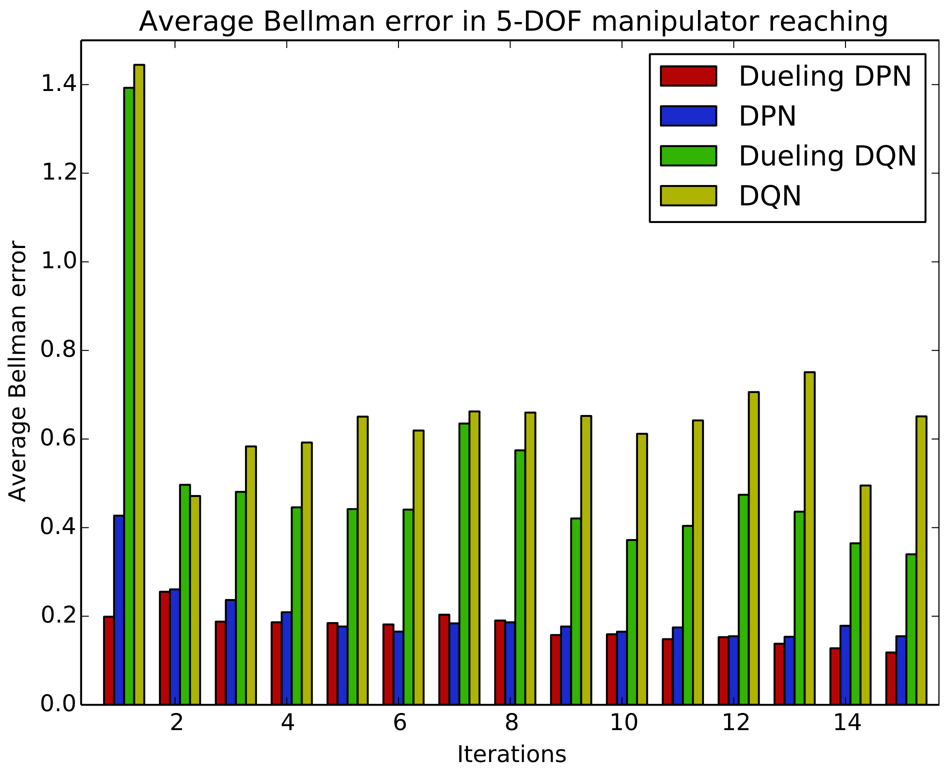


Figure 3.7. Average Bellman error in each iteration

3.2.4. Bellman Error in DPN

Next we investigated the effect of smooth policy update in DPN's function approximation. We define the Bellman error as:

$$BE(\hat{V}^{t+1}(s), r^a_{ss'}, \hat{V}^t(s)) = \|\hat{V}^{t+1}(s) - (r^a_{ss'} + \gamma \hat{V}^t(s))\|. \tag{3.6}$$

The Bellman error measures the approximation error of the value function during learning. State value function $\hat{V}^t$ of DPN can be calculated following Eq. (2.9), and the calculation in DQN follows $\hat{V}^t(s) = \mathbb{E}_\pi\Big[Q^t(s,a)\Big]$. Fig. 3.7 shows the average Bellman error in the first 15 iterations of ten learnings in 5-DOF manipulation reaching tasks. As a result, we achieved more accurate value function approximation in DPN and Dueling DPN than in DQN and Dueling DQN due to limiting the overly large updates that result in stable and efficient learning.

3.3. Real Robot Experiment

In this section, we applied the Dueling DPN to the NEXTAGE robot, a 15-DOF humanoid robot with sufficient precision for manufacturing, to learn two robotic cloth manipulation tasks. Following [43], we focus on learning a policy with high-level discrete actions, i.e., the grasp and release points on a cloth to solve two tasks: 1) flipping over a handkerchief and 2) folding a t-shirt.

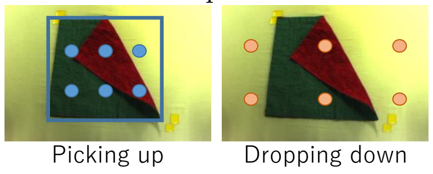
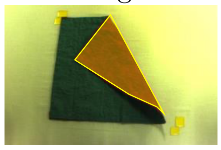
3.3.1. Flipping a Handkerchief

Setting

The environment and the DPN settings of this task are respectively shown in Tables 4.4a and 3.2b. The network architecture, the optimizer, and the hardware/software mainly follow the setting in Section 3.2.1, and the input layer is fixed to $84 \times 84 \times 3$ nodes for the RGB state image. The software is built on the Robot Operating System (ROS) [44].

Results

The learning results of three experiments are shown in Fig. 3.9. Each experiment took about four hours, including 40 minutes for manually initializing the handkerchief ($\approx$ 30 seconds per episode) to generate 2400 samples (= 16 iterations). The samples used for the Dueling DPN initialization were generated by a human operator who selected the actions for several states. Note that these samples are collected from non-expert demonstrations and they are insufficient to learn a good policy. The performance of the policy learned from these samples using supervised learning is shown as "Supervised" in Fig. 3.9. It is better than the random action but cannot be as good as the proposed methods. From the results, without prior policy initialization, Dueling DPN converged faster and achieved higher reward than Dueling DQN. With only 300 demonstration samples for initialization, Dueling DPN learned better policies by exploring with 2400 additional samples. Both Dueling DPN with/without prior policy initialization outperformed the corresponding Dueling DQN algorithms. Fig. 3.8 shows one example of a handkerchief flipping process learned by Dueling DPN, which turned

MDP setting	**Description**
State	The input state is a 84×84 px RGB image from NEXTAGE's integrated camera.
Action	$6 \times 6 = 36$ gripper actions are defined as picking up the handkerchief from 2×3 points over its current area and dropping it down to 2×3 points over the table. 
Reward	The reward is defined as the ratio of the red area over the whole image in the current state. 
Initial state	The handkerchief is initially placed green side up by a human.

(a) Parameter setting of flipping handkerchief task

Parameter	**Meaning**	**Value**
η	Parameters controlling the effect of smooth policy update	1
M	Number of episodes for one iteration	5
T	Number of steps for one episode	30
E	Size of memory $\mathcal{D}$ to store samples	450
N	Number of target network updates in one iteration	2
U	Epoch number of DNN updates	40
L	Number of samples for policy initialization	300
K	Number of P-network updates at policy initialization	20

(b) Parameter setting of DPN algorithm

Table 3.2. Settings and learning parameters of flipping handkerchief task

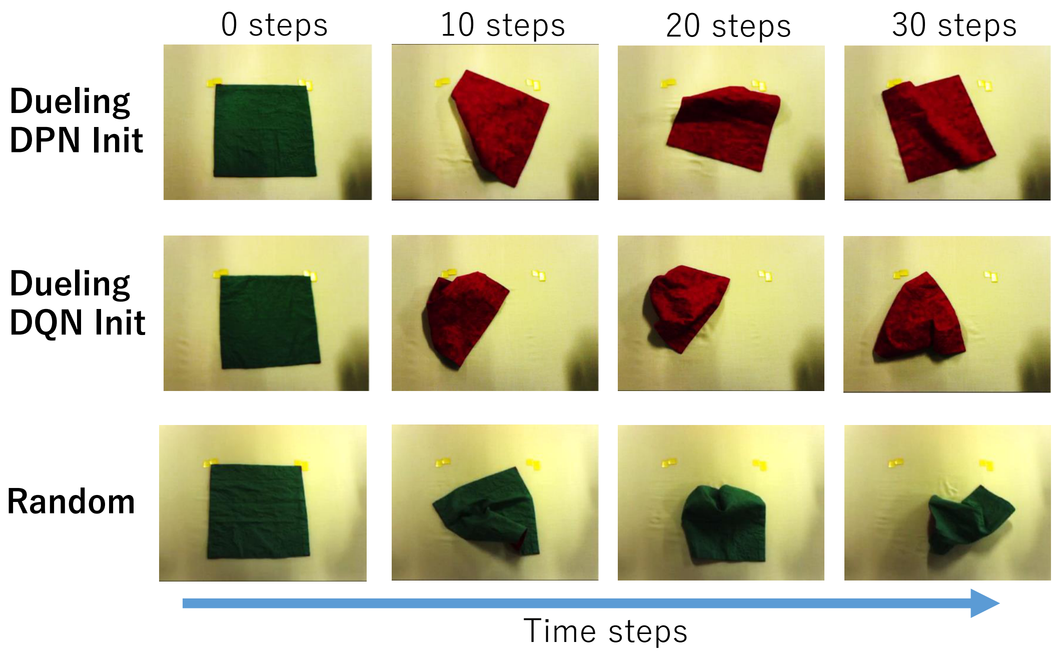


Figure 3.8. Handkerchief folding trajectory generated from policy learned from 2400 samples

about 80% of the handkerchief over in about 15 steps.

3.3.2. Folding a T-shirt

Setting

The next task was to fold a t-shirt. The details of the environment and the algorithm settings are respectively summarized in Tables 4.4a and 3.2b. The network architecture, the optimizer, and the hardware/software settings are the same as in the handkerchief flipping task. It is a more challenging task than flipping handkerchief due to 1) a larger action space to deal with various clothing operations, 2) a more complex reward function that shows the stepwise folding achievement degree as in Algorithm 3, 3) fewer steps in one rollout, 4) fewer samples (80) generated by a human operator for prior policy initialization.

Results

The averaged learning results based on three experiments are shown in Fig. 3.11. Dueling DPN successfully learned the task with only 80 demonstration samples for

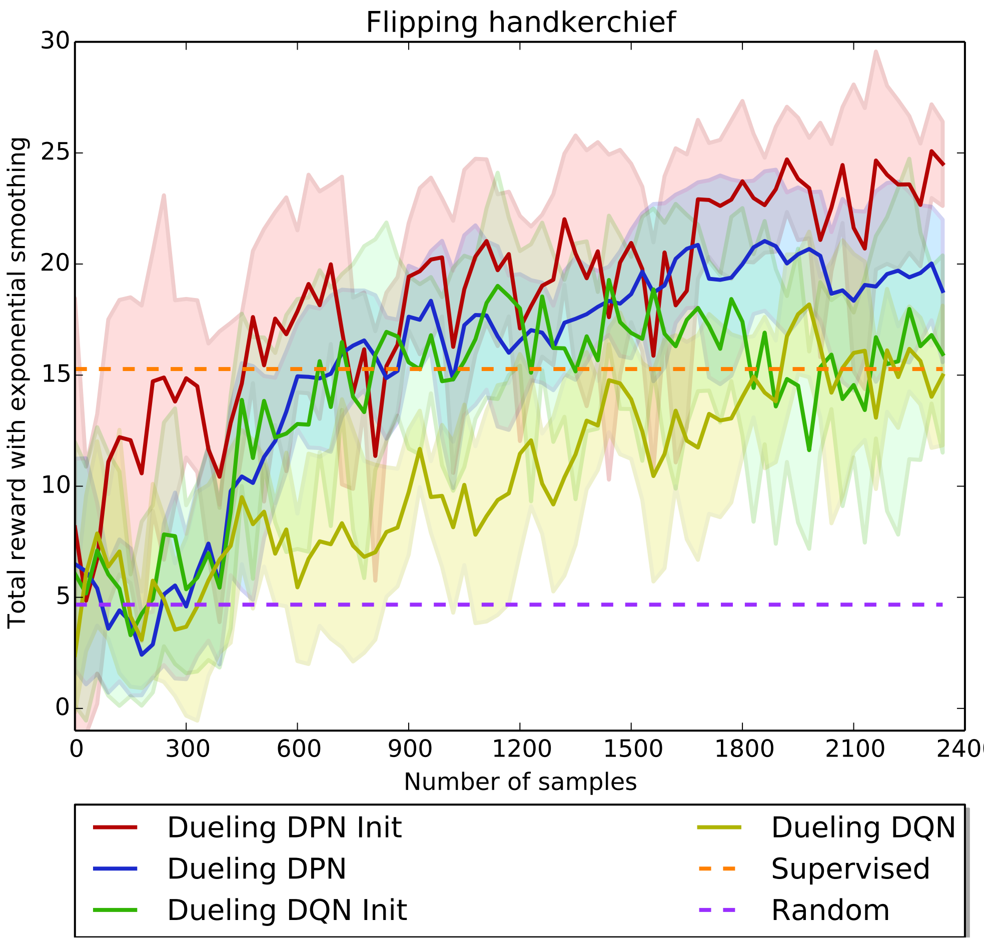
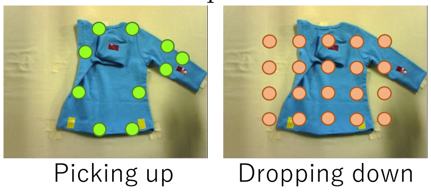
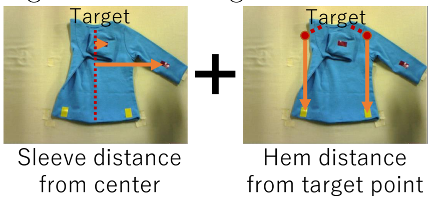


Figure 3.9. Learning curve of flipping a handkerchief

MDP setting	**Description**
State	The input state is a 84×84 px RGB image from the NEXTAGE's integrated camera.
Action	$10 \times 20 = 200$ gripper actions are defined as picking up the t-shirt from 10 points over its current area and dropping it down to 5×4 points over the table. 
Reward	The reward function is designed to trigger an action to fold the hem after folding the sleeve. The processing is shown in Algorithm 3.  Target Target Sleeve distance from center Hem distance from target point
Initial state	T-shirt is initialized to a state in which it is spread by a human.

(a) Parameter setting of folding t-shirt task

Parameter	**Meaning**	**Value**
η	Parameters controlling the effect of smooth policy update	1
M	Number of episodes for one iteration	5
T	Number of steps for one episode	8
E	Size of memory $\mathcal{D}$ to store samples	120
N	Number of target network updates in one iteration	2
U	Epoch number of DNN updates	30
L	Number of samples for policy initialization	80
K	Number of P-network updates at policy initialization	5

(b) Parameter setting of DPN algorithm

Table 3.3. Settings and learning parameters of folding t-shirt task

Algorithm 3: Reward function of t-shirt folding task

```
Initialize $InitHemR = [0.675, 0.8]$, $InitHemL = [0.325, 0.8]$
Initialize $TargetHemR = [0.675, 0.208]$, $TargetHemL = [0.325, 0.208]$
Function HemReward(SleevePoint, CenterHem):
    Initialize reward = 0
    reward = -Sum(|SleevePoint - CenterHem|)
    return reward
Function SleeveReward(HemPoint, InitHem, TargetHem):
    Initialize reward = 0
    Initalize $Distance = |InitHem - TargetHem|$
    reward = Sum($Distance - |HemPoint - TargetHem|$)
    return reward
Function ShirtReward():
    Initialize reward = 0
    Update color marker
    Get HemPointR, HemPointL, SleevePointR, SleevePointL
    if Detect hem marker then
        CenterHem = (HemPointR + HemPointL)/2
        reward = SleeveReward(SleevePointR, CenterHem) +
          SleeveReward(SleevePointL, CenterHem)
    else
        reward = 1
    if Detect sleeve marker then
        reward =
          reward + HemReward(HemPointR, InitHemR, TargetHemR) +
          HemReward(HemPointL, InitHemL, TargetHemL)
    return reward
```

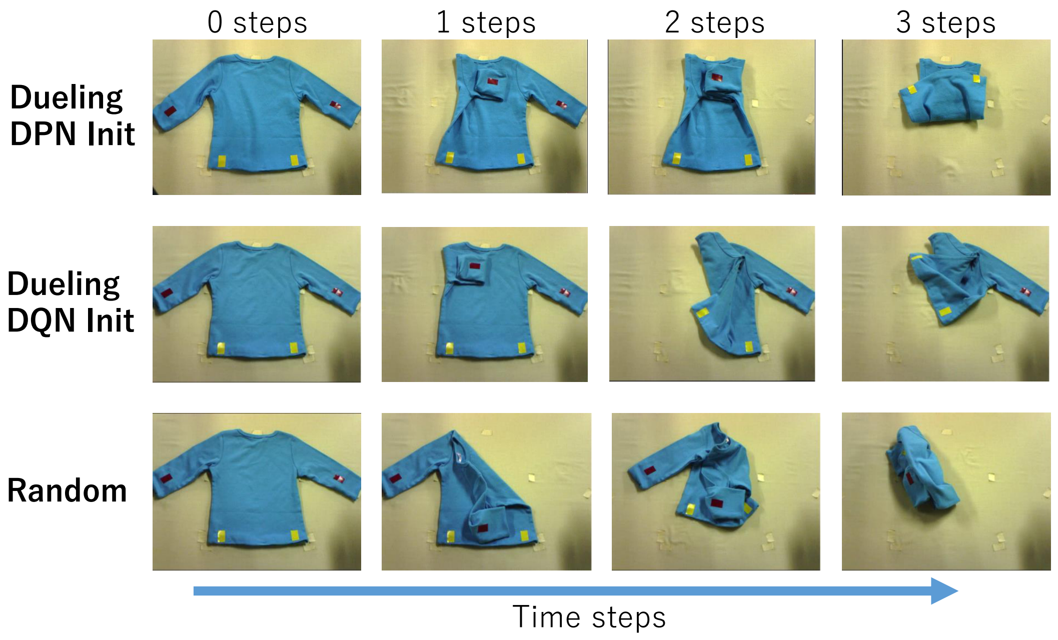


Figure 3.10. T-shirt folding trajectory generated from policy learned from 2400 samples

policy initialization and 112 additional samples generated during reinforcement learning. The samples for initialization are collected from non-expert demonstrations. The line "Supervised" in Fig. 3.11 shows that the policy learned by these samples using supervised learning could not lead to a good policy. According to Fig. 3.11, only Dueling DPN was able to improve its performance based on the given non-expert demonstration while other methods gradually improved their performance but never learned sufficient policies for folding the t-shirt. These results suggest that only our proposed method, Dueling DPN, has the capability to learn tasks with a large action space and a complex reward function even with very limited samples. Fig. 3.10 shows one t-shirt folding procedure learned by Dueling DPN and DQN with prior policy initialization. Dueling DPN init successfully folded it by appropriately selecting three actions per step, but the corresponding Dueling DQN could only achieve the first step.

Several examples of high-level features learned by Dueling DPN are visualized by Grad-CAM [45] in Fig. 3.12. These heat maps where the red/blue colors indicate high/low attention of the agent indicate that our proposed method successfully learned useful and meaningful features. The t-shirt's sleeves drew the

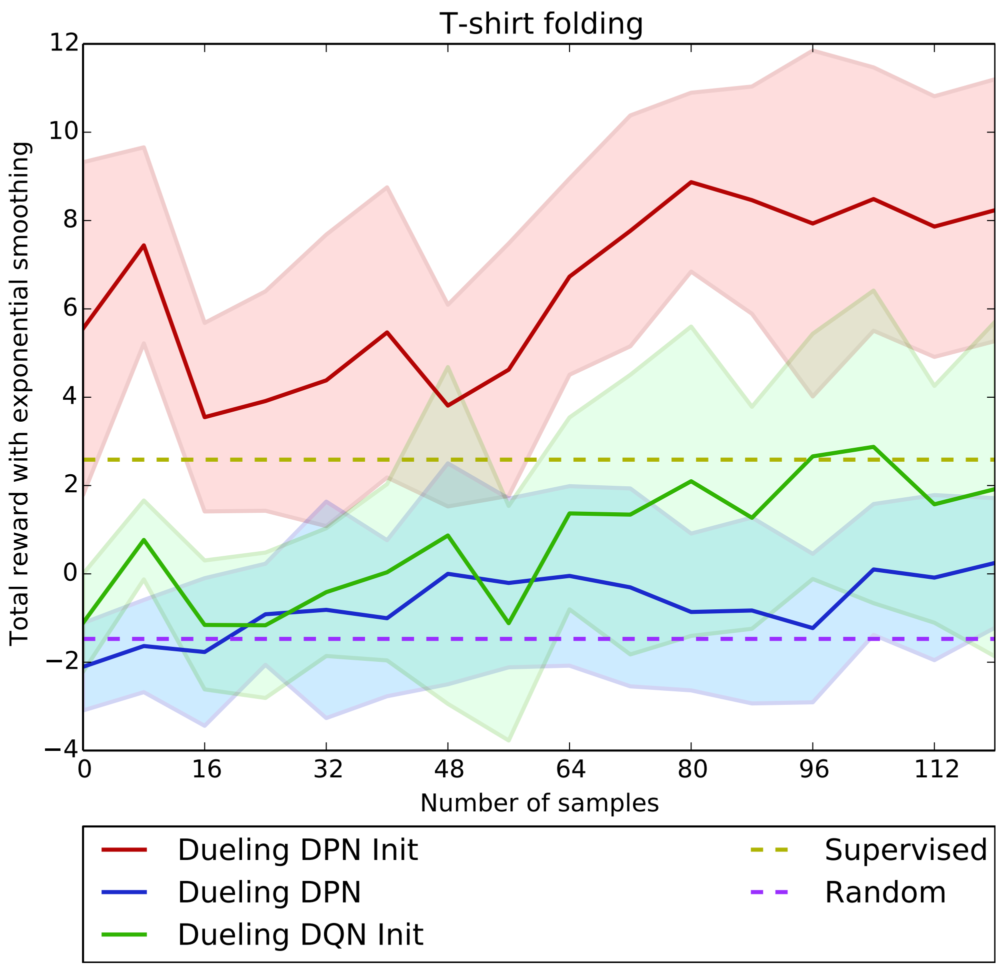


Figure 3.11. Learning curve of t-shirt folding task

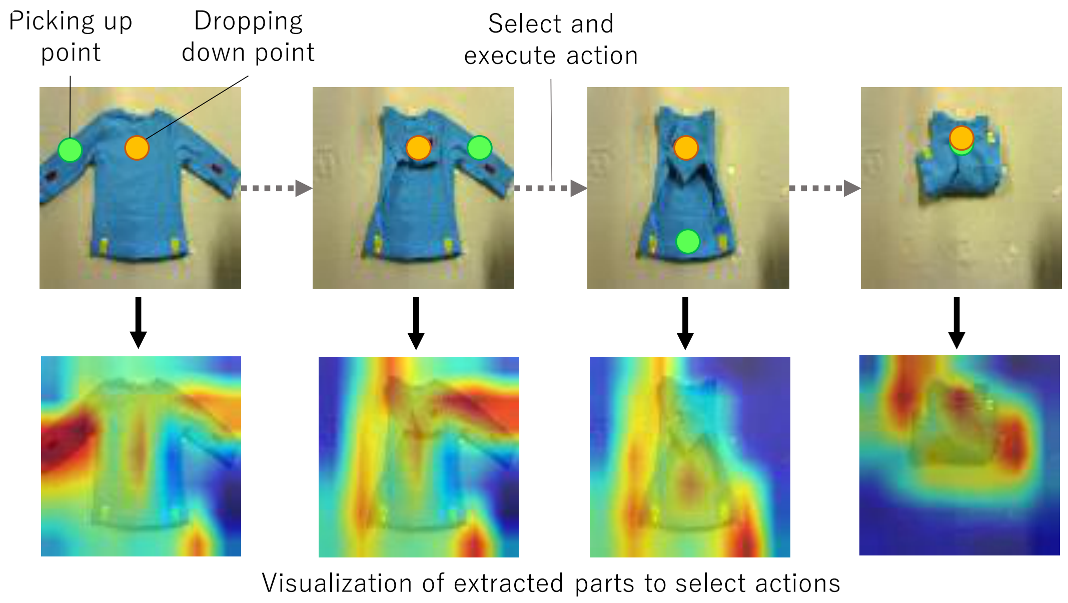

Picking up point
Dropping down point
Select and execute action
Visualization of extracted parts to select actions

Figure 3.12. Visualization of extracted parts in images for action selection using Grad-CAM. Heat map shows parts extracted when actions are selected.

agent's attention following the order of operations in the first two steps. Then the hem's corner was concerned more to finish the folding task.

3.4. Summary of Chapter 3

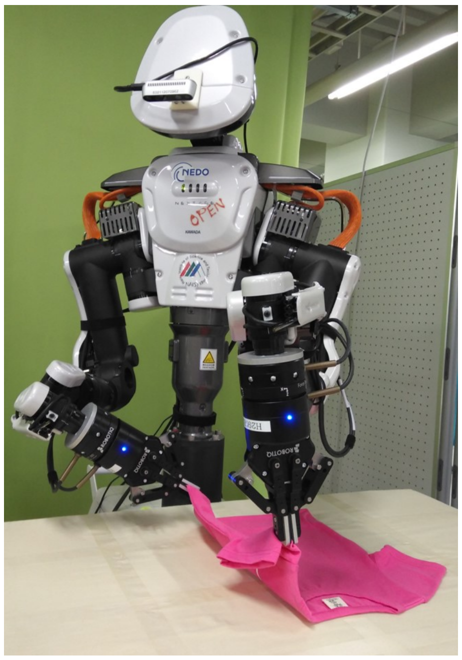
The contribution of this chapter is twofold. We proposed two new deep reinforcement learning algorithms, Deep P-Network (DPN) and Dueling Deep P-Network (DDPN). The core idea shared by them is to combine the nature of smooth policy update in value function based reinforcement learning (Dynamic Policy Programming) with the capability of automatic feature extraction in deep neural networks to enhance the sample efficiency and the learning stability process fewer samples. We compared them with previous DRL methods in a simulated n DOF manipulator reaching task to investigate our proposed methods' performance. Furthermore, we applied them to two robotic cloth manipulation tasks with a dual-arm robot, NEXTAGE: 1) flipping of a handkerchief and 2) folding a t-shirt with a limited number of samples. We confirmed in all the experiments that our method achieved more sample efficiency and stabilized learning than the previous methods.

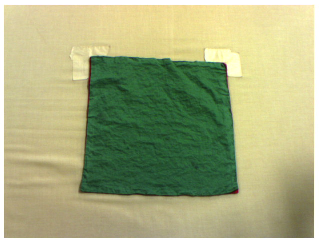
4. Generative Adversarial Imitation Learning with Human Demonstration and Target State Label

In the previous chapter, the reward function design such as folding a t-shirt has become complex because evaluating the flexible clothes state is difficult. Therefore, this chapter focuses on a Generative Adversarial Imitation Learning framework (GAIL), which allows an agent to learn near-optimal behaviors from demonstrations and self explorations without explicit reward function design.

When applying GAIL to a real robot task, humans use the robot's action spaces to collect demonstrations. In this case, it is not easy to collect suitable demonstrations because human's and robot's state action spaces are different. For example, when the robot's action space in a folding task is a folding line, the human selects a folding line and collects a demonstration while watching the monitor. Whether the state transitions as expected by humans can only be known after the robot acts. Thus, it is difficult for a beginner to collect a complete demonstration. On the other hand, a human can present the target state of the task. In this chapter, we propose the P-Generative Adversarial Imitation Learning with Bi-Discriminator (PGAIL-BiD), which includes two discriminators: an expert discriminator and a target discriminator. The proposed method encourages reaching the target state via the target discriminator and learns the better performing policy even when demonstrations is imperfect.

PGAIL-BiD is first applied to a n DOF simulated manipulator reaching task to evaluate their learning performances and compare the effect of different demon-


(a) NEXTAGE: dual arm humanoid robot


(b) Handkerchief

(c) Shirt

(d) Trousers

Figure 4.1. Real robot setting: Our targets are two robotic cloth manipulation tasks with a dual-arm humanoid robot NEXTAGE (a) 1) flipping of a handkerchief (b) and 2) folding shirt (c) and trousers (d).

strations and parameter settings with previous GAIL methods. Then PGAIL-BiD is applied to real robotic cloth manipulation tasks to control a dual-arm humanoid robot NEXTAGE (Fig. 4.1a) to learn 1) the flipping of a handkerchief (Fig. 4.1b) and 2) folding a shirt and trousers (Fig. 4.1c, Fig. 4.1d)

4.1. Proposed Method

4.1.1. Structure of PGAIL-BiD

As shown in Fig. 4.2, PGAIL-BiD consists of following three components:

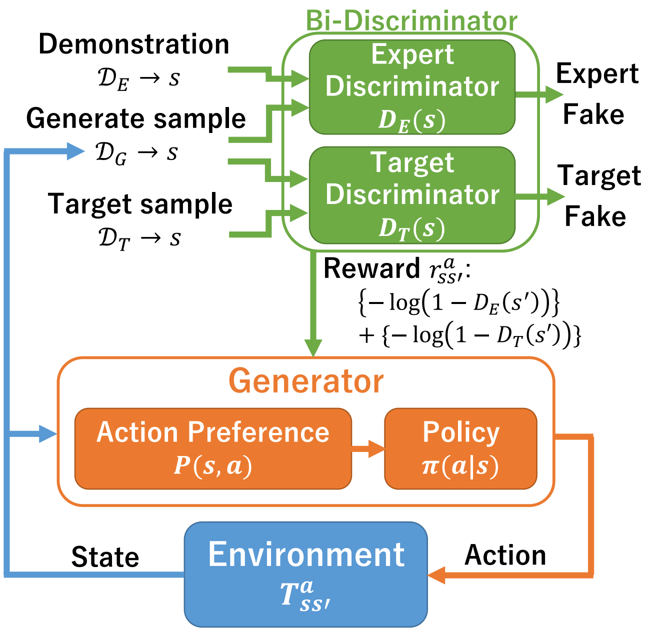


Figure 4.2. Overview of proposed PGAIL-BiD.

- Expert discriminator $D_E(s)$ distinguishing between demonstration states and generated states.
- Target discriminator $D_T(s)$ distinguishing between target states and generated states.
- Generator $\pi(a|s)$ learns policies to achieve the target state while imitating the demonstration.

Compared to the previously proposed GAIL frameworks in Fig. 4.3. The target sample and the target discriminator are increased, and the reward consists of an expert discriminator and a target discriminator. Since the target sample $\mathcal{D}_T$ is a state selected from within demonstrations $\mathcal{D}_E$, the relation is $\mathcal{D}_T \subset \mathcal{D}_E$. Two discriminators learn a better classification ability while the generator learns to confuse two discriminators. Since the input of the target discriminator is state only, the expert discriminator's input is also state only.

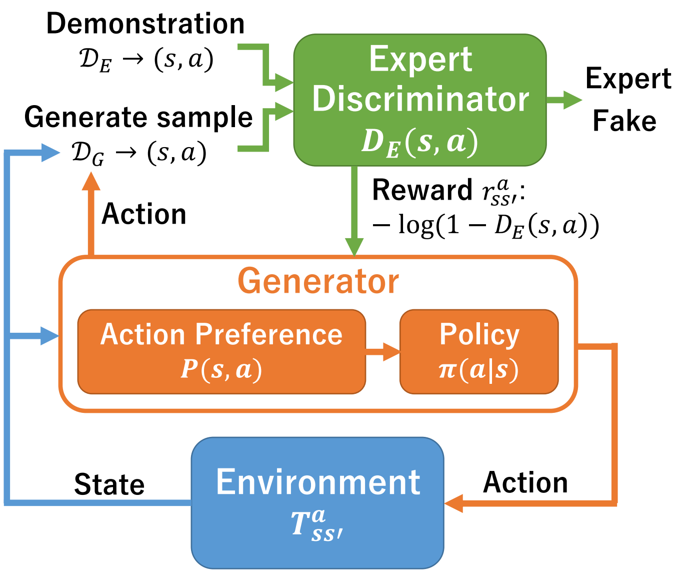


Figure 4.3. Overview of proposed PGAIL.

The objective of the PGAIL-BiD is formulated as follows:

$$\begin{aligned}\max_{\pi}\min_{D_E}\min_{D_T} \quad & \mathbb{E}_{\pi}[-\log(1-D_E(s)) - \log(1-D_T(s))] \\ & + \mathbb{E}_{\pi_E}[-\log(D_E(s))] + \mathbb{E}_{d_T}[-\log(D_T(s))].\end{aligned} \tag{4.1}$$

Given the state s as input, two discriminator outputs the probability $D_E(s) \in [0,1], D_T(s) \in [0,1]$. π is the policy learned by the generator, π_E is the demonstration policy. $d_T(s)$ is stationary distribution of observe target state. Since the generator and two discriminator are approximated by DNNs, we take a gradient-based numerical simultaneous optimization approach same as GAIL. More details of the optimization are given in subsequent sections.

4.1.2. Generator Optimization

Given expert discriminator $D_E(s)$ and target discriminator $D_T(s)$, the updated policy π of generator is formulated as:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}[-\log(1-D_E(s)) - \log(1-D_T(s))]. \tag{4.2}$$

The reward function of RL is formulated as follows:

$$r_{s'} = \{-\log(1 - D_E(s'))\} + \{-\log(1 - D_T(s'))\} \tag{4.3}$$

The reward is calculated from the transition state s' and two discriminators. In order to stably learn the policy from the reward changing by the learning of the discriminator, it is necessary to update the policy smoothly while maintaining the diversity of the policy. PGAIL-BiD learns the policy of generator by EDPN by following DPN with an important modification so that it has such two properties as smooth update and diversity in the policy. To this end, the reward function is designed by

$$r_{EDPN} = r_{s'} - \frac{1}{\eta}\mathrm{KL}(\ \pi \parallel \bar{\pi}\ ) + \sigma H(\ \pi\ ) \tag{4.4}$$

where

$$\mathrm{KL}(\ \pi \parallel \bar{\pi}\ ) = \sum_{a \in \mathcal{A}} \pi(a|s) \log \frac{\pi(a|s)}{\bar{\pi}(a|s)} \tag{4.5}$$

and

$$H(\ \pi\ ) = \sum_{a \in \mathcal{A}} -\pi(a|s) \log\left(\pi(a|s)\right). \tag{4.6}$$

The second term is promoting smooth policy update, and the amount of update is quantified by the Kullback-Leibler divergence $\mathrm{KL}(\pi \parallel \bar{\pi})$ between the current policy π and the baseline policy $\bar{\pi}$. The third term is promoting the diversity of actions in policy, and calculate the entropy $H(\pi)$. η and σ are coefficients that control their balance. By learning the policy that maximizes this total reward, the policy is smoothly updated while maximizing the entropy of the policy. Value function-based reinforcement learning with these two constraints has been robust to the approximation error of value function [46].

Following [29], the update rule of EDPN's action preferences function is derived as:

$$P_{t+1}(s,a) = \frac{1}{1+\sigma\eta}(P_t(s,a) - V^t_{\bar{\pi}}(s)) + \sum_{s\in\mathcal{S}} T^a_{ss'}\left(r_{s'} + \gamma V^t_{\bar{\pi}}(s')\right) \tag{4.7}$$

where

$$\bar{\pi}_t(a|s) = \frac{\exp(\frac{\eta}{1+\sigma\eta}P_t(s,a))}{\sum_{a'\in\mathcal{A}}\exp(\frac{\eta}{1+\sigma\eta}P_t(s,a'))} \tag{4.8}$$

and

$$V^t_{\bar{\pi}}(s) = \frac{1+\sigma\eta}{\eta}\log\sum_{a\in\mathcal{A}}\exp(\frac{\eta}{1+\sigma\eta}P_t(s,a)). \tag{4.9}$$

$T^a_{ss'}$ is the probability of transitioning from state s to state s' under action a. $\gamma \in (0,1)$ is the discount parameter.

When learning a policy for image-based inputs, the action preferences function is approximated by DNNs. θ is defined as the DNNs' parameters that approximate the action preferences function and calculates teaching signal $y(\theta^-)$ following Eq. (4.7):

$$y(\theta^-) = \frac{1}{1+\sigma\eta}(\hat{P}_t(s,a;\theta^-) - \hat{V}^t_{\bar{\pi}}(s;\theta^-)) + \sum_{s\in\mathcal{S}} T^a_{ss'}\left(r^a_s + \gamma\hat{V}^t_{\bar{\pi}}(s';\theta^-)\right). \tag{4.10}$$

The current parameters as θ^- are saved to build a target network. Then, the gradient of the following error function $J(\theta,\theta^-)$ is computed and then used to update the parameter θ using the stochastic gradient descent:

$$J(\theta,\theta^-) \triangleq (y(\theta^-) - \hat{P}(s,a;\theta))^2. \tag{4.11}$$

Note that if $\sigma = 0$, the EDPN becomes equivalent to DPN, thus, EDPN is a generalized version of DPN to be suitable for GAIL.

4.1.3. Discriminator Optimization

Two discriminator's error function is defined as:

$$J_E(\phi) = \mathbb{E}_{\pi}[-\log(1 - D_E(s;\phi))] + \mathbb{E}_{\pi_E}[-\log(D_E(s;\phi))] \tag{4.12}$$

$$J_T(\omega) = \mathbb{E}_{\pi}[-\log(1 - D_T(s;\omega))] + \mathbb{E}_{d_T}[-\log(D_T(s;\omega))] \tag{4.13}$$

where ϕ, ω is defined as the DNNs parameters that approximate the discriminator. To update the parameter ϕ, ω using the stochastic gradient descent, since the discriminator can be a simpler function than that of policy, the learning progress would be faster than the policy. Thus, the parameters ϕ, ω needs to be updated conservatively to avoid over-fitting.

4.1.4. Summary

The pseudocode of training PGAIL-BiD is shown in Algorithm 4. After policy initialization, the training of PGAIL-BiD has four steps in one iteration, and executes them a fixed number of times.

1. generate new samples following current policy π based on the current action preferences function
2. update the discriminator via updating ϕ, ω to minimize Eq. (4.12) and Eq. (4.13)
3. calculate the reward of $\mathcal{D}_G$ following the updated two discriminator D_E, D_T
4. update the generator using EDPN

4.2. Simulation Results

4.2.1. Simulation Setting

In this section, the PGAIL-BiD is first evaluated in a n-DOF manipulator reaching task ($n = 2, 5$) in simulation. The state is the color image $84 \times 84 \times 3$ px

Algorithm 4: PGAIL-BiD

```
Initialize local memory $\mathcal{D}_G$ its size $N$
Load demonstrations $\mathcal{D}_E$
Load target samples $\mathcal{D}_T$
Initialize value network weights $\theta$, target network weights $\theta^- = \theta$
Initialize discriminator network weights $\phi, \omega$
Initialize PGAIL-BiD iteration number $I$
Initialize generator parameters $M, T$
Initialize discriminator network iteration number $J$
Initialize value network iteration number $K$
for $i = 1, 2, ..., I$ do
    for $episode = 1, 2, ..., M$ do
        for $t = 1, 2, ..., T$ do
            Execute policy $\pi_i$ represented by $P^i(s, a; \theta)$, generate samples $\mathcal{D}_G$
    for $j = 1, 2, ..., J$ do
        Update the parameters $\phi, \omega$ of discriminator $D^i_E(s; \phi), D^i_T(s; \omega)$ to minimize Eq. (4.12, 4.13) based on $\mathcal{D}_E, \mathcal{D}_T$ and $\mathcal{D}_G$
    Calculate the reward of generated samples $\mathcal{D}_G$ according to $D^i_E(s; \phi), D^i_T(s; \omega)$ following Eq. (4.3)
    for $k = 1, 2, ..., K$ do
        Update the parameters $\theta$ of $P^i(s, a; \theta)$ using $\mathcal{D}_G$ and $\theta^-$, following Eq. (4.11)
    Update target network $\theta^- = \theta$
```

image where the n DOF manipulator is drawn, as shown in Fig. 4.4. The length of each limb between the adjoining joints is set to $\frac{1}{n}$. The environment and the PGAIL-BiD parameter settings are respectively shown in Tables 4.1a and 4.1b. In this simulation, we investigated three things: 1) learning from imperfect demonstrations, 2) comparing PGAIL-BiD and PGAIL with target discriminator only, and 3) visualization of estimated reward functions. Through these investigations, we show that PGAIL-BiD learns better performing policies from imperfect demonstrations.

MDP setting	Description
State	The entirety of a 84×84 px color image. The number of state dimensions is $84 \times 84 \times 3 \quad (21,168)$
Action	Discrete actions $[-0.0875, -0.0175, 0, 0.0175, 0.0875]$ (rad) to increment the joint with the respective angle for each DOF. We define an action at each time step as one move per joint to reduce the number of actions to $(N \times 5)$.
Reward for evaluation	Reward function is set as $r'_s = -(\lvert X_{\text{target}} - X\rvert + \lvert Y_{\text{target}} - Y\rvert)$ where X, Y is the current position of the manipulator's end-effector, and $X_{\text{target}} = 0.6830, Y_{\text{target}} = 0$ is the target position.
Initial state	The first joint is set to position $[0, 0]$. All angles are initialized to 0 rad at the start of the simulation.
Demonstrations	Demonstrations (9 samples) are sampled from an RL policy that maximizes the reward for evaluation. Demonstrations (20 steps $\times$ 5 trajectory = 100 samples) are generated from exploring policies during the learning process. The target samples are selected from demonstrations, with the distance between the manipulator's end-effector and the target position less than 0.1.

(a) Parameter setting of n DOF manipulator reaching task.

Parameter	Meaning	Value
η	Parameters controlling the effect of smooth policy update	1
σ	Parameters controlling the effect of causal entropy	0.05
M	Number of episodes for one iteration	10
T	Number of steps for one episode	20
N	Size of memory $\mathcal{D}_G$ to store sample	2000
J	Iteration number of discriminator updates	10
K	Iteration number of value network updates	20

(b) Parameter setting of PGAIL-BiD algorithm.

Table 4.1. Settings and learning parameters of n DOF manipulator reaching task

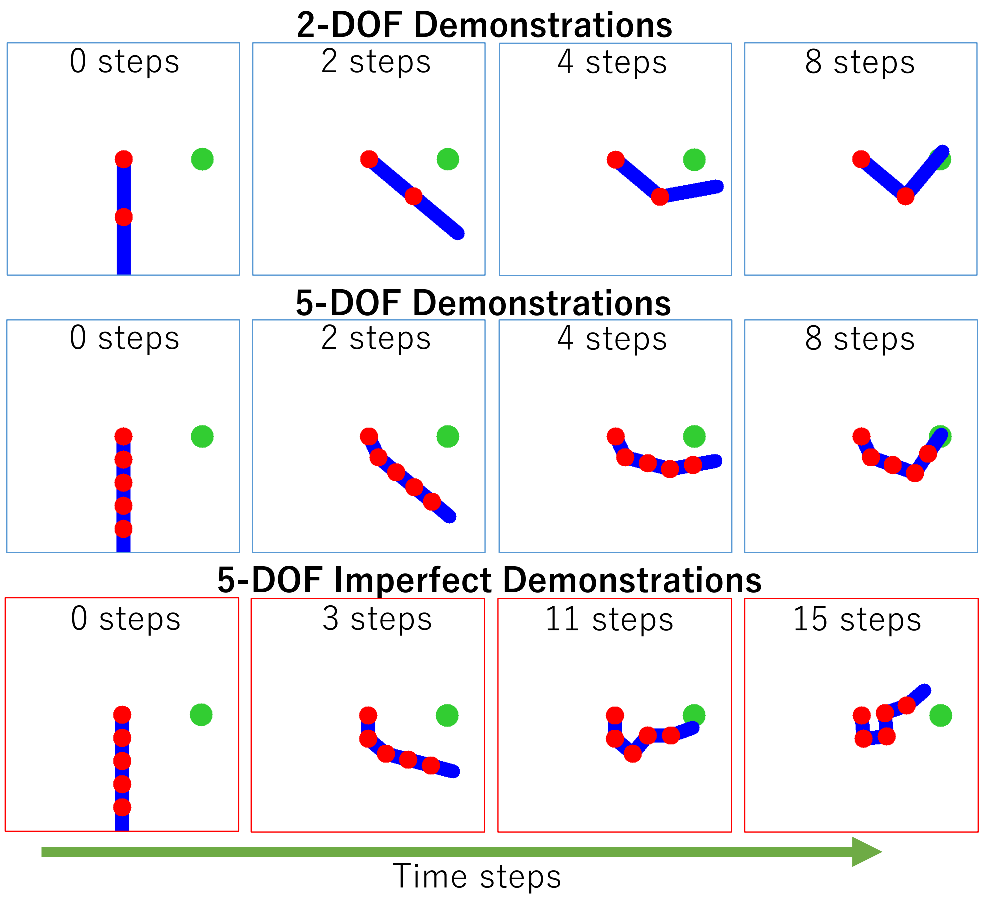


Figure 4.4. The snapshot show demonstrations of n-DOF manipulator reaching task. Demonstrations are sampled from RL policies. RL exploration policies generate imperfect demonstrations.

4.2.2. Learning from Imperfect Demonstrations

We first investigate the learning policy performances from imperfect demonstrations. To get different demonstrations of performance, we collected imperfect demonstrations from RL exploration policies with $[5, 7, 9, 11]$ updates. To verify the effect of smooth policy update and entropy-maximizing, we compared EDPN and naive DQN. After this, GAIL with DQN is called "QGAIL."

The learning results with different demonstrations are shown in Fig. 4.5 where the Y axis is the performance defined as a scaled total baseline reward $r'_s = -\Big(|X_{\text{target}} - X| + |Y_{\text{target}} - Y|\Big)$. The demonstration policy's performance

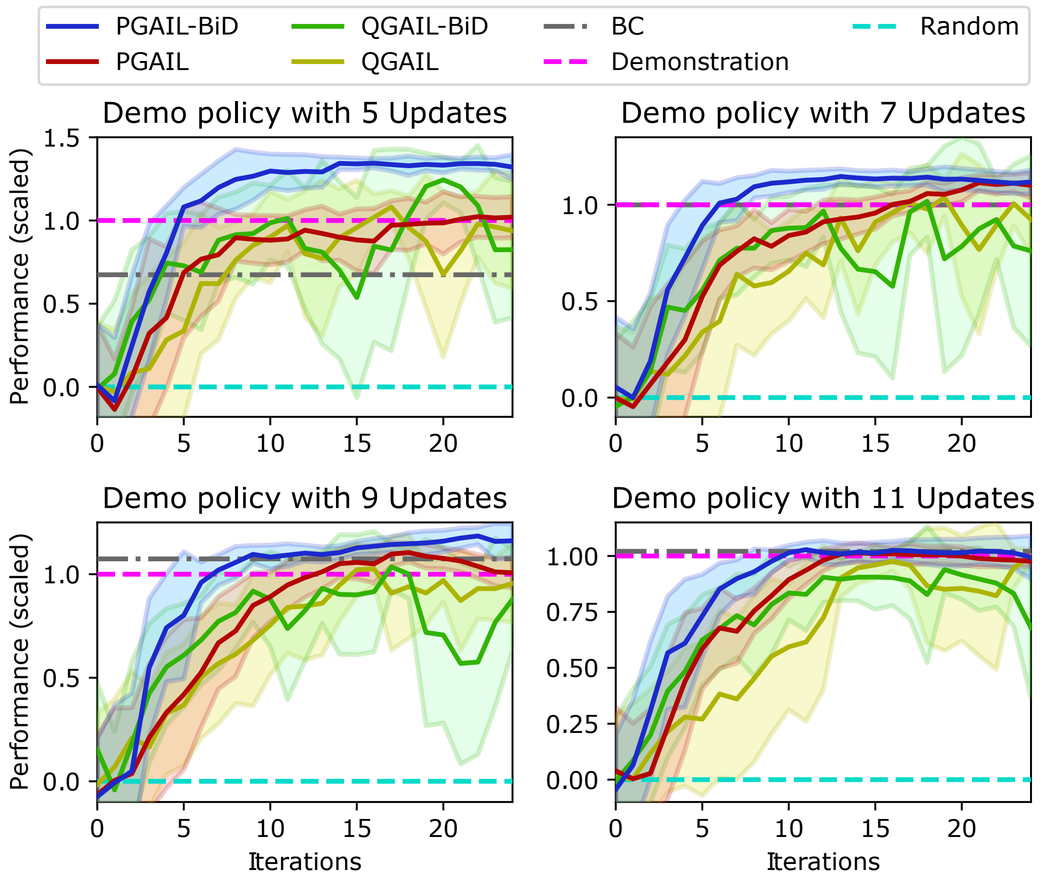


Figure 4.5. Learning curve of 5-DOF manipulator reaching task with imperfect demonstrations.

is scaled to 1 while one of the random policies is scaled to 0. The learning curves are the average of five trials. Without the Bi-Discriminator, PGAIL and QGAIL converged to the performance of demonstration. The learning curves of QGAIL-BiD are unstable, and performance is similar to demonstrations. The proposed method, PGAIL-BiD approaches the same or more than the performance of demonstration.

4.2.3. Comparison of PGAIL-BiD and PGAIL with Target Discriminator

To verify the effect of Bi-Discriminator, we compared PGAIL-BiD, PGAIL and PGAIL with Target Discriminator. The environment setting was the same as in the previous section, and demonstrations were collected from the updated 5th RL policy.

The learning results with different discriminator are shown in Fig. 4.6. PGAIL with $D_E(s)$ learned policies that outperformed experts at four iterations, but performance declined after six iterations. This result indicates that the policy generates many target states, and thus the reward of the target state is reduced. PGAIL with $D_T(s)$ outperforms the experts but converges to the local minimum. PGAIL-BiD outperforms the experts after four iterations and keeps high performance after that.

4.2.4. Visualization of Estimated Reward Function

In this subsection, to compare reward function represented by discriminator, we visualize reward function in 2-DOF manipulator reaching task. Since the state of the 2-DOF manipulator is in the two dimension of the joint angle, the reward function is drawn as shown in Fig. 4.7a. The learned reward functions from imperfect demonstrations are visualized in Fig. 4.7b. From the baseline reward function, it is clear that the optimal joint angles for reaching task are 50° and 90°. PGAIL-BiD learned an accurate reward function that offers value near $[50°, 90°]$ in all iterations. PGAIL learned a reward function with high near $[50°, 90°]$ after five iterations. QGAIL-BiD learned a reward function with high near $[50°, 90°]$ after three iterations but reward function after five iteration is highest near $[75°, 90°]$. These results indicate that EDPN learns policies stably from Bi-Discriminator.

All the simulation experiment results suggest that the proposed framework, PGAIL-BiD, can learn a better reward function from imperfect demonstration with stable learning thanks to the support of both the smooth policy update and causal entropy of policy in EDPN.

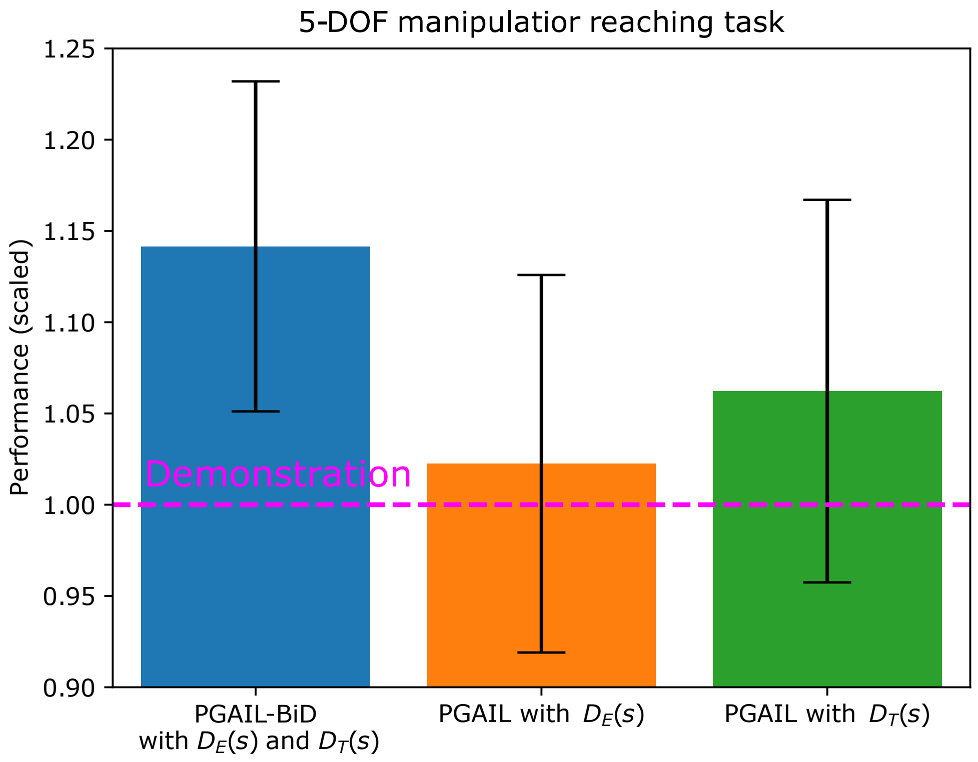
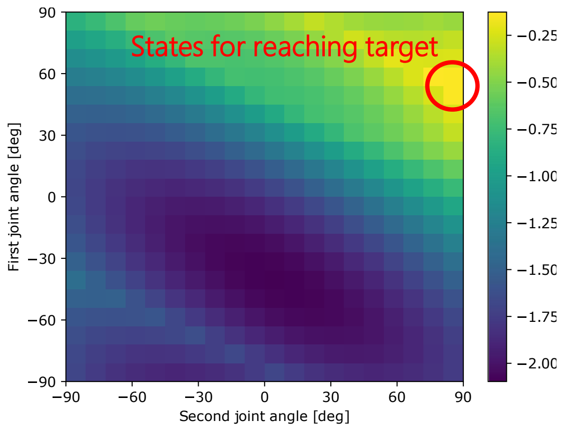


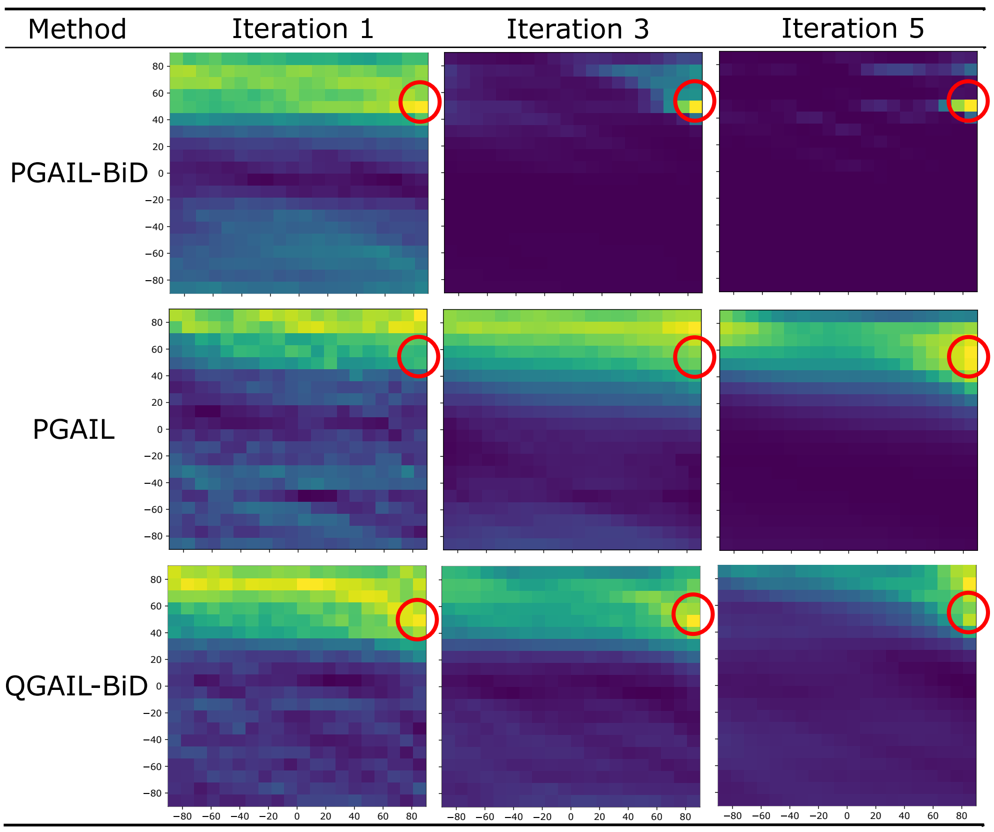
Figure 4.6. Evaluation of Bi-Discriminator Performance in the 5-DOF manipulator reaching task. Compare performance of policies learned from rewards containing different discriminators. The left bar is PGAIL with Bi-Discriminator including expert discriminator $D_E(s)$ and target discriminator $D_T(s)$ (PGAIL-BiD). The middle bar is PGAIL with only expert discriminator $D_E(s)$ as reward. The right bar is PGAIL with only target discriminator $D_T(s)$ as reward.

4.3. Real Robot Experiment

In this section, we applied PGAIL with EDPN to the NEXTAGE (`www.nextage.kawada.jp/en/`), a 15-DOF humanoid robot with sufficient precision for complex manufacturing tasks, to learn a policy that flips a handkerchief and folds clothes from demonstrations. We focus on learning a policy with high-level discrete actions, i.e., grasp-release points and folding lines on a cloth to solve two tasks: 1) flipping over a handkerchief and 2) folding clothes.



(a) The baseline reward function value of 2 DOF manipulator reaching.



(b) The reward function values of 2 DOF manipulator reaching learned by different methods.

Figure 4.7. Learned reward functions from demonstration.

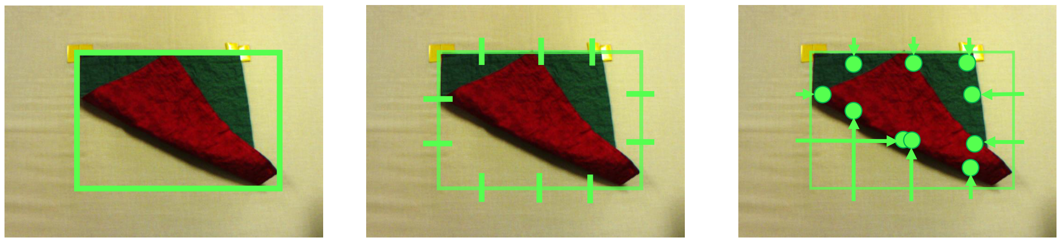

(a) Recognize cloth and enclose in a square. (b) Divide each side at equal intervals. (c) Scan to the edge of cloth.

Figure 4.8. Image processing of picking up points.

4.3.1. Flipping a Handkerchief

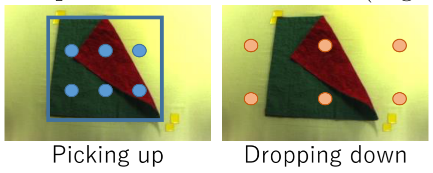
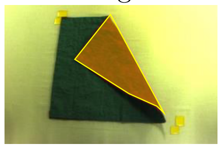
Setting

In this flipping a handkerchief task, we evaluate the learned policy's performance in a real-world environment with an easy design of the reward function. This handkerchief has different colors on the back and front, and the target state is the red color spreading on up. The environment and the PGAIL-BiD settings of this task are respectively shown in Tables 4.4a and 4.2b. $10 \times 6 = 60$ gripper actions are defined as picking up the handkerchief from 10 points over its current area and dropping it down to 2×3 points over the table (Fig. 4.8).

Results

The learning results of PGAIL-BiD and PGAIL over three experiments are shown in Fig. 4.9 where the Y axis is the success rate. During the test, the handkerchief is initialized to a green spreading on up and we test three trials per one policy (3 experiments $\times$ 3 trials $=$ 9 evaluations). The successful manipulation is defined as the red area finally becomes more than 80%. Each experiment took about eight hours, including 47 minutes for automatically initializing the handkerchief ($\approx$ 20 seconds per episode) to generate 4200 samples ($=$ 14 iterations). After exploring with 3900 samples, PGAIL-BiD learned to reach around 80%. On the other hand, PGAIL learned to reach around 30% with the same number of samples.

As compared to the control policies learned by PGAIL-BiD and PGAIL, and random policies shown in Fig. 4.10, the proposed method generated the trajectory

MDP setting	Description
State	The input state is a 84×84 px RGB image from NEXTAGE's integrated camera.
Action	$10 \times 6 = 60$ gripper actions are defined as picking up the handkerchief from 10 points over its current area and dropping it down to 2×3 points over the table (Fig. 4.8). 
Reward	The reward is defined as the ratio of the red area over the whole image in the current state. 
Initial state	The robot executes three random actions and initializes the handkerchief.
Demonstrations	The sample is collected from humans for trajectory of 5 episodes (100 samples). The target state is set as the final state of the episode.

(a) Parameter setting of flipping handkerchief task

Parameter	Meaning	Value
η	Parameters controlling the effect of smooth policy update	1
σ	Parameters controlling the effect of causal entropy	0.05
M	Number of episodes for one iteration	10
T	Number of steps for one episode	30
N	Size of memory $\mathcal{D}_G$ to store sample	3000
J	Iteration number of discriminator updates	5
K	Iteration number of value network updates	10

(b) Parameter setting of PGAIL-BiD algorithm.

Table 4.2. Settings and learning parameters of flipping handkerchief task

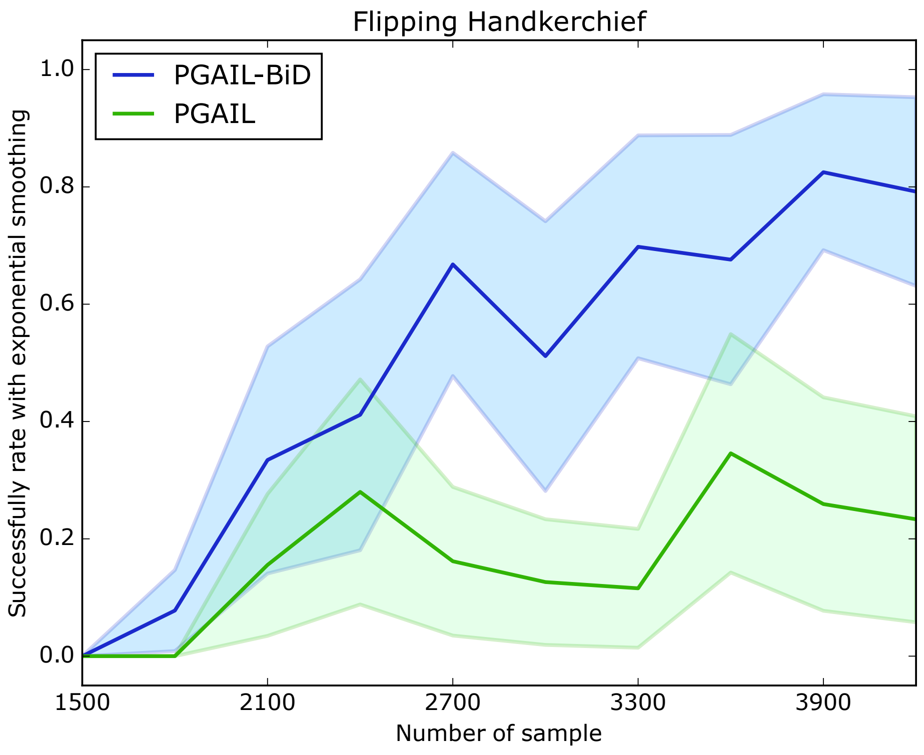


Figure 4.9. Learning results of the NEXTAGE humanoid robot on flipping a handkerchief task. PGAIL-BiD and PGAIL are compared on the learning curve evaluated by task success rate. Each method learn the policy 3 times, and each policy is tried for 3 episodes. Thus, each method is evaluated by task success rate from 9(3) episodes.

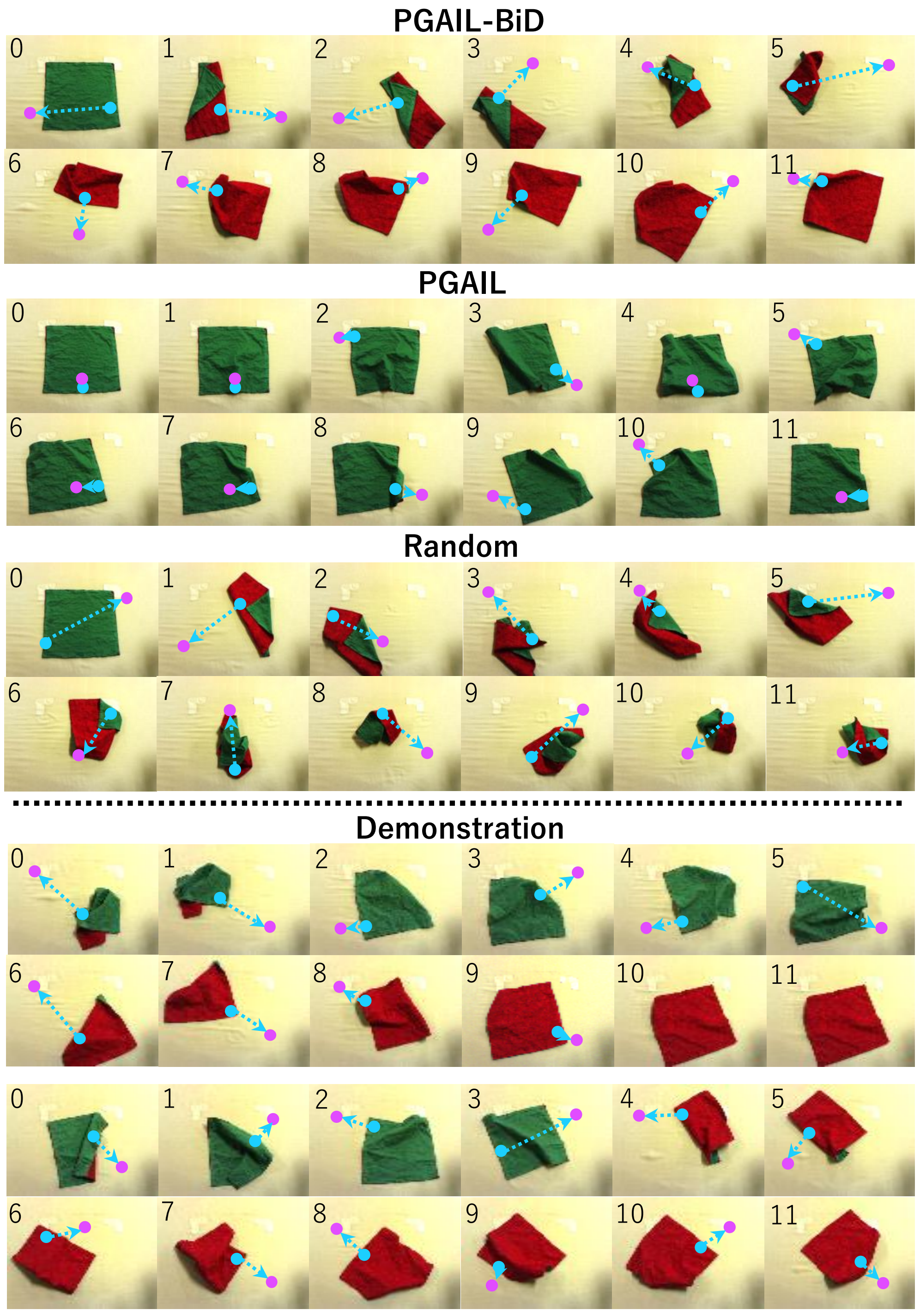


Figure 4.10. The snapshot of the learned policy per step. The blue and purple dots are picking up and dropping positions respectively for actions at each step.

of successfully flipping the handkerchief. PGAIL keeps the green state and could not flip the handkerchief. This reason is that demonstrations contain a state similar to the initial state.

Evaluation of Learned Policy's Robustness

The robustness of the policies learned by PGAIL-BiD is evaluated with different initial states shown in Table 4.3. For each initial state, we test the policy with five trials. The successful manipulation is defined as the red area finally becomes more than 80%. According to the result in Table 4.3, behavior cloning could not select the appropriate action when the observed initial states do not exist in demonstrations. PGAIL-BiD shows a better generalization ability to maximize the red area in various initial states. This result indicates that PGAIL-BiD can smoothly explore the flipping task following demonstrations. Rather than copying demonstrations, it is able to learn a robust policy for states out of demonstrations.

4.3.2. Folding Clothes

Setting

To verify that the proposed method could learn different clothes manipulation in the same environment setting, we applied the proposed method to folding clothes tasks for shirts and trousers. The environment and the PGAIL-BiD settings of this task are respectively shown in Tables 4.4b and 4.4. The folding action is executed as shown in Fig. 4.11. The folding direction, grasping points, and the folding path are automatically calculated according to the current clothes shape and the selected fold line.

Results

Since this folding task is challenging to design a reward function, we evaluate the learned policies through learned trajectories, task success rates, and CNN trained feature extraction of the policies. PGAIL-BiD learned folding policies from a 390 sample of shirts collected in about 5 hours and 220 samples of trousers collected in about 3 hours.

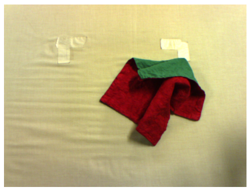
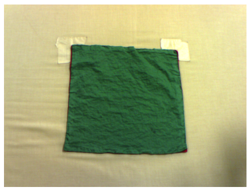
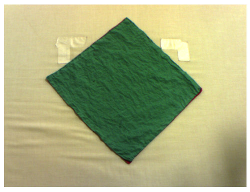
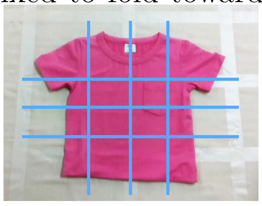
Initial state	PGAIL-BiD	PGAIL	Behavior Cloning
 Random	4/5	3/5	2/5
	4/5	1/5	4/5
	3/5	2/5	2/5

Table 4.3. Evaluation of success rate in flipping handkerchief task. Each method is evaluated with 5 trials in each initial state.

MDP setting	Description
State	The input state is a 84×84 px RGB image from realsense camera.
Action	All fold line are defined according to the current clothes shape ($3 + 3 = 6$ line). The fold direction is fixed to fold towards the center. 
Initial state	Shirt or trousers is initialized to a state in which it is spread by a human.
Demonstrations	The sample is collected from humans for trajectory of 5 episodes (shirt: 5 episodes $\times 3$ steps $= 15$ samples, trousers: 5 episodes $\times 2$ steps $= 10$). The target state is set as the final state of the episode.

(a) Parameter setting of flipping handkerchief task

Parameter	Meaning	Value
η	Parameters controlling the effect of smooth policy update	0.7
σ	Parameters controlling the effect of causal entropy	0.03
M	Number of episodes for one iteration	10
T	Number of steps for one episode	3 (shirt) or 2 (trousers)
N	Size of memory $\mathcal{D}_G$ to store sample	800
J	Iteration number of discriminator updates	10
K	Iteration number of value network updates	20

(b) Parameter setting of PGAIL-BiD algorithm.

Table 4.4. Settings and learning parameters of folding Clothes task

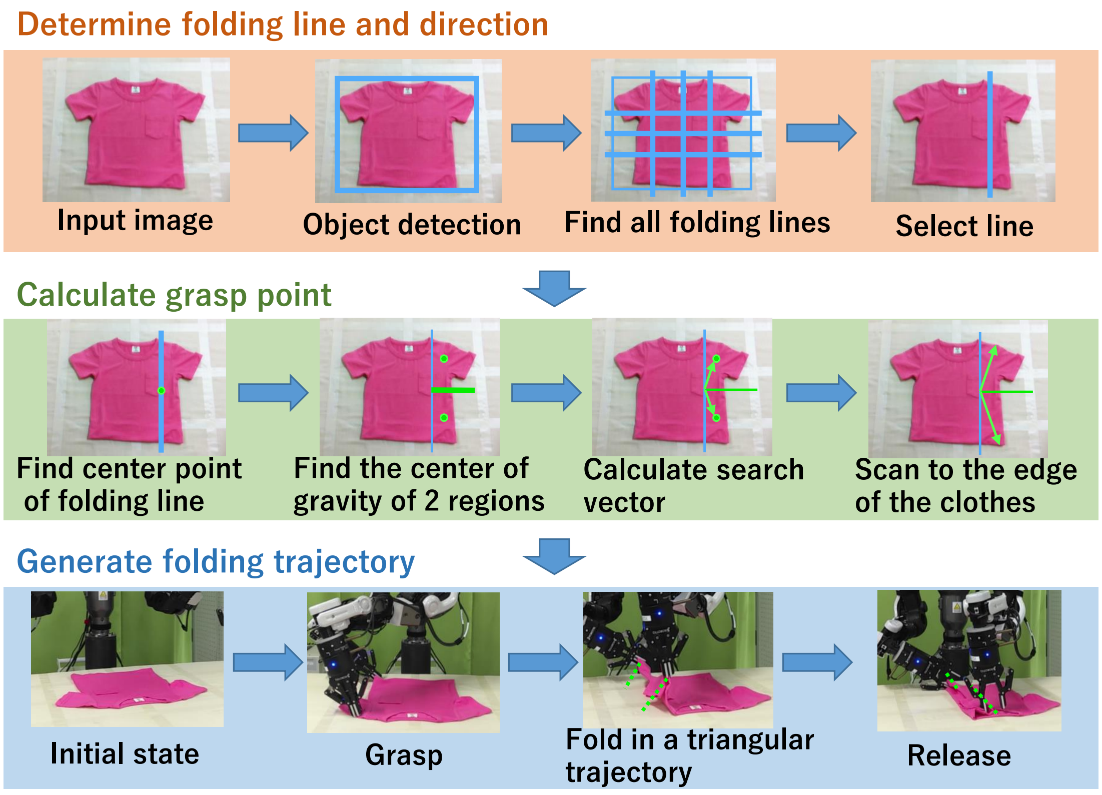

Determine folding line and direction
Input image
Object detection
Find all folding lines
Select line
Calculate grasp point
Find center point of folding line
Find the center of gravity of 2 regions
Calculate search vector
Scan to the edge of the clothes
Generate folding trajectory
Initial state
Grasp
Fold in a triangular trajectory
Release

Figure 4.11. Overview of folding action.

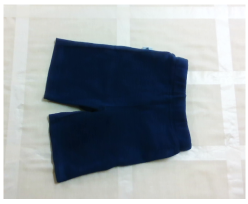
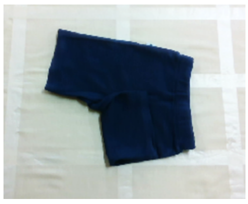
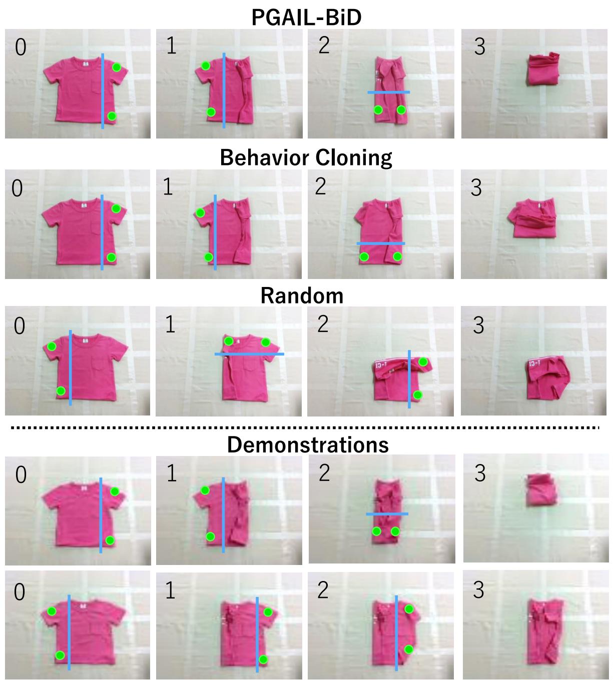
Initial state	PGAIL-BiD	Behavior Cloning
	4/5	1/5
	4/5	0/5
	5/5	3/5
	4/5	2/5

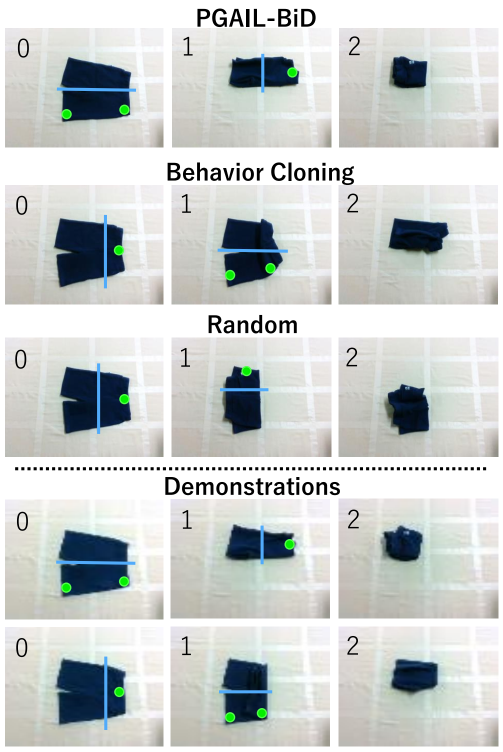
Table 4.5. Evaluation of success rate in folding clothes task. Each method is evaluated with 5 trials in each initial state. Success is defined as achieving the target state.

Fig. 4.12 shows the trajectories generated from the policies learned for two clothes folding tasks. PGAIL-BiD learning policies have a high success rate for both shirts and trousers, while the BC learning policies have a low success rate due to imperfect demonstrations. Table 4.5 shows the results of evaluating the success rate for each initial state. PGAIL-BiD policies have a higher success rate at different initial states, indicating that these policies are more robust than the BC policies.

Grad-CAM [45] visualizes several examples of high-level features learned by PGAIL-BiD in Fig. 4.13. These heat maps where the red/blue colors indicate



(a) Shirt folding trajectory.



(b) Trousers folding trajectory.

Figure 4.12. The snapshot of the learned policy per step. Blue lines are the folding the picking up, and green points are grasping points.

high/low attention of the agent indicate that our proposed method successfully learned useful and meaningful features. PGAIL-BiD and BC policies are compared with each other. PGAIL-BiD policies extract the details such as spreading areas and edges of clothes, while BC policies extract the areas other than clothes. This result indicates that PGAIL-BiD policies are robust to position, wrinkles, and some creases.

4.4. Summary of Chapter 4

There are two contributions in this chapter. The algorithmic aspect is PGAIL-BiD's proposal, a new adversarial imitation learning framework with two discriminators; PGAIL-BiD learns imitation policies from rewards with a high estimate of the target state. Thus, the influence of imperfect sample in demonstrations is reduced, and PGAIL-BiD policies achieve better than demonstration performance. EDPN contains a constraint that maximizes the policy's entropy with smooth policy updates, thus encouraging search while reducing policy over-learning. This approach's application contribution is that it learns the clothing manipulation policy without any specific reward function design. The proposed method was applied to a real environment consisting of a dual-armed robot NEXTAGE and learned better performance cloth manipulation policies.

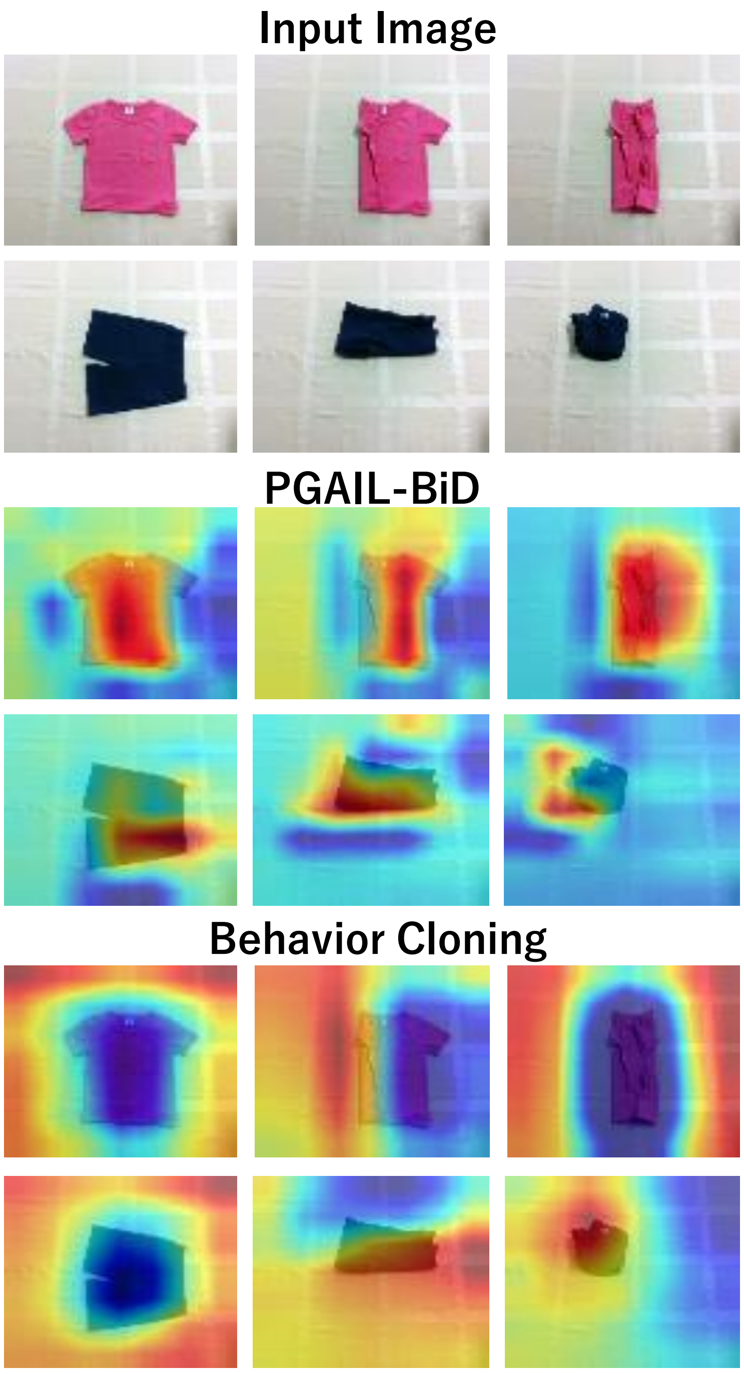


Figure 4.13. Visualization of extracted parts in images for action selection using Grad-CAM. Heat map shows parts extracted when actions are selected.

5. Discussions

5.1. Related Works

5.1.1. RL with Smooth Policy Update

To improve the sample efficiency and learning stability with fewer samples in RL, smooth policy update is exploited to limit the information that is lost during learning [47]. The main idea is to introduce the Kullback-Leibler divergence to limit the differences between the current and new policies into the reward function. The related approaches include both value function-based, e.g., [23], and policy search, e.g., relative entropy policy search [37] and guided policy search [48]. In the robot control domain, the smooth policy update was applied to learn hierarchical policies [49] and achieve sample efficiency and learning stability with kernel trick in robot hand control with a 32-dimensional state space [32]. The current combination of smooth policy update and DRL [50] focuses on learning end-to-end motor policies represented by linear Gaussian controllers in continuous action space. On the other hand, combining the value function based DRL with smooth policy update has not been intensively studied.

5.1.2. GAIL with Improved Discriminator

In order to focus on a specific problem, many previous studies improve the discriminator of GAIL. Methods focusing on reaching tasks adjust a reward for target states. For example, [51] first learned a reward function from the target state collected from a human and then learned a policy using the learned reward. In the end, the policy is improved by an evaluated performance from a human. [52] represented reinforcement learning and imitation learning integrally with probabilistic graphical models and learns a policy from imitation rewards and task-dependent achievement rewards. [53] accelerated the convergence of policies by updating the target state in the expert sample according to the learning progress.

5.1.3. Imitation Learning

Imitation learning is divided into two categories, Behavior Cloning (BC) and Inverse Reinforcement Learning (IRL) [54]. BC learns policies by directly regressing expert data [55]. Other prior BC studies learn robust policies by collecting data that add noise to expert policies [56, 57]. IRL has been challenging to apply to large scale tasks [58, 59], but GAIL based on IRL has been successfully applied to large scale tasks by updating policy and reward function like a GAN framework [60,61]. BC and GAIL differ in robustness and learning costs. BC has fewer interactions with the environment and lower learning costs. GAIL has a high learning cost because it explores a large area from a massive number of interactions. When compared in the aspect of robustness, GAIL is more robust than BC.

The proposed method of the GAIL framework with two discriminators estimates higher target state rewards than expert rewards. It is possible to learn more policies than expert policies. In the handkerchief flipping task, PGAIL-BiD policy can generate a different trajectory than the experts. The proposed method learns other strategy policies than the experts, and the BC learns policies to keep the expert trajectory.

5.2. Open Issues

5.2.1. Design of Action Space

In this research, the cloth manipulation environment employs high-level discrete action space as a folding line. This action space facilitates learning complex tasks from a small number of samples, but discrete action spaces need to be designed for each task. On the other hand, low-level continuous action space can be applied to various environments without action space design. In continuous policy with value-based RL, previous studies have approximated continuous policies from the value function. [62] updated a value function that maximizes the causal entropy of a policy and learned continuous policy by implementing with Actor-Critic. [63] extends softmax policy to continuous policy with Deep RBF networks.

One future work is to design compact and efficient action space for more complicated deformable object manipulation tasks. In our current work, the action space was set by all the combinations of picking and dropping points allocated on the plane in a grid manner which is not compact or sufficient for more complex tasks. Several previous works could be applied to improve the actions' dexterity like defining meaningful predefined patterns [64], and exploiting synergies [65]. Furthermore, cooperating motion trajectories between two or more robot arms should be considered as actions for challenging tasks like wrapping clothes in [66]. It is also interesting to design an action space to efficiently manipulate the clothes like human beings based on several related works: [67] detected the wrinkle condition and operate a robot to extend necessary wrinkle for better folding performance; [68] gripped the cloth edge to reduce the wrinkles caused during operation; [69] directly folded the hem and sleeve of clothes based on a clothing model.

5.2.2. Reduction of Learning Costs in Real Environments

In our robot experiments, initialization required the most time and labor: returning a cloth to its initial state in every iteration. The automation of the initialization enable to increase the number of sample, DRL learns complex policy from a large number of samples. Recent work suggests a potentially helpful approach to alleviate this issue by simultaneously learning a rest policy as a usual policy [70, 71]. Extending our methods by combining them with previous work [70, 71] is also interesting future work.

Previous studies with simulation have learned clothing manipulation policies with DRL [20, 72] and reduce learning costs in real environments. However, DRL with cloth simulation has only been successful with simple shape clothes manipulation. It is challenging to simulate complex shapes at high speed due to wrinkles, friction, and the like. Recent clothing simulation research [73, 74] employs deep learning to simulate clothing to accelerate the computation of complex shapes. There is a possibility of learning real clothes manipulation policy using sim-to-real technology [19–21] to close the gap between the simulation and the real world.

5.2.3. Conservative Reinforcement Learning

If DRL of real robot policies is unstable, sample efficiency is reduced due to unnecessary exploration. Thus DRL is necessary to stabilize the learning of policies. Previous research [75, 76] uses an analytical approach to prevent unstable learning. Although Conservative Policy Iteration [75] is challenging to apply to complex simulation tasks, [77] applied it to DRL and learned policies that were more stable and performed better than conventional methods in some atari game tasks.

5.2.4. Application to Various Tasks

This research applied DRL to the clothing manipulation task folding and handkerchief flipping. Although the tasks in this experiment could be performed by roughly moving the arms without any equipment, dexterity and equipment are essential for everyday tasks such as washing, cleaning, and cooking. In order to achieve hand dexterity, a robotic hand with a high degree of finger freedom and tactile sensors and control of the hand is needed. For example, when cutting food in cooking, it is necessary to hold the food in different shapes with one hand while grasping the knife and adjusting the force applied to the food. In the tidying up task, the robot needs to grasp objects of various shapes and sizes. Previous studies exist that achieve dexterous hand finger movements [32, 78] and dexterous arm movements [79, 80], but few studies have implemented complex tasks with dexterous finger and arm movements. The author believes that human-level robots will be realized with the proper utilization of these robotics and machine learning technologies.

6. Conclusion

First, we presented two sample efficient DRL algorithms: DPN and Dueling DPN. The contribution of DPN/Dueling DPN chapter is twofold. We proposed two new deep reinforcement learning algorithms, Deep P-Network (DPN) and Dueling Deep P-Network (DDPN). The core idea shared by them is to combine the nature of smooth policy update in value function-based reinforcement learning (Dynamic Policy Programming) with the capability of automatic feature extraction in deep neural networks to enhance the sample efficiency and the learning stability process with fewer samples. To investigate our proposed methods' performance, we compared them with previous DRL methods in a simulated n DOF manipulator reaching task. Furthermore, we applied them to two robotic cloth manipulation tasks with a dual-arm robot, NEXTAGE: 1) flipping of a handkerchief and 2) folding a t-shirt with a limited number of samples. We confirmed in all the experiments that our method achieved more sample efficiency and stabilized learning than the previous methods.

Second, we proposed PGAIL-BiD that is a GAIL framework with two discriminators. When applying GAIL to real robot tasks, collecting complete expert samples from a human is difficult. PGAIL-BiD estimates high rewards for target states with the target discriminator and improves learning policies' reaching performance when learning from incomplete expert samples. To stabilize adversarial learning with two discriminators, PGAIL-BiD employs EDPN to consider both smooth and causal entropy in policy updates. PGAIL-BiD was applied to n DOF simulated manipulator reaching task and learned policies better than expert samples. In real robot experiments, PGAIL-BiD learned two cloth manipulation tasks: 1) flipping of a handkerchief and 2) folding two types of clothes from incomplete expert samples.

In summary, this thesis proposes two methods that reduce the cost of applying

DRL to a real robot task. By proposing sample-efficient DRL with smooth policy update, we relaxed the learning cost on real robots. We applied it to cloth manipulation, which is difficult to design in state space. We present PGAIL-BiD, GAIL framework with two discriminators. PGAIL-BiD learn cloth manipulation policy without task-specific reward function design from incomplete expert samples collected by a human.

A. Appendix: Model-Beased Deep Reinforcement Learning with Latent Discrete State Action Space

In this chapter, we present Variationally Autoencoded Dynamic Policy Programming (VAE-DPP), a framework for the action plan of cloth folding, as shown in Fig. A.1. VAE-DPP employs Variationally Autoencoded Hidden Markov Decision Model (VAE-HMDM) to train the cloth folding model. VAE-HMDM learns the latent space of the image with Variational Autoencoder (VAE) and learns the latent space of dynamics with hidden Markov model (HMM) with action space. Previous research indicates that a discrete set can represent the state-action space at the cloth folding task [81], and we model the state-action space of latent dynamics with a discrete set. The proposed method employed Dynamic Policy Programming (DPP) [23] to action planning and generated an action sequence.

In a real-world experiment, we applied VAE-DPP to a cloth folding task by a dual-arm robot. We demonstrated that the proposed method train a cloth folding model and plan trajectories to different target states.

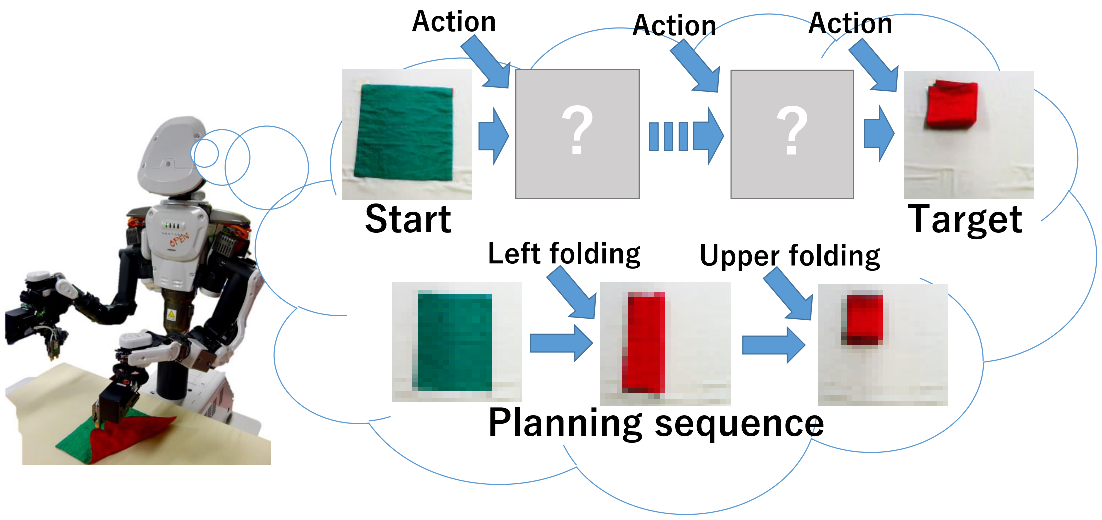


Figure A.1. Action planning of cloth folding task. The proposed method generates a state action sequence from initial state and target state.

A.1. Proposed Method

A.1.1. Variationally Autoencoded Hidden Markov Decision Model

Fig. A.2 shows a graphical model of VAE-HMDM, which learns the latent variable $\mathbf{a}$ of the observation image $\mathbf{x}$ and the latent variable $\mathbf{z}$ of the dynamics. By optimizing $p_\psi(\mathbf{a}|\mathbf{x})$ and $q_\phi(\mathbf{x}|\mathbf{a})$ simultaneously, latent variable $\mathbf{a}$ is learned to decode the observation image $\mathbf{x}$. In this model, $\mathbf{a}$ is assumed to be generated from the dynamics latent variable $\mathbf{z}$, and the latent variable $\mathbf{z}'$ at the next time transitions with probability distribution $p_\theta(\mathbf{z}'|\mathbf{z}, \mathbf{u})$ that depends on the previous latent variable $\mathbf{z}$ and action $\mathbf{u}$. Even though image transitions are partially observable Markov decision process, the transitions of the dynamics latent variable $\mathbf{z}$ are Markov decision processes.

The model learns the parameter ψ of encoder $p_\psi(\mathbf{a}|\mathbf{x})$, the parameter ϕ of decoder $q_\phi(\mathbf{x}|\mathbf{a})$ and the parameter of latent dynamics θ. The parameters ψ, ϕ, θ are optimized to maximize the following lower bound.

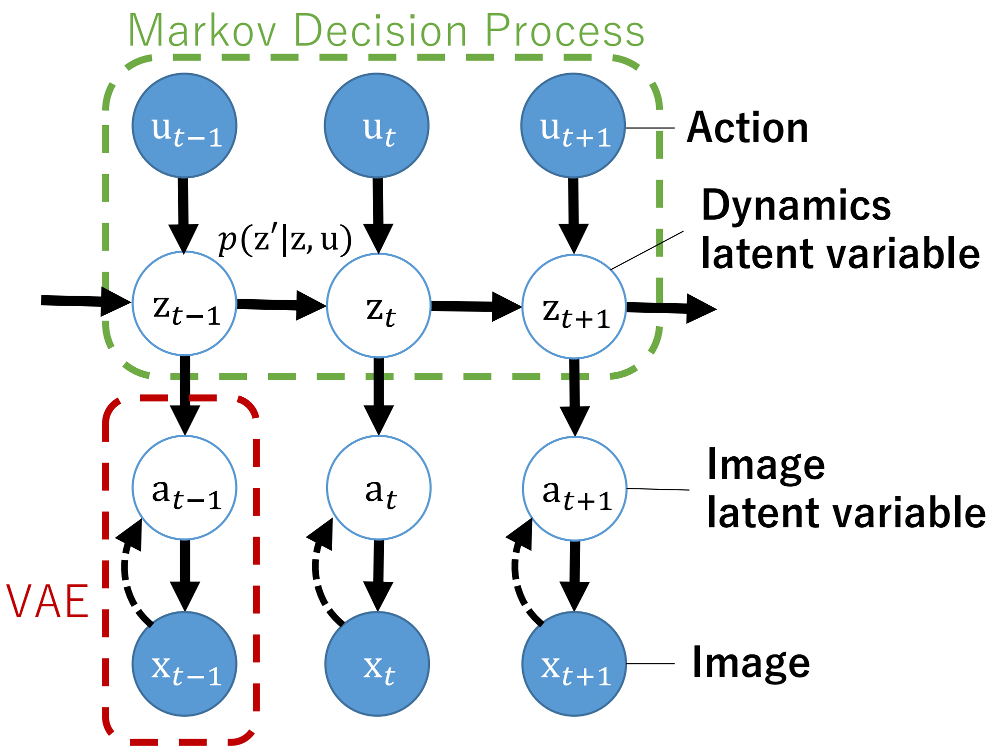


Figure A.2. The graphical model of VAE-HMDM

$$\begin{aligned}\mathcal{F}(\psi,\phi,\theta) &= \mathbb{E}_{q_\phi(\mathbf{a}|\mathbf{x})}\Bigg[\log\frac{p_\psi(\mathbf{x}|\mathbf{a})}{q_\phi(\mathbf{a}|\mathbf{x})} \\ &\quad +\mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{a},\mathbf{u})}\left[\log\frac{p_\theta(\mathbf{a}|\mathbf{z})p_\theta(\mathbf{z}|\mathbf{u})}{p_\theta(\mathbf{z}|\mathbf{a},\mathbf{u})}\right]\Bigg] \qquad \text{(A.1)}\end{aligned}$$

$\{\tilde{\mathbf{a}}^{(i)},\tilde{\mathbf{z}}^{(i)}\}_{i=1}^{I}$ is sampled from the VAE decoder $q_\phi(\mathbf{a}|\mathbf{x})$ and the latent dynamics $p_\theta(\mathbf{z}|\mathbf{a},\mathbf{u})$. Therefore, lower bound can be estimated by the Monte Carlo integral as follows:

$$\begin{aligned}\hat{\mathcal{F}}(\psi,\phi,\theta) &= \frac{1}{I}\sum_i\Big\{\log p_\psi(\mathbf{x}|\tilde{\mathbf{a}}^{(i)}) + \log p_\theta(\tilde{\mathbf{a}}^{(i)},\tilde{\mathbf{z}}^{(i)}|\mathbf{u}) \\ &\quad -\log q_\phi(\tilde{\mathbf{a}}^{(i)}|\mathbf{x}) - \log p_\theta(\tilde{\mathbf{z}}^{(i)}|\tilde{\mathbf{a}}^{(i)},\mathbf{u})\Big\} \qquad \text{(A.2)}\end{aligned}$$

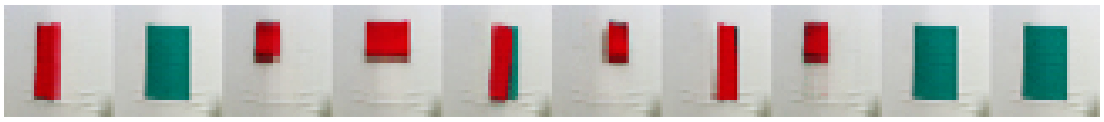


Figure A.3. Decoding image from latent variable $\mathbf{z}$ of VAE-HMDM

The parameter ψ,ϕ,θ maximizing the lower bounds are optimized with gradient-based numerical optimization.

A.1.2. Action Planing with Dynamic Policy Programming

The proposed method plan action on latent space $\mathbf{z}$ of VAE-HMDM. A policy $\pi(\mathbf{u}|\mathbf{z})$ generates sequence of $[\mathbf{z}, \mathbf{u}]$ reaching target latent variable $\mathbf{z}$. The policy is learned with Dynamic Policy Programming (DPP) [23]. By decoding the sequence of generated latent variable $\mathbf{z}$ to images, it is able to confirm the planned transition image sequence. DPP learns a policy $\pi(\mathbf{u}|\mathbf{z})$ to maximize total reward through interactions with dynamics $p(\mathbf{z}'|\mathbf{z}, \mathbf{u})$.

A.2. Real Robot Experiment

We applied VAE-DPP to the NEXTAGE robot, a 15-DOF humanoid robot with sufficient manufacturing precision, to learn folding cloth tasks. In the experiment, square cloths with different colors on the reverse side were employed, and the initial state was set with green on the upper side. The state is defined as $32\times32\times3$ px RGB images, folding action is defined as up-folding, right-folding and left-folding. One sequence was two folded actions, and the model was trained from 200 sequences sampled from the random policy. The dimension of the latent variable $\mathbf{z}$ is set to 10. For the reward function, the following equation was employed to learn the policy and plan the action.

$$reward(\mathbf{z}) = \begin{cases} 1 & (\mathbf{z} = \mathbf{z}_{target}) \\ 0 & (otherwise) \end{cases} \tag{A.3}$$

Fig. A.3 show decoding images from latent variable $\mathbf{z}$ of VAE-HMDM. Decoding images indicate that latent variable $\mathbf{z}$ has acquired a variety of folding states.

Fig. A.4 show sequences planed with VAE-DPP. The upper left corner numbers represent the steps, and the image in the upper right corner is a decoding image of the latent variable $\mathbf{z}$ generated during the action plan. The action plan in Fig. A.4 generates a sequence aimed at two different target latent variables $\mathbf{z}_{target}$, and the results indicate that a suitable action sequence for the target state has been generated.

A.3. Summary of VAE-DPP

In this chapter, the VAE-DPP is proposed as a framework for a cloth folding action plan. We confirmed that VAE-DPP could plan the operation of cloth folding through real experiments.

The future main task is to investigate whether it is possible to plan the folding action for shirts, trousers, and other clothes.

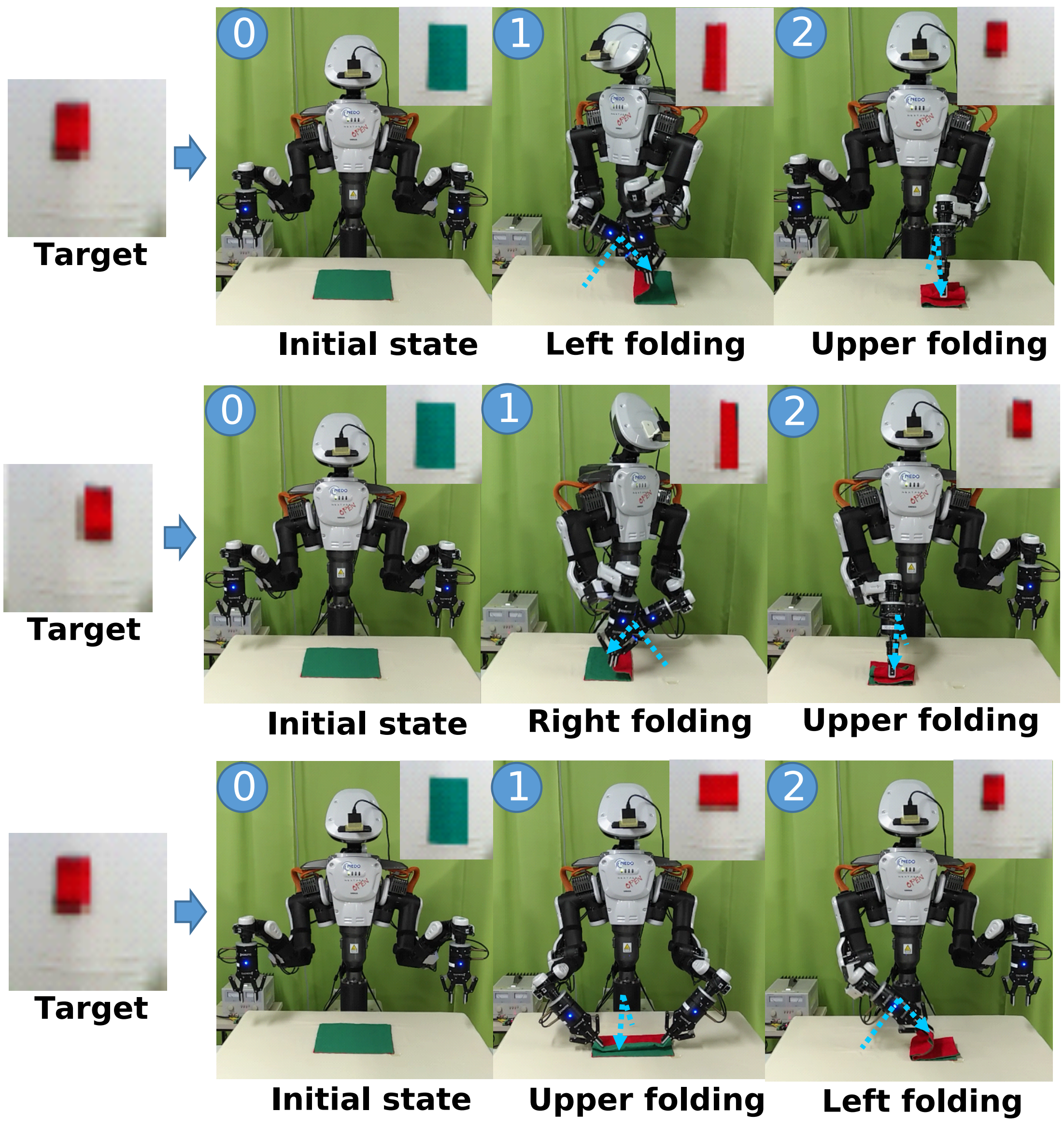


Figure A.4. Planning and execution of folding cloth manipulation with VAE-DPP

Acknowledgements

In my research days at Nara Institute of Science and Technology (NAIST), I've been blessed with many people, environments, and sometimes good fortune. I was able to experience a lot of precious experiences, thanks to the support of many people.

First, I would like to express my appreciation to Associate Professor Takamitsu Matsubara. He taught me various things such as the knowledge, techniques, and approaches required for research and gave me advice on my research activities and career path. I am also profoundly grateful for the opportunity to research a theme that I hope to explore. I really appreciate Professor Kenji Sugimoto. He has supported me as my supervisor since I was a master student. I also want to thank you for your good comments and suggestions. I also thank Professor Tsukasa Ogasawara to review this thesis. His comments on the practical and robotic aspects gave me an excellent opportunity to reconfirm my research.

I would like to thank all members of Robot Learning Laboratory and Intelligent System Control Laboratory. I luckily got a lot of opportunities to discuss and talk with them. Particular thank you to Associate Professor Yunduan Cui. He was a good example and a good co-author for me. Finally, I want to thank you for acknowledging my family and friends for their supports.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press Cambridge, 1998.

[2] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *International Symposium on Experimental Robotics (ISER)*, pp. 363–372, 2006.

[3] T. Hester, M. Quinlan, and P. Stone, "RTMBA: A real-time model-based reinforcement learning architecture for robot control," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 85–90, 2012.

[4] E. Theodorou, J. Buchli, and S. Schaal, "Reinforcement learning of motor skills in high dimensions: A path integral approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2397–2403, 2010.

[5] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, "Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 213–228, 2008.

[6] S. Bitzer, M. Howard, and S. Vijayakumar, "Using dimensionality reduction to exploit constraints in reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3219–3225, 2010.

[7] H. Durrant-Whyte, N. Roy, and P. Abbeel, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," in *Robotics: Science and Systems (RSS)*, pp. 57–64, 2012.

[8] K. Yamazaki, R. Ueda, S. Nozawa, M. Kojima, K. Okada, K. Matsumoto, M. Ishikawa, I. Shimoyama, and M. Inaba, "Home-assistant robot for an aging society," *Proceedings of the IEEE*, vol. 100, no. 8, pp. 2429–2441, 2012.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT Press, 2016.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

[11] V. Mnih, *Machine learning for aerial image labeling.* PhD thesis, University of Toronto, 2013.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, June 2015.

[13] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio Speech and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[14] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6645–6649, 2013.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, vol. 37, pp. 1889–1897, 2015.

[17] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale

data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[18] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *The Conference on Robot Learning (CoRL)*, vol. 87, pp. 651–673, 2018.

[19] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conference on Robot Learning (CoRL)*, pp. 262–270, 2017.

[20] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning (CoRL)*, vol. 87, pp. 734–743, 2018.

[21] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," *arXiv preprint arXiv:1810.05687*, 2018.

[22] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 4565–4573, 2016.

[23] M. G. Azar, V. Gómez, and H. J. Kappen, "Dynamic policy programming," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 3207–3245, 2012.

[24] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, pp. 1995–2003, 2016.

[25] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[27] R. S. Sutton, “Generalization in reinforcement learning: Successful examples using sparse coarse coding,” *Advances in Neural Information Processing Systems (NIPS)*, pp. 1038–1044, 1996.

[28] M. G. Lagoudakis and R. Parr, “Least-squares policy iteration,” *Journal of Machine Learning Research*, vol. 4, no. 44, pp. 1107–1149, 2003.

[29] M. G. Azar, V. Gómez, and B. Kappen, “Dynamic policy programming with function approximation,” in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 119–127, 2011.

[30] E. Todorov, “Linearly-solvable markov decision problems,” in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1369–1376, 2006.

[31] Y. Cui, T. Matsubara, and K. Sugimoto, “Pneumatic artificial muscle-driven robot control using local update reinforcement learning,” *Advanced Robotics*, pp. 1–16, 2017.

[32] Y. Cui, T. Matsubara, and K. Sugimoto, “Kernel dynamic policy programming: Applicable reinforcement learning to robot systems with high dimensional states,” *Neural networks*, vol. 94, pp. 13–23, 2017.

[33] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, “Gotta Learn Fast: A New Benchmark for Generalization in RL,” *arXiv preprint arXiv:1804.03720*, 2018.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[35] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, “Reinforcement learning with deep energy-based policies,” in *International Conference on Machine Learning (ICML)*, pp. 1352–1361, 2017.

[36] Y. Tsurumine, Y. Cui, E. Uchibe, and T. Matsubara, “Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation,” *Robotics and Autonomous Systems*, vol. 112, pp. 72–83, 2019.

[37] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *Association of the Advancement of Artificial Intelligence (AAAI)*, pp. 1607–1612, 2010.

[38] S. Levine, N. Wagoner, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *IEEE Conference on Robotics and Automation (ICRA)*, 2015.

[39] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Learning from demonstrations for real world reinforcement learning," *Computing Research Repository (CoRR)*, vol. abs/1704.03732, 2017.

[40] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *Computing Research Repository (CoRR)*, abs/1707.08817, 2017.

[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[42] F. Chollet *et al.*, "Keras." `https://github.com/keras-team/keras`, 2017.

[43] D. Tanaka, S. Arnold, and K. Yamazaki, "EMD Net: An encode-manipulate-decode network for cloth manipulation," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1771–1778, 2018.

[44] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, p. 5, 2009.

[45] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626, 2018.

[46] T. Kozuno, E. Uchibe, and K. Doya, "Theoretical analysis of efficiency and robustness of softmax and gap-increasing operators in reinforcement learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 89, pp. 2995–3003, 2019.

[47] H. Van Hoof, G. Neumann, and J. Peters, "Non-parametric policy search with limited information loss," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2472–2517, 2017.

[48] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning (ICML)*, pp. 1–9, 2013.

[49] C. Daniel, G. Neumann, and J. Peters, "Hierarchical relative entropy policy search," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 273–281, 2012.

[50] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[51] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine, "End-to-end robotic reinforcement learning without reward engineering," in *Robotics: Science and Systems*, 2020.

[52] A. Kinose and T. Taniguchi, "Integration of imitation learning using gail and reinforcement learning using task-achievement rewards via probabilistic graphical model," *Advanced Robotics*, 2020.

[53] Y. Ding, C. Florensa, P. Abbeel, and M. Phielipp, "Goal-conditioned imitation learning," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 15324–15335, 2019.

[54] T. Osa, J. Pajarinen, G. Neumann, J. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends in Robotics*, vol. 7, pp. 1–179, Mar. 2018.

[55] D. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” in *Advances in Neural Information Processing Systems (NIPS)*, pp. 305 –313, Morgan Kaufmann, December 1989.

[56] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” vol. 15 of *Proceedings of Machine Learning Research*, pp. 627–635, 2011.

[57] M. Laskey, J. Lee, R. Fox, A. D. Dragan, and K. Goldberg, “DART: noise injection for robust imitation learning,” in *Conference on Robot Learning (CoRL)*, vol. 78, pp. 143–156, 2017.

[58] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning.,” in *Association of the Advancement of Artificial Intelligence (AAAI)*, pp. 1433–1438, 2008.

[59] A. Boularias, J. Kober, and J. Peters, “Relative entropy inverse reinforcement learning,” in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 15, pp. 182–189, 2011.

[60] C. Finn, S. Levine, and P. Abbeel, “Guided cost learning: Deep inverse optimal control via policy optimization,” in *International Conference on Machine Learning (ICML)*, vol. 48, pp. 49–58, 2016.

[61] E. Uchibe, “Deep inverse reinforcement learning by logistic regression,” in *International Conference on Neural Information Processing (ICONIP)*, vol. 9947, pp. 23–31, 2016.

[62] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *The International Conference on Machine Learning (ICML)*, vol. 80, pp. 1861–1870, 2018.

[63] K. Asadi, R. E. Parr, G. D. Konidaris, and M. L. Littman, “Deep rbf value functions for continuous control,” *arXiv preprint arXiv:2002.01883*, 2020.

[64] T. Jung and D. Polani, “Kernelizing lspe(λ),” in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 338–345, 2007.

[65] C. Alessandro, I. Delis, F. Nori, S. Panzeri, and B. Berret, “Muscle synergies in neuroscience and robotics: from input-space to task-space perspectives,” *Frontiers in Computational Neuroscience*, vol. 7, p. 43, 2013.

[66] N. Hayashi, T. Suehiro, and S. Kudoh, “Planning method for a wrapping-with-fabric task using regrasping,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1285–1290, 2017.

[67] L. Sun, G. Aragon-Camarasa, S. Rogers, and J. P. Siebert, “Accurate garment surface analysis using an active stereo robot head with application to dual-arm flattening,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 185–192, 2015.

[68] H. Yuba, S. Arnold, and K. Yamazaki, “Unfolding of a rectangular cloth from unarranged starting shapes by a dual-armed robot with a mechanism for managing recognition error and uncertainty,” *Advanced Robotics*, vol. 31, no. 10, pp. 544–556, 2017.

[69] S. Miller, J. Van-Den-Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, “A geometric approach to robotic laundry folding,” *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 249–267, 2012.

[70] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, “Leave no trace: Learning to reset for safe and autonomous reinforcement learning,” in *International Conference on Learning Representations (ICLR)*, 2018.

[71] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine, “The ingredients of real world robotic reinforcement learning,” in *International Conference on Learning Representations (ICLR)*, 2020.

[72] R. Jangir, G. Alenya, and C. Torras, “Dynamic cloth manipulation with deep reinforcement learning,” *arXiv preprint arXiv:1910.14475*, 2019.

[73] Z. Lähner, D. Cremers, and T. Tung, "Deepwrinkles: Accurate and realistic clothing modeling," in *European Conference on Computer Vision (ECCV)*, vol. 11208, pp. 698–715, 2018.

[74] D. Holden, B. C. Duong, S. Datta, and D. Nowrouzezahrai, "Subspace neural physics: fast data-driven interactive simulation," in *The Annual ACM Symposium on Theory of Computing (STOC)*, pp. 6:1–6:12, 2019.

[75] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *International Conference on Machine Learning (ICML)*, pp. 267–274, 2002.

[76] L. Zhu and T. Matsubara, "Ensuring monotonic policy improvement in entropy-regularized value-based reinforcement learning," *arXiv preprint arXiv:2008.10806*, 2020.

[77] N. Vieillard, O. Pietquin, and M. Geist, "Deep conservative policy iteration," in *Association of the Advancement of Artificial Intelligence (AAAI)*, pp. 6070–6077, AAAI Press, 2020.

[78] M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *International Journal of Robotics Research (IJRR)*, vol. 39, no. 1, 2020.

[79] T. Tamei, T. Matsubara, A. Rai, and T. Shibata, "Reinforcement learning of clothing assistance with a dual-arm robot," in *IEEE-RAS International Conference on Humanoid Robots*, pp. 733–738, 2011.

[80] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, 2018.

[81] A. Doumanoglou, J. Stria, G. Peleka, I. Mariolis, V. Petrik, A. Kargakos, L. Wagner, V. Hlaváč, T.-K. Kim, and S. Malassiotis, "Folding clothes au-

tonomously: A complete pipeline," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1461–1478, 2016.

Publication List

Journal

1. Yoshihisa Tsurumine, Yunduan Cui, Eiji Uchibe, and Takamitsu Matsubara, “Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation,” Robotics and Autonomous Systems, vol. 112, pp. 72-83, 2019.

2. Yoshihisa Tsurumine, and Takamitsu Matsubara, “Double Discriminator P-Generative Adversarial Imitation Learning with Human Demonstration for Robotic Cloth Manipulation,” Robotics and Autonomous Systems, In preparation.

International Conference

1. Junki Matsuoka, Yoshihisa Tsurumine, Yuhwan Kwon, Takamitsu Matsubara, Takeshi Shimmura, and Sadao Kawamura, "Learning Food-arrangement Policies from Raw Images with Generative Adversarial Imitation Learning," IEEE International Conference on Ubiquitous Robots (UR), pp. 93-98, 2020.

2. Yuhwan Kwon, Takumi Kaneko, Yoshihisa Tsurumine, Hikaru Sasaki, Kimiko Motonaka, Seiji Miyoshi, and Takamitsu Matsubara, "Combining Model Predictive Path Integral with Kalman Variational Auto-Encoder for Robot Control from Raw Images," IEEE/SICE International Symposium on System Integration (SII), pp. 271-276, 2020.

3. Yoshihisa Tsurumine, Yunduan Cui, Kimitoshi Yamazaki, and Takamitsu Matsubara, "Generative Adversarial Imitation Learning with Deep P-Network for Robotic Cloth Manipulation," IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp.290—296, 2019.

4. Takumi Kaneko, Yoshihisa Tsurumine, James Poon, Yukio Onuki, Yingda Dai, Kaoru Kawabata, and Takamitsu Matsubara, "Learning Deep Dynamical Models of a Waste Incineration Plant from In-furnace Images and Pro-

cess Data," IEEE International Conference on Automation Science and Engineering (CASE), pp. 873-878, 2019.

5. Yoshihisa Tsurumine, Yunduan Cui, Eiji Uchibe, and Takamitsu Matsubara, "Deep Dynamic Policy Programming for Robot Control with Raw Images," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1545-1550, 2017.

Domestic Conference

1. 鶴峯義久, 崔允端, 山崎公俊, 松原崇充, ”変分オートエンコーデッド動的方策計画による布折り畳み動作の学習”, 日本ロボット学会学術講演会 (RSJ), 1A2-07, 2019.

2. 鶴峯義久, 崔允端, 山崎公俊, 松原崇充, ”最大エントロピー動的方策計画による柔軟物操作の模倣学習”, 日本機械学会ロボティクス・メカトロニクス講演会 (ROBOMECH), 1P2-A11, 2019.

3. 鶴峯義久, 崔允端, 山崎公俊, 松原崇充, ”変分オートエンコーデッドカルバックライブラー制御による物理制約を考慮したタスク達成に導く画像系列の合成”, 日本ロボット学会学術講演会 (RSJ), 2E1-03, 2018.

4. 鶴峯義久, 崔允端, 内部英治, 松原崇充, "生画像を入力とするサンプル効率の良い深層強化学習と双腕ロボットによる布操作への応用", 日本ロボット学会学術講演会 (RSJ), 3I3-01, 2017.

5. 鶴峯義久, 崔允端, 松原崇充, 内部英治, 杉本謙二, "動的方策計画に基づく深層強化学習", 計測自動制御学会制御部門マルチシンポジウム (MSCS), PS-16, 2017.

Feature Column

1. 松原崇充, 鶴峯義久, "方策を滑らかに更新する深層強化学習と双腕ロボットによる布操作タスクへの適用", 人工知能学会誌「人工知能」, Vol. 35, No. 1, pp. 47–53, 2020.

2. 松原崇充, 鶴峯義久, "動的方策計画法を用いた敵対的模倣学習とその応用", 日本ロボット学会誌「ロボ學」, Vol. 36, No. 6, pp. 5–8, 2020.