NAIST-IS-DD1461206

Doctoral Dissertation

Practical Model-free Reinforcement Learning in Complex Robot Systems with High Dimensional States

Yunduan Cui

September 9, 2017

Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of Engineering

Yunduan Cui

Thesis Committee:

Professor	Kenji Sugimoto (Supervisor)
Professor	Tsukasa Ogasawara (Co-supervisor)
Associate Professor	Takamitsu Matsubara (Co-supervisor)
Assistant Professor	Masaki Ogura (Co-supervisor)
Assistant Professor	Taisuke Kobayashi (Co-supervisor)

Practical Model-free Reinforcement Learning in Complex Robot Systems with High Dimensional States*

Yunduan Cui

Abstract

As a promising learning paradigm in recent years, reinforcement learning learns good policies by interacting with an unknown environment and thus being suitable to the scenario of controlling robots to explore in challenging tasks. On the other hand, both the main two groups of reinforcement learning algorithms, the value function approach and the policy search, are still impractical in model-free learning of complex robot systems due to several limitations. The value function approach learns value function over all states and actions without any prior knowledge but suffers from both the unstable learning process with insufficient real world samples and the intractable computational complexity in high dimensional systems. The policy search efficiently finds an optimal solution in a local area while being sensitive to the initialization of a well parameterized policy based on some knowledge of the task and model.

The motivation of this thesis is to explore practical model-free reinforcement learning algorithm to control complex robot systems. Our main idea is to take advantages of both the value function approach and the policy search. We propose a new approach that focuses on learning the global value function from the local sample space defined by the current policy. The Kullback-Leibler divergence is employed to limit the over large policy update in order to generate samples in continuous and local areas. Other machine learning methods are then applied

*Doctoral Dissertation, Graduate School of Information Science,
Nara Institute of Science and Technology, NAIST-IS-DD1461206, September 9, 2017.

on the local samples to locally approximate the value function. This framework solves the high sampling cost and intractable computational complexity without requiring any prior knowledge of the model or task.

Two algorithms are proposed based on this framework as examples: Local Update Dynamic Policy Programming (LUDPP) and Kernel Dynamic Policy Programming (KDPP). We first investigate the learning performance of the proposed methods in a range of simulation tasks including pendulum swing up and multiply DOF manipulator reaching, the proposed algorithms significantly outperform the conventional algorithms in high dimensional cases. Both LUDPP and KDPP are then successfully applied to control a Pneumatic Artificial Muscle (PAM) driven robotic hand, a high-dimensional system in finger position control and unscrew bottle cap task respectively while given limited samples and with ordinary computing resources. All results indicate the practicability of the proposed framework in controlling complex robot systems.

Keywords:

Reinforcement Learning, Robotic Learning, Pneumatic Artificial Muscles

Contents

List of Figures

1 Introduction

1.1 Background

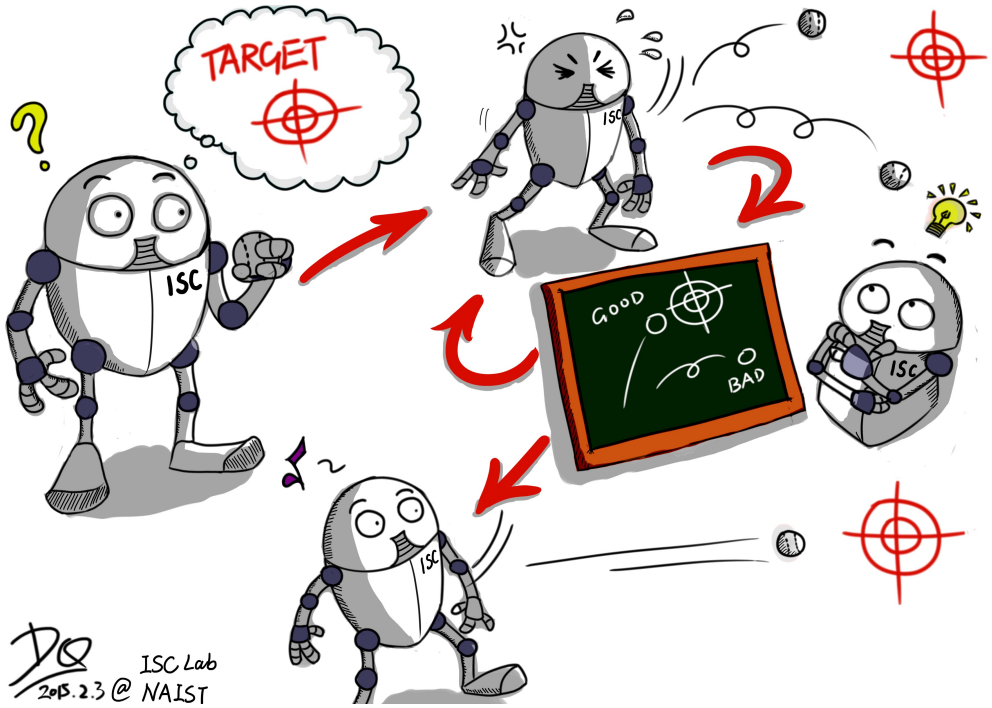


Figure 1.1: An example of Robot throwing ball using reinforcement learning.

Image that we hope one robot to automatically learn how to throw a ball to the target. As shown in Fig. 1.1, the robot firstly throws the ball according to its current knowledge (e.g., random actions at the beginning), then judge how good the performance is (the closer to the target, the better performance) and updates its knowledge. Repeating this strategy, the robot will iteratively improve its performance and finally achieve the task. This learning strategy is called reinforcement

learning [1, 2], a learning paradigm inspired by behaviorist psychology where the agent learns to interact with the environment in order to maximize some long term cumulative rewards. As one integral part of contemporary machine learning, reinforcement learning is different to other learning methods like supervised learning [3] and imitation learning [4,5]. The learning reinforces the good decision making while penalizes the bad decision making according to the reward signal collected from the environment. With capability of exploring unknown worlds, reinforcement learning becomes an popular approach that expresses a remarkably broad range of robot control problems in a natural manner [6,7], i.e., deriving the robots to learn complex tasks by exploring and interacting with the environment.

The main problem solved in reinforcement learning is to calculate the value function to indicate the long-term reward of all states. It naturally turns to a good policy by following the value function to reach high reward state. In robot control domain, the reinforcement learning algorithms are mainly divided into two groups according to [6,7]: the value function approach and the policy search, which can be viewed as direct and indirect methods to learn the value function. The value function approach, e.g. Q-learning [8, 9], SARSA [10], Least squares policy evaluation (LSPE) [11], and Least Squares Policy Iteration (LSPI) [12–14], attempts to learn a complete optimal value function among all states and actions and generates an optimal policy for discrete actions in accordance to it based on the Bellman Principle of Optimality [15]. Its applications in real robot control include multiple robot cooperation in soccer game [16], three-link and two-joint robot [17], bimanual reaching task using humanoid robot [18] and autonomous vehicle control [19]. The policy search [20], e.g., REINFORCE method [21], Natural Actor-Critic [22], Relative Entropy Policy Search [23], Policy Improvement with Path Integrals (PI^2) [24] and Guided Policy Search [25], initiates a control policy parameterized by prior task knowledge (e.g. a Central Pattern Generator (CPG) [26] or Dynamic Movement Primitives (DMP) [27, 28]) and gradually updates the parameters within a localized region of state-action space. It has been successfully applied to several robots in different tasks including: simple biped robot [29], helicopter flight [30], robot swings baseball [31], humanoid biped robot [32] and 12 DOF robot dog [33].

1.2 Motivation

Besides the current applications of both the value function approach and the policy search in robot control domain, on the other hand, applying them to complex robot systems with high dimensional states still remains a big challenge due to several issues including the high cost of generating samples in real robots, the lack of the model knowledge and the intractable computational complexity. The value function approach, where function approximation is commonly utilized to represent the value function, becomes problematic when working with complex robot systems featuring continuous states. The insufficient sample quantity due to the difficulties of gathering enough samples to densely fill the relevant region of state-action space on real complex robot systems, rapidly results in both unstable value function approximation and overly large policy updates that cause divergence during learning. It is called the curse of insufficient samples. The computational complexity of approximating the value function quickly becomes intractable as system dimensionality increases. It is the curse of dimensionality.

By contrast, the policy search is believed to be more popular in controlling complex robots according to [6, 7]. It improves the stability of learning and reduces computational complexity by only searching the parameter space of control policy instead of the entire-state-action space. By carefully selecting a suitable parameterized policy via knowledge from simulation, model dynamics [34], human demonstration [5] or imitation learning [4, 5], the learning process will optimize the performance of policy to a quite good one quickly. Unfortunately, the policy search requires suitable parameterized and initialized control policies based on prior knowledge of tasks and models that are not always available.

Compared with the model-free methods mentioned above, the model-based reinforcement learning refers to learning optimal behavior indirectly by learning a model of the environment [34, 35]. On the other hand, it is challenging to learn a enough good model with given limited samples, especially in large dimensional state space. The learned model may worsen the learning compared with the model-free reinforcement learning. In this thesis, we will not touch model-based reinforcement learning.

The motivation of this thesis rises from the challenge above. We aim to explore practical model-free reinforcement learning algorithms in complex robot systems

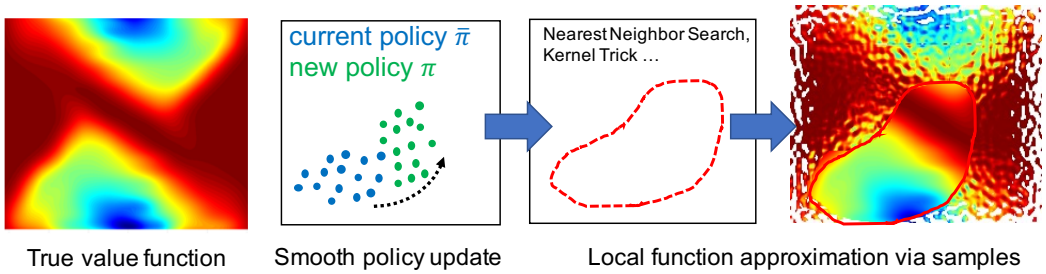


Figure 1.2: Proposed framework: learning value function from local area.

to fulfill: no estimation of model during learning, stably learning with insufficient real world samples, tractable computational complexity in high dimensional systems and no requirement of suitable parameterized control policies. Our main idea is to take advantages of both the value function approach and the policy search. Following the strategy of the policy search, we learn the global value function from samples generated in local areas in order to stabilize the learning process. Once the global value function can be efficiently approximated via these local samples, the curse of dimensionality could be solved. On the other hand, we learn a parameterized value function rather than a policy following the value function approach to be pure free to the prior knowledge of tasks and models.

1.3 Contribution

A new reinforcement learning framework is proposed to accomplish this idea. It is inspired by a optimize control algorithm Dynamic Policy Programming (DPP) [36–38] that employs Kullback-Leibler divergence to limit the over large policy update. We first extend DPP to a reinforcement learning algorithm with exploration. By smoothing the update of the exploration policy, the insufficient samples are generated in local areas that smoothly move following this policy to construct a continue area in global state-action space. These local samples are further utilized to efficiently approximating value function with other machine learning methods. Figure. 1.2 shows the proposed framework.

Two new value function approach based algorithms are introduced as examples of the proposed framework: Local Update Dynamic Policy Programming

(LUDPP) and Kernel Dynamic Policy Programming (KDPP). Both two algorithms stabilize the learning by adding the Kullback-Leibler divergence between current and new policies as a regularization term. Different machine learning methods, Nearest Neighbor Search (NNS) [39] and kernel trick [40], are employed in LUDPP and KDPP to reduce the computational complexity in high-dimensional systems.

The learning performance of LUDPP and KDPP are first investigated in simulation tasks with increase number of system dimensions and compared with conventional value function approach based learning algorithms with the same machine learning methods (NNS and kernel trick). With the increase number of system dimensions and insufficient samples, the proposed methods greatly outperform the conventional ones. Then LUDPP and KDPP are applied to control the Shadow Dexterous Hand [41], a Pneumatic Artificial Muscle (PAM) driven humanoid robot hand. LUDPP successfully learns one finger position reaching task with a limited computational resource while other conventional methods cannot. KDPP learns unscrewing a bottle cap with the aid of touch sensors. This combined system has a 32 dimensional state space with 625 discrete actions whose high state-action dimensionality renders it impractical for conventional value function approach based algorithms to the best of our knowledge, whereas KDPP converged to a viable solution within a small number of learning iterations while given limited samples.

Besides the proposed framework and LUDPP [42, 43], KDPP [44, 45], the author has also carried out research in extending DPP to deep reinforcement learning [46–50] for controlling robot handle deformable objects [51], intelligent driving assistance based on DPP for disabled people [52] and Environment-adaptive Interaction Primitives (EalPs), a new algorithm for motor skill learning for human robot cooperation [53][1]. For the reason of consistency, this thesis will only introduce EaIPs as a work related to learning policy parameterized by DMP ib appendix.

The remainder of this thesis is organized as follows. In Chapter 2, preliminaries of reinforcement learning and Markov Decision Process are introduced. The

[1] these two works are cooperations between Nara Institute of Science and Technology and University of Technology, Sydney

smooth policy update using Kullback-Leibler divergence is presented in Chapter 3 as the foundation of the proposed framework. LUDPP, KDPP and the corresponding simulation results, analysis and real robot experiment of Pneumatic Artificial Muscle (PAM) driven humanoid robot hand are in Chapters 4 and 5 respectively. The discussions and conclusions are in Chapters 6 and 7. Lastly, an appendix gives more details of EaIPs, another proposed algorithm for adaptive motion planning in human-robot cooperation.

2 Preliminaries

In this chapter, the basic definitions and properties of the Markov decision process (MDP), and the notation that will be used in the reminder of this thesis are first introduced. Then a brief introduction of the value function approach and the policy search is presented.

2.1 Markov Decision Processes

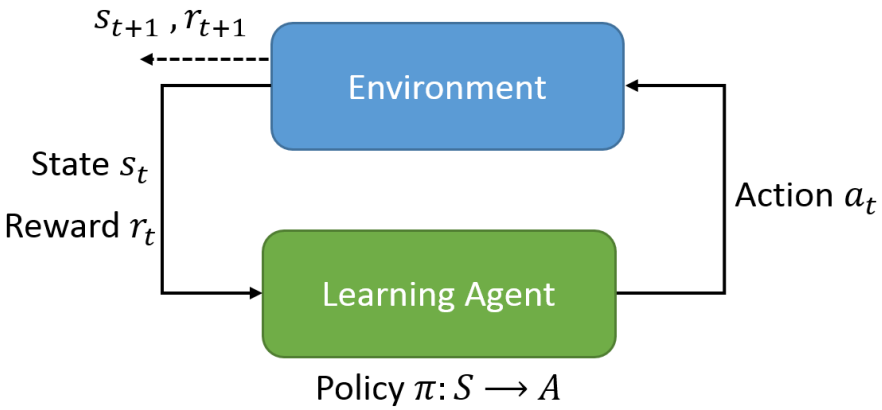


Figure 2.1: The framework of reinforcement learning, $\boldsymbol{s}$ is the state of the environment, $\boldsymbol{a}$ is the action the learning agent make, r is the reward signal.

The Markov decision processes (MDP) [54, 55] is usually used to represent problems in reinforcement learning. It models the system we hope to control to several states, and allows the agent to do different actions to the environment. The solution of a MDP is to select actions according to states to obtain high

reward. In each time step, the learning agent interacts with a dynamic environment by taking action $\boldsymbol{a}$ according to its current policy π, observes the state of the environment in next step and obtains the reward signal. The policy of making decision of action $\boldsymbol{a}$ will be reinforced if the reward is good/positive and vice versa. By repeating this learning loop, the agent will be able to iteratively learn a good policy that maximize the long term reward.

MDP gives a formalism of planning actions for environment in the face of uncertain. It is a discrete time stochastic process defined by a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ consisting of:

- $\mathcal{S} = \{\boldsymbol{s}_1, \boldsymbol{s}_2, ..., \boldsymbol{s}_n\}$ is a finite set of control states that represents the different situations in which decisions must be made.
- $\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_m\}$ is a finite set of control actions for agent.
- $\mathcal{P}$ is the transition model of the process; $\mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$ is the probability of transitioning from state $\boldsymbol{s}$ to state $\boldsymbol{s}'$ under action $\boldsymbol{a}$.
- $\mathcal{R}$ is the reward (or cost) function of the process that maps transitions to real numbers, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$; when transitioning from state $\boldsymbol{s}$ to state $\boldsymbol{s}'$ under the action $\boldsymbol{a}$, the agent gets the reward $\mathcal{R}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$; for simplicity, we define the expected reward for taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$ by:

$$R(\boldsymbol{s}, \boldsymbol{a}) = \sum_{\boldsymbol{s}' \in \mathcal{S}} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\mathcal{R}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \tag{2.1}$$

- $\gamma \in (0, 1)$ is the discount factor that exponentially discounts future rewards of MDP in an infinite horizon. It is an implicit way to introduce the notion of elapsed time in the process.

As one discrete-time process, MDP begins at time t_0 in state $\boldsymbol{s}_0 \in \mathcal{S}$. The agent observes the state at each time step $\boldsymbol{s}_t$ and makes an action $\boldsymbol{a}_t \in \mathcal{A}$. The state of next step $\boldsymbol{s}_{t+1}$ is obtained from $\mathcal{P}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ while the reward is determined by $\mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$.

MDP's horizon h, i.e., the span of the process over time, can be infinite (the process runs forever), finite (the process will stop in a finite number of steps) and fixed (the process runs for a fixed number of steps). We also define episode as

a complete run of MDP in h steps which consists a sequence of states, actions, and rewards. The expected discounted total reward of one episode is therefore calculated by:

$$\mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{P}} \left[\sum_{t=0}^{h} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \middle| \boldsymbol{s}_0 = \boldsymbol{s} \right]. \tag{2.2}$$

2.2 Properties of Markov Decision Processes

The goal of the agent in reinforcement learning is to optimize the expected discounted total reward in Eq. (2.2) by selecting actions $\boldsymbol{a}_0, \boldsymbol{a}_1, ..., \boldsymbol{a}_h$ and interacting with the process. A stochastic policy $\pi : \mathcal{S} \mapsto \Omega(\mathcal{A})$, indicates the agent's probability of choosing all actions in each state. $\pi(\boldsymbol{a}|\boldsymbol{s})$ represents the probability of taking action $\boldsymbol{a}$ at state $\boldsymbol{s}$. $\pi : \mathcal{S} \mapsto \mathcal{A}$ represents the function of choosing a good policy at the state $\boldsymbol{s}$ that fulfills $\pi(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s})$. The expected discounted total reward with policy π is defined by:

$$\mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{P}; \boldsymbol{a}_t \sim \pi} \left[\sum_{t=0}^{h} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \middle| \boldsymbol{s}_0 = \boldsymbol{s} \right], \tag{2.3}$$

an optimal policy π^* maximizing the expected discounted total reward follows:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{P}; \boldsymbol{a}_t \sim \pi} \left[\sum_{t=0}^{h} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \middle| \boldsymbol{s}_0 = \boldsymbol{s} \right]. \tag{2.4}$$

According to [54], there exists at least one optimal policy for each MDP.

In order to evaluate the goodness of each state, Eq. (2.3), the expected total reward when the process starts in state $\boldsymbol{s}$ following policy π, is defined by the state value function $V^\pi : \mathcal{S} \mapsto \mathbb{R}$:

$$V^\pi(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{P}; \boldsymbol{a}_t \sim \pi} \left[\sum_{t=0}^{h} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \middle| \boldsymbol{s}_0 = \boldsymbol{s} \right], \tag{2.5}$$

It is often convenient to associate state value functions with state-action pairs rather than states. The expected total discounted reward upon choosing action $\boldsymbol{a}$ from state $\boldsymbol{s}$ and then following policy π is defined by the state-action value function $Q^\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$:

$$Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{P}; \boldsymbol{a}_t \sim \pi} \left[\sum_{t=0}^{h} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \middle| \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a} \right], \tag{2.6}$$

The optimal expected value functions $V^* : \mathcal{S} \mapsto \mathbb{R}$ and $Q^* : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ are then defined by:

$$V^*(\boldsymbol{s}) = \max_{\pi} V^{\pi}(\boldsymbol{s}), \tag{2.7}$$

$$Q^*(\boldsymbol{s}, \boldsymbol{a}) = \max_{\pi} Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}). \tag{2.8}$$

The state value function satisfies Bellman equations that are used to efficiently solve MDP. They are recursive definitions of the value function which consists of two parts, the immediate reward, and the expected sum of future discounted rewards:

$$V^{\pi}(\boldsymbol{s}) = \sum_{\boldsymbol{a} \in \mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) \Big(R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V^{\pi}(\boldsymbol{s}') \Big), \tag{2.9}$$

Since the V^{π}, V^* and Q^{π}, Q^* have relationship:

$$V^{\pi}(\boldsymbol{s}) = Q^{\pi}\Big(\boldsymbol{s}, \pi(\boldsymbol{s})\Big), \tag{2.10}$$

The state-action value function also satisfies Bellman equations following:

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \sum_{\boldsymbol{a}' \in \mathcal{A}} \pi(\boldsymbol{a}^{'}|\boldsymbol{s}^{'}) Q^{\pi}(\boldsymbol{s}^{'}, \boldsymbol{a}^{'}). \tag{2.11}$$

It is also convenient to represent the Bellman equations of state-action function using matrix notation:

$$Q^{\pi} = R + \gamma \boldsymbol{P} \prod_{\pi} Q^{\pi} \tag{2.12}$$

where Q^{π} and R are victors with size $|\mathcal{S}||\mathcal{A}|$, $\boldsymbol{P}$ is a $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$ matrix of transition model $\mathcal{P}$ that fulfills:

$$\boldsymbol{P}\Big((\boldsymbol{s}, \boldsymbol{a}), \boldsymbol{s}^{'}\Big) = \mathcal{P}(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{s}^{'}), \tag{2.13}$$

The learning result of unscrewing bottle cap using Shadow Dexterous Hand.The learning result of unscrewing bottle cap using Shadow Dexterous Hand. One $|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|$ stochastic matrix $\prod_{\pi}$ is defined to describe π:

$$\prod_{\pi} \Big(\boldsymbol{s}^{'}, (\boldsymbol{s}^{'}, \boldsymbol{a}^{'})\Big) = \pi(\boldsymbol{a}^{'}|\boldsymbol{s}^{'}). \tag{2.14}$$

In order to analytically or iteratively to obtain the value of Q^{π}, the linear system based Equation (2.12) is built:

$$\Big(\boldsymbol{I} - \gamma \boldsymbol{P} \prod_{\pi}\Big) Q^{\pi} = R. \tag{2.15}$$

Searching an optimal policy π^* following Eq. (2.4), the corresponding optimal state value function and state-action value function satisfy bellman equations according to Eqs. (2.9) and (2.11):

$$V^*(\boldsymbol{s}) = \max_{\boldsymbol{a} \in \mathcal{A}} \Big[R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') V^*(\boldsymbol{s}') \Big], \tag{2.16}$$

$$Q^*(\boldsymbol{s}, \boldsymbol{a}) = R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \max_{\boldsymbol{a}' \in \mathcal{A}} Q^*(\boldsymbol{s}', \boldsymbol{a}'). \tag{2.17}$$

The corresponding matrix format is:

$$Q^* = R + \gamma \boldsymbol{P} \prod_{\max}^{Q^*} Q^* \tag{2.18}$$

where Q^* and R are victors with size $|\mathcal{S}||\mathcal{A}|$, $\boldsymbol{P}$ is a $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$ matrix of transition model $\mathcal{P}$ following Eq. (2.13). Another stochastic matrix $\prod_{\max}^{Q^*}$ with same size of $\prod_{\pi}$ is defined by:

$$\prod_{\max}^{Q^*} \Big(\boldsymbol{s}', (\boldsymbol{s}', \boldsymbol{a}') \Big) = \begin{cases} 1 & \text{if} \quad \boldsymbol{a}' = \arg\max_{\boldsymbol{a}} Q^*(\boldsymbol{s}, \boldsymbol{a}') \\ 0 & \text{, otherwise} \end{cases}. \tag{2.19}$$

The solution of the k-th iteration is converged to Q^* by iteratively approximating following:

$$Q^{k+1} = R + \gamma \boldsymbol{P} \prod_{\max}^{Q^k} Q^k. \tag{2.20}$$

As another view of the the Bellman equations of state-action value function Q, the Bellman operators $\mathcal{T}^{\pi}$ and Bellman optimality operator $\mathcal{T}$ are defined by:

$$(\mathcal{T}^{\pi} Q^{\pi})(\boldsymbol{s}, \boldsymbol{a}) \triangleq R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \sum_{\boldsymbol{a}' \in \mathcal{A}} \pi(\boldsymbol{a}' | \boldsymbol{s}') Q^{\pi}(\boldsymbol{s}', \boldsymbol{a}'), \tag{2.21}$$

$$(\mathcal{T} Q^*)(\boldsymbol{s}, \boldsymbol{a}) \triangleq R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \max_{\boldsymbol{a}' \in \mathcal{A}} Q^*(\boldsymbol{s}', \boldsymbol{a}'). \tag{2.22}$$

According to [56], both two operators above are contraction mappings. For every policy π and any two state-action value functions Q and Q', it follows:

$$\|\mathcal{T}^{\pi} Q - \mathcal{T}^{\pi} Q'\| \leq \gamma \|Q - Q'\|, \qquad \|\mathcal{T} Q - \mathcal{T} Q'\| \leq \gamma \|Q - Q'\|. \tag{2.23}$$

Algorithm 1 Value Iteration and Policy Iteration

Require: $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\gamma, \epsilon$ and initial state-action function Q_0

Initialize state-action function $Q^{'} \leftarrow Q_0$
repeat
  $Q \leftarrow Q^{'}$
  $Q^{'} \leftarrow R + \gamma \boldsymbol{P} \prod_{\max}^{Q} Q$
until $\|Q - Q^{'}\|_{\infty} < \epsilon$
$\pi(\boldsymbol{s}) \leftarrow \arg\max_{\boldsymbol{a} \in \mathcal{A}} Q^{'}(\boldsymbol{s}, \boldsymbol{a}), \forall \boldsymbol{s} \in \mathcal{S}$
return π

Algorithm 2 Policy Iteration

Require: $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\gamma$ and initial policy π_0

Initialize policy $\pi^{'} \leftarrow \pi_0$
repeat
  $\pi \leftarrow \pi^{'}$
  $Q^{\pi} \leftarrow \left(\boldsymbol{I} - \gamma \boldsymbol{P} \prod_{\pi}\right)^{-1} R$
  $\pi^{'}(\boldsymbol{s}) \leftarrow \arg\max_{\boldsymbol{a} \in \mathcal{A}} Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}), \forall \boldsymbol{s} \in \mathcal{S}$
until $\pi = \pi^{'}$
return π

2.3 Value Function based Approach

The value function approach, e.g. Q-learning [8, 9], SARSA [10], Least squares policy evaluation (LSPE) [11], and Least Squares Policy Iteration (LSPI) [12–14], attempts to learn a complete optimal value function among all states and actions and generates an optimal policy for discrete actions in accordance to it. It starts from dynamic programming on MDP [55] which includes value iteration [54] and policy iteration [57].

As the pseudo code of value iteration shown in Algorithm 1, the value iteration calculates the optimal state-action value function Q^* by iteratively applying Bellman optimality operator on initial state-action value function Q_0 since the optimal policy π^* can be obtained by selecting actions according to Q^* greedily [54]. On the other hand, policy iteration keeps an arbitrary policy π in memory and search the optimal policy by iteratively improving this policy [57] following Al-

gorithm 2. Combining them with function approximation, we get approximate value iteration (AVI) and approximate value iteration (API) to solve MDP with continuous states. According to [56, 58], both AVI and API have error bounds without committing to any particular approximation method that indicates their practicability following:

Theorem 2.3.1. *Define $\hat{Q}^{\pi_k}$ to be the approximate value function on Q^{π_k}, a boundary of approximate policy evaluation ϵ is defined by:*

$$\epsilon = \limsup_{k\to\infty} \|\hat{Q}^{\pi_k} - Q^{\pi_k}\|_\infty.$$

If π_k is the greedy policy of $Q^{\hat{\pi}_k}$, the loss bound in the presence of approximation error bound follows:

$$\limsup_{k\to\infty} \|Q^* - Q^{\pi_k}\|_\infty \leq \frac{2\gamma\epsilon}{(1-\gamma)^2}$$

where γ is the discount factor of MDP.

Both AVI and API require the knowledge of the transition model $\mathcal{P}$ which is difficult to obtain in real world problems. One solution is model-free learning that directly learns the value function via operating a appropriate tradeoff between exploration of the new actions/states and exploitation of the current knowledge without knowing the transition model. This thesis focus on model-free reinforcement learning. The famous and widely used value function approaches include Q-learning [8], SARSA [10], Least-Squares Policy Iteration (LSPI) [12, 59] and Least Squares Policy Evaluation (LSPE) [11].

In principle, the value function based approach requires a global convergence of the state-action space. Once the optimal value function is obtained, it is easy to find the optimal policy by simply greedily choosing actions to optimize it according to Eqs. (2.16) and (2.17). However, the value function based approach becomes problematic when working with complex robot systems featuring continuous states, where function approximation is commonly utilized to represent the value function. Insufficient sample quantity due to the difficulties of gathering enough samples to densely fill the relevant region of state-action space on real complex robot systems, rapidly results in both unstable value function approximation and overly large policy updates that cause both divergence during

learning and dangerous actions generated by over large policy deviations in real world systems. It turns to some practical approaches that learn first step solution in simulations and then move to real robots [17]. Moreover, the computational complexity of exploring the entire state-action space and approximating the state-value function exponentially with the increase of the system dimensionality increases and quickly becomes intractable with common computational resources. All the challenges above result in limited applications of the value function based approach in robot control problems including multiple robot cooperation in soccer game [16], three-link and two-joint robot [17], bimanual reaching task using humanoid robot [18] and autonomous vehicle control [19]. It is believed that the recent research has given up to approximate the whole value function and rather directly learns the control policy from local trajectories/rollouts due to the intractable calculation and insufficient samples in real world [6, 7].

2.4 Policy Search

As another alternative solution to the robot control problems with high dimensional state space besides the value function approach, the policy search [22, 60] is the most popular way to generate control policy in robotics that naturally supports continuous actions and be able to benefit from well selected initial policies according to [6, 7]. Several algorithms are developed in this group including REINFORCE method [21], Natural Actor-Critic [22], Relative Entropy Policy Search [23], Policy Improvement with Path Integrals (PI^2) [24] and Guided Policy Search [25].

The central idea of the policy search is to learn a policy without calculating or approximating the value function V or Q. It efficiently represents and learns a good approximated policy $\hat{\pi}(*, \boldsymbol{\theta}_k)$ where $\boldsymbol{\theta}_k$ is a free parameter. The optimization of policy is operated locally around the current $\hat{\pi}(*, \boldsymbol{\theta}_k)$ by calculating $\Delta\boldsymbol{\theta}_k$ that increase the expected reward of MDP in Eq. (2.3). The approximated policy is then updated by $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_k$ where $\Delta\boldsymbol{\theta}_i$ is obtained by performing gradient algorithms on the expected reward:

$$\Delta\boldsymbol{\theta}_k = \alpha \nabla_{\boldsymbol{\theta}_k} \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{P}; \boldsymbol{a}_t \sim \hat{\pi}(*, \boldsymbol{\theta}_k)} \left[\sum_{t=0}^{h} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \middle| \boldsymbol{s}_0 = \boldsymbol{s} \right], \tag{2.24}$$

$\alpha \in [0, 1]$ is the learning rate. The optimal parameter $\boldsymbol{\theta}^*$ follows:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{P}; \boldsymbol{a}_t \sim \hat{\pi}(*, \boldsymbol{\theta}_k)} \left[\sum_{t=0}^{h} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \middle| \boldsymbol{s}_0 = \boldsymbol{s} \right]. \tag{2.25}$$

Compared with the value function approaches, on the other hand, the policy search improves the stability of learning and reduces computational complexity by only searching the parameter space of control policy instead of the entire-state-action space and therefore be popular in robot control: simple biped robot [29], helicopter flight [30], robot swings baseball [31], humanoid biped robot [32] and 12 DOF robot dog [33]. On the other hand, the policy search has its own limitations: the performance is gradually improved by only considering the current policy and its neighborhood in the policy search. Furthermore, it requires properly designed policy model with tunable parameters (e.g., a Central Pattern Generator [26] or Dynamic Movement Primitives [27, 28]) based on the prior knowledge of robot systems and tasks. Even though such a knowledge could be obtained from model dynamics [34], human demonstration [5] or imitation learning [4, 5], the application of the policy search is still limited when no prior knowledge is available.

3 Dynamic Policy Programming

In this chapter, we first introduce Dynamic Policy Programming (DPP) [36–38] and its extension with linear function approximation, Sampling-based Approximate Dynamic Policy Programming (SADPP). As the foundation of the proposed framework in this thesis, both DPP and SADPP stably solve MDPs with smooth policy updates by employing the Kullback-Leibler divergence between current and new policies as a regularization term. Then we investigate the learning performance of SADPP with samples reuse in a pendulum swing up simulation, and compare it with LSPI. The results indicate the positive effect of smooth policy update in the learning using limited number of samples.

3.1 Kullback-Leibler Divergence in Bellman Equation

The Kullback–Leibler divergence [61, 62], also called information divergence, information gain, relative entropy and KL divergence is defined to measure the non-symmetric difference between two probability distributions. In MDP, having the current policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ and the baseline policy $\bar{\pi}(\boldsymbol{a}|\boldsymbol{s})$, the Kullback-Leibler divergence measures the difference between these two probability distributions following:

$$KL\Big(\pi(\cdot|\boldsymbol{s})\|\bar{\pi}(\cdot|\boldsymbol{s})\Big) = \sum_{\boldsymbol{a}\in\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) \log\left(\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\bar{\pi}(\boldsymbol{a}|\boldsymbol{s})}\right). \tag{3.1}$$

According to [36–38], a new algorithm called Dynamic Policy Programming (DPP) is proposed by adding the Kullback-Leibler divergence to the expected reward r_{s_t} as a penalty term to build a new value function:

$$V^{\pi}_{\bar{\pi}}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{s}_t\sim\mathcal{P};\boldsymbol{a}_t\sim\pi}\left[\sum_{t=0}^{h}\gamma^t\Big(\mathcal{R}(\boldsymbol{s}_t,\boldsymbol{a}_t,\boldsymbol{s}_{t+1}) - \frac{1}{\eta}KL\Big(\pi(\cdot|\boldsymbol{s}_t)\|\bar{\pi}(\cdot|\boldsymbol{s}_t)\Big)\Big)\middle|\boldsymbol{s}_0=\boldsymbol{s}\right], \tag{3.2}$$

where $\eta \in (0, 1]$ is the inverse temperature to control the effect of Kullback-Leibler divergence term: the the reward is less related to the difference between $\pi(\boldsymbol{a}|\boldsymbol{s})$ and $\bar{\pi}(\boldsymbol{a}|\boldsymbol{s})$ with the increase of η. Combine Eqs. (3.2) and (2.9), the Bellman equations of state value function with Kullback-Leibler divergence satisfies:

$$V_{\bar{\pi}}^{\pi}(\boldsymbol{s}) = \sum_{\boldsymbol{a}\in\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) \Bigg[R(\boldsymbol{s},\boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') V_{\bar{\pi}}^{\pi}(\boldsymbol{s}') - \frac{1}{\eta} \log\left(\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\bar{\pi}(\boldsymbol{a}|\boldsymbol{s})}\right) \Bigg], \quad (3.3)$$

As a modified version of Eq. (2.5), Eq. (3.3) minimizes the difference between the current policy π and the baseline policy $\bar{\pi}$ while maximizing the expected reward. The balance of these two sides is controlled by η. Following [36, 63], let η be a positive constant, for all $\boldsymbol{s} \in \mathcal{S}$ the optimal value function $V_{\bar{\pi}}^{*}(\boldsymbol{s})$ and for all $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}$ the optimal policy $\bar{\pi}^{*}(\boldsymbol{a}|\boldsymbol{s})$ respectively satisfy:

$$V_{\bar{\pi}}^{*}(\boldsymbol{s}) = \max_{\pi} \sum_{\boldsymbol{a}\in\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) \Bigg[R(\boldsymbol{s},\boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') V_{\bar{\pi}}^{*}(\boldsymbol{s}') - \frac{1}{\eta} \log\left(\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\bar{\pi}(\boldsymbol{a}|\boldsymbol{s})}\right) \Bigg]. \quad (3.4)$$

Following [36, 63], the maximization of Eq. (3.4) can be performed in a closed form. let η be a positive constant, for all $\boldsymbol{s} \in \mathcal{S}$ the optimal value function $V_{\bar{\pi}}^{*}(\boldsymbol{s})$ and for all $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}$ the optimal policy $\bar{\pi}^{*}(\boldsymbol{a}|\boldsymbol{s})$ respectively satisfy:

$$V_{\bar{\pi}}^{*}(\boldsymbol{s}) = \frac{1}{\eta} \log \sum_{\boldsymbol{a}\in\mathcal{A}} \bar{\pi}(\boldsymbol{a}|\boldsymbol{s}) \exp\left[\eta \sum_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') \Big(\mathcal{R}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V_{\bar{\pi}}^{*}(\boldsymbol{s}')\Big)\right] \quad (3.5)$$

$$\bar{\pi}^{*}(\boldsymbol{a}|\boldsymbol{s}) = \frac{\bar{\pi}(\boldsymbol{a}|\boldsymbol{s}) \exp\left[\eta \sum\limits_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') \Big(\mathcal{R}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V_{\bar{\pi}}^{*}(\boldsymbol{s}')\Big)\right]}{\exp\Big(\eta V_{\bar{\pi}}^{*}(\boldsymbol{s})\Big)}. \quad (3.6)$$

These results are derived by applying Lagrangian multipliers to Eq. (3.3) and calculating the maximization with constraints $\sum_{\boldsymbol{a}\in\mathcal{A}} \pi(\boldsymbol{a}|\boldsymbol{s}) = 1$, $0 < \pi(\boldsymbol{a}|\boldsymbol{s}) < 1$.

Since $\bar{\pi}^{*}(\boldsymbol{a}|\boldsymbol{s})$ is a function of the baseline policy $\bar{\pi}(\boldsymbol{a}|\boldsymbol{s})$, the optimal state value function $V_{\bar{\pi}}^{*}$ and transition model $\mathcal{P}$, we can first obtain the optimal state value function by a fixed-point iteration:

$$V_{\bar{\pi}}^{k+1}(\boldsymbol{s}) = \frac{1}{\eta} \log \sum_{\boldsymbol{a}\in\mathcal{A}} \bar{\pi}(\boldsymbol{a}|\boldsymbol{s}) \exp\left[\eta \sum_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') \Big(\mathcal{R}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V_{\bar{\pi}}^{k}(\boldsymbol{s}')\Big)\right]. \quad (3.7)$$

In order to solve the MDP and calculate the optimal policy π^{*}, a double-loop algorithm is repeated by repeating the baseline policy $\bar{\pi}$ is replaced by new policy

calculated via Eq. (3.6) at each step. It is the main loop of DPP that contains both state value function update and policy update as:

$$V_{\bar{\pi}}^{k+1}(\boldsymbol{s}) = \frac{1}{\eta} \log \sum_{\boldsymbol{a}\in\mathcal{A}} \bar{\pi}^k(\boldsymbol{a}|\boldsymbol{s}) \exp\Big[\eta \sum_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\Big(\mathcal{R}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V_{\bar{\pi}}^k(\boldsymbol{s}')\Big)\Big], \tag{3.8}$$

$$\bar{\pi}^{k+1}(\boldsymbol{a}|\boldsymbol{s}) = \frac{\bar{\pi}^k(\boldsymbol{a}|\boldsymbol{s}) \exp\Big[\eta \sum\limits_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\Big(\mathcal{R}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V_{\bar{\pi}}^k(\boldsymbol{s}')\Big)\Big]}{\exp\Big(\eta V_{\bar{\pi}}^{k+1}(\boldsymbol{s})\Big)}. \tag{3.9}$$

To simply DPP's loop, the action preferences function Ψ_k [1] for all state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ is defined by:

$$\Psi_{k+1}(\boldsymbol{s},\boldsymbol{a}) \triangleq \frac{1}{\eta} \log \bar{\pi}^k(\boldsymbol{a}|\boldsymbol{s}) + \sum_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\Big(\mathcal{R}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma V_{\bar{\pi}}^k(\boldsymbol{s}')\Big). \tag{3.10}$$

By plugging Eq. (3.10) in to Eqs. (3.8) and (3.9), we deduce a simple loop of DPP:

$$V_{\bar{\pi}}^{k+1}(\boldsymbol{s}) = \frac{1}{\eta} \log \sum_{\boldsymbol{a}\in\mathcal{A}} \exp\Big(\eta \Psi_k(\boldsymbol{s},\boldsymbol{a})\Big), \tag{3.11}$$

$$\bar{\pi}^{k+1}(\boldsymbol{a}) = \frac{\exp\Big(\eta \Psi_k(\boldsymbol{s},\boldsymbol{a})\Big)}{\sum_{\boldsymbol{a}'\in\mathcal{A}} \exp\Big(\eta \Psi_k(\boldsymbol{s},\boldsymbol{a}')\Big)}. \tag{3.12}$$

Finally, a Bellman-like recursion of DPP is obtained by plugging the loop above to Eq. (3.10):

$$\begin{aligned}\Psi_{k+1}(\boldsymbol{s},\boldsymbol{a}) &= \Psi_k(\boldsymbol{s},\boldsymbol{a}) \Psi_k(\boldsymbol{s}) + \sum_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\Big(\mathcal{R}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') + \gamma \mathcal{L}_\eta \Psi_k(\boldsymbol{s}')\Big) - \mathcal{L}_\eta \Psi_k(\boldsymbol{s}) \\ &= \Psi_k(\boldsymbol{s},\boldsymbol{a}) + R(\boldsymbol{s},\boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}'\in\mathcal{S}} \mathcal{P}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}') \mathcal{L}_\eta \Psi_k(\boldsymbol{s}') - \mathcal{L}_\eta \Psi_k(\boldsymbol{s}).\end{aligned} \tag{3.13}$$

where the operator $\mathcal{L}_\eta$ is defined by:

$$\mathcal{L}_\eta \Psi(\boldsymbol{s}) \triangleq \frac{1}{\eta} \log \sum_{\boldsymbol{a}\in\mathcal{A}} \exp(\eta \Psi(\boldsymbol{s},\boldsymbol{a})). \tag{3.14}$$

To get a more analytically tractable recursion according to [36], $\mathcal{L}_\eta \Psi(\boldsymbol{s})$ is replaced by a Boltzmann soft-max operator:

$$\mathcal{M}_\eta \Psi(\boldsymbol{s}) = \sum_{\boldsymbol{a} \in \mathcal{A}} \frac{\exp\Big(\eta \Psi(\boldsymbol{s}, \boldsymbol{a})\Big) \Psi(\boldsymbol{s}, \boldsymbol{a})}{\sum_{\boldsymbol{a}' \in \mathcal{A}} \exp\Big(\eta \Psi(\boldsymbol{s}, \boldsymbol{a}')\Big)}. \tag{3.15}$$

It is proved in [64] that the difference between the $\mathcal{M}_\eta$ and $\mathcal{L}_\eta$ is limited following:

$$|\mathcal{L}_\eta(x) - \mathcal{M}_\eta(x)| \leq \frac{\log(|\mathcal{A}|)}{\eta}. \tag{3.16}$$

Then the final recursion with Boltzmann soft-max operator is as follows:

$$\begin{aligned} \Psi_{k+1}(\boldsymbol{s}, \boldsymbol{a}) &\triangleq \mathcal{O}\Psi_k(\boldsymbol{s}, \boldsymbol{a}) \\ &= \Psi_k(\boldsymbol{s}, \boldsymbol{a}) + R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \mathcal{M}_\eta \Psi_k(\boldsymbol{s}') - \mathcal{M}_\eta \Psi_k(\boldsymbol{s}), \end{aligned} \tag{3.17}$$

$\mathcal{O}$ is an operator defined on Ψ_k and the Bellman operator $\mathcal{T}^{\pi_k} \Psi_k(\boldsymbol{s}, \boldsymbol{a})$ represents $R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} \mathcal{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \mathcal{M}_\eta \Psi_k(\boldsymbol{s}')$.

Like other AVI and API algorithms, DPP with function approximation have error boundary without committing to any particular approximation method. According to [36–38], DPP has a better error boundary than other value function approaches following theorem 3.1.1:

Theorem 3.1.1. *Define $\hat{Q}^{\pi_0}, \hat{Q}^{\pi_1}, ..., \hat{Q}^{\pi_k}$ to be the approximate value function on k steps sequence, a average boundary of function approximation $\bar{\epsilon}$ is defined by:*

$$\bar{\epsilon} = \limsup_{k \to \infty} \frac{\sum_{j=0}^{k} \|\mathcal{T}^{\pi_j} \hat{Q}^{\pi_j} - \hat{Q}^{\pi_{j+1}}\|_\infty}{k+1}.$$

If π_k is the greedy policy of $Q^{\hat{\pi}_k}$, the loss bound in the presence of approximation error bound follows:

$$\limsup_{k \to \infty} \|Q^* - Q^{\pi_k}\|_\infty \leq \frac{2\gamma\bar{\epsilon}}{(1-\gamma)^2}$$

where γ is the discount factor of MDP.

Obviously, the average error $\bar{\epsilon}$ is usually smaller than the supremum norm of error ϵ. Therefore, compared with other AVI and API algorithms, DPP is

guaranteed to achieve a near-optimal performance in the approximation of value function even the individual errors ϵ_k are very large.

In this section, we introduce the algorithm of the original dynamic policy programming. By applying Kullback–Leibler divergence, DPP is able to limited the over large policy update represented by the action preferences function, ie, a smooth policy update. As the key feature we utilized in our proposed framework, the corresponding sampling distribution will thus be changed slightly between two iterations and make the generated samples located in a local area of the whole state space.

3.2 Action Preferences with Linear Function Approximation

Original DPP is not a reinforcement learning algorithm since it is only applicable to problems with discrete states and prior knowledge about the underlying model. In order to extend DPP to model-free reinforcement Learning of large scale problems with continuous states $\boldsymbol{s} \in \mathcal{S}$, Sampling-based Approximate Dynamic Policy Programming (SADPP),L is proposed in [36]. Defining the n-th state-action pair from a set of N samples as $\boldsymbol{x}_n = [\boldsymbol{s}_n, \boldsymbol{a}_n]_{n=1:N}$, SADPP approximates action preferences function via Linear Function Approximation (LFA): $\hat{\Psi}_k(\boldsymbol{x}_n) = \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{\theta}_k$ where $\boldsymbol{\phi}(\boldsymbol{x}_n)$ denotes the $M \times 1$ output vector of m basis functions, $[\varphi_1(\boldsymbol{x}_n), ..., \varphi_M(\boldsymbol{x}_n)]^{\mathrm{T}}$ where $\varphi_i : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a bounded real value function, and $\boldsymbol{\theta}_k$ is the corresponding $M \times 1$ weight vector. There are many types of basis functions for function approximation, e.g. the Radial Basis Function (RBF) is defined as:

$$\varphi(\boldsymbol{x}) = \exp\left(- \frac{\|\boldsymbol{x} - \boldsymbol{c}\|_2^2}{\rho^2} \right), \tag{3.18}$$

$\boldsymbol{c}$ is the center of the RBF in the state-action space and ρ is a bandwidth parameter.

Using LFA, the DPP operator is approximated by finding a new weight vector $\boldsymbol{\theta}_{k+1}$ that projects $\mathcal{O}\hat{\Psi}_k$ on the column space spanned by the vector of basis functions $\boldsymbol{\Phi} = \left[\boldsymbol{\phi}(\boldsymbol{x}_1), ..., \boldsymbol{\phi}(\boldsymbol{x}_N)\right]^{\mathrm{T}}$. The loss function for LFA to minimize is

Algorithm 3 Sampling-based Approximate Dynamic Policy Programming

Require: η, γ, σ, K and N

initialize weights vector $\boldsymbol{\theta}_0$

for $k = 0, 1, ..., K-1$ **do**

  generate samples $\{(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_n^{'})\}_{n=1:N}$ from distribution μ

  for each $n = 1, 2, 3, ..., N$ **do**

    for each $\boldsymbol{a} \in \mathcal{A}$ **do**

      $\hat{\Psi}_k(\boldsymbol{s}_n, \boldsymbol{a}) = \boldsymbol{\theta}_k^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{s}_n, \boldsymbol{a})$

      $\hat{\Psi}_k(\boldsymbol{s}_n^{'}, \boldsymbol{a}) = \boldsymbol{\theta}_k^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{s}_n^{'}, \boldsymbol{a})$

    $\mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}_n) = \sum_{\boldsymbol{a} \in \mathcal{A}} \frac{\exp\left(\eta \hat{\Psi}_k(\boldsymbol{s}_n, \boldsymbol{a})\right) \hat{\Psi}_k(\boldsymbol{s}_n, \boldsymbol{a})}{\sum_{\boldsymbol{b} \in \mathcal{A}} \exp\left(\eta \hat{\Psi}_k(\boldsymbol{s}_n, \boldsymbol{b})\right)}$

    $\mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}_n^{'}) = \sum_{\boldsymbol{a} \in \mathcal{A}} \frac{\exp\left(\eta \hat{\Psi}_k(\boldsymbol{s}_n^{'}, \boldsymbol{a})\right) \hat{\Psi}_k(\boldsymbol{s}_n^{'}, \boldsymbol{a})}{\sum_{\boldsymbol{b} \in \mathcal{A}} \exp\left(\eta \hat{\Psi}_k(\boldsymbol{s}_n^{'}, \boldsymbol{b})\right)}$

    $\mathcal{O}\hat{\Psi}_k(\boldsymbol{s}_n, \boldsymbol{a}_n) = \boldsymbol{\phi}(\boldsymbol{s}_n, \boldsymbol{a}_n)^{\mathrm{T}} \boldsymbol{\theta} + \mathcal{R}(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_n^{'}) + \gamma \mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}_n^{'}) - \mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}_n)$

  $\boldsymbol{A} = \sum_{n=1:N} \boldsymbol{\phi}(\boldsymbol{s}_n, \boldsymbol{a}_n) \boldsymbol{\phi}^{\mathrm{T}}(\boldsymbol{s}_n, \boldsymbol{a}_n) + \sigma^2 N \boldsymbol{I}$

  $\boldsymbol{B} = \sum_{n=1:N} \mathcal{O}\hat{\Psi}_k(\boldsymbol{s}_n, \boldsymbol{a}_n) \boldsymbol{\phi}(\boldsymbol{s}_n, \boldsymbol{a}_n)$

  $\boldsymbol{\theta}_{k+1} = \boldsymbol{A}^{-1} \boldsymbol{B}$

return $\boldsymbol{\theta}_K$

defined by:

$$J(\boldsymbol{\theta}; \hat{\Psi}) \triangleq \|\boldsymbol{\Phi}\boldsymbol{\theta} - \mathcal{O}\hat{\boldsymbol{\Psi}}_k\|_{2,\mu}^2 \tag{3.19}$$

where $\mathcal{O}\hat{\boldsymbol{\Psi}}_t$ is $N \times 1$ matrix with elements $\mathcal{O}\hat{\Psi}_t(\boldsymbol{s}, \boldsymbol{a})$ following Eq. (3.17) and μ is a probability measure on $\mathcal{S} \times \mathcal{A}$.

Applying least-squares regression to get the optimal $\boldsymbol{\theta}$ that minimizes $J(\boldsymbol{\theta}; \hat{\Psi})$, an orthogonal projection is operated to project $\mathcal{O}\hat{\boldsymbol{\Psi}}_k$ to the space spanned by $\boldsymbol{\Phi}$ by multiplying $\boldsymbol{\Phi}(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}$ following [65]:

$$\boldsymbol{\Phi}\boldsymbol{\theta} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathcal{O}\hat{\boldsymbol{\Psi}}_k \tag{3.20}$$

The update of SADPP therefore follows:

$$\boldsymbol{\theta}_{k+1} = \arg\min J(\boldsymbol{\theta}; \hat{\Psi}) = \left[\mathbb{E}(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi})\right]^{-1} \mathbb{E}(\boldsymbol{\Phi}^{\mathrm{T}}\mathcal{O}\hat{\boldsymbol{\Psi}}_k). \tag{3.21}$$

where the expectation is taken w.r.t. $(\boldsymbol{x}, \boldsymbol{a}) \sim \mu$. Since it is infeasible to compute $\mathcal{O}\hat{\boldsymbol{\Psi}}_k$) for all states and actions in large scale problems, the least-squares solution can be estimated by N i.i.d. samples $\boldsymbol{x}_n = [\boldsymbol{s}_n, \boldsymbol{a}_n]_{n=1:N}$ drawn from the distribution μ following the empirical loss:

$$\tilde{J}(\boldsymbol{\theta}; \hat{\Psi}) \triangleq \frac{1}{N} \sum_{n=1}^{N} \left(\boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}} \boldsymbol{\theta} - \mathcal{O}\hat{\Psi}_k \right)^2 + \sigma^2 \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}, \tag{3.22}$$

σ is a regularization term to avoid over-fitting due to limited sample quantity. The least-square solution of SADPP therefore is:

$$\boldsymbol{\theta}_{k+1} = [\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} + \sigma^2 \mathbf{I}]^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \mathcal{O}\hat{\boldsymbol{\Psi}}_k. \tag{3.23}$$

The pseudo code of SADPP is showed in Algorithm 3.

3.3 SADPP with Sample Reuse

In the studies of DPP [36–38], SADPP with linear function approximate is only applied in toy experiments as a supplement of theory. It does not have a exploration policy but only generates samples from a distributution μ. In this section, we combines SADPP and sample reuse to achieve a rollout based algorithm with exploration.

With sample reuse, the samples are obtained by rollout based interaction with the robot in each iteration and will be reused in the future learning. In the k-th iteration, the data are generated under the current policy π^k and defined as:

$$\mathcal{D}^{\pi^k} = \{T_1^{\pi^k}, T_2^{\pi^k}, ..., T_M^{\pi^k}\}. \tag{3.24}$$

It contains M rollout trajectories with totally $M \times N$ samples:

$$T_m^{\pi^k} = \{(\boldsymbol{s}_{m,n}^{\pi^k}, \boldsymbol{a}_{m,n}^{\pi^k}, \boldsymbol{s}_{m,n}^{\prime\pi^k})\}_{n=1:N} \tag{3.25}$$

In order to achieve an exploration based learning, at the beginning of the k-th iteration, SADPP obtains data by a soft-max exploration:

$$\pi_{\text{explore}}(\boldsymbol{a}|\boldsymbol{s}) = \frac{\exp(\eta_{\text{explore}} \hat{\Psi}_k(\boldsymbol{s}, \boldsymbol{a}))}{\sum_{\boldsymbol{a}' \in \mathcal{A}} \exp(\eta_{\text{explore}} \hat{\Psi}_k(\boldsymbol{s}, \boldsymbol{a}'))} \tag{3.26}$$

Algorithm 4 SADPP with Sample Reuse

Require: $\eta, \eta_{explore}, \gamma, \sigma, K, M$ and N

initialize weights vector $\boldsymbol{\theta}_0$

for $k = 0, 1, ..., K-1$ **do**

  generate samples $\mathcal{D}^{\pi^k}$ by following π_{explore}

  for each $\left\{(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}, \boldsymbol{s}'^i_{m,n})\right\}_{i=0:k,\ m=1:M,\ n=1:N}$ **do**

    for each $\boldsymbol{a} \in \mathcal{A}$ **do**

      $\hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}) = \boldsymbol{\theta}_k^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{s}^i_{m,n}, \boldsymbol{a})$

      $\hat{\Psi}_k(\boldsymbol{s}'^i_{m,n}, \boldsymbol{a}) = \boldsymbol{\theta}_k^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{s}'^i_{m,n}, \boldsymbol{a})$

    $\mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}^i_{m,n}) = \sum_{\boldsymbol{a}\in\mathcal{A}} \frac{\exp(\eta \hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a})) \hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a})}{\sum_{\boldsymbol{b}\in\mathcal{A}} \exp(\eta \hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{b}))}$

    $\mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}'^i_{m,n}) = \sum_{\boldsymbol{a}\in\mathcal{A}} \frac{\exp(\eta \hat{\Psi}_k(\boldsymbol{s}'^i_{m,n}, \boldsymbol{a})) \hat{\Psi}_k(\boldsymbol{s}'^i_{m,n}, \boldsymbol{a})}{\sum_{\boldsymbol{b}\in\mathcal{A}} \exp(\eta \hat{\Psi}_k(\boldsymbol{s}'^i_{m,n}, \boldsymbol{b}))}$

    $\boldsymbol{A} = \sum_{n=1:N} \boldsymbol{\phi}(\boldsymbol{s}_n, \boldsymbol{a}_n) \boldsymbol{\phi}^{\mathrm{T}}(\boldsymbol{s}_n, \boldsymbol{a}_n) + \sigma^2 N \boldsymbol{I}$

    $\boldsymbol{B} = \sum_{n=1:N} \mathcal{O}\hat{\Psi}_k(\boldsymbol{s}_n, \boldsymbol{a}_n) \boldsymbol{\phi}(\boldsymbol{s}_n, \boldsymbol{a}_n)$

    $\mathcal{O}\hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}) = \boldsymbol{\theta}'^{\mathrm{T}} \boldsymbol{\phi}'(s^i_{m,n}, \boldsymbol{a}^i_{m,n}) + \mathcal{R}(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}, \boldsymbol{s}'^i_{m,n}) + \gamma \mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}'^i_{m,n}) - \mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}^i_{m,n})$

  $\boldsymbol{A} = \sum_{i=0:k, m=1:M, n=1:N} \Phi(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}) \Phi^{\mathrm{T}}(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}) + \sigma^2 N \boldsymbol{I}$

  $\boldsymbol{B} = \sum_{i=0:k, m=1:M, n=1:N} \mathcal{O}\hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}) \Phi(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n})$

  $\boldsymbol{\theta}_{k+1} = \boldsymbol{A}^{-1} \boldsymbol{B}$

return $\boldsymbol{\theta}_K$

where the parameter $\eta_{explore}$ is the temperature to control the randomness of the soft-max function during exploration. The pseudo code of SADPP with sample reuse is showed in Algorithm 4.

In this section, SADPP is applied in pendulum swing up with comparison of Least Square Policy Iteration (LSPI) [12, 13]. In the simulation of pendulum swing up shown in Fig. 3.1, the pendulum mass $m = 0.8kg$, the bar's length $l = 1.5m$, gravity $g = 9.8m/s^2$ and dissipation $d = 0.3$, the time step $\Delta t = 0.1s$. The control state is $[\ \theta, \dot{\theta}\ ]$ where the pendulum angle $\theta \in [-\pi, \pi)\ rad$ and the angle velocity $\dot{\theta} \in [-3\pi, 3\pi]\ rad/s$. The action is the torque $\tau \in \{-1, 0, 1\}\ N \cdot m$. The reward function is defined as $\mathcal{R} = -(\theta - 0)^2$ as the swing up angle is 0.

For both SADPP and LSPI, the gaussian function is used in linear function

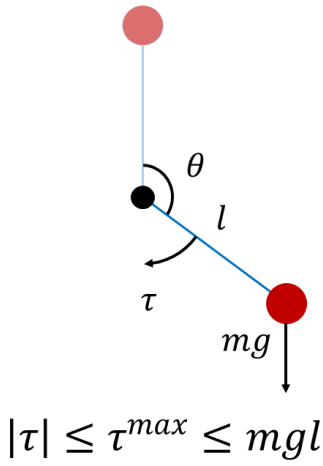


Figure 3.1: The simulation of pendulum swing up.

approximate as basis function. In this experiment, 11 basis functions in one state dimension, and 3 basis functions for each action are used. The total number of basis functions is $11^2 \times 3 = 363$. The parameters in SADPP is set as: $\alpha = 0.01, \gamma = 0.95, \eta = \eta_{explore} = 0.01$. In each rollout of one iteration, the pendulum state is initialized to a randomly perturbed state very close to $[-\pi, 0]$. We define the reward as the accumulated $\mathcal{R}$ from the initial state to the final state in one rollout following the current greedy policy. The stop criteria is reward $\geqslant -300$. Fig.3.2 shows the learning result (all data is the average value of 10 times experiments).

According to the result, due to the lack of smoothness in the policy update with limited number of samples, LSPI was easy to diverge while SADPP has a more stable learning and converge to a larger reward. To converge to the same high reward, LSPI needed double times length of rollout trajectory than SADPP. The effect of Kullback–Leibler divergence in the smooth policy update is also analysed in this simulation task. Figures 3.3 and 3.4 show the comparison of SADPP and LSPI in both approximate value function V and policy map according to the largest value action in the approximate action preferences and Q function during one time learning. Initialized by RBFs with random weights in the 0-th iteration with limited number of samples generated per iteration, it is difficult for LSPI to learn a stable approximate value function. The over large policy update results

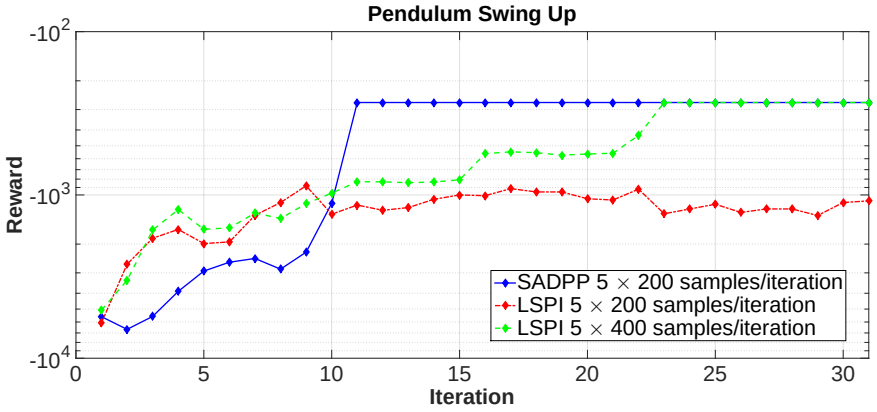


Figure 3.2: Comparison between SADPP and LSPI with different number of samples in pendulum swinging up task.

to a messy policy map that could not achieve the swing up task. On the other hand, under the restriction of the Kullback–Leibler divergence, SADPP is able to smoothly update the policy map to achieve the swing up task under the same experiment setting. This result indicate the importance of smooth policy update in a successful learning with limited number of samples.

The convergence of DPP has been theoretically proven according to [36–38], while DPP with function approximation has also been proven to have a better asymptotic performance-loss bound than other approximate policy iteration and approximate value iteration approaches with the assumption that a generative model of an MDP is obtained to generate the next sample for all state-action pairs [66, 67]. The results of toy simulation above indicate that SADPP with sample reuse outperformed LSPI in the presence of function approximation and limited sampling budget per iteration. The positive effect of smooth policy update on sampling policy which results to a more stable learning with limited number of samples is also demonstrated. However, besides one application in inverse reinforcement learning [68], DPP/SADPP is still unable to fully utilize its smooth policy update in real world robot control due to the intractable computational complexity with exponentially growing size of RBFs $\boldsymbol{\phi}$ in high dimensional systems.

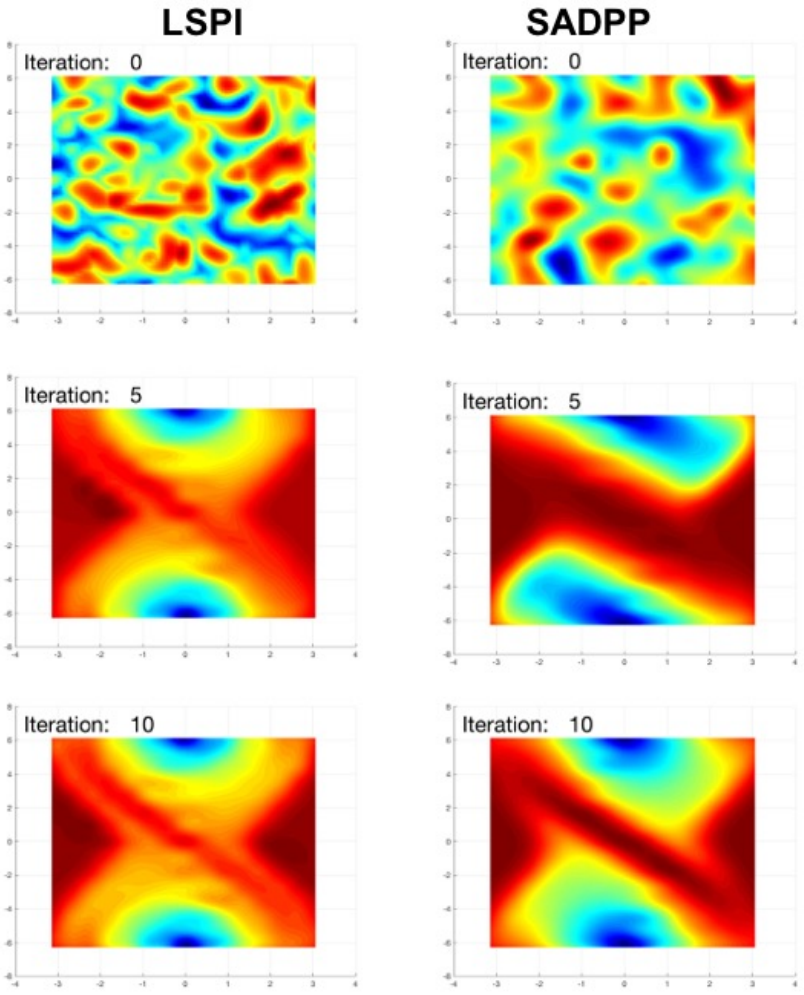


Figure 3.3: Comparison of approximate value function V between SADPP (right) and LSPI (left), both with 5×200 samples per iteration. X axis is the state of angle, Y axis is the state of angle velocity.

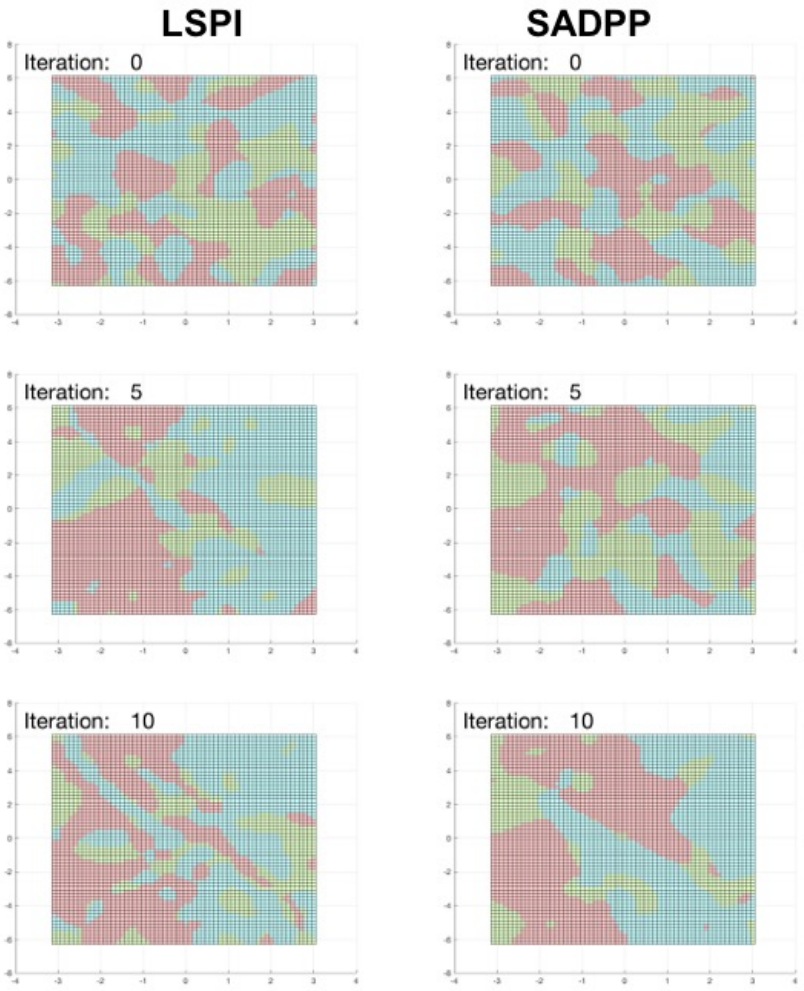


Figure 3.4: Comparison of policy map generate from approximate Q function and action preferences function between SADPP (right) and LSPI (left), both with 5×200 samples per iteration. X axis is the state of angle, Y axis is the state of angle velocity. Red, green and blue areas represent action -1, 0 and 1 torque respectively.

3.4 Learn Global Value Function via Local Samples Space

Let's turn back to the current limitations of reinforcement learning in robot control domain mentioned in Chapter 1. The value function approach is able to learn a global value function but be easy to diverge with insufficient samples. Furthermore, the computational complexity in high dimensional systems is intractable. On the other hand, the policy search learns a parameterized policy rather than value function. The optimal is searched in a local area to reduce the complexity while selecting and initializing the parameterized policy suitably always need extra prior knowledge of the models and tasks.

A new reinforcement learning framework is proposed in this thesis to take advantages of both the value function approach and the policy search. The main idea comes from the nature of DPP. Since the over large policy update can be restrained by Kullback–Leibler divergence, we can iteratively approximate the global value function via samples in a local area driven by a smoothly updated exploration policy. Thus, even the real world samples are insufficient, we could focus on accurately approximate the value function in a local area defined by generated samples. This strategy is similar to the policy search but no parameterized policy is required. Furthermore, the foundation fo solving the curse of dimensionality in high dimensional systems is provided by the smooth policy update. Once the learning progress is stabilized locally, other machine learning algorithms could be easily applied to the corresponding area to efficiently approximate the value function. Intuitively, this framework update the value function instead of the parameterized policy in a local area based on rollout exploration while the value function is able to be efficiently approximated via other machine learning tricks.

In the following two chapters, two examples of the proposed framework will be introduced: Local Update Dynamic Policy Programming (LUDPP) and Kernel Dynamic Policy Programming (KDPP). They utilize different machine learning methods, nearest neighbor search (NNS) [39] and kernel trick [40] respectively, but all be practical to MDP with high dimensional state space, especially when the samples are insufficient.

4 Local Update Dynamic Policy Programming

In this chapter, a new value function approach based reinforcement learning algorithm, Local Update Dynamic Policy Programming (LUDPP) is proposed to work in the high dimensional state-action space with much reduced computational complexity. LUDPP exploits the nature of the smooth policy update inherited by Dynamic Policy Programming (DPP) [36–38]. It is able to update its value function in a far smaller local area selected by the current samples of state and action every iteration and therefore considerably reduce the computational complexity in both time and space.

4.1 Proposed Method

The idea of locally updating the approximated value function in reinforcement learning algorithms is inspired by the locally updated Gaussian process regression [69] and the locally weighted learning in control [70] which achieve efficiency by limiting the calculation in a local area rather than a global space. LUDPP limits the updating of value function (action preferences function) in a local area instead of the global state-action space to simplify the calculation every iteration. LUDPP is naturally suitable to this idea because of its sampling efficiency and stable learning while such a local update should not be recommended in general reinforcement learning algorithms since the approximated value function becomes cruder: the distribution of samples will be generated following this inaccurate value function and turns to a cruder value function.

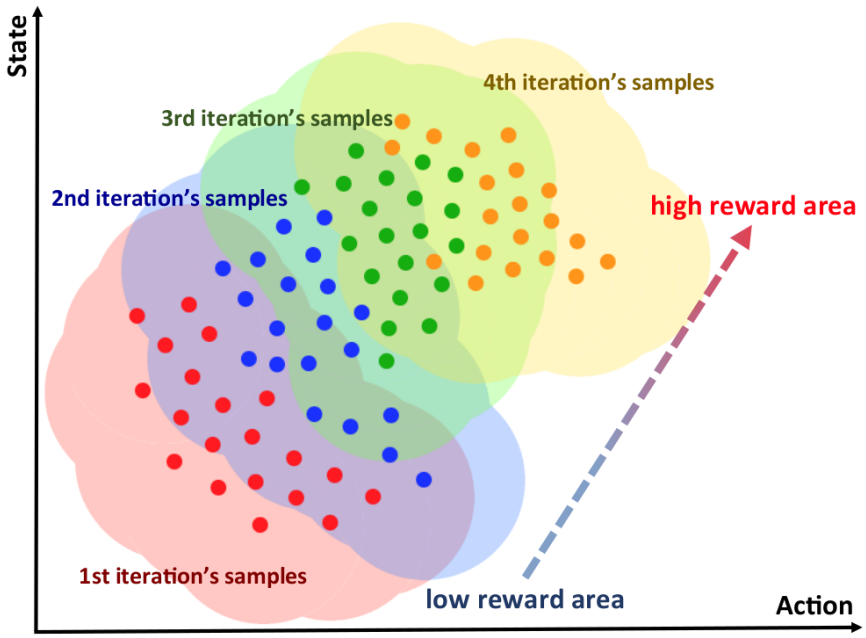


Figure 4.1: The framework of local update dynamic policy programming.

4.1.1 Local Update of Value Function

According to the Chapter 3, DPP is suitable to be combined with the local update as its control policy is updated smoothly by adding Kullback-Leibler divergence between the current and previous policy in the reward function. LUDPP exploits the nature of smooth policy update from DPP and be able to keep a stable local update of value function. Figure 4.1 illustrates the framework of LUDPP. The samples in each iteration are generated following the current value function. A local area covering all current samples (translucent area) is selected to locally update the value function. Thanks to Kullback-Leibler divergence, LUDPP generates samples smoothly from the low reward area to the high reward area. The large overlapped local area between the previous and current iteration lead to a accurate approximation of the value function since it contributes to the learning continuously.

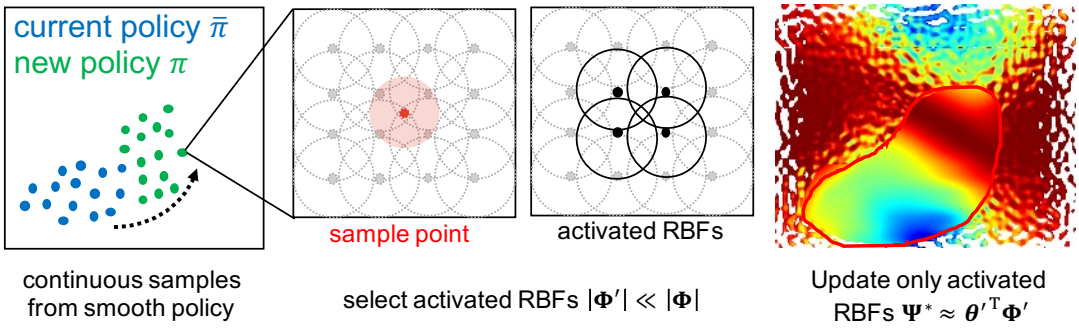


Figure 4.2: The principle of applying Nearest Neighbor Search: calculating/updating the weights of activated RBFs instead of all is more tractable.

4.1.2 Sample Reuse and Exploration

Samples are generated by roll-out-based interaction in LUDPP. Since the weights vector will be re-constructed every iteration according to Eq. (3.23), sample reuse is applied to efficiently utilize the limited learning samples. We define the data generated in the k-th iteration following control policy π^k by:

$$\mathcal{D}^k = \{T_1^k, T_2^k, ..., T_M^k\} \tag{4.1}$$

which contains M roll-out trajectories and each trajectory contains N samples:

$$T_m^k = \left\{(\boldsymbol{s}_{m,n}^k, \boldsymbol{a}_{m,n}^k, \boldsymbol{s}_{m,n}'^k)\right\}_{n=1:N}. \tag{4.2}$$

A soft-max exploration policy is used for the exploration in LUDPP. In the k-th iteration, the exploration policy $\pi_{explore}$ following Eq. (3.12):

$$\pi_{\text{explore}}(\boldsymbol{a}|\boldsymbol{s}) = \frac{\exp(\eta_{\text{explore}}\hat{\Psi}_k(\boldsymbol{s},\boldsymbol{a}))}{\sum_{\boldsymbol{a}'\in\mathcal{A}}\exp(\eta_{\text{explore}}\hat{\Psi}_k(\boldsymbol{s},\boldsymbol{a}'))} \tag{4.3}$$

The temperature η_{explore} controls the randomness of the soft-max function.

4.1.3 LUDPP with RBFs

In this subsection, the detail of LUDPP using RBFs is introduced as one implementation. the RBFs basis function ϕ_{RBF} is defined follows Eq. (3.18):

$$\phi_{RBF}(\boldsymbol{s}, \boldsymbol{a}) = \exp\left(- \frac{\|\boldsymbol{x} - \boldsymbol{c}\|_2^2}{\xi^2} \right) \tag{4.4}$$

where $\boldsymbol{x} = [\boldsymbol{s}, \boldsymbol{a}]$ is the input vector, $\boldsymbol{c}$ is the center of this RBF in the state-action space and ξ is a bandwidth parameter. Since basis function whose center has closer Euclidean distance to the $\boldsymbol{x}$ contributes more to the approximation of the value function in this point, it is convenient to applying Nearest Neighbor Search (NNS) [39, 71] to search the active basis function close enough to the samples in current iteration and determine the local area in Fig. 4.1. With only the weights of active basis functions being updated, this local update is assumed to be sufficient to represent the approximated value function. Figure 4.2 shows the main principle of this strategy. There are many methods for NNS, e.g., K-Dimensional Tree [72] and Locality-Sensitive Hashing [73]. Here we abstract the process of NNS to a function:

$$(\boldsymbol{\theta}^{'}, \boldsymbol{\phi}^{'}, \boldsymbol{i}_{active}) = NNS(\boldsymbol{\theta}_k, \boldsymbol{\phi}, \mathcal{D}^k, P_{NNS}). \tag{4.5}$$

The input parameter $\boldsymbol{\theta}_k$ is the weights vector in the k-th iteration, $\boldsymbol{\phi}$ is the original basis functions vector, $\mathcal{D}^k$ is the generated data, P_{NNS} controls the number of nearest basis functions to search for each samples in $\mathcal{D}^k$. $NNS()$ searches the active basis function $\boldsymbol{\phi}^{'}$ and the corresponding weights vector $\boldsymbol{\theta}^{'}$ and saves the index of active basis functions in $\boldsymbol{i}_{active}$.

LUDPP has an internal smoothness in policy update following the principle of DPP. The exploration policy $\pi_{explore}$ is defined to be smoothly updated as the action preferences $\hat{\Psi}_k$'s update is smooth. With this smooth exploration policy, the generated samples are assumed to move to the high-reward area of state-action space smoothly. This feature may reduce the sampling's bias during learning, but also fulfills the requirement of LUDPP's local update: this partial update highly requires no steep update of policy occurs in order to make the locally active area of basis functions moves smoothly so that the active basis functions make strong effect continuously in the next iterations.

Algorithm 5 LUDPP using RBFs

Require: $\boldsymbol{\phi}, \eta_{\text{explore}}, \eta, \gamma, \sigma, K, M, N$ and P_{NNS}

1: initialize weights vector $\boldsymbol{\theta}_0$
2: **for** $k = 0, 1, ..., K-1$ **do**
3: generate samples $\mathcal{D}^k$ by following π_{explore} following Eq. (4.3)
4: $(\boldsymbol{\theta}^{'}, \boldsymbol{\phi}^{'}, \boldsymbol{i}_{active}) = NNS(\boldsymbol{\phi}, \boldsymbol{\theta}_k, \mathcal{D}^k, P_{NNS})$
5: **for each** $\left\{(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}, \boldsymbol{s}^{'i}_{m,n})\right\}_{i=0:k,\ m=1:M,\ n=1:N}$ **do**
6: **for each** $\boldsymbol{a} \in \mathcal{A}$ **do**
7: $\hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}) = \boldsymbol{\theta}^{'\mathrm{T}}_k \boldsymbol{\phi}^{'}(\boldsymbol{s}^i_{m,n}, \boldsymbol{a})$
8: $\hat{\Psi}_k(\boldsymbol{s}^{'i}_{m,n}, \boldsymbol{a}) = \boldsymbol{\theta}^{'\mathrm{T}}_k \boldsymbol{\phi}^{'}(\boldsymbol{s}^{'i}_{m,n}, \boldsymbol{a})$
9: $\mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}^i_{m,n}) = \sum_{\boldsymbol{a}\in\mathcal{A}} \frac{\exp\left(\eta \hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a})\right) \hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a})}{\sum_{\boldsymbol{b}\in\mathcal{A}} \exp\left(\eta \hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{b})\right)}$
10: $\mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}^{'i}_{m,n}) = \sum_{\boldsymbol{a}\in\mathcal{A}} \frac{\exp\left(\eta \hat{\Psi}_k(\boldsymbol{s}^{'i}_{m,n}, \boldsymbol{a})\right) \hat{\Psi}_k(\boldsymbol{s}^{'i}_{m,n}, \boldsymbol{a})}{\sum_{\boldsymbol{b}\in\mathcal{A}} \exp\left(\eta \hat{\Psi}_k(\boldsymbol{s}^{'i}_{m,n}, \boldsymbol{b})\right)}$
11: $\mathcal{O}\hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}) = \boldsymbol{\theta}^{'\mathrm{T}} \boldsymbol{\phi}^{'}(s^i_{m,n}, \boldsymbol{a}^i_{m,n}) + \mathcal{R}(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}, \boldsymbol{s}^{'i}_{m,n}) + \gamma \mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}^{'i}_{m,n}) - \mathcal{M}_\eta \hat{\Psi}_k(\boldsymbol{s}^i_{m,n})$
12: $\boldsymbol{A} = \sum_{i=0:k, m=1:M, n=1:N} \boldsymbol{\phi}^{'}(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}) \boldsymbol{\phi}^{'\mathrm{T}}(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}) + \sigma^2 N \boldsymbol{I}$
13: $\boldsymbol{B} = \sum_{i=0:k, m=1:M, n=1:N} \mathcal{O}\hat{\Psi}_k(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n}) \boldsymbol{\phi}^{'}(\boldsymbol{s}^i_{m,n}, \boldsymbol{a}^i_{m,n})$
14: $\boldsymbol{\theta}_{k+1}(\boldsymbol{i}_{active}) = \boldsymbol{A}^{-1} \boldsymbol{B}$
15: return $\boldsymbol{\theta}_K$

Algorithm 5 is the pseudo code of LUDPP using RBFs. After generating samples following the current exploration policy $\pi_{explore}$ in the k-th iteration, the active RBFs are calculated by Eq. (4.5) in line 4. From lines 5 to 13, $\mathcal{O}\hat{\Psi}_k$ among all samples is calculated. Lines 14 to 16 calculate the new weights vector locally following Eq. (3.23). Notice line 16 calls a matrix inversion with time complexity $\mathcal{O}(n^3)$ [74] when the matrix size is $(n \times n)$. This local update cuts considerably the size of matrices $\boldsymbol{A}$ from $\boldsymbol{\phi} \times \boldsymbol{\phi}$ to $\boldsymbol{\phi}^{'} \times \boldsymbol{\phi}^{'}$ in some high-dimensional problems (e.g., in the case of matrices with several millions rows and columns). If P_{NNS} is suitable to find a small local RBFs vector that $|\boldsymbol{\phi}| \gg |\boldsymbol{\phi}^{'}|$, the local update will reduce the computation complexity of updating weights vector greatly.

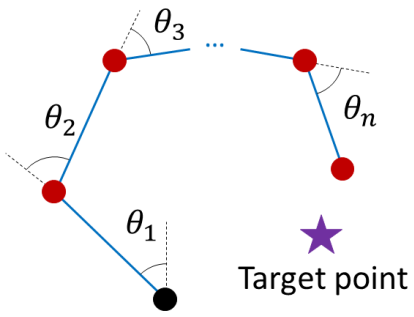


Figure 4.3: The simulation of n DOF manipulator reaching tasks.

4.2 Simulation Results

In this section, LUDPP using RBFs is applied to the simulation of pendulum swing up (shown in Fig.3.1) and n (two to four) DOF manipulator reaching tasks (shown in Fig. 4.3). As one of the most popular and efficient algorithms in this area, LSPI [12, 75, 76] shares the same least squares update law with DPP. The only difference is that LSPI does not consider the Kullback-Leibler divergence between policies to control the smoothness of policy update. Therefore we combine NNS with LSPI to achieve a Local Update Least Square Policy Iteration (LULSPI) using RBFs as comparison in this simulation.

4.2.1 Pendulum Swing Up

In this simulation, the parameters of pendulum swing up were set as: mass $m = 0.8\,\mathrm{kg}$, the bar's length $l = 1.5\,\mathrm{m}$, gravity $g = 9.8\,\mathrm{m/s^2}$, time step $\Delta t = 0.1\,\mathrm{s}$ and the parameter of dissipative loss of energy (e.g., friction in the elements of the pendulum) $d = 0.3$. The continuous state is $[\theta,\, \dot{\theta}]^{\mathrm{T}}$ where $\theta \in [-\pi, \pi)\,\mathrm{rad}$ is the angle between the current pendulum position and the swing up position, $\dot{\theta} \in [-3\pi, 3\pi]\,\mathrm{rad/s}$ is the angle velocity. The discrete action is torque $\tau \in \{-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2\}\,\mathrm{N \cdot m}$. The reward function is defined as $\mathcal{R} =$

$-\theta^2$ to give high reward or low cost when the pendulum is close to the swing up position. Defining the swing up position as 12 o'clock position, $\theta = 0\,\text{rad}$, the state was initialized to 6 o'clock position, $\theta = \pi\,\text{rad}$ with 0 angle velocity at the beginning of each roll-out. While the proposed approach allows agent to learn from various initial states and generate good solution, we set one initial state in simulations in order to simplify the task and reduce the requirement of samples and iterations.

For the function approximation, 20 RBFs with $\xi = 0.5$ were set in each dimension of state and 9 RBFs with $\xi = 0.2$ for the action space. 3600 RBFs were arranged in a $20 \times 20 \times 9$ grid over a 3-dimensional state-action space. The parameters of SADPP and LUDPP were set as $\alpha = 0.01, \gamma = 0.95, \eta = 0.0001, \beta = 0.01$. Totally 10×200 samples, 10 trajectories and each one has 200 steps, were generated every iteration in learning. A test roll-out with a greedy control policy was operated 500 steps before each iteration. The learning stopped when the accumulative reward in the last 400 steps in test roll-out was more than 0.5. It requires the pendulum to swing up in the first 100 steps and then kept stable for the next 400 steps. All data was the average over 10 times experiments.

The learning result is shown in Fig. 4.4, both SADPP and LSPI worked well without the local update. SADPP converged to the highest reward with 10×200 samples/iteration. On the other hand, setting P_{NNS} to 100 and 10, LUDPP converged to a close reward with almost same number of iterations compared with SADPP while LULSPI converged to a lower reward of SADPP with several iterations more. When we set $P_{NNS} = 1$, LUDPP was able to converge to a high reward with a bit more iterations while LULSPI could not converge at all. For the number of RBFs, LUDPP averagely updated 30.6% RBFs (1101.46 of 3600) when $P_{NNS} = 1$.

Figure 4.5 shows one example of LUDPP and LULSPI's learning in the first 10 iterations under the view point of value function (action preferences function) update ($P_{NNS} = 1$). The first and second rows show the highest value of action preferences function with the corresponding action among the state-action space in SADPP and LUDPP, respectively. The third and fourth rows show LSPI and LULSPI's value function, i.e., the highest value of Q function with the corresponding action, among the state-action space. In each small figure, the X axis

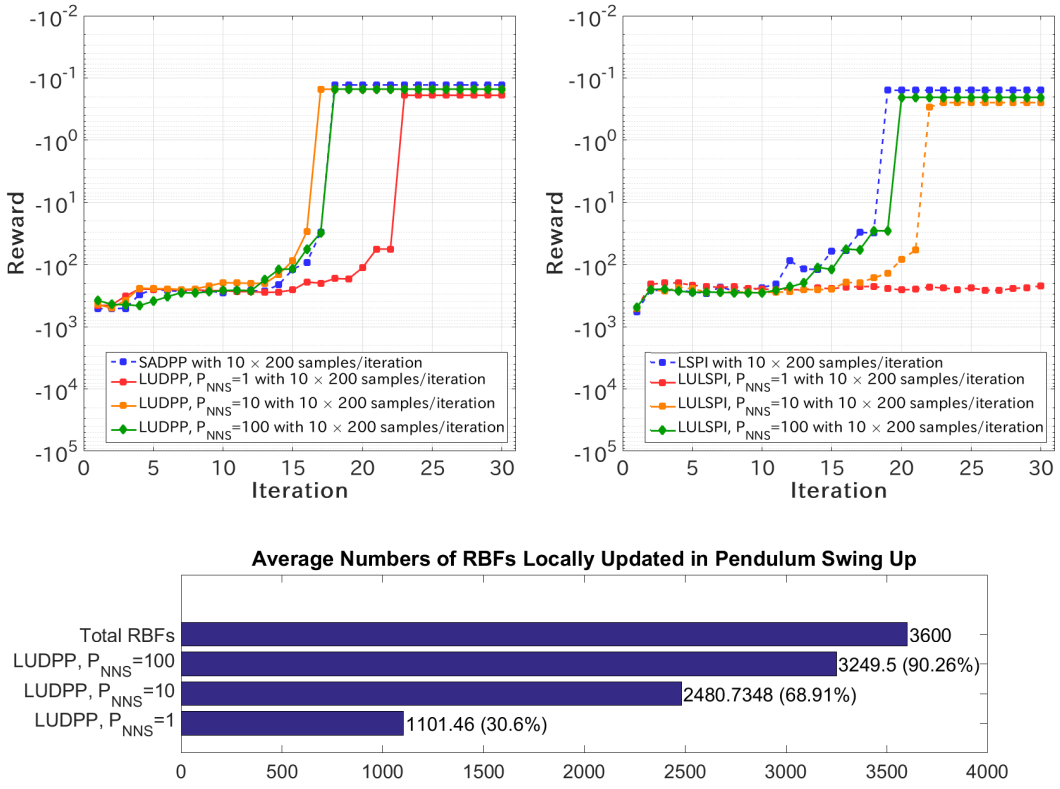


Figure 4.4: The learning result of pendulum swing up. Top: comparison between LUDPP (left) and LULSPI (right). Bottom: the average size of local update RBFs in LUDPP.

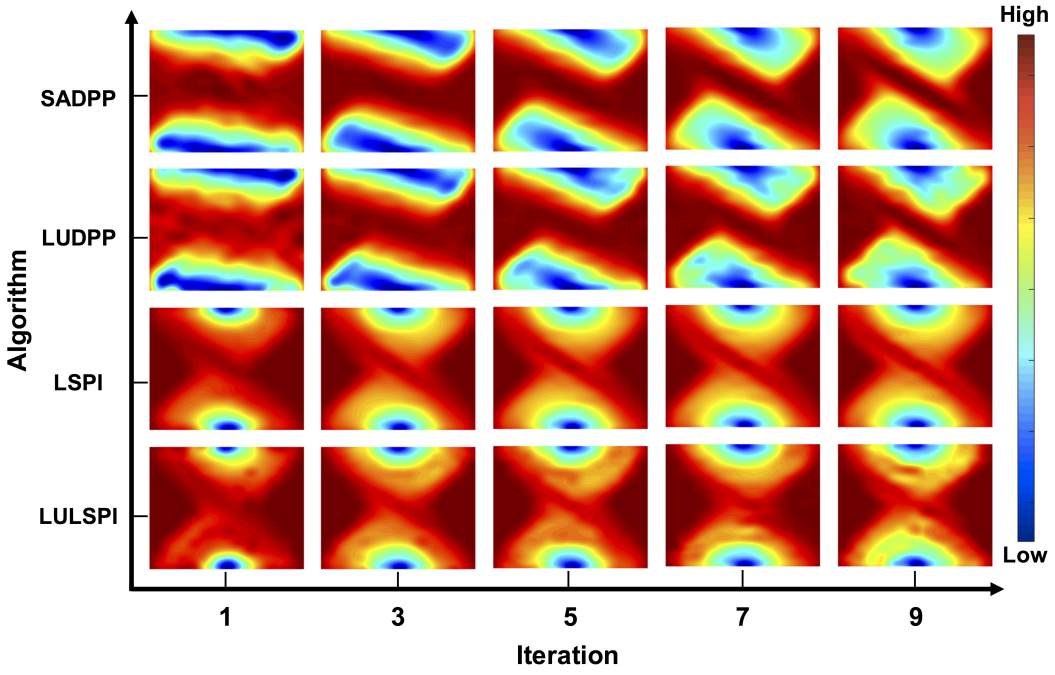


Figure 4.5: One example of the update of value function (action preferences function) in pendulum swing up (the high value area is red and low area is blue).

is the angle, Y axis is the angle velocity. In LUDPP, the crude approximation of local update worked good. LUDPP's smooth policy update kept the local update stable. Even though the action preferences function in the first iteration had several incorrect high value areas, LUDPP finally learned a similar action preferences function of SADPP's. On the other hand, the value function of LULSPI was updated incorrectly compared with LSPI's.

4.2.2 Multiple DOF Manipulator Reaching

In the n DOF manipulator reaching ($n = 2, 3, 4$), we investigate the learning performance of LUDPP in higher state-action space with different setting of parameter P_{NNS}. The continuous state in this task is $[\theta_1, \theta_2, ..., \theta_n]^{\mathrm{T}}$ where $\theta_i \in [-\frac{\pi}{2}, \frac{\pi}{2}]\,\mathrm{rad}$ is the angle of the i-th joint. Discrete actions $[u_1, u_2, ..., u_n]^{\mathrm{T}}$ have three values for each joint: increase/decrease $0.0875\,\mathrm{rad}$ joint angle and maintain the current angle. The first joint is set to the position $[0, 0]$ and the length of one limb between two joints is set to $\frac{1}{n}\,\mathrm{m}$. All joint angles are initialized to 0 at the beginning of each roll-out. The reward function is defined as $\mathcal{R} = -1000 \times ((p_x - X_{target})^2 + (p_y - Y_{target})^2)$ where $[p_x, p_y]$ is the end-effector's position and $X_{target} = 0.6830, Y_{target} = 0$ are the targets to reach. We set the parameter setting of algorithm like pendulum swing up. The stopping criterion is that the reward of the last state in the test roll-out is more than -1. All data was the average over 10 times experiments.

For the function approximation, nine RBFs with $\xi = 0.5$ were set in each dimension of state and three RBFs with $\xi = 0.2$ for the action space. In two DOF manipulator reaching task, totally 729 RBFs were arranged in a $9 \times 9 \times 3 \times 3$ grid over a four dimensional state-action space. LUDPP was trained with 10 samples/iteration and $P_{NNS} = \{1, 10, 50, 100\}$. In three DOF manipulator reaching task, 19683 RBFs were arranged in a $9 \times 9 \times 9 \times 3 \times 3 \times 3$ grid over a six dimensional state-action space. LUDPP was trained with 10 samples/iteration and $P_{NNS} = \{1, 10, 100, 1000\}$. In four DOF manipulator reaching task, the total number of RBFs becomes $9 \times 9 \times 9 \times 9 \times 3 \times 3 \times 3 \times 3 = 531441$ that is intractable to SADPP/LSPI using our computational server. We trained LUDPP with 10 samples/iteration. In order to investigate the effect of different setting of meta parameter P_{NNS}, we set $P_{NNS} \in [1, 1000]$ with the same number of interval.

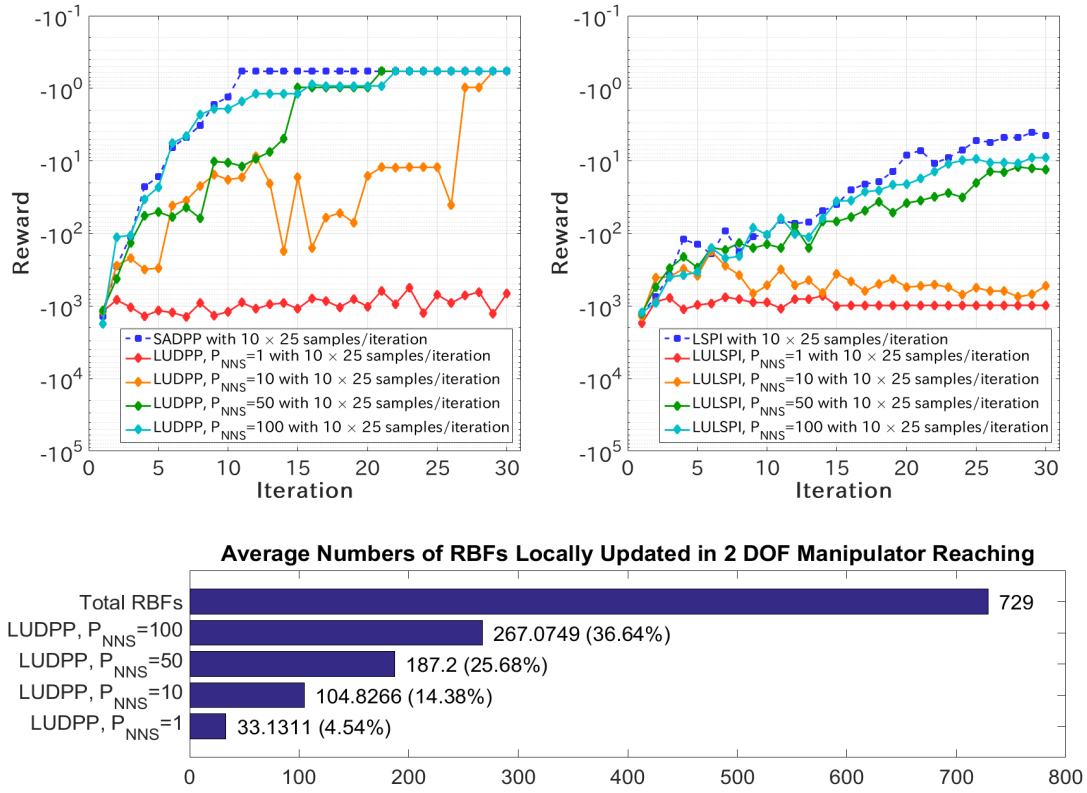


Figure 4.6: The learning result of two DOF manipulator reaching task. Top: comparison between LUDPP (left) and LULSPI (right). Bottom: the average size of local update RBFs in LUDPP.

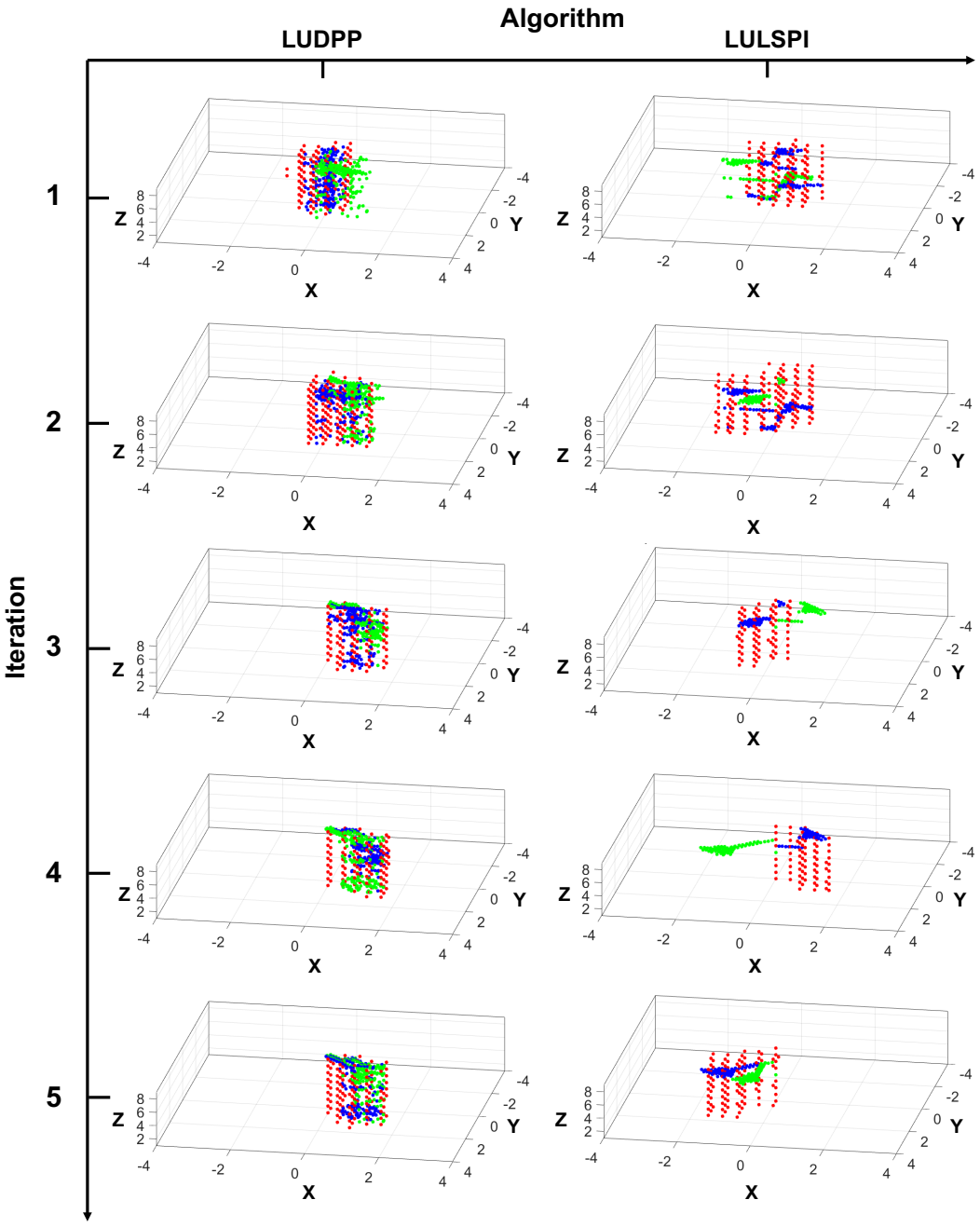


Figure 4.7: Samples and active basis functions in the first 5 iterations of two DOF manipulator reaching with LUDPP (left) and LULSPI (right). ●: Center points of active basis functions; ●: Samples generated in current iteration; ●: Samples generated in next iteration. The X axis and Y axis represent θ_1 and θ_2, respectively and the Z axis represents u_{mix}.

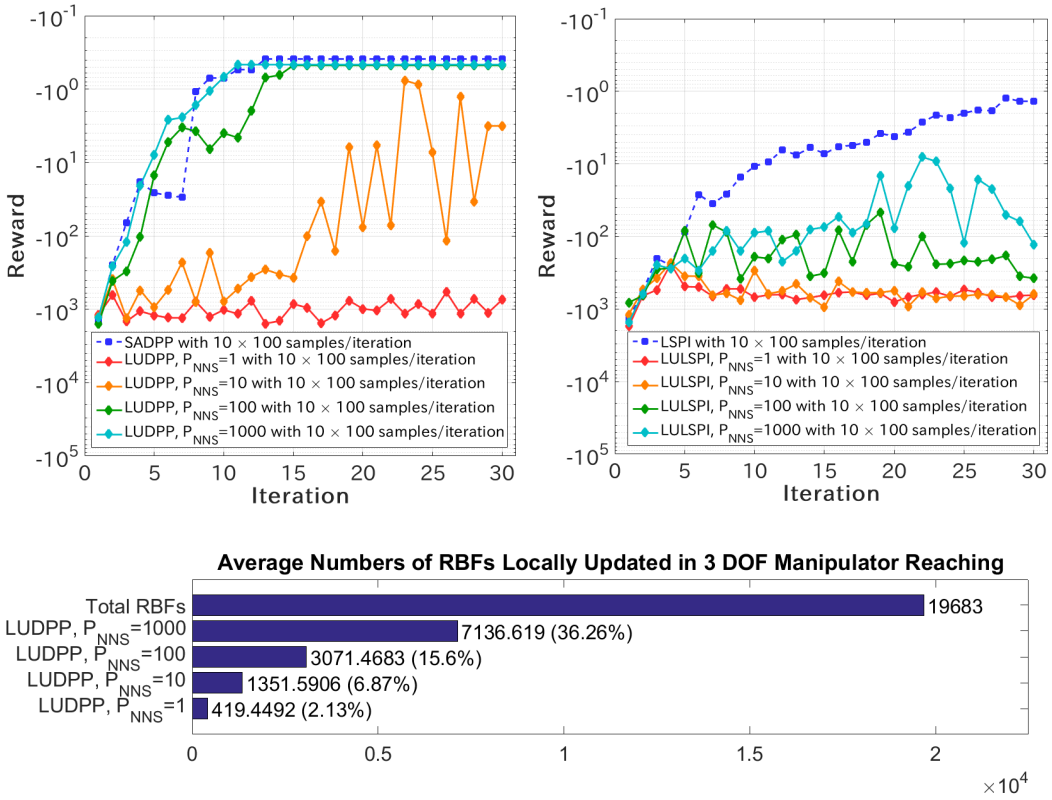


Figure 4.8: The learning result of three DOF manipulator reaching task. Top: comparison between LUDPP (left) and LULSPI (right). Bottom: the average size of local update RBFs in LUDPP.

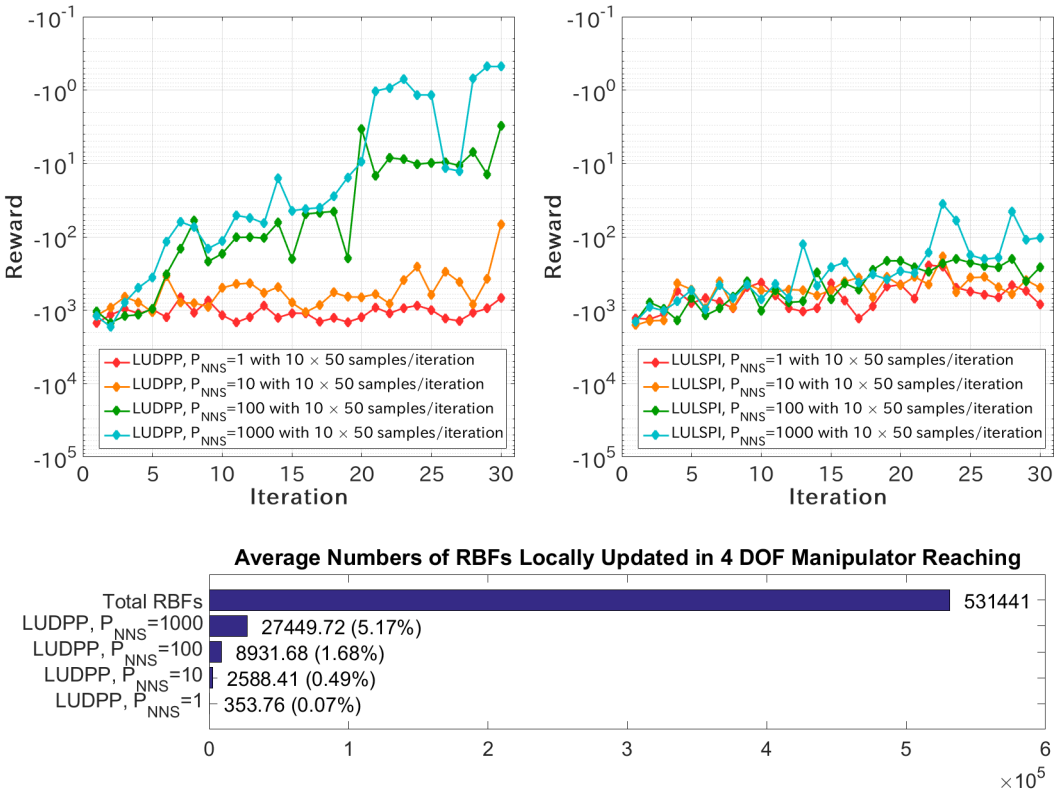


Figure 4.9: The learning result of four DOF manipulator reaching task. Top: comparison between LUDPP (left) and LULSPI (right). Bottom: the average size of local update RBFs in LUDPP.

Figure 4.6 (top) shows the learning results of LUDPP (left) and LULSPI (right) in two DOF manipulator reaching. The average RBFs updated in each iteration is illustrate in Fig. 4.6 (bottom). Because SADPP has a better samples complexity than LSPI, with the same number of samples, SADPP converged to a better reward. Applying the local update, LUDPP was able to converge to a very high reward within 30 iterations' learning with P_{NNS} being set as 10, 50 and 100. It learned to a similar result of SADPP while only update 14.38% ($P_{NNS} = 10$), 25.68% ($P_{NNS} = 50$) and 36.64% ($P_{NNS} = 100$) RBFs. On the other hand, LULSPI resulted in a poor performance.

The four dimensional data of two DOF manipulator reaching ($[\theta_1, \theta_2, u_1, u_2]^{\mathrm{T}}$) was translated to a three dimensional one ($[\theta_1, \theta_2, u_{mix}]^{\mathrm{T}}$) to investigate the the local update of LUDPP from an intuitive viewpoint. We defined u_{mix} with nine discrete actions to represent all nine different setting of $[u_1, u_2]^{\mathrm{T}}$. One example of LUDPP and LULSPI's learning in the first five iterations was shown in Fig. 4.7. The red dots are the active RBFs' center which determined the local area of the whole state-action space. The blue dots are samples in the current iteration. They are used to find the active RBFs by nearest neighbor search. The samples generated in the next iteration were represented by green dots. According to the left column of Fig. 4.7, LUDPP updated its policy smoothly. The samples generated in the next iteration were close to previous samples and therefore be able to utilize the active RBFs efficiently. In contrast, LULSPI generated samples in a steep way.

In three DOF manipulator reaching, LUDPP similarly outperformed LULSPI according to Figs. 4.8. Only 36.26% ($P_{NNS} = 1000$), 15.60% ($P_{NNS} = 100$), 6.87% ($P_{NNS} = 10$) RBFs being updated to converge to a close result of SADPP within 30 iterations in LUDPP. Both LSPI and LULSPI could not converge to a good solution stably.

Due to the limitation of both calculation time and hardware, no SADPP and LSPI was trained in four DOF manipulator reaching with a eight dimensional state-action space and 531441 RBFs. Only LUDPP and LULSPI with 10×50 samples/iteration were trained. Parameter P_{NNS} is set as 1, 10, 100 and 1000. As shown in Fig. 4.9, LUDPP's learning result was improved with the increasing P_{NNS}. Its successful rate of learning is high with 5.17% ($P_{NNS} = 1000$) and

1.68% ($P_{NNS} = 100$) RBFs being updated while LULSPI could not converge at all.

4.3 Real Robot Experiment

In this section, we apply the proposed reinforcement algorithm, LUDPP, to Shadow Dexterous Hand [41], a Pneumatic Artificial Muscle (PAM) driven humanoid robot hand in position reaching task. The related works of applying reinforcement learning to control PAM driven robots are all using policy search algorithms [77, 78], where the prior knowledge of robot systems is obtained by software simulator or human demonstrations. On the other hand, the proposed framework achieves purely model-free and self-exploration reinforcement learning can be used without any prior knowledge of model/simulator/demonstrations.

We first give an introduction of the features and model of Pneumatic Artificial Muscle (PAM) driven robots. Then we go the details of the platform, Shadow Dexterous Hand. Lastly, we show the real experiment of LUDPP in this platform.

4.3.1 Pneumatic Artificial Muscles Driven Robots

Pneumatic Artificial Muscle (PAM) is an attractive device for use as an actuator for a wide variety of robots after having been widely employed in industrial automation. Thanks to its flexible, lightweight, compliant structure that approximates real muscles and high power-to-weight ratio, PAM-driven robot is highly suitable to execute tasks with our nearby living environments in dealing with the several objects and tools as required, physically interact to the humans for supporting their daily activities or rehabilitation purposes (e.g., elastic orthoses [79] and rehabilitation/training exoskeleton systems [80]). Unfortunately, the PAM-driven robot has not only high system dimensionality but also nonlinearities from pressure dynamics, mechanical structure, hysteresis phenomena and mass flow rate [81–83]. It is difficult to accurately model and control by traditional model-free control [84] from intelligent PID [85] to neural networks [86] and fuzzy system [87] approaches.

As one potential solution, reinforcement learning is able to iteratively learn control policies without model. However, applying the policy search algorithms

to PAM-driven robot is limited since the control signal for PAM-driven robot is discrete (to control the valve to inflate, deflate and hold the current pressure) while the initial policy could not be intuitively obtained. Even though the value function approach typically employs discrete control actions and be able to learn without prior model knowledge for initial policy, applying it to control PAM-driven robot is still challenging: the curse of high system dimensionality results to an impractical calculation complexity while the the learning is fragile due to curse of insufficient learning samples. Therefore it is suitable to evaluate the work of this thesis on the PAM-driven robot since the motivation according to our motivation.

The McKibben pneumatic artificial muscle has an internal rubber tube surrounded by a cylindrical mesh braided by inextensible threads [83]. The cylindrical form of PAM is retained by sealing both ends of its two-layered tube by caps that have connectors to supply compressed air. A contraction force is generated by PAM when the long axis shortens due to the inextensible threads by filling compressed air to the tube. With such a unique structure, PAM has high power-to-weight, good flexibility and be suitable to drive robots to interact with humans safely. On the other hand, PAM is difficult to model and control due to the nonlinearities in its structure like pressure dynamics, hysteresis phenomena and the effect of mass flow rate. According to [81–83], a simple equation of PAM's axial tension F follows:

$$F = -(P - P_0)\frac{dV}{dL} \tag{4.6}$$

where P is the internal gas pressure and P_0 is the environment pressure (101.325 kPa), dV and dL are the change of PAM volume and the axial displacement of PAM, respectively. For one governable joint is driven by two antagonistically placed PAMs (denoted by subscript 1 and 2), the torque follows:

$$\tau = F_1 - F_2 = -(P_1 - P_0)\frac{dV_1}{dL_1} + (P_2 - P_0)\frac{dV_2}{dL_2} \tag{4.7}$$

According to Eqs. (4.6) and (4.7), the modeling and controlling of PAM-driven robot by the traditional methods is challenging because of both the difficulty of accurately measuring the volume and length of PAM in real world and the consideration of the moment arm of every muscle depending on joint angles.

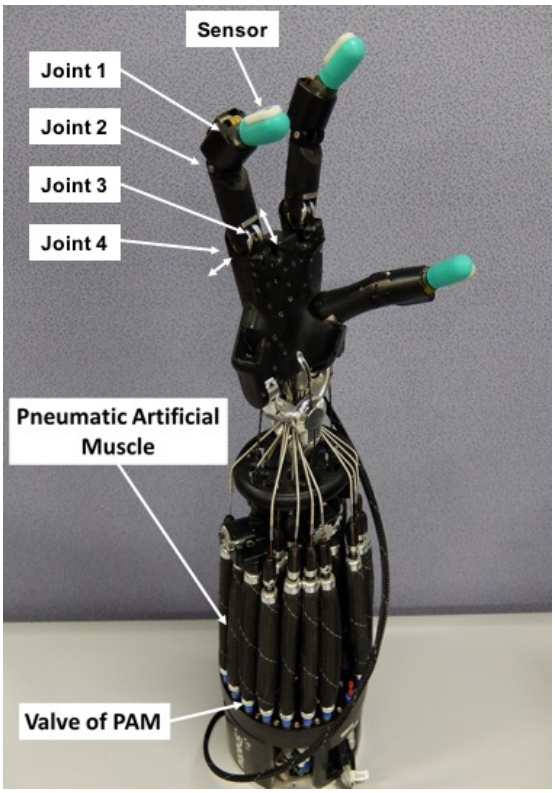


Figure 4.10: The Shadow Dexterous Hand.

As another feasible approach, reinforcement learning is able to learn the control policy of PAM-driven robot without the knowledge of dynamics and therefore does not require the measurement of PAM's volume and length. However, the unstable function approximation and infeasible computational complexity makes previous value function approach based algorithms (e.g., Q-learning [8] and LSPI [12]) infeasible in the application of such a high-dimensional robot system.

4.3.2 Platform: Shadow Dexterous Hand

In this thesis, the platform for real robot experiments the Shadow Dexterous Hand, one PAM-driven humanoid robot hand, which is an advanced humanoid robot hand system that reproduces as closely as possible the kinematics and dexterity of the human hand using pneumatic artificial muscles [41].

According to Fig. 4.10, the Shadow Dexterous Hand has three fingers while each finger has four joints. Two extra joints are used to control the wrist. The joint 1 of each finger is disable since a BioTac tactile sensor with capability of measuring forces, vibrations and heat flow (http://www.syntouchllc.com) is attached on. Each governable joint is driven by two antagonistically placed PAMs. There are three control actions for one PAM's valve: inflating, deflating and holding the current pressure. The state space of one governable joint in the Shadow Dexterous Hand is a four dimensional vector $[\theta, \dot{\theta}, P_1, P_2]^{\mathrm{T}}$ where θ and $\dot{\theta}$ are the the joint angle and angular velocity respectively, P_1 and P_2 are the air pressures of the two antagonistic PAMs. The action space is $[u_1, u_2]^{\mathrm{T}}$ where u_1 and u_2 being their corresponding actions.

For the computational resource, a computational server with Intel Xeon E5-2697 v2 CPU and 250 GB memory was used. The control framework of the Shadow Dexterous Hand was achieved by Robot Operating System (ROS) [88]. The reinforcement learning algorithms were written in Matlab. Its Robotics System Toolbox is used for communication between Matlab and ROS.

4.3.3 Position Reaching Control using LUDPP

In this section, we apply LUDPP to a two DOF finger reaching control of a Shadow Dexterous Hand (Fig. 4.10) as one application of controlling PAM-driven robots. This task has a 12-dimensional state-action space that is intractable to typical value function based RL algorithms.

Experimental Setting

Our task is controlling the joint 2: J_a (proximal-phalangeal joint) and the joint 3: J_b(metacarpal-phalangeal joint) of the shadow dexterous hand (Fig. **??**) to reach one finger to the target position. According to Section 2, one joint has a 6 dimensional state-action space. In this 2 joints control, the control state is $\mathbf{s} = [\theta_a, \dot{\theta}_a, P_{a1}, P_{a2}, \theta_b, \dot{\theta}_b, P_{b1}, P_{b2}]^{\mathrm{T}}$ and the action is $\mathbf{u} = [u_{a1}, u_{a2}, u_{b1}, u_{b2}]$ with 3 discrete actions: filling air in, releasing air out and hold the current air pressure. Thus, a 12 dimensional state-action space is generated.

For function approximation, we set three RBFs in each dimension of the state-

action space. The total number of RBFs was $3^{12} = 531441$. Such a huge number of RBFs made the calculation intractable to SADPP and LSPI under our computation server. More than 2 TB space is required to save a 531441×531441 matrix in Matlab and calculating its inversion needs a very huge number of memory and extremely long time.

We apply LUDPP to face the challenge of the curse of dimensionality in this experiment. For the parameters of LUDPP, we set $P_{NNS} = 500$ and other parameters as same as the simulation tasks' in Section 4. The target positions were set to $\theta_{target,a} = 1.5\,\text{rad}$, $\theta_{target,b} = 1.0\,\text{rad}$. The reward function was defined by $\mathcal{R} = -1000 \times ((\theta_a - \theta_{target,a})^2 + (\theta_b - \theta_{target,b})^2)$. At the beginning of each roll-out, the initial state of the shadow dexterous hand was set as: $\theta_a = 0\,\text{rad}$, $\theta_b = 0.6\,\text{rad}$, $\dot{\theta}_a = \dot{\theta}_b = 0\,\text{rad/s}$. The air pressures in all relative PAMs were set from 100 to 300 kPa. We generated 5 trajectories with totally 5×50 samples every iteration. The control loop runs at 1.25 Hz while each action operates for 0.25 s. A test roll-out with a greedy control policy is carried out at the beginning of each iteration. The reward of its last state $\mathcal{R}_{test}$ is recorded. The stopping criterion is $\mathcal{R}_{test} \geq -100$. The stopping criterion is not strict, since we focus on investigating the learning performance of our proposed algorithm. We used a computational server with Intel Xeon E5-2697 v2 CPU and 250 GB memory. The average calculate time of one iteration is around 10 minutes and the whole learning time of one experiment with 20 iterations including generating samples and updating weights is around 4 hours.

Results

The learning result based on 5 times experiments is shown in Fig. 4.11 where LUDPP converged to a stable policy with a limited number of samples (250 samples/iteration) and iterations (15 iteration). The average number of locally updated RBFs over 5 times experiments is 51606.17 that is only 9.71% of the totally 531441 RBFs among the 12-dimensional state-action space.

Figure 4.12 shows one learning's snapshot of the test roll-out's control trajectories (15 time steps) in the first, 5-th, 10-th and 15-th iteration. The white lines on figure demonstrate the target position. In the first iteration, two antagonistically placed PAMs could not cooperate to move the corresponding joint to the target

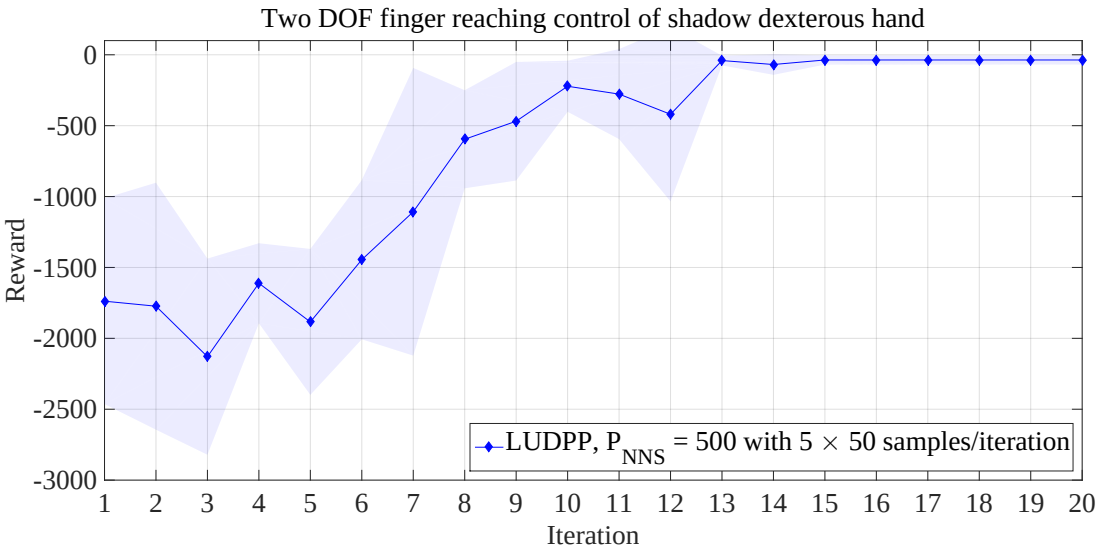


Figure 4.11: The learning result of two DOF finger reaching control of shadow dexterous hand with error bar of standard deviation.

position as control policy π follows the initial action preferences function approximated by RBFs with random weights. The shadow hand started to move its finger to the target position iteratively according to the 5-th and 10-th iterations and finally get a good solution in the 15-th iteration.

Figure 4.13 shows the analysis of the 15-th iteration's control trajectory. The left row shows four PAMs' action trajectories. The middle row shows their corresponding air pressure. The angle trajectories of two joints is in the right row. In this iteration, LUDPP learned to cooperate the two antagonistically placed PAMs with opposite actions (filling air in/releasing air out) to move the joint following Eq. (4.7). On the other hand, LUDPP also considered the nonlinear dynamics of air pressure and axial displacement (unobservable state) to fix and keep the position of joints accurately by some small actions.

In this experiment, LUDPP reduces the intractable calculation of a PAM-driven robot system with 12 dimensional state-action space to a tractable one and smoothly converges to a good solution with limited number of samples and iterations. This result suggests that LUDPP has a good potential to be applied in advanced robot systems with highly nonlinear dynamics while other value

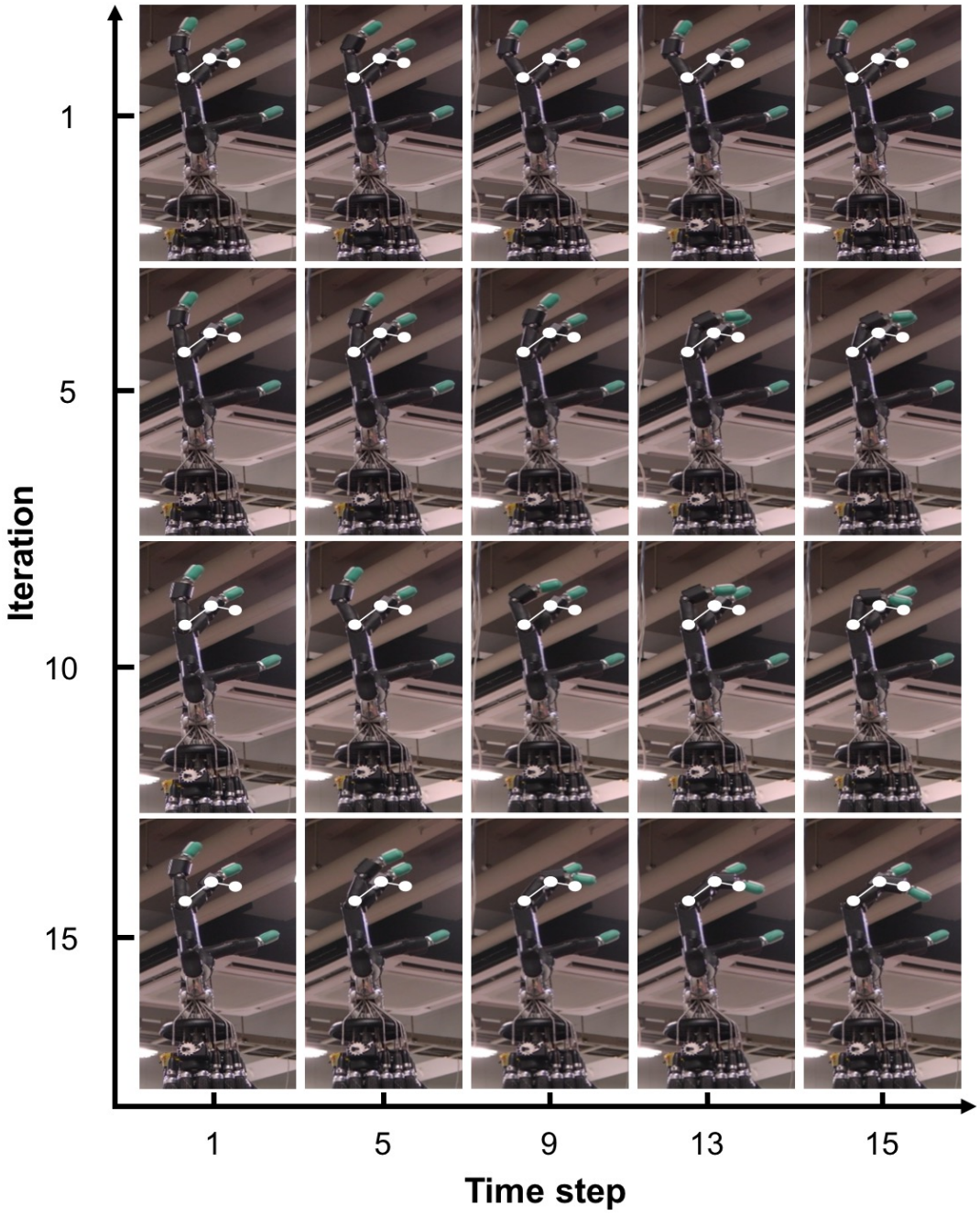


Figure 4.12: Snapshot of two DOF finger reaching control of shadow dexterous hand. (the white line demonstrates the target position)

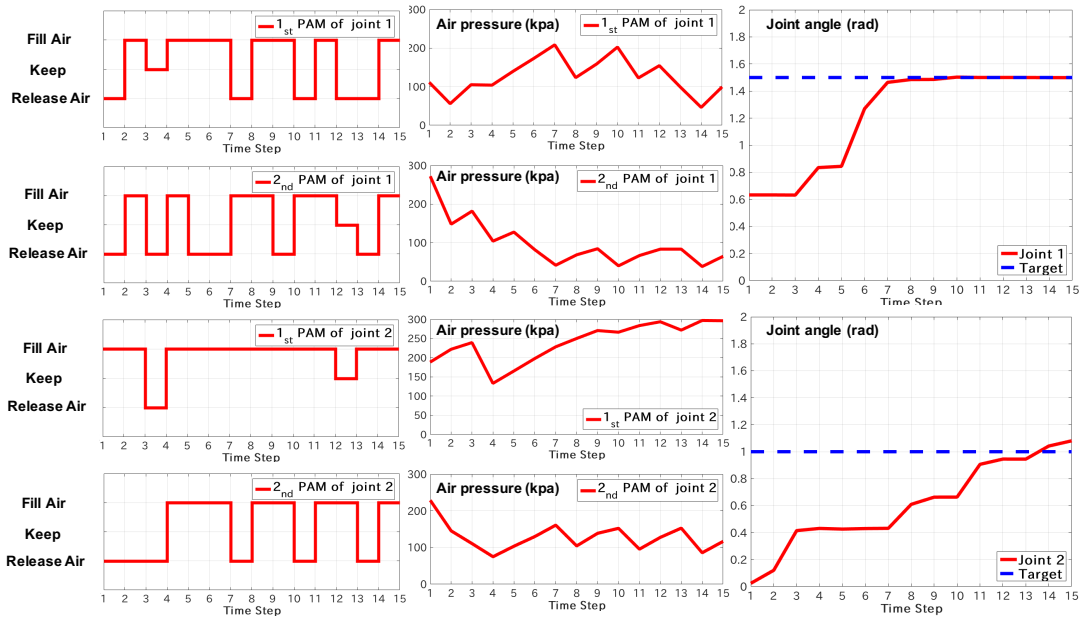


Figure 4.13: The analysis of one learned policy in two DOF finger reaching control.

function based RL algorithms are commonly limited to some small scale problems.

4.4 Summary of LUDPP

All simulation and real robot experiment results in this section support that LUDPP worked good with its action preference function being updated locally. Once a suitable P_{NNS} is set, LUDPP is able to learn as good as SADPP to learn a good solution and considerably reduces the computational complexity while LULSPI approximates its value function incorrectly and diverges easily without smooth policy update. This result met our assumption that the smooth policy update stabilizes the local update. With the increase of the system dimensions, the percentage of the locally updated RBFs will shrink to a very small as seen in all simulations. Supported by Figs. 4.5 and 4.7, the error of function approximation increased iteratively and made the learning difficult to converge in LULSPI. The non-smooth policy update results in a crude approximation of value function as the active RBFs updated in the current iteration contributed less to the samples in the next iteration due to the long distance between them. The simulation results also show that LUDPP also has a better sample complexity similar to SADPP. With the same number of samples and suitable setting of P_{NNS}, LUDPP outperformed LSPI in the converged reward.

On the the other hand, the computational complexity of LUDPP is still intractable in complex tasks. In real robot experiment, LUDPP need to calculate 50000 RBFs (9.71% of all) is still too expensive due to the large total number of the RBFs. In the next chapter, we will further update it to a more efficient algorithm, KDPP by representing the approximate value function by infinite RBFs using kernel trick [40, 89].

5 Kernel Dynamic Policy Programming

In this chapter another new value function approach based Reinforcement Learning algorithm, Kernel Dynamic Policy Programming (KDPP), is proposed to learn tasks represented by high dimensional states Markov decision process with both increased stability and much reduced computational complexity. KDPP inherits the smooth policy update of Dynamic Policy Programming (DPP) [36–38] which allows for stable value function approximation when faced with insufficient samples in high dimensional state-action spaces, by considering the Kullback-Leibler divergence between current and new policies as a regularization term. The stable learning of an approximated value function then allows for application of the kernel trick [40,89] to implicitly represent and update the approximate value function of high dimensional state-action spaces using the inner product of pairs of generated samples, to reduce both learning divergence and sometimes intractable computational complexity.

We investigate the scalability of KDPP by comparing it with existing kernel trick based value function approaches: kernel based least squares policy iterations (KLSPI) [90, 91] and kernel based least squares policy evaluation (KLSPE) [92] in a simulated n DOF manipulator reaching task ($n = 2, 5, 10, 20, 40$). A further analysis of the accuracy of kernel based value function approximation in KDPP, KLSPI and KLSPE is conducted to demonstrate the improved performance of our proposed algorithm. As an application to a real high dimensional robot system, we also apply KDPP to control the Shadow Dexterous Hand, a Pneumatic Artificial Muscle (PAM) driven humanoid hand, to unscrew a bottle cap with the aid of touch sensors. This combined system has a 32 dimensional state space with 625 discrete actions whose high state-action dimensionality renders it impractical

for conventional value function approach algorithms to the best of our knowledge, whereas KDPP converged to a viable solution within a small number of learning iterations while given limited samples.

5.1 Proposed Method

The idea of applying the kernel trick in value function approaches is proposed in [90–92] in order to efficiently approximate value function in high-dimensional systems. However, to the best of our knowledge, the current research is limited in simulations with low dimensionality because the kernel selected from an overly aggressively updated policy results in both a brittle kernel function approximation and diverged learning. In this section, combining SADPP with kernel function approximation results in our proposed Kernel Dynamic Policy Programming (KDPP) with the ability to iteratively select kernels based on a smoothly updated sampling policy, which leads to a stable kernel function approximation in high-dimensional systems with a limited number of samples.

5.1.1 Kernel Trick

According to [93], define a kernel as a symmetric function between two points as $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = k(\boldsymbol{x}_j, \boldsymbol{x}_i)$, a kernel matrix as $\boldsymbol{K}$ storing kernel values for all pairs in a dataset with $[\boldsymbol{K}]_{ij} = [\boldsymbol{K}]_{ji} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The kernel function can be interpreted as a inner product between the two points in a higher-dimensional space following Mercer's theorem [94] if $\boldsymbol{K}$ is positive semi-definite. The kernel trick is to use this property to represent a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the pairs of data in the feature space. It is a widely applied technology in machine learning with several popular kernels, e.g., Fisher, Polynomial and RBF kernels.

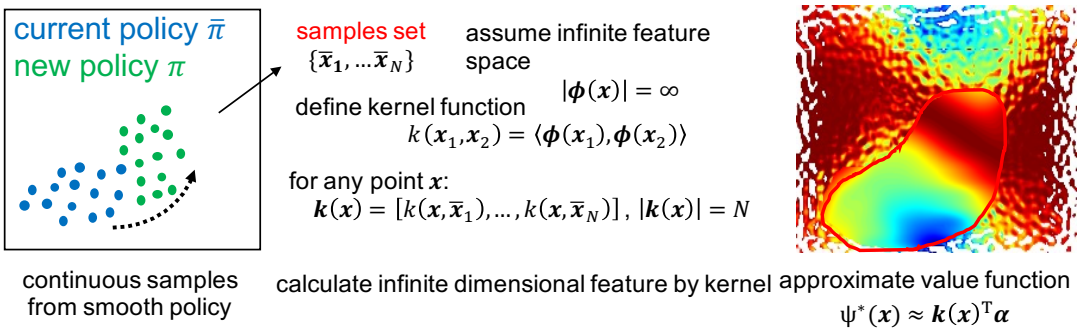


Figure 5.1: The principle of applying Kernel trick: calculating infinite dimensional basis functions on state-action space by kernel functions on samples space.

5.1.2 Action Preferences with Kernel Function Approximation

In this research, we apply the kernel trick to action preferences with LFA to overcome the curse of dimensionality: rather than explicitly computing the coordinates of the matrix $\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}$ which is intractable with high system dimensionality, computationally cheaper inner products among samples are employed to efficiently approximate action preferences.

Start from the approximated action preferences via LFA: $\hat{\Psi}_{t+1}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\theta}_{t+1}$, plug Eq. (3.23) into it, the approximated action preferences function becomes:

$$\hat{\Psi}_{t+1}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}[\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \sigma^2\mathbf{I}]^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathcal{O}\hat{\boldsymbol{\Psi}}_t \tag{5.1}$$

which can be represented using the Woodbury identity:

$$\hat{\Psi}_{t+1}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\Phi}^{\mathrm{T}}[\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \sigma^2\mathbf{I}]^{-1}\mathcal{O}\hat{\boldsymbol{\Psi}}_t. \tag{5.2}$$

Defining a $N \times N$ kernel matrix $\boldsymbol{K} := \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$ that $[\boldsymbol{K}]_{ij} = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle =: k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and a $N \times 1$ vector $\boldsymbol{k}(\boldsymbol{x}) = [k(\boldsymbol{x}, \tilde{\boldsymbol{x}}_1), ..., k(\boldsymbol{x}, \tilde{\boldsymbol{x}}_N)]^{\mathrm{T}}$, the approximated action preferences function represented by the kernel is obtained as:

$$\hat{\Psi}_{t+1}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^{\mathrm{T}}[\boldsymbol{K} + \sigma^2\mathbf{I}]^{-1}\mathcal{O}\hat{\boldsymbol{\Psi}}_t = \boldsymbol{k}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\alpha}_{t+1} \tag{5.3}$$

where $\boldsymbol{\alpha}_{t+1} = [\boldsymbol{K} + \sigma^2\mathbf{I}]^{-1}\mathcal{O}\hat{\boldsymbol{\Psi}}_t$ is the $N \times 1$ dual variable vector of $M \times 1$ weight vector $\boldsymbol{\theta}_{t+1}$. The main principle of KDPP is shown in Fig 5.1. As a further extension of LUDPP, KDPP is able to calculate even infinite dimensional basis functions, i.e., $|\boldsymbol{\phi}(\boldsymbol{x})| = \infty$, by translating it to kernel functions on samples space.

5.1.3 Online Selection of a Regression Subset

Employing the kernel trick, the action preferences function is efficiently approximated on N sample pairs following Eq. (5.3). The next step is to select a subset of samples $\mathcal{D}_{kernel} = [\tilde{\boldsymbol{x}}_n]_{n=1:N'}, N' \ll N$ to reduce computational complexity. According to [91,92], the output vector of basis functions $\boldsymbol{\phi}(\boldsymbol{x}) = [\varphi_1(\boldsymbol{x}), ..., \varphi_M(\boldsymbol{x})]^{\mathrm{T}}$ should be approximated by $\mathcal{D}_{kernel}$ with the corresponding $N' \times 1$ weight vector $\boldsymbol{\alpha}$:

$$\boldsymbol{\phi}(\boldsymbol{x}) \approx [\phi(\tilde{\boldsymbol{x}}_1), ..., \phi(\tilde{\boldsymbol{x}}_{N'})]\boldsymbol{\alpha}. \tag{5.4}$$

To minimize the error of approximation $\|\boldsymbol{\phi}(\boldsymbol{x}) - [\phi(\tilde{\boldsymbol{x}}_1), ..., \phi(\tilde{\boldsymbol{x}}_{N'})]\boldsymbol{\alpha}\|^2$, $\boldsymbol{\alpha}$ follows:

$$\boldsymbol{\alpha} = \boldsymbol{K}_{N'N'}^{-1}\boldsymbol{k}_{N'}(\boldsymbol{x}) \tag{5.5}$$

where $\boldsymbol{K}_{N'N'}$ is a $N' \times N'$ kernel matrix of subset with $[\boldsymbol{K}_{N'N'}]_{ij} = k(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)$. $\boldsymbol{k}_{N'}(\boldsymbol{x})$ represents vector $[k(\boldsymbol{x}, \tilde{\boldsymbol{x}}_1), ..., k(\boldsymbol{x}, \tilde{\boldsymbol{x}}_M)]^{\mathrm{T}}$. Defining $[\boldsymbol{K}_{NN'}]_{ij} = k(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}_j)$, the calculation of kernel matrix is reduced from all samples to a subset $\mathcal{D}_{kernel}$:

$$\boldsymbol{K} \approx \boldsymbol{K}_{NN'}\boldsymbol{K}_{N'N'}^{-1}\boldsymbol{K}_{NN'}^{\mathrm{T}}, \tag{5.6}$$

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) \approx \boldsymbol{k}_{N'}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{K}_{N'N'}^{-1}\boldsymbol{k}_{N'}(\boldsymbol{x}_j). \tag{5.7}$$

Define the new generated samples including state and action as $\boldsymbol{x}^* = [\boldsymbol{s}^*, \boldsymbol{a}^*]$. For on-line selection of $\mathcal{D}_{kernel}$ during the learning process, we calculate the variance of each generated sample $\boldsymbol{x}^*$ approximated by the current $\mathcal{D}_{kernel}$ that indicates its informativeness (i.e., approximate linear dependency (ALD) analysis):

$$\delta^* = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}_{N'}(\boldsymbol{x}^*)^{\mathrm{T}}(\boldsymbol{K}_{N'N'})^{-1}\boldsymbol{k}_{N'}(\boldsymbol{x}^*). \tag{5.8}$$

If the corresponding δ^* exceeds a given threshold, $\boldsymbol{x}^*$ is considered to be sufficiently unique to add to $\mathcal{D}_k$ as a new feature since the current $\mathcal{D}_k$ could not

accurately approximate δ^*. Intuitionally, the kernel function approximation with online selection of regression set can be treated as adaptively selecting RBFs from samples to efficiently approximate the function among the current known state-action space. It considerably reduces the computational complexity in high dimensional problem where the observe space is usually far smaller than the whole one.

5.1.4 Kernel Dynamic Policy Programming

Here we detail the pseudocode of KDPP in Algorithm 6. KDPP is a model-free reinforcement learning algorithm, its samples are obtained by rollout-based interaction with the plant/environment every iteration and reused in future learnings. The data generated in the t-th iteration is defined as $\mathcal{D}_t = \{D_t^1, ..., D_t^I\}$ containing I rollout trajectories. Each trajectory has J samples: $D_t^i = \{(\boldsymbol{s}_t^{i,j}, \boldsymbol{a}_t^{i,j}, \boldsymbol{s'}_t^{i,j})\}_{j=1:J}$.

According to Algorithm 6, the inputs are the temperature of the Kullback-Leibler divergence term η, regularization term σ, number of iterations T, number of rollouts I, rollout length J, and **TOL** as the threshold for on-line regression subset selection. In line 1, KDPP initializes the regression subset $\mathcal{D}_{kernel}$ as an empty set, and $\boldsymbol{\alpha}$ as an empty vector with size $N^{'} = 0$. From lines 3 to 6, KDPP first generates samples. At the first iteration ($t = 0$), the samples are generated by a purely random policy π^{random}. For later iterations $t \geq 1$, samples are generated by a soft-max exploration policy defined as:

$$\pi^{\text{explore}}(\boldsymbol{a}|\boldsymbol{s}) = \frac{\exp(\eta_{\text{explore}}\hat{\Psi}_t(\boldsymbol{s}, \boldsymbol{a}))}{\sum_{\boldsymbol{a'}\in\mathcal{A}} \exp(\eta_{\text{explore}}\hat{\Psi}_t(\boldsymbol{s}, \boldsymbol{a'}))}. \tag{5.9}$$

It is based on the baseline policy $\bar{\pi}$ of each iteration following Eq. (3.26) where $\Psi(\boldsymbol{s}, \boldsymbol{a})$ is approximated by the current subset of kernel ridge regression [95] $\mathcal{D}_{kernel}$. The randomness of exploration is controlled by a temperature parameter η_{explore}. The baseline policy is determined by the current subset of kernel with initial weights at the first iteration, and then be selected as the policy updated in the previous iteration. It is implicitly represented by the action preference functions and be unnecessary to calculated. From lines 7 to 15, the subset for kernel ridge regression $\mathcal{D}_{kernel}$ is built from samples $\mathcal{D}_0$, following Eq. (5.8) with threshold **TOL** (if $\mathcal{D}_{kernel}$ is empty, the current sample is added). $\boldsymbol{\alpha}$ is expanded

Algorithm 6 Kernel Dynamic Policy Programming

Require: η, γ, σ, T, I, J and **TOL**.

initialize $\mathcal{D}_{kernel} = \emptyset$, $\boldsymbol{\alpha} = \emptyset$ and $N^{'} = 0$.
for t= 0, 1, 2, ..., T
  if t == 0
    generate $\mathcal{D}_t$ by π^{random}
  else
    generate $\mathcal{D}_t$ by π_t^{explore}
  for each $\{\boldsymbol{x}_t^{i,j} = [\boldsymbol{s}_t^{i,j}, \boldsymbol{a}_t^{i,j}]\}_{i=1:I,j=1:J} \in \mathcal{D}_t$
    if $\mathcal{D}_{kernel} == \emptyset$
      $N^{'} = N^{'} + 1$
      add $\tilde{\boldsymbol{x}}_{N'} = \boldsymbol{x}_t^{i,j}$ to $\mathcal{D}_{kernel}$ and $\alpha_{N'} = 0$ to $\boldsymbol{\alpha}$
    else
      calculate $\delta = k(\boldsymbol{x}_t^{i,j}, \boldsymbol{x}_t^{i,j}) - \boldsymbol{k}_{N'}(\boldsymbol{x}_t^{i,j})^{\mathrm{T}}(\boldsymbol{K}_{N'N'})^{-1}\boldsymbol{k}_{N'}(\boldsymbol{x}_t^{i,j})$
      if $\delta >$ **TOL**
        $N^{'} = N^{'} + 1$
        add $\tilde{\boldsymbol{x}}_{N'} = \boldsymbol{x}_t^{i,j}$ to $\mathcal{D}_{kernel}$ and $\alpha_{N'} = 0$ to $\boldsymbol{\alpha}$
  for each $\{(\boldsymbol{s}_l^{i,j}, \boldsymbol{a}_l^{i,j}, \boldsymbol{s}_l^{\prime i,j})\}_{l=0:t,i=1:I,j=1:J}$
    for each $\boldsymbol{a} \in \mathcal{A}$
      $\hat{\Psi}_t([\boldsymbol{s}_l^{i,j}, \boldsymbol{a}]) = \sum_{n=1}^{N^{'}} k(\tilde{\boldsymbol{x}}_n, [\boldsymbol{s}_l^{i,j}, \boldsymbol{a}])\alpha_n$
      $\hat{\Psi}_t([\boldsymbol{s}_l^{\prime i,j}, \boldsymbol{a}]) = \sum_{n=1}^{N^{'}} k(\tilde{\boldsymbol{x}}_n, [\boldsymbol{s}_l^{\prime i,j}, \boldsymbol{a}])\alpha_n$
    $\mathcal{M}_\eta \hat{\Psi}_t(\boldsymbol{s}_l^{i,j}) = \sum_{\boldsymbol{a}\in\mathcal{A}} \frac{\exp(\eta\hat{\Psi}_t([\boldsymbol{s}_l^{i,j},\boldsymbol{a}]))\hat{\Psi}_t([\boldsymbol{s}_l^{i,j},\boldsymbol{a}])}{\sum_{\boldsymbol{b}\in\mathcal{A}}\exp(\eta\hat{\Psi}_t([\boldsymbol{s}_l^{i,j},\boldsymbol{b}]))}$
    $\mathcal{M}_\eta \hat{\Psi}_t(\boldsymbol{s}_l^{\prime i,j}) = \sum_{\boldsymbol{a}\in\mathcal{A}} \frac{\exp(\eta\hat{\Psi}_t([\boldsymbol{s}_l^{\prime i,j},\boldsymbol{a}]))\hat{\Psi}_t([\boldsymbol{s}_l^{\prime i,j},\boldsymbol{a}])}{\sum_{\boldsymbol{b}\in\mathcal{A}}\exp(\eta\hat{\Psi}_t([\boldsymbol{s}_l^{\prime i,j},\boldsymbol{b}]))}$
    $\mathcal{O}\hat{\Psi}_t([\boldsymbol{s}_l^{i,j}, \boldsymbol{a}_l^{i,j}]) = \sum_{m=1}^{M} k(\tilde{\boldsymbol{x}}_m, [\boldsymbol{s}_l^{i,j}, \boldsymbol{a}_l^{i,j}])\alpha_m + r_{\boldsymbol{s}_l^{i,j},\boldsymbol{s}_l^{\prime i,j}}^{\boldsymbol{a}_l^{i,j}}$
      $+\gamma\mathcal{M}_\eta\hat{\Psi}_t(\boldsymbol{s}_l^{\prime i,j}) - \mathcal{M}_\eta\hat{\Psi}_t(\boldsymbol{s}_l^{i,j})$
  update $\boldsymbol{\alpha} = [\boldsymbol{K}_{N'N'} + \sigma^2\mathbf{I}]^{-1}\mathcal{O}\hat{\boldsymbol{\Psi}}_t$
return $\mathcal{D}_{kernel}$ and $\boldsymbol{\alpha}$

by adding the corresponding new dual variable term $\alpha_{N'} = 0$, and then updating from lines 16 to 23.

5.2 Simulation Results

5.2.1 Simulation Setting

In this section the learning performance of KDPP is investigated in a simulated n DOF manipulator reaching task ($n = 2, 5, 10, 20, 40$). It is an interesting and suitable task to investigate the performance of value function approach based reinforcement learning in robot control problems with high dimensional state space because it keeps the basic framework of robot control while it can be easily extended to different dimensional state space with the corresponding DOFs with low computational complexity. The continuous states in this are defined as $[\theta_1, \theta_2, ..., \theta_n]^{\mathrm{T}}$. $\theta_i \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ rad represents the angle in the i-th joint of manipulator. Each joint has five discrete actions, $[-0.0875, 0.0175, 0, 0.0175, 0.0875]$ rad, i.e., $-5°$, $-1°$, $0°$, $1°$ and $5°$) in angle space, to increment the joint with the respective angle. We define an action at each time step as one move on one joint so the total number of actions is reduced to $(n \times 4) + 1$ (four valid movements for each joint and one stop action).

The aim of this definition is to simplify the problem and avoid the intractable larger action space and focus on to investigate the performance in tasks with higher dimensional state space since it is the focus feature of the proposed method. If we redefine the actions to make them move all the joints at a time in 20 DOF manipulator reaching task, the action space will have $5^{20} \gg 10^{13}$ actions which is intractable to calculate. The first joint is set to position $[0, 0]$. The length of each limb between two joints is set to $\frac{1}{n}$ m. All angles are initialized to 0 rad at the start of the simulation. The target position to reach in two dimensional axes is set as $X_{\text{target}} = 0.6830, Y_{\text{target}} = 0$, and the reward function is set as $\mathcal{R} = -1000 \times \left((X_{\text{target}} - X)^2 + (Y_{\text{target}} - Y)^2\right)$ where X, Y is the current position of end-effector. Learning is deemed finished when the reward from a test rollout of the last state is more than -1. The following results are derived from 100 repetitions of the same experiment.

Table 5.1: List of algorithms compared in simulation

Algorithm	Update type	Kernel trick	KL term
LSPI [12, 59]	one-shot	×	×
LSPE [11]	iterative	×	×
SADPP [36–38]	one-shot	×	◯
KLSPI [90, 91]	one-shot	◯	×
KLPE [92]	iterative	◯	×
KDPP	one-shot	◯	◯

A comparison with five other algorithms (Table 5.1) is conducted. Least Squares Policy Programming (LSPI) [12, 59], Least Squares Policy Evaluation (LSPE) [11] and Sampling-based Approximate Dynamic Policy Programming (SADPP) [36–38] are conventional value function approaches using linear function approximation while SADPP employs the Kullback-Leibler divergence (KL term). Both LSPI and SADPP directly obtain least square solutions from the projected Bellman equation, their new weight vectors are rebuilt every iteration following $\boldsymbol{\theta}_{new} = \boldsymbol{A}^{-1}\boldsymbol{B}$, i.e., "one-shot" update, whereas LSPE updates its weight vector iteratively following $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} + \beta\boldsymbol{A}^{-1}\boldsymbol{B}$, where β is the learning rate [96]. Therefore, LSPI has a non-smooth "one-shot" update, LSPE has a "iterative" update controlled by β and SADPP has a smooth "one-shot" update. We also compare the respective kernelized versions of LSPI and LSPE that use kernel function approximation: Kernel-based Least Squares Policy Programming (KLSPI) [90, 91] and Kernelizing Least squares policy evaluation (KLSPE) [92]. To the best of the authors' knowledge, KLSPI and KLSPE are the only existing kernel trick based value function approaches to date.

The parameters of all algorithms in two, five and ten DOF manipulator reaching tasks are set to $\gamma = 0.95, \sigma = 0.1, T = 50, I = 5, J = 100$ (Algorithm 6). Both SADPP and KDPP have $\eta = 0.01$. All kernel based algorithms share $\mathbf{TOL} = 0.025$. For 20 and 40 DOF manipulator reaching tasks, the rollout number is increased to $I = 10$, while threshold $\mathbf{TOL} = 0.05$. All kernel based algorithms use RBF kernels.

5.2.2 Results

The learning performance in n DOF manipulator, $(n = 2, 5, 10, 20, 40)$, reaching tasks is demonstrated in Fig. 5.2. The best learning policies represented by a trajectory in 50 steps and the success rate over 10 repetitions of all kernel based algorithms are shown in Fig. 5.3. The average number of features used for approximate value function is shown in Fig. 5.4.

The learning result of three DOF manipulator reaching task is shown in Fig. 5.2a where all algorithms learned good control policies. For linear function approximation, with 9 RBFs added in each dimension of state, a total $9^2 \times (4 \times 2 + 1) = 729$ RBFs are required while a subset with less than 60 features is used in kernel function approximation. Both function approximation methods worked well in this task. In the five DOF manipulator reaching task, the required RBFs for linear function approximation are significantly increased to $9^5 \times (4 \times 5 + 1) = 1240029 \gg 10^5$. Saving and calculating such huge weight matrices is impossible using ordinary computing hardware. Instead, we only put 5 RBFs in each dimension of state to employ totally $5^5 \times (4 \times 5 + 1) = 78125$ RBFs. According to Fig. 5.2b, even though SADPP converged to a higher reward compared to LSPI and LSPE in the five DOF manipulator reaching task, its learning performance is far worse than all kernel value function approaches due to the poor function approximation by its limited number of RBFs, whereas all kernel trick based value function approaches efficiently represent the value function by tiny subsets built from samples (less than 200 features). Starting from 10 DOF, SADPP, LSPI and LSPE were not attempted because the total number of RBFs required by linear function approximation grows to $9^{10} \times 41 \gg 10^{10}$, $9^{20} \times 81 \gg 10^{20}$ and $9^{40} \times 161 \gg 10^{40}$ at 10, 20 and 40 DOF respectively. On the other hand the application of KLSPI, KLSPE and KDPP is not limited by high DOF systems, requiring only several thousand RBFs to build the kernel even at 40 DOF. However, the learning performance of KLSPI and KLSPE degenerated as DOF increased in both reward (Figs. 5.2c, 5.2d and 5.2e) and success rate (Fig. 5.3), whereas KDPP kept learning viable solutions to obtain high rewards with high stability.

We also trained KDPP with different settings of η in a five DOF reaching task to investigate its effect on learning. Figure 5.5 illustrates our results based the average of ten repetitions. With a decrease of η, the Kullback-Leilber term in

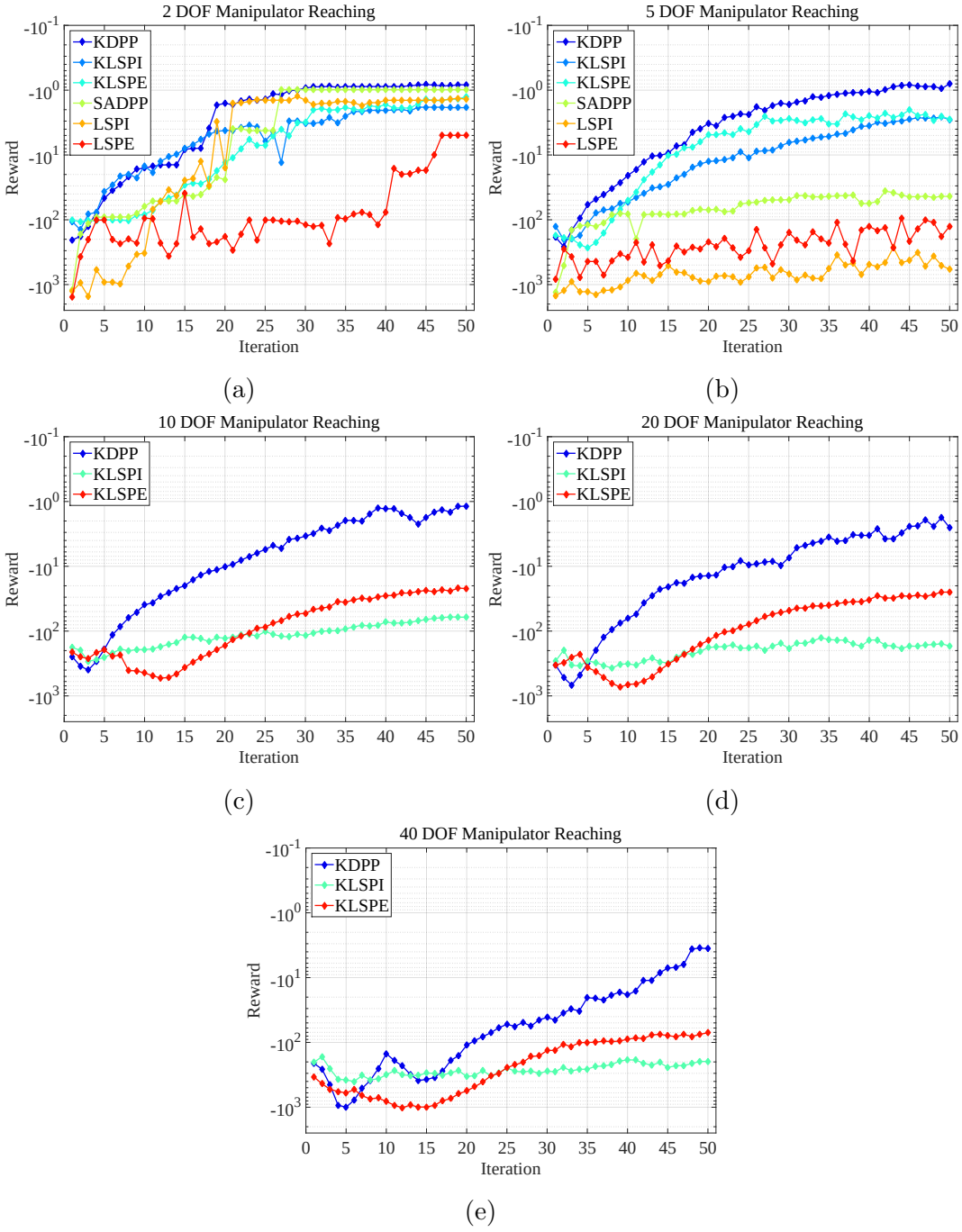


Figure 5.2: The average learning results of the n DOF manipulator reaching task over 100 repetitions.

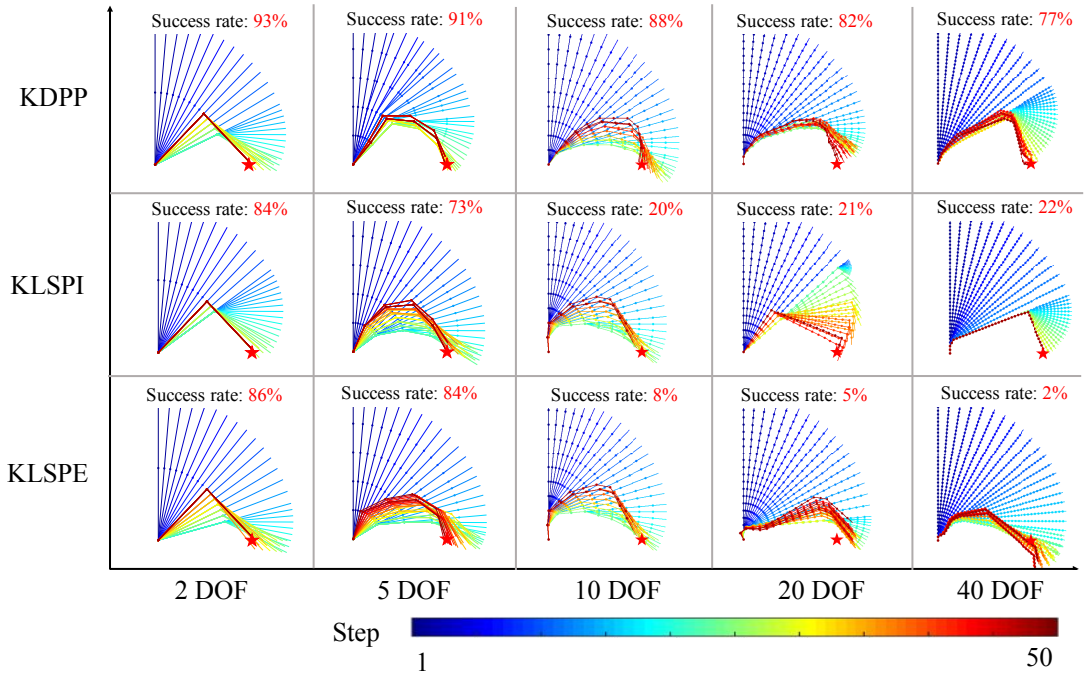


Figure 5.3: An example of iteratively generated samples in the n DOF manipulator reaching task; the red dot is the target position.

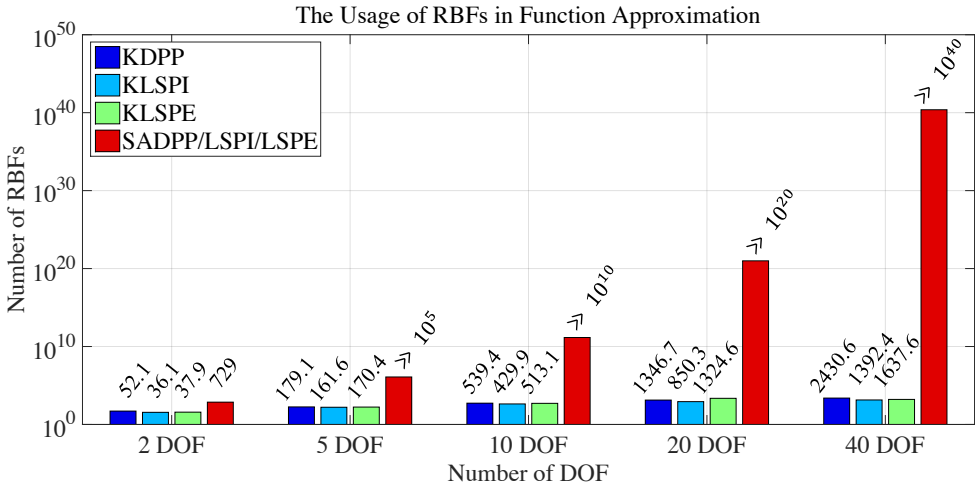


Figure 5.4: The average number of RBFs used for function approximation in the n DOF manipulator reaching task over 100 repetitions.

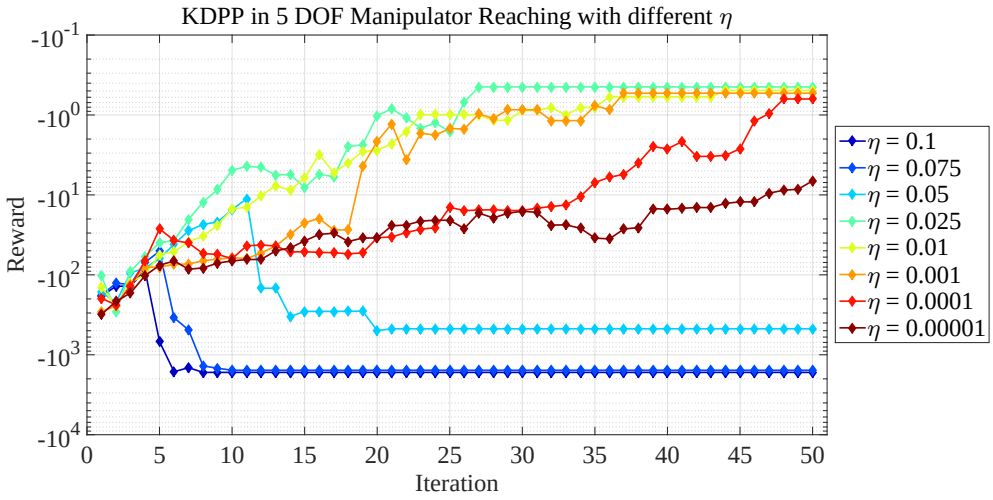


Figure 5.5: The average learning results of KDPP with different set of η in the five DOF manipulator reaching task over ten repetitions.

Eq. (3.4) limits the policy update more, and therefore slows down the rate of learning. On the other hand by increasing η, SADPP/KDPP will increasingly ignore the Kullback-Leilber term. While SADPP/KDPP theoretically turns into KLSPI/LSPI with $\eta \to +\infty$ according to [37], a large η causes numerical instabilities in real applications, resulting in excessively large weights that in turn cause divergences in learning.

The effect of different settings of **TOL** in the five DOF reaching task was investigated in Figs. 5.6 and 5.7. With a increase of **TOL**, less RBFs were selected to build the kernel and therefore limits its capability of function approximation. (when **TOL** = 1, KDPP could not learn to a good result with only one RBF in its kernel set) On the other hand, **TOL** turns to a huge kernel size which is intractable to high-dimensional systems. The selection of **TOL** should be well considered to balance of both approximation ability and computational complexity.

To investigate all three kernel based algorithms' capabilities of function approximation, we define the Bellman error for one sample $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$ in the k-th iteration following [93]:

$$BN\left(\hat{V}_{k+1}(\boldsymbol{s})\right) = \|\hat{V}_{k+1}(\boldsymbol{s}) - \left(\mathcal{R}^{\boldsymbol{a}}_{\boldsymbol{ss}'} + \gamma \hat{V}_k(\boldsymbol{s}')\right)\|_2 \tag{5.10}$$

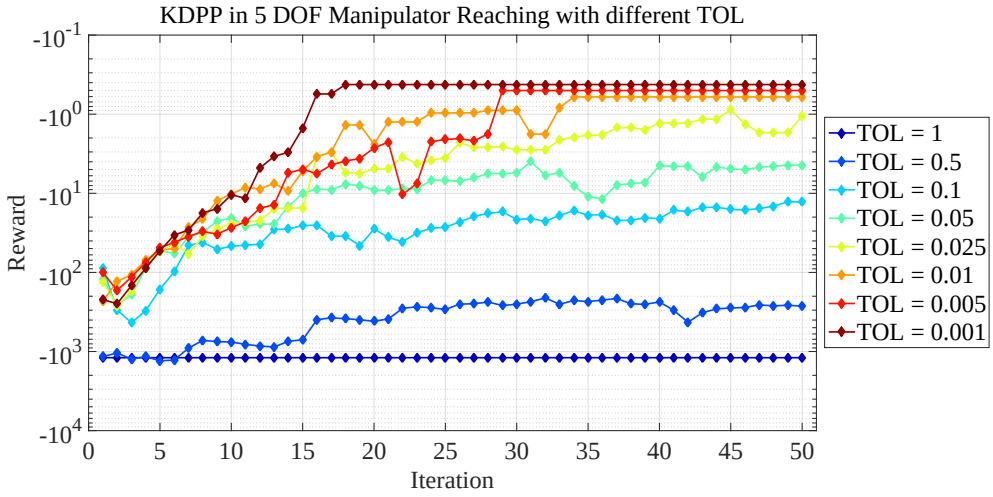


Figure 5.6: The average learning results of KDPP with different set of **TOL** in the five DOF manipulator reaching task over ten repetitions.

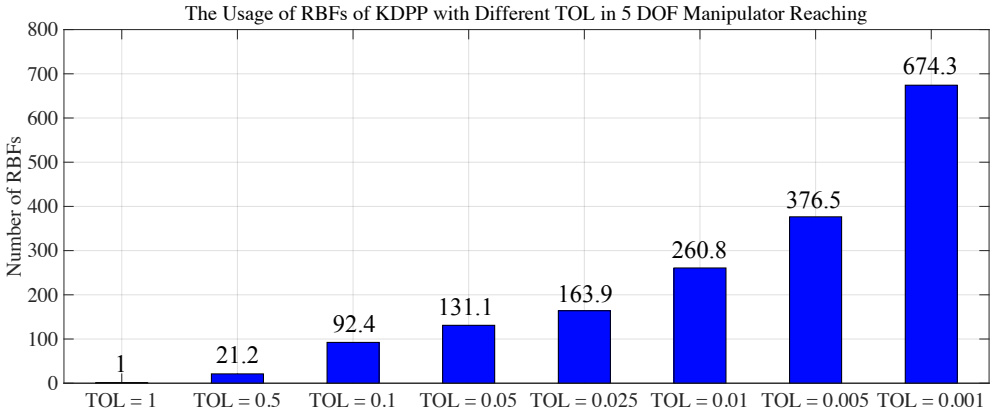
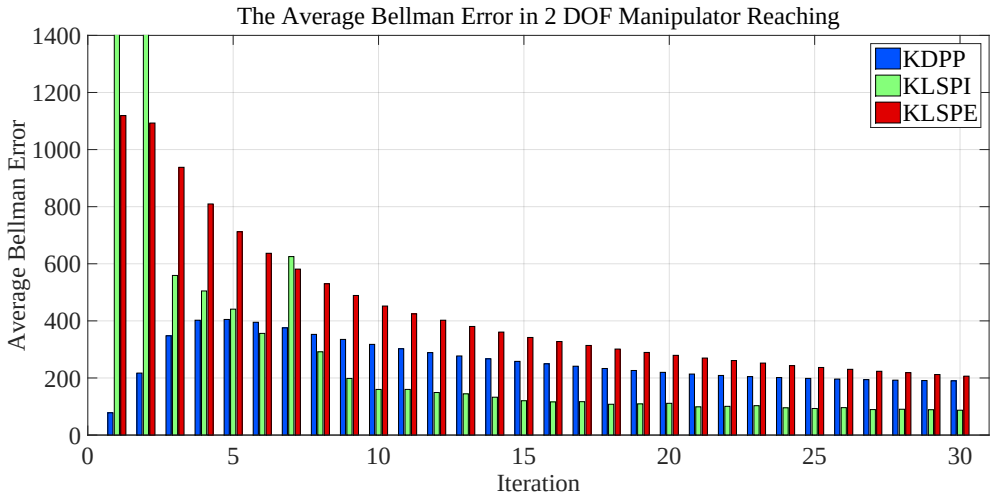
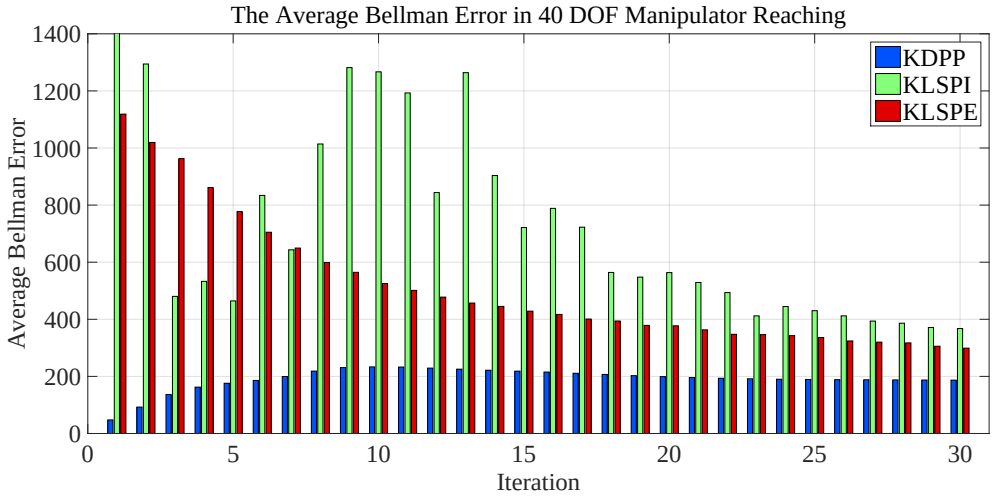


Figure 5.7: The average number of RBFs used for function approximation of KDPP with different set of **TOL** in the five DOF manipulator reaching task over ten repetitions.



(a)



(b)

Figure 5.8: The average Bellman error in each iterations of one learning.

where $\hat{V}_k$ indicate the approximated value function translated from the action preferences function/Q function approximated via kernel, e.g., in KDPP it follows Eq. (3.11), in the k-th iteration. $BN(\hat{V}_{k+1}(\boldsymbol{s}))$ represents the error between the baseline and updated approximated value functions in state $\boldsymbol{s}$. The average Bellman errors among the current samples in the first 30 iteration's learning of 2 and 40 DOF reaching tasks using all three kernel based algorithms are measured in Fig. 5.8.

For the 2 DOF case in Fig. 5.8a, KDPP and KLSPE smoothly decrease their average Bellman error owing to the Kullback-Leibler divergence and the iterative update style respectively, while KLSPI has a steeper decrease but finally reaches a lower error. All three algorithms are able to build suitable subsets of kernels to accurately approximate the value function, however only KDPP reaches a small average Bellman error iteratively while both KLSPI and KLSPE could not in the 40 DOF case (Fig. 5.8b). This result indicates the superiority of our proposed algorithm in high dimensional state space: leveraging the smooth policy update, KDPP limits overly large policy updates and generates samples that iteratively move to a high reward area. The kernel built by such smooth movement of samples contributes towards better approximated value function in high dimensional state space. In comparison, KLSPI and KLSPE could not generate samples that smoothly move to high reward areas as learning suffers without control over the policy update: excessively large policy updates at the beginning of the learning yield poor samples that are far away from high reward areas. These poor samples result in a less expressive subset to approximate the value function in high reward areas via the kernel trick and therefore worsens the policy in the next iteration.

5.3 Real Robot Experiment

In this section, we apply the proposed reinforcement algorithm, KDPP, to Shadow Dexterous Hand to learn unscrewing bottle cap. Compared with the position control task using LUDPP in Chapter 4, this task is more chaillenging with a far higher dimensional state space. The experimental setting of Shadow Dexterous Hand follows Section 4.3.2.

5.3.1 Learning Unscrewing Bottle Cap using KDPP

In this section, KDPP is applied to control the Shadow Dexterous Hand to learn unscrewing a bottle cap using two fingers without modeled dynamics and an initial policy. It forms an appealing task to be solved by reinforcement learning because the knowledge of physical interactions between robot and environment (e.g. the amount of normal force required to generate enough friction in order to rotate the cap using a single finger) is difficult to model.

Experimental Setting

Figure 5.9 shows our experimental setting. Each finger of the Shadow Dexterous Hand has three joints. Each joint is controlled by two antagonistic PAMs with discrete actions u: inflating, deflating and holding the current pressure. The control state of a joint is four dimensional $[\theta, \dot{\theta}, P_1, P_2]^{\mathrm{T}}$ with two dimensional action $[u_1, u_2]^{\mathrm{T}}$, where θ is the joint angle, $\dot{\theta}$ is the angular velocity, P_1 and P_2 are the air pressures of the two antagonistic PAMs, and u_1 and u_2 being their corresponding actions. A BioTac tactile sensor (http://www.syntouchllc.com) is attached to the end of each finger. Capable of measuring forces, vibrations and heat flow, the BioTac tactile sensor is only utilized to measure pressure in this experiment. We divide the touch area into four parts (up, down, left and right), and regularize the pressure values of each part for a four dimensional state $[T_{up}, T_{down}, T_{left}, T_{right}]^{\mathrm{T}}$ to $[0, 1]$. Thus, the control state of each finger has $3 \times 4 + 4 = 16$ dimensions.

Controlling two full fingers to unscrew bottle cap requires a 32 dimensional state space. Here we create 625 actions that limit one joint in each finger to inflate/deflate at each time step. A Microsoft Kinect was set in front of the Shadow Dexterous Hand to obtain the current state of the bottle cap, by capturing the position of a small red marker attached to the cap. During learning, if the cap was unscrewed; i.e. the marker moved down along the vertical Y axis, a high reward is obtained. Ten trajectories with 10×50 samples were generated every iteration, with the initial position of the fingers fixed to be close to the bottle cap. The same parameters for KDPP as used in the simulation experiment were used. The control loop runs at 1.25Hz while each action operates for 0.25 seconds. Before each iteration, a test rollout with a greedy control policy is carried out

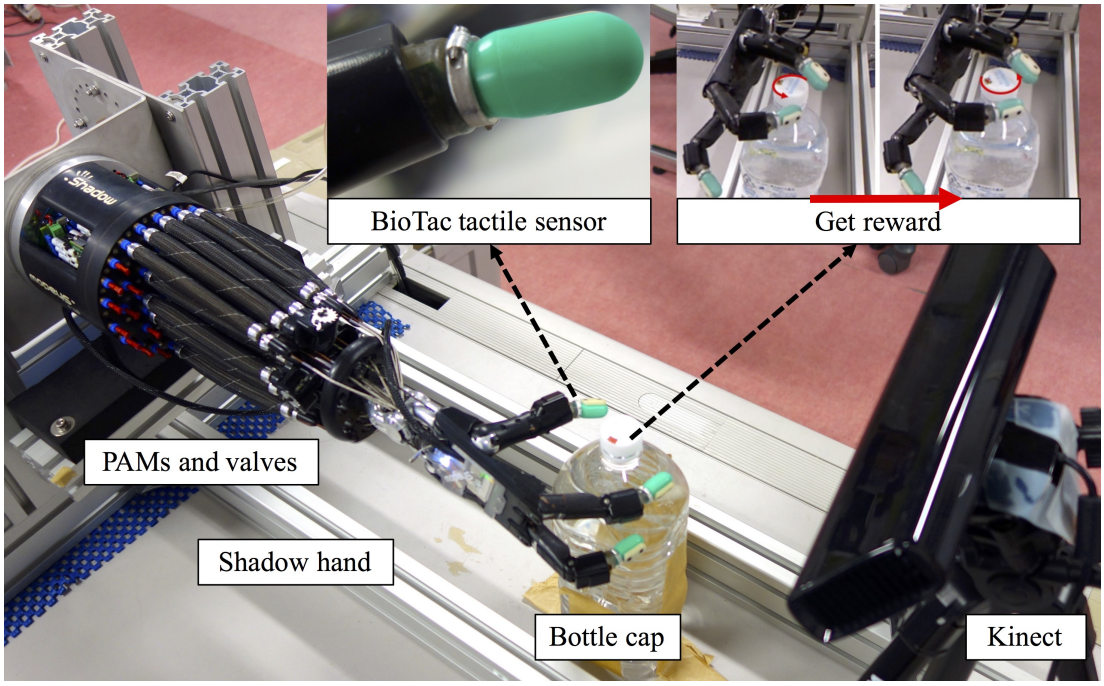


Figure 5.9: Experimental setting of unscrewing a bottle cap via the Shadow Dexterous Hand.

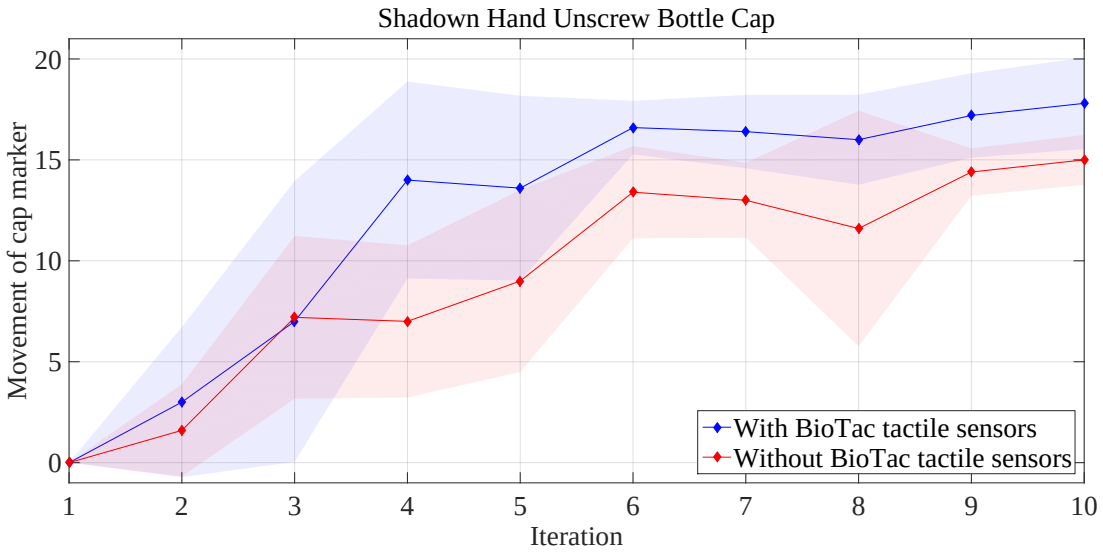


Figure 5.10: The average learning result of unscrewing a bottle cap via the Shadow Dexterous Hand over five repetitions, with error bar of standard deviation.

twice; the reward $\mathcal{R}_{\text{test}} = (Y_{\text{start}} - Y_{\text{end}})$ represents the movement between the initial and final position of marker along the Y axis as captured by the Kinect. The stopping criterion is $\mathcal{R}_{\text{test}} \geq 18$, when the cap has rotated more than 120°. Policies for unscrewing the cap both with and without the use of the tactile sensors were learned five times each by KDPP. We used a computational server with Intel Xeon E5-2697 v2 CPU and 250 GB memory.

Results

Figure 4.11 shows the average five repetitions learning results over . KDPP successfully learned the value function for both the 24 dimensional state (without tactile sensing) and 32 dimensional state (with tactile sensing) cases, resulting in good policies to unscrew the bottle cap within ten iterations. Learning performance is improved by incorporating tactile information compared to learning without tactile information, while the computational complexity is kept tractable for operating on ordinary computing hardware: by the 10-th iteration, on average KDPP approximated the value function by 2493.8 RBFs while the entire

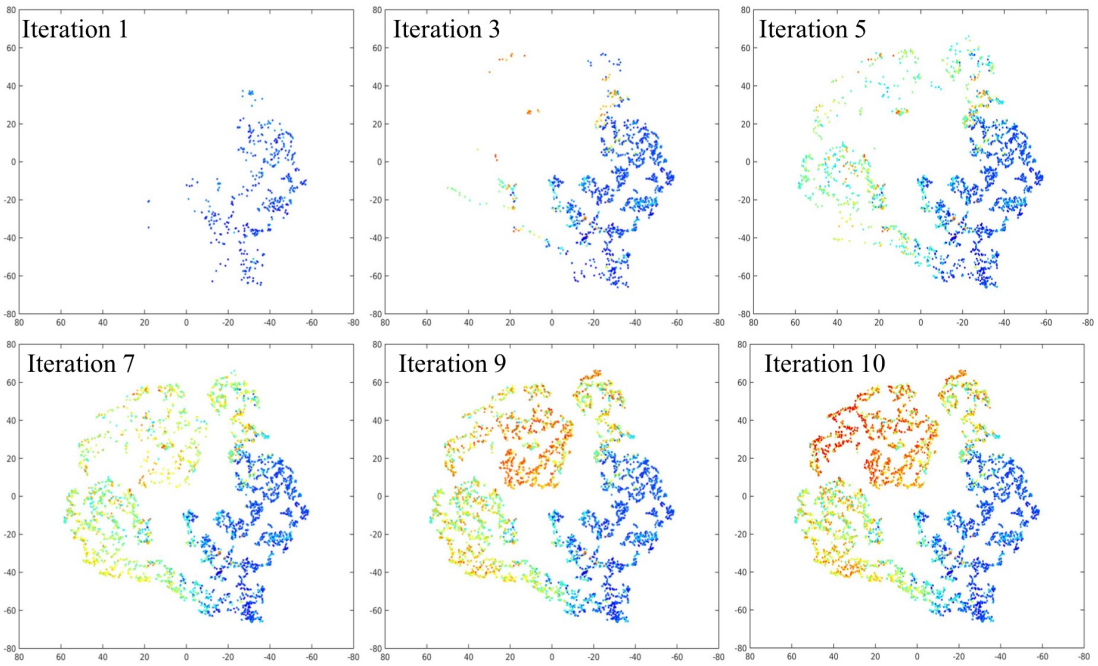


Figure 5.11: The 32 dimensional state of learning samples and test rollouts in ten iterations by t-SNE. The color demonstrates the value function value of each state. Gray arrows show the movement of robot actions in 1, 5, 7 and 9 iteration over this 2D space, the corresponding screen shots are shown in the left.

computation time was 62.8 seconds.

Applying t-Distributed Stochastic Neighbor Embedding (t-SNE) [97,98], a reliable dimensionality reduction technique for the visualization of high dimensional datasets, allows transforming all the learning samples and test rollouts from one trial into a 2D figure representing all 32 dimensional states explored (the visualization is based on states without the value function' value). Then we calculate the corresponding value function's value of each 2D points and demonstrate them by color. In order to track the learning of KDPP, we select some sequential snapshots from the Kinect during the test rollouts of iterations 1, 5, 7 and 9 iteration and follow their states in Fig. 5.12. In the first iteration, KDPP could not reach any high value area when the fingers failed to reach the cap, the test rollout only

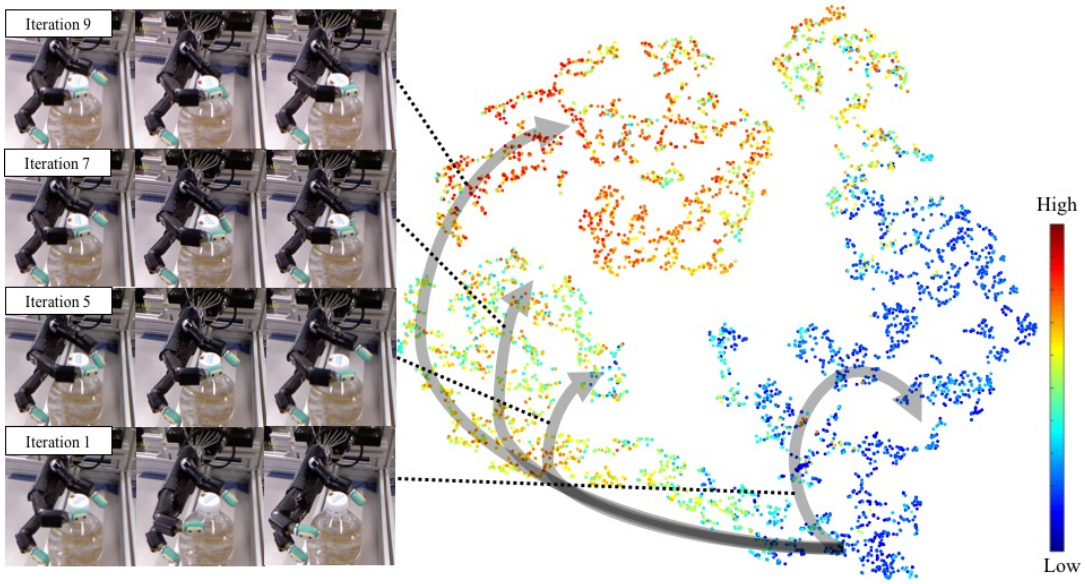


Figure 5.12: The 32 dimensional state of learning samples and test rollouts in ten iterations by t-SNE. The color demonstrates the value function value of each state. Gray arrows show the movement of robot actions in 1, 5, 7 and 9 iteration over this 2D space, the corresponding screen shots are shown in the left.

moved a circle in low value area (blue) while the robot failed to even touch the bottle cap. From iterations 3 to 9, KDPP smoothly explored unknown areas and progressively updated its approximated value function to build a policy to move from the blue area to the red. Correspondingly, the robot performs increasingly successful movements towards unscrewing the bottle cap.

5.4 Summary of KDPP

In this section, KDPP is intriduced as an extension of LUDPP to approximate the value function by infnite number of features which is implicitly calculated by kernel function. As a practical extension of DPP and kernelized value function approach based reinforcement learning methods, KDPP stabilizes value function approximation by adding the Kullback-Leibler divergence as a regularization term to keep policy updates smooth. Exploiting this smoothness, KDPP successfully utilizes the kernel trick to represent and update the high dimensional approximate value function using generated samples which considerably reduces computational complexity. From our simulation results KDPP was able to learn viable solutions in very high dimensional systems with a small number of samples, while other kernel value function approaches such as KLSPI could not. In the real robot experiment, we successfully apply KDPP to learn unscrewing bottle cap task in a complex system with 32 dimensional state space. It not only learned good policy, but also achieve very efficient calculation in both time and space.

6 Discussions

The framework proposed in this thesis combines smooth policy update and different machine learning algorithms to solve both the curses of insufficient samples and dimensionality in order to control robot systems with high dimensional state space. According to the results in this study, the framework outperformed several conventional methods and show its potential in robot control domain. On the other hand, the current frameworks has several limitations in both algorithm and application that need to be improved further. We discuss them in this chapter.

6.1 Related Works

The proposed framework is to update the value function from local area of samples. To keep samples generated in a local area, we employ the Dynamic Policy Programming (DPP) [36–38] which firstly add the Kullback-Leibler divergence into the value function update. We extend DPP to robot control domain by combining it with sample reuse and softmax exploration. Furthermore, different machine learning methods, NNS and kernel trick, are introduced to work with DPP to efficiently approximate the value function in a local area and therefore be practical in complex robot systems. Our work can be viewed as a value function approach version of Relative Entropy Policy Search [23, 99] and Guided Policy Search [25, 100] that are all policy search algorithms using Kullback-Leibler divergence. While avoiding several limitations of both the value function approach and the policy search, the proposed framework still has some issues on both algorithm and real robot control parts.

6.2 Open Issues in Algorithm

6.2.1 Support of Continuous Actions

The first issue is the lack of supporting continuous and/or large number of actions. Be similar to all other value function approach algorithms, the current framework naturally support only discrete actions. It could not efficiently approximate value function with huge number or continuous actions since it calculate the action preference function over all actions. To extend the value function approach to support continuous actions, One popular solution is the deterministic policy gradient (DPG) [101]. It maintains a parameterized actor function $\mu(\boldsymbol{s}, \theta^{\mu})$ which specifies the current policy by deterministically mapping states to a specific action. This actor function could be obtained by applying gradient descent methods besides the learning of Q function. Another solution comes from Continuous-action Approximate Policy Iteration (CAPI) [102]. It using kernel functions and linear polynomials to get the gradient information of a differentiable approximated value function for policy search in continuous action space. However, it is challenging to apply both DPG and CAPI to DPP since the softmax function over all actions required by DPP becomes intractable with continuous actions. Moreover, it is still intractable for the methods above to handle high dimensional continuous actions.

6.2.2 Combination with Deep Reinforcement Learning

Another interesting issue is to apply the proposed framework to deep reinforcement learning [46–50] which usually learn features with high dimensional raw image via deep neural networks [103] in robot control domain. Since sufficient training samples are too arduous for physical robot systems, our framework may contribute to a faster convergence with limited number of samples in deep reinforcement learning. Our first step work called Deep Dynamic Policy Programming has been in [51]. In this work, we combine deep reinforcement learning with DPP to control a humanoid robot to learn flipping deformable objects with sample efficiency where the only input state is the high dimensional raw image captured by camera.

6.2.3 Utilization of the Kernel

The next issue is to further explore the power of the kernel method in our framework since the kernel trick used in KDPP is not yet developed sufficiently. For example, the parameters of kernel function is not carefully selected in this thesis. Only the Gaussian kernel function is considered in current work, while other kernel functions also contribute in kernel value function approach [102]. Moreover, applying kernel embeddings [104, 105] to smoothly learn the value function and an implicit dynamics model from limited samples should be a suitable strategy in robot control domain. Following the related works in [106, 107], it will be interesting to estimate the dynamics model during learning of KDPP which turns the current framework from model-free to model based one.

6.3 Open Issues in Robot Control

6.3.1 Multiple Targets Task

On limitation of the current study is the only one target task in both simulation and real robot experiments, i.e., one target position to reach and unscrewing bottle cap in order to simplify the experiments. It is possible and interesting to extend the proposed framework to multiple tasks, e.g., controlling the finger to reach multiple positions, learning unscrewing/screwing bottle cap, by designing several reward (cost) functions and applying the proposed framework to learn generative solutions of complex tasks following [108].

6.3.2 Better Designed Action Set

It is also important to extend the action set to more complex and better designed one, e.g., defining the meaningful predefined patterns in octopus arm simulation [92] and applying muscle synergies [109]. For a wider range of applications, we intend to apply the proposed framework to more interesting tasks like working in kitchen. Since the proposed algorithms are value function based approach that support discrete actions and has a global view of the whole task, they are suitable to be a rough decision maker in a large robot learning system where the details

of actions are achieved by other algorithms like policy search. One good example is to build action set by robot movement learned by EaIP proposed in Appendix, then learn challenging tasks using the proposed framework.

6.3.3 Application in Other Areas

Lastly, we would like to continue applying the proposed algorithms in different algorithms. One good example is our cooperation work with University of Technology, Sydney in robot wheelchair driving assistance [52]. In this work, we applied the smooth policy update of DPP to help disabled and aged peoples to drive their wheel chair easily by softly considering both the user input and experts' demonstrations.

7 Conclusions

A new reinforcement learning framework is proposed in this thesis that aims to practical applications of model-free reinforcement learning in controlling complex robot systems with limited number of real world samples and a tractable computational complexity. Taking the advantages of both value function approach and the policy search, the proposed framework updates and approximates the value function from samples generated by a Kullback-Leibler divergence based smoothly updated exploration policy. It can be viewed as a local samples space and turns to a stable learning when the samples is insufficient to cover the whole state-action space and therefore provide foundation to run other machine learning algorithms to efficiently approximate the value function. While policy search algorithms with Kullback-Leibler divergence (e.g., Relative Entropy Policy Search [23, 99] and Guided Policy Search [25, 100]) are successful in the robot control literature, to the best our knowledge, this study is the first time to explore the capability of learning the global value function via a local viewpoint by employing the Kullback-Leibler divergence in robot control domain. Compared with the most popular solution in the robot control domain, the policy search that benefits from well selected initial policies and naturally supports continuous actions, the proposed framework is a good alterative solution when a carefully selected parameterized policy according to different tasks is unavailable. It is pure model-free and self-exploration and can be used without any prior knowledge while they are difficult to be applied to control problems with continuous actions and large action space.

In this thesis, two corresponding algorithms, LUDPP and KDPP, are proposed as examples. Both of them share the smooth policy update strategy, LUDPP limits the update of approximate value function in a local area selected by the current samples of state and action every iteration via Nearest Neighbor Search

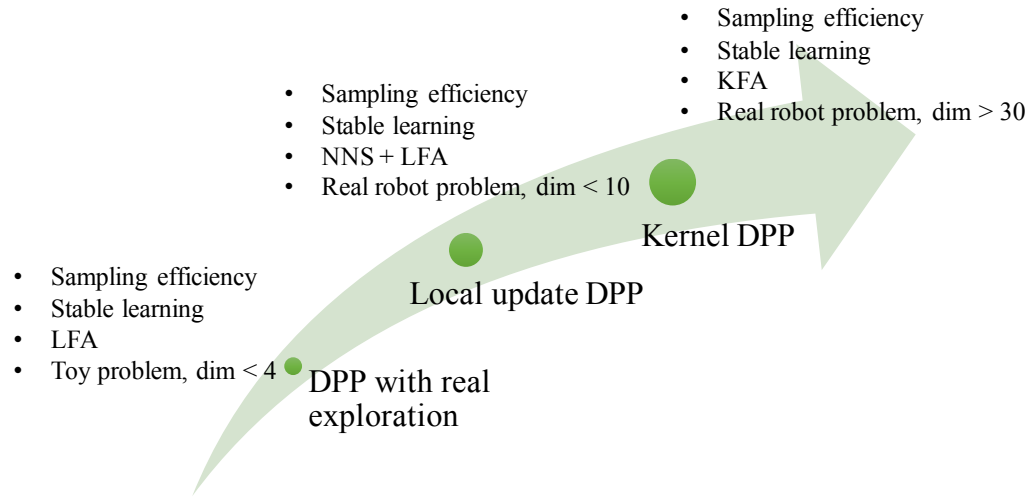


Figure 7.1: The overall view of all proposed methods in this thesis.

(NNS) [39] rather than the whole state-action space; As a further update of LUDPP, KDPP employs the kernel trick [40] to implicitly represent and update the approximate value function of high dimensional state spaces using the inner product of pairs of generated samples, i.e., adaptively selecting basis functions from samples to efficiently approximate the function among the current known state-action space.

The learning performances of both LUDPP and KDPP were firstly investigated by simulation tasks, N DOF manipulator reach control with comparison of the conventional algorithms without smooth policy update. Applying NNS, LUDPP is able to achieve 4 DOF reaching tasks by updating only about 5% of 10^5 RBFs which is required by common linear function approximation. On the other hand, the local update trick could not work on the compared algorithm without smooth policy update. For KDPP, it is able to achieve even 40 DOF manipulator reaching task with a only around 2500 RBFs, as comparison, common linear function approximation need more than 10^{40} RBFs which is intractable to any current computational resource. As comparison, all other kernel based value function approaches could not learn in such a challenging task without smooth policy update.

Then the two proposed algorithms are applied to Shadow Dexterous Hand [41],

a Pneumatic Artificial Muscle (PAM) driven humanoid robot hand as real applications: LUDPP was applied to a two DOF finger reaching control. This task has a 12-dimensional state-action space and therefore requires a huge number of basis functions to approximate the value function. It successfully learned a good control policy within 20 iterations and only updated less than 10% basis functions. KDPP was applied to the task of unscrewing a bottle cap using Shadow Dexterous Hand. The whole system has a 32 dimensional state space with 625 discrete actions. Generating only 500 samples per iteration, KDPP successfully learned viable solutions within 10 iterations. The average size of the implicit features using the kernel trick is only less than 3000 whereas $\gg 10^{20}$ RBFs are required for linear function approximation in conventional value function approach based algorithms. These results indicate both LUDPP and KDPP's potential for application on other complex robot systems. Figure 7.1 summarize the proposed methods with their performances in this thesis.

In summary, the contribution of this thesis is twofold. For the part of algorithm, this research explored the potential of applying value function approach to complex robot control problems and provided solutions to the curses of insufficient samples and dimensionality. We proposed a prior model knowledge free framework to face the limitations of the policy search. For the part of real robot experiment, we successfully applied the proposed algorithms to control Pneumatic Artificial Muscle (PAM) driven robot to learn challenging task like unscrewing bottle cap. Our framework does not require huge number of samples and knowledge of tasks and models. It turns to an efficient solution to model-free learning control of a variety of PAM driven devices, e.g., wearable robots that physically interact to the humans for supporting their daily activities or rehabilitation purposes.

A Appendix: Environment-adaptive Interaction Primitives

A.1 Introduction

In this chapter, we introduce Environment-adaptive Interaction Primitives (EaIPs), a cooperation research between Nara Institute of Science and Technology and University of Technology, Sydney. The purpose of EaIPs is to design a learning-from-demonstration framework for human-robot cooperative tasks with additional environmental conditions (e,g, the size of target object). After learning from training samples of both human and robot movement to finish a task under different environmental settings, the framework must be able to predict suitable robot motor skills to satisfy both a short partner observation period and novel environmental conditions.

EaIPs build upon the Interaction Primitives [110] (IPs) framework, which allow for a robot and human to perform collaborative tasks by converging upon a suitable parameter set for the execution of Dynamic Movement Primitives [28] (DMPs) after observing some initial period of human partner movement. In order to allow IPs to function in more complex situations, we integrate environmental parameters about the task to be accomplished into the parameter inference step for Environment-adaptive Interaction Primitives (EalPs) (Fig A-1), which give inferences that consider both partner behavior and parameters describing environmental conditions. This is more aligned with the approach a human would undertake when collaborating with another person; information about the best action for them to perform must not come solely from his partner, but also from

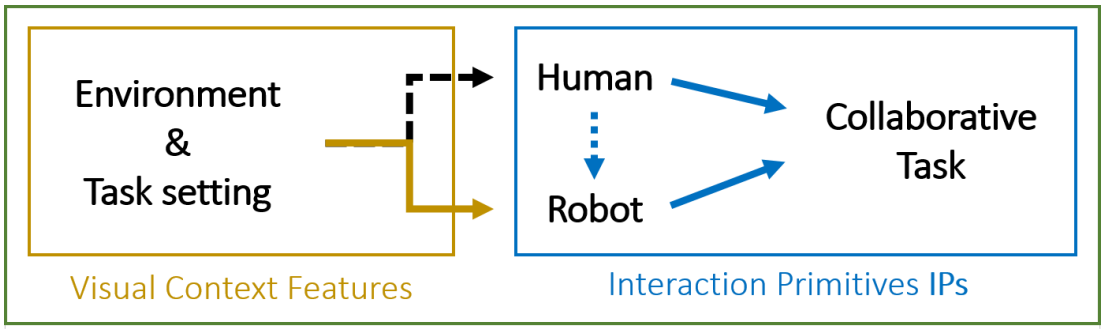


Figure A-1: Schematic diagram of EaIPs: an extension of IPs to adapt to environmental conditions. In IPs, a robot predicts trajectory parameters to cooperate with a human partner after observing a brief movement period. EaIPs enable robots to consider additional environmental conditions during trajectory prediction.

their shared environment. The impact of this contribution is twofold; the first being that the delay caused by partner observation can be significantly reduced, which allows for more immediate and fluent robot motion in situations where the human actions are ambiguous in the initial moments. The second effect is the capacity of adapting different environmental conditions in prediction. When faced with novel environmental parameters the prediction is inevitably tuned to parameters that allow for a similar basic trajectory structure, whereas conventional IPs require additional training data in scenarios where scaling a DMP on human motion no longer ensures safety. These improvements allow the EaIPs framework to be robust against poor inferences from noisy observations of partner behavior.

A.2 Approach

A.2.1 Dynamic Movement Primitives

Encoding trajectory of human or robot movements, DMP [28] is formally written as a dynamic system:

$$\ddot{y}(t) = \Big(\alpha_y\Big(\beta_y\big(g - y(t)\big) - \big(\frac{\dot{y}(t)}{\tau}\big)\Big) + f(x_t)\Big)\tau^2 \tag{A.1}$$

where α_y and β_y are constants, y is the state variable of the trajectory, g is the target position, τ is a time constant and t is the time step. $f(x_t)$ is the forcing function built by M Gaussian basis functions and a corresponding $M \times 1$ weights vector $\boldsymbol{w}$:

$$f(x_t) = \frac{\sum_{i=1}^{M} \psi_i(x_t) w_i x_t}{\sum_{j=1}^{M} \psi_j(x_t)} = \phi(x_t)^T \boldsymbol{w}, \tag{A.2}$$

x follows a canonical system: $\dot{x} = -\alpha_x x \tau$ where $x_0 = 1$.

To learn a weight vector $\boldsymbol{w}$ of DMP encoding a T step trajectory $\boldsymbol{y} = [y(t), \dot{y}(t), \ddot{y}(t)]_{t=1:T}^T$, the forcing function that reproduces the sample trajectory from the t-th step is calculated according to Eq. (A.1):

$$f(x_t) = \frac{1}{\tau^2}\ddot{y}(t) - \alpha_y\Big(\beta_y\big(g - y(t)\big) - \frac{\dot{y}(t)}{\tau}\Big). \tag{A.3}$$

The system can be resolved with $\boldsymbol{f} = \boldsymbol{\Phi}\boldsymbol{w}$ where $\boldsymbol{\Phi} = [\phi(x_1), ..., \phi(x_T)]^T$ and $\boldsymbol{f} = [f(x_1), ..., f(x_T)]^T$. Its least squares solution follows:

$$\boldsymbol{w} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{f} \tag{A.4}$$

A.2.2 Interaction Primitives in human-robot cooperation tasks

According to [110], applying Interaction primitives (IPs) to human-robot cooperation tasks has two steps: 1. Estimating phase of observed human movement. 2. predicting robot motor skills with a partially observation of only human's movement.

Dynamic Time Warping (DTW) [111] is employed to estimate the phase of observed human movement. Given one partially observed human movement $[\boldsymbol{y}_1^*, ..., \boldsymbol{y}_{T'}^*]^T$ and one reference movement $[\boldsymbol{y}_1, ..., \boldsymbol{y}_T]^T$, the full human movement during the original demonstration of the task. DTW measures the similarity between these two temporal sequences and provides the index t^* reflecting the frame in the reference movement which produces minimal costs with respect to the observed query movement, i.e., $[\boldsymbol{y}_1^*, ..., \boldsymbol{y}_{T'}^*]^T$ is close to $[\boldsymbol{y}_1, ..., \boldsymbol{y}_{t^*}]^T$. The estimated phase of partially observed human movement is therefore:

$$x^* = \exp\left(- \alpha_x \left(\frac{t^*}{T} \right) \tau \right). \tag{A.5}$$

For the prediction of robot motor skills with a partially observation of human's movement using IPs, we firstly prepare S sets of N DoFs trajectories that are time-scaled to the same length T as the training samples:

$$\boldsymbol{Y} = [\boldsymbol{Y}_{human}, \boldsymbol{Y}_{robot}] = \begin{bmatrix} \boldsymbol{y}_1^1 & ... & \boldsymbol{y}_N^1 \\ \vdots & \ddots & \vdots \\ \boldsymbol{y}_1^S & ... & \boldsymbol{y}_N^S \end{bmatrix} \tag{A.6}$$

where N is the totally number of DoFs for both human and robot. Defining $\boldsymbol{y}_i^j$, $\boldsymbol{w}_i^j$ and g_i^j as the trajectory, weights vector and target position of the i-th DoF in the j-th demonstration respectively, $\boldsymbol{\theta}^{[j]} = [\boldsymbol{w}_1^{jT}, g_1^j, ..., \boldsymbol{w}_N^{j\,T}, g_N^j]^T, j = 1, ..., S$ is the DMPs parameter vector learned from $[\boldsymbol{y}_1^j, ..., \boldsymbol{y}_N^j]$. Thus $p(\boldsymbol{\theta})$, the distribution among the parameter vector samples $\boldsymbol{\theta}^{[j]}, j = 1, ..., S$, follows:

$$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}}), \tag{A.7}$$

$$\boldsymbol{\mu}_{\boldsymbol{\theta}} = \frac{\sum_{j=1}^S \boldsymbol{\theta}^{[j]}}{S}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \frac{\sum_{j=1}^S (\boldsymbol{\theta}^{[j]} - \boldsymbol{\mu}_{\boldsymbol{\theta}})(\boldsymbol{\theta}^{[j]} - \boldsymbol{\mu}_{\boldsymbol{\theta}})^T}{S}. \tag{A.8}$$

Note that $\boldsymbol{\theta} = [\boldsymbol{\theta}_{human}, \boldsymbol{\theta}_{robot}]^T$ contains the parameter vectors of both human and robot.

Partially observing human's movement and estimating its phase x^* according to a reference movement by DTW, the trajectories $\boldsymbol{Y}_{human}^* = [\boldsymbol{y}_1^*, ..., \boldsymbol{y}_n^*]^T$ are resampled from the observed movement where $n < N$ is the DoFs of human

movement. The unavailable trajectories of robot $\boldsymbol{Y}^*_{robot}$ are set to $\mathbf{0}$. Defining $\boldsymbol{Y}^* = [\boldsymbol{Y}^*_{human}, \boldsymbol{Y}^*_{robot}]$, the prediction of both human and robot's parameter vector is represented by:

$$p(\boldsymbol{\theta}|\boldsymbol{Y}^*) \propto p(\boldsymbol{Y}^*|\boldsymbol{\theta})p(\boldsymbol{\theta}). \tag{A.9}$$

The likelihood $p(\boldsymbol{Y}^*|\boldsymbol{\theta})$ is modeled by a Gaussian distribution of the forcing function:

$$p(\boldsymbol{Y}^*|\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{F}^*|\boldsymbol{\Omega\theta}, \sigma^2\boldsymbol{I}) \tag{A.10}$$

where $\boldsymbol{F}^*$ has two parts: $\boldsymbol{F}^*_{human} = [\boldsymbol{f}^*_1, ..., \boldsymbol{f}^*_n]^T$ is the observed forcing function of $\boldsymbol{Y}^*_{human}$, its element is given by:

$$f^*_i(x_t) = \frac{1}{\tau^2}\ddot{y}^*_i(t) - \alpha_y\Big(- \beta_y y^*_i(t) - \frac{\dot{y}^*_i(t)}{\tau}\Big). \tag{A.11}$$

$\boldsymbol{F}^*_{robot}$ is the unavailable forcing function of robot and set as $\mathbf{0}$. The matrix $\boldsymbol{\Omega\theta}$ contains the forcing function with relationship to $\tilde{\boldsymbol{\Phi}}_t = [\phi(x_t)^T, \alpha_y\beta_y]$ over learning samples for $1 \leq t \leq t^*$:

$$\boldsymbol{\Omega\theta} = \begin{bmatrix} \tilde{\boldsymbol{\Phi}} & 0 & ... & ... \\ 0 & \tilde{\boldsymbol{\Phi}} & 0 & ... \\ \vdots & \vdots & \vdots & \vdots \\ 0 & ... & ... & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_1 \\ g_1 \\ \vdots \\ \boldsymbol{w}_N \\ g_N \end{bmatrix} \tag{A.12}$$

with the $\tilde{\boldsymbol{\Phi}}$ related to $\boldsymbol{\theta}_{robot}$ in $\boldsymbol{\Omega}$ being set to 0. σ^2 is the observation noise variance.

The joint distribution $p(\boldsymbol{Y}^*, \boldsymbol{\theta})$ is also a Gaussian distribution given the likelihood $p(\boldsymbol{\theta}|\boldsymbol{Y}^*)$:

$$\mathcal{N}\left(\begin{bmatrix} \boldsymbol{F}^* \\ \boldsymbol{\theta} \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\Omega\theta} \\ \mu_\theta \end{bmatrix}, \begin{bmatrix} \boldsymbol{A} & \boldsymbol{\Sigma}_\theta\boldsymbol{\Omega}^T \\ \boldsymbol{\Omega\Sigma}^T_\theta & \boldsymbol{\Sigma}_\theta \end{bmatrix} \right) \tag{A.13}$$

where $\boldsymbol{A} = \sigma^2\boldsymbol{I} + \boldsymbol{\Omega\Sigma}_\theta\boldsymbol{\Omega}$. The mean and variance of conditional distribution $p(\boldsymbol{\theta}|\boldsymbol{Y}^*)$ is derived as:

$$\begin{aligned} \mu_{\theta|\boldsymbol{y}^*} &= \mu_\theta + \boldsymbol{\Sigma}_\theta\boldsymbol{\Omega}^T A^{-1}(\boldsymbol{F}^* - \boldsymbol{\Omega}\mu_\theta), \\ \Sigma_{\theta|\boldsymbol{y}^*} &= \Sigma_\theta - \boldsymbol{\Sigma}_\theta\boldsymbol{\Omega}^T A^{-1}\boldsymbol{\Omega\Sigma}_\theta. \end{aligned} \tag{A.14}$$

After obtaining $\boldsymbol{\theta}$, the robot motor skills are operated by running DMPs with parameter vector $\boldsymbol{\theta}_{robot}$ with estimated phase x^*.

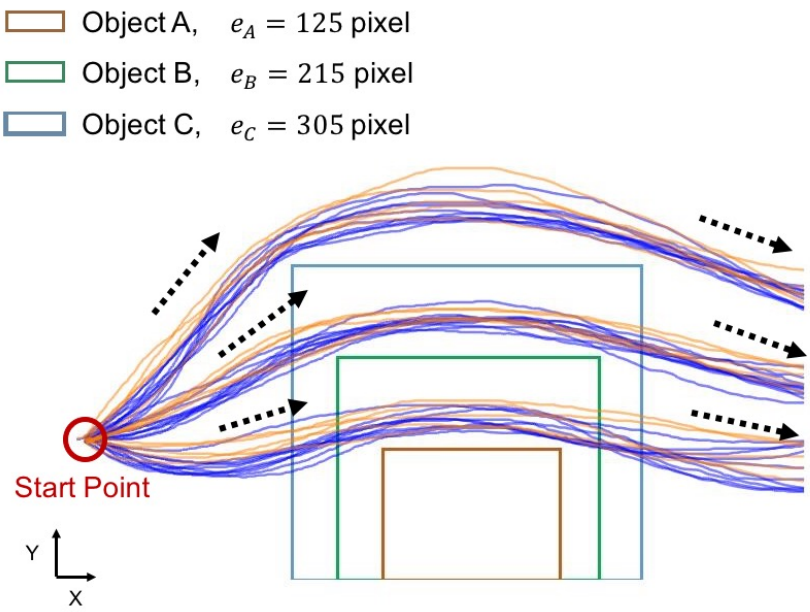


Figure A-2: Trajectories of training samples (blue) and testing samples (orange) to cross three objects on a 2D plan in simulation.

A.3 Simulation Results

EalPs are firstly applied to a toy simulation that simplifies our task of covering objects with a plastic bag in three-dimensional space, to passing over rectangular objects of different size in a two-dimensional space. As shown in Fig. A-2 there are three rectangular objects, each allocated an environmental attribute as a measure of their height: 125, 215 and 305 pixels respectively. 30 training trajectories (blue) and 15 testing (orange) trajectories (each with 500 steps) are generated by hand. All trajectories in simulation have two DoFs (X and Y axis). We compare the inference accuracy of IPs and EalPs between two situations: a full observation (both horizontal and vertical movement) and a partial observation (horizontal movement only) over 15 testing trajectories with different lengths of observed trajectories. The inference accuracy is represented by the average dynamic time warping distance between the predicted and original trajectories.

According to the results shown in Fig. A-4, EalPs considerably improve the inference accuracy in this simulation. In the full observation case where both X and

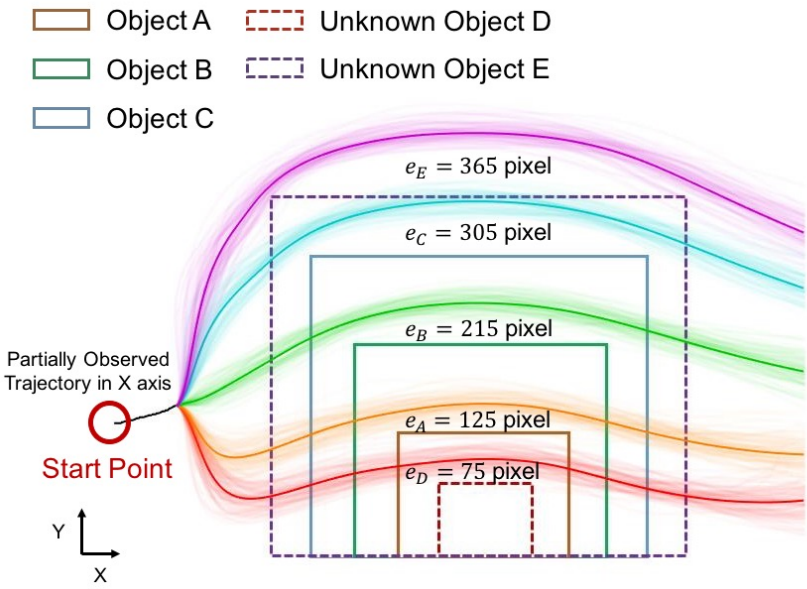
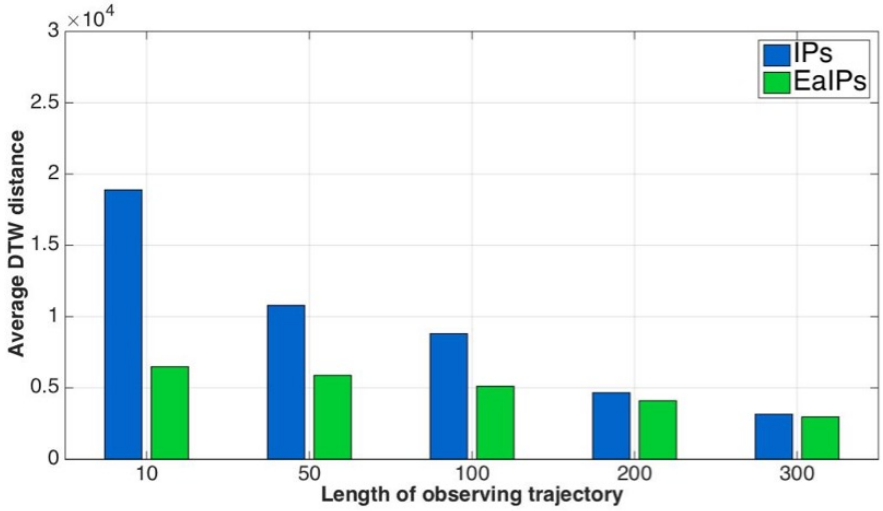


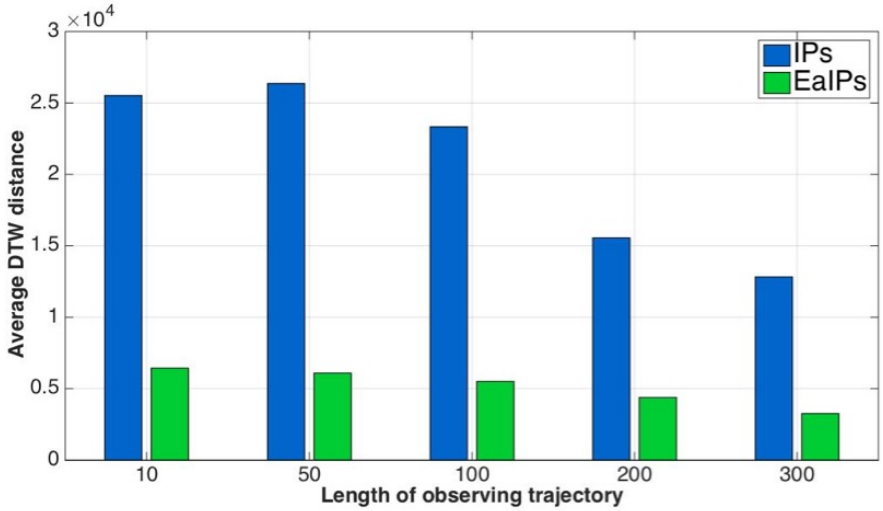
Figure A-3: Predicted trajectories of one testing samples ($t^* = 100$, only X axis is observable) with different environmental parameters. (Objects D and E are not included in training samples)

Y axis trajectories are observable, both IPs and EalPs have better prediction with the increase of the observed trajectories' length. On the other hand EalPs predict better than IPs with shorter observed trajectories ($t^* \leq 100$ steps) because the environment parameter indicates the target object's class, whereas the early steps in both horizontal and vertical movements contain less information to determine which object is being covered. As all trajectories in X axis are very similar in Fig. A-2, the prediction is more challenging in the partial observation case where only horizontal movements are available. IPs cannot predict accurately even after observing 60% of the trajectory while EalPs have almost identical performance to the full observation case, showing the environmental parameter's capability of improving prediction performance with partially observed trajectories that lack sufficient information.

The environment parameter not only improves the inference accuracy, but also enables EalPs to generalize trajectories for tasks with unknown environmental setting according to training samples. Given a 100 steps test trajectory in the



(a) The average inference accuracy of full observation.



(b) The average inference accuracy of partial observation.

Figure A-4: Comparison of trajectory prediction accuracy between IPs and EaIPs. DTW distance (vertical axis) is a unitless measure of error between two temporally aligned signals.

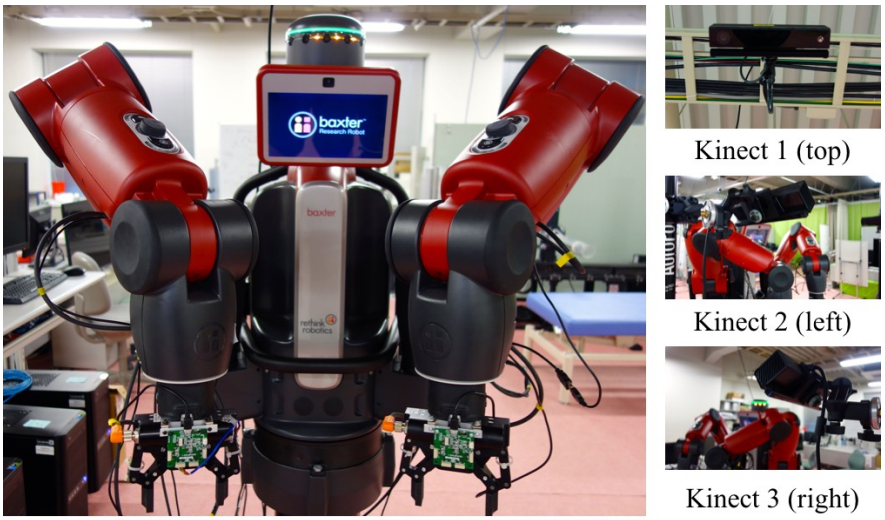
single observation case, trajectories based on different environmental parameters are shown in Fig. A-3 where 100 trajectories generated by $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$ for each object are represented by transparent lines, and solid lines represent the mean trajectories. We add new objects D and E that are not included in training samples to investigate EalPs' generalization ability. This shows the EalPs' capability of generating suitable trajectories to new situations; EalPs successfully obtain movement style from samples and automatically adjust the scale of trajectories to fit new environments according to the relationship between the trained and new environmental parameters.

These simulation results show the potential of EalPs in the early stages of movement inference of human-robot tasks. EalPs are able to give accurate prediction with short and partially observed trajectories from a very brief observation and can generate suitable co-operative robot movement. Moreover, EalPs adjust their prediction to fit new environments based on the knowledge learned from training samples. It reduces the requirement of training samples to cover many possible cases, while still allowing robots to intelligently co-operate with humans when dealing with various objects.

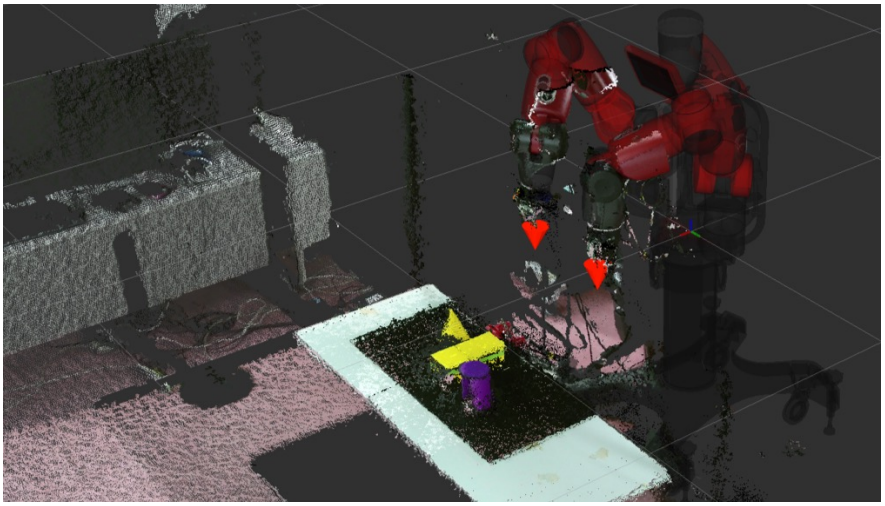
A.4 Experimental Results

For validation in a real experiment, the NAIST Baxter robot (Fig A-5) was made to hold one corner of a large plastic bag stationary with its right gripper, while its left gripper held another corner of the bag and swept it over a large object to cover it in tandem with his partner, who moves both hands. The NAIST Baxter has three Kinect V2 sensors (Fig A-5a) providing visualization of the robot from left, right and birds-eye views (Fig A-5b). Brightly colored cleaning gloves are worn by the interaction partner throughout gathering of training data and testing, so the Kinects can track the partner's hands in real-time. Data communication and logging was managed via the Robot Operating System (www.ros.org) middleware.

Five training trajectories were gathered for each of three objects (Fig A-6a): a 75 cm high stool, an 80 cm high office chair and a 90 cm high cabinet. Sample trajectories (Fig A-6b-A-6c) containing the poses of the Baxter's grippers and the positions of the partner's hands were then cut and interpolated in Matlab to be of



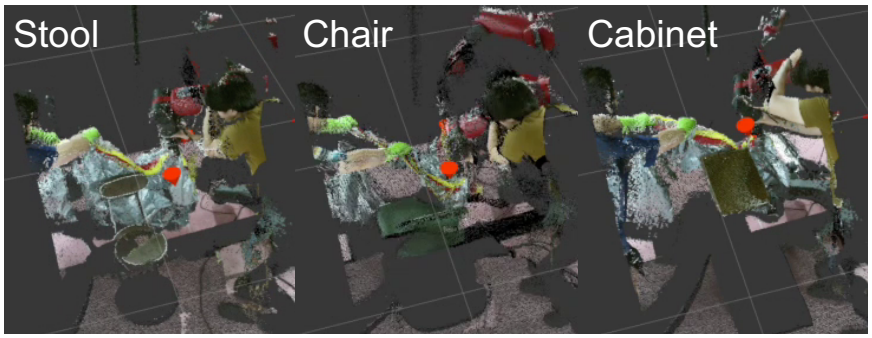
(a) The Baxter research robot with three Kinect V2.



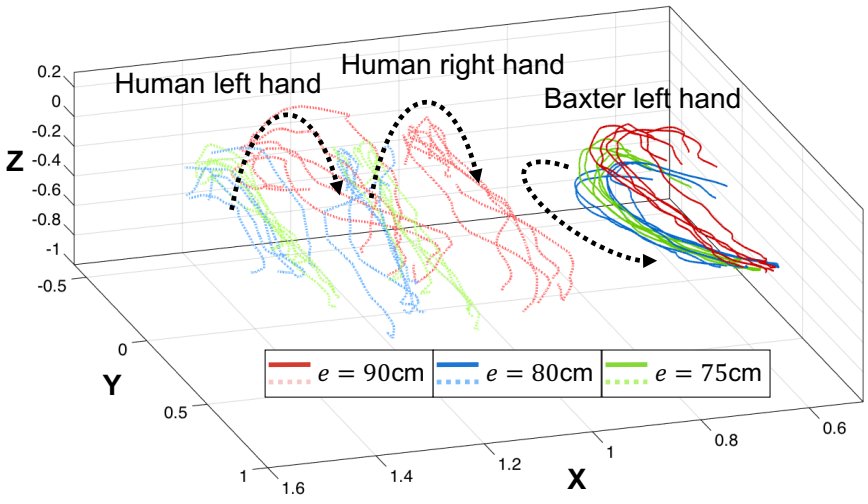
(b) Point cloud generated by Kinects.

Figure A-5: NAIST Baxter research robot learning system.

(a) Objects for training samples with e as height.



(b) Examples of training sample generation, captured by Kinect V2 sensors.



(c) 15 training sets (includes human left/right hand, and Baxter left hand gripper).

Figure A-6: Training trajectories for three objects.

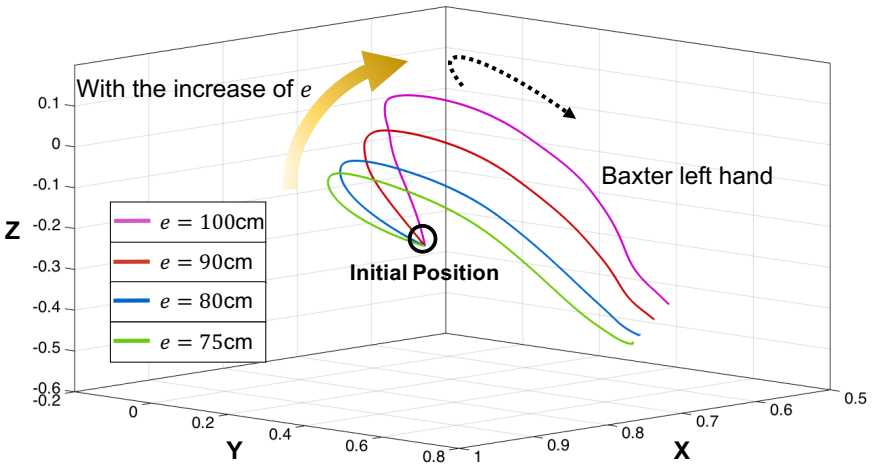


Figure A-7: Baxter left gripper trajectory from EalP across various objects.

uniform length for EalP training. For ease of differentiability, the imaginary axes of gripper orientation Quaternions were learned instead of their Euler equivalents. Figure A-7 shows the Baxter's left gripper trajectory in the execution of EalP for covering each of the three known objects with a new interaction partner, as well as a novel fourth object: the 90 cm high cabinet elevated by an additional 10 cm. It can be seen that each trajectory is suitably changed to satisfy each environmental parameter, and is robust to both inconsistent starting positions of the partner's hands and significant noise from glove tracking throughout their motion. Only 50 partner observations (1.5 seconds) were required to converge on the parameter sets that resulted in the trajectories shown. Figure A-8 shows object coverage by the interaction partner and the EalP-generated trajectories from Figure A-7, showing appropriate movements within the Baxter arm's range of motion.

A.5 Conclusion

This work presents an extension of the Interaction Primitives framework to allow for more natural human-robot interaction. By taking into account environmental conditions along with parameters describing human behavior, the robot can

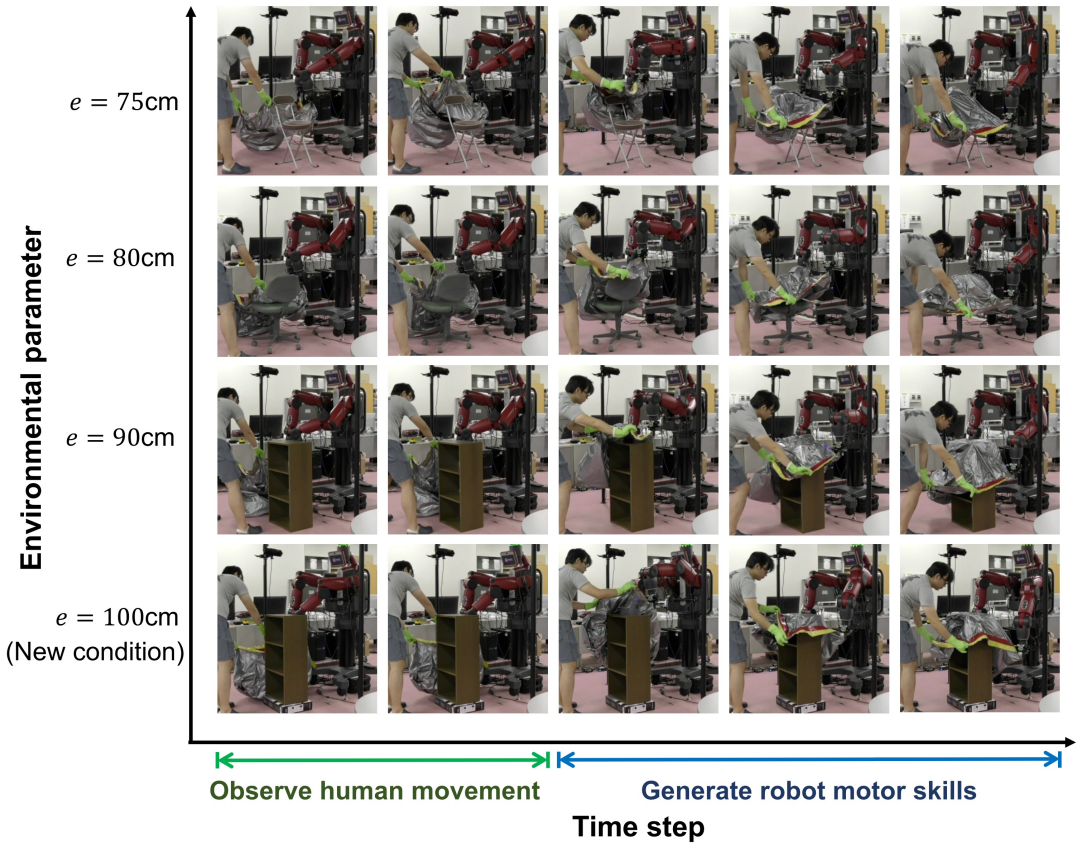


Figure A-8: Results of EalP trajectory generation in cooperative covering task with different environmental conditions.

quickly and confidently determine suitable parameters of motor skills even when faced with ambiguous behavior from the human partner during initial moments of their collaborative activity. The additional environmental parameter allows some knowledge of behavioral 'style' desirable of the robot to be learned independently from the human. Experimental results in both simulation of a toy problem and on a real humanoid platform show that these allow for fast and accurate task generalizations despite poorly informative early partner behavior. For future investigation we intend to pursue the application of dimensionality reduction techniques for learning more rigidly structured trajectories [112], and the tracking of more of the human partner's body for achieving increasingly complex collaborative human-robot activities with stochastic environmental parameters from sensor data. Also, we are going to combine EaIPs with deep neural networks [103] to learn from automatically generated multiple environment parameters.

Acknowledgements

My first and deepest gratitude goes to Associate Professor Takamitsu Matsubara for his encouragement, patience in supervision. He taught me how to become a true researcher via both his creative ideas and positive attitude to any challenge. I fully enjoy exploring with him. It is my proud to be his student and friend. Then I would like to express my heartfelt gratitude to Professor Kenji Sugimoto for his kind instruction and massive support. I will never forget his help and encouragement in my hard time during these three years. I also would like to thank other thesis committee members: Professor Tsukasa Ogasawara, Assistant Professor Masaki Ogura and Assistant Professor Taisuke Kobayashi, for their valuable comments and kind support.

I appreciate Associate Professor Jaime Valls Miro from University of Technology, Sydney for his supervision and beer when I was in Australia, Associate Professor Sang-Ho Hyon from Ritsumekan University for his taking care during my short visiting, and Assistant Professor Ming Ding from Robotics Laboratory, NAIST for his kind assist in experiments. Furthermore, my thanks would go to all members in Intelligent System Control Laboratory, NAIST for their kind help during these three years and the staff in NAIST office for their great job and assistance. Another gratitude goes to one of my best friends, and my best co-worker, Mr. James Poon for his great friendship. I enjoy our cooperation in both Japan and Australia.

My thanks would also go to my beloved parents for their loving considerations, great confidence, and other family members for their encouragement. Finally, my gratitude goes to my wife for her constant love, understanding and encouragement during my journey of PhD. Her daily talk via Internet helps me relax from stressful research while her tolerance of my occasional vulgar moods is a testament in itself of her unyielding devotion and love.

References

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

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

[4] S. Schaal, A. Ijspeert, and A. Billard, "Computational approaches to motor learning by imitation," *Philosophical transactions of the royal society of london b: biological sciences*, vol. 358, no. 1431, pp. 537–547, 2003.

[5] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[6] S. Schaal and C. G. Atkeson, "Learning control in robotics," *IEEE robotics & automation magazine*, vol. 17, no. 2, pp. 20–29, 2010.

[7] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The international journal of robotics research (IJRR)*, vol. 32, no. 11, pp. 1238–1274, 2013.

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

[9] C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[10] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," pp. 1038–1044, 1996.

[11] A. Nedić and D. P. Bertsekas, "Least squares policy evaluation algorithms with linear function approximation," *Discrete event dynamic systems*, vol. 13, no. 1-2, pp. 79–110, 2003.

[12] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The journal of machine learning research (JMLR)*, vol. 4, no. 44, pp. 1107–1149, 2003.

[13] M. G. Lagoudakis, *Efficient approximate policy iteration methods for sequential decision making in reinforcement learning*. PhD thesis, Duke University, Durham, NC, 2003.

[14] D. P. Bertsekas, "Approximate policy iteration: A survey and some new methods," *Journal of control theory and applications*, vol. 9, no. 3, pp. 310–335, 2011.

[15] R. Bellman, "Dynamic programming and lagrange multipliers," *Proceedings of the national academy of sciences*, vol. 42, no. 10, pp. 767–769, 1956.

[16] E. Uchibe, M. Asada, and K. Hosoda, "Cooperative behavior acquisition in multi-mobile robots environment by reinforcement learning based on state vector estimation," in *IEEE international conference on robotics and automation (ICRA)*, vol. 2, pp. 1558–1563, 1998.

[17] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and autonomous systems*, vol. 36, no. 1, pp. 37–51, 2001.

[18] 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.

[19] 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.

[20] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Neural information processing systems (NIPS)*, pp. 1057–1063, 2000.

[21] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[22] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[23] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *Association of the advancement of artificial intelligence (AAAI)*, pp. 1607–1612, 2010.

[24] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *The journal of machine learning research (JMLR)*, vol. 11, pp. 3137–3181, 2010.

[25] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *IEEE international conference on robotics and automation (ICRA)*, pp. 156–163, 2015.

[26] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review," *Neural networks*, vol. 21, no. 4, pp. 642–653, 2008.

[27] S. Schaal, "Dynamic movement primitives - a framework for motor control in humans and humanoid robotics," in *Adaptive motion of animals and machines*, pp. 261–280, 2006.

[28] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[29] R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, vol. 3, pp. 2849–2854, 2004.

[30] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," *Experimental robotics IX*, pp. 363–372, 2006.

[31] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 2219–2225, 2006.

[32] 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.

[33] 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.

[34] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *International conference on machine learning (ICML)*, pp. 465–472, 2011.

[35] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Neural information processing systems (NIPS)*, pp. 1–8, 2007.

[36] M. G. Azar, V. Gómez, and H. J. Kappen, "Dynamic policy programming with function approximation," in *International conference on artificial intelligence and statistics AISTATS*, pp. 119–127, 2011.

[37] M. G. Azar, V. Gómez, and H. J. Kappen, "Dynamic policy programming," *The journal of machine learning research (JMLR)*, vol. 13, no. 1, pp. 3207–3245, 2012.

[38] M. G. Azar, *On the theory of reinforcement learning: methods, convergence analysis and sample complexity*. PhD thesis, Radboud University Nijmegen, 2012.

[39] A. Andoni, *Nearest neighbor search: the old, the new, and the impossible*. PhD thesis, Massachusetts Institute of Technology, 2009.

[40] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[41] R. Walker, "Shadow dextrous hand technical specification," *Shadow robot company*, 2013.

[42] Y. Cui, T. Matsubara, and K. Sugimoto, "Local update dynamic policy programming in reinforcement learning of pneumatic artificial muscle-driven humanoid hand control," in *IEEE-RAS international conference on humanoid robots (Humanoids)*, pp. 1083–1089, 2015.

[43] Y. Cui, T. Matsubara, and K. Sugimoto, "Pneumatic artificial muscle-driven robot control using local update reinforcement learning," *Advanced robotics*, vol. 31, no. 8, pp. 397–412, 2017.

[44] Y. Cui, T. Matsubara, and K. Sugimoto, "Kernel dynamic policy programming: Practical reinforcement learning for high-dimensional robots," in *IEEE-RAS international conference on humanoid robots (Humanoids)*, pp. 662–667, 2016.

[45] 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.

[46] 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.

[47] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Association of the advancement of artificial intelligence (AAAI)*, pp. 2094–2100, 2016.

[48] 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 lLearning (ICML)*, pp. 1995–2003, 2016.

[49] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International conference on machine learning (ICML)*, pp. 1329–1338, 2016.

[50] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[51] Y. Tsurumine, Y. Cui, E. Uchibe, and T. Matsubara, "Deep dynamic policy programming for robot control with raw images," in *IEEE/RSJ international conference on intelligent robots and systems (IROS), accepted*, 2017.

[52] J. Poon, Y. Cui, J. Valls Miro, T. Matsubara, and K. Sugimoto, "Local driving assistance from demonstration for mobility aids," in *IEEE international conference on robotics and automation (ICRA)*, pp. 5935–5941, 2017.

[53] Y. Cui, J. Poon, T. Matsubara, J. V. Miro, K. Sugimoto, and K. Yamazaki, "Environment-adaptive interaction primitives for human-robot motor skill learning," in *IEEE-RAS international conference on humanoid robots (Humanoids)*, pp. 711–717, IEEE, 2016.

[54] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[55] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[56] D. P. Bertsekas, *Dynamic programming and optimal control*, vol. 2. Athena Scientific Belmont, MA, 1995.

[57] R. A. Howard, *Dynamic Programming and Markov Processes*. The MIT Press, Cambridge, Massachusetts, 1960.

[58] A. Farahmand, C. Szepesvári, and R. Munos, "Error propagation for approximate policy and value iteration," in *Neural information processing systems (NIPS)*, pp. 568–576, 2010.

[59] M. G. Lagoudakis, R. Parr, *et al.*, "Model-free least-squares policy iteration," in *Neural information processing systems (NIPS)*, vol. 14, pp. 1547—-1554, 2001.

[60] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and trends in robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[61] S. Kullback, *Information theory and statistics*. Courier Corporation, 1997.

[62] J. M. Joyce, "Kullback-leibler divergence," in *International encyclopedia of statistical science*, pp. 720–722, Springer, 2011.

[63] E. Todorov, "Linearly-solvable markov decision problems," in *Neural information processing systems (NIPS)*, pp. 1369–1376, 2006.

[64] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[65] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning," in *Recent advances in reinforcement learning*, pp. 33–57, 1996.

[66] C. Thiery and B. Scherrer, "Least-squares λ policy iteration: Bias-variance trade-off in control problems," in *International conference on machine learning (ICML)*, 2010.

[67] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1st ed., 1996.

[68] E. Uchibe and K. Doya, "Inverse reinforcement learning using dynamic policy programming," in *International conference on development and learning and on epigenetic robotics*, pp. 222–228, 2014.

[69] Y. Shen, A. Ng, and M. Seeger, "Fast Gaussian process regression using KD-trees," in *Neural information processing systems (NIPS)*, pp. 1225–1232, 2006.

[70] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," *Artificial intelligence review*, vol. 11, no. 1-5, pp. 75–113, 1997.

[71] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[72] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[73] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Annual symposium on computational geometry*, pp. 253–262, 2004.

[74] K. E. Atkinson, *An introduction to numerical analysis*. John Wiley & Sons, 2008.

[75] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuska, "Online least-squares policy iteration for reinforcement learning control," in *American control conference (ACC)*, pp. 486–491, 2010.

[76] L. Li, M. L. Littman, and C. R. Mansley, "Online exploration in least-squares policy iteration," in *International conference on autonomous agents and multiagent systems*, pp. 733–739, 2009.

[77] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," in *IEEE international conference on robotics and automation (ICRA)*, pp. 378–383, 2016.

[78] A. Gupta, C. Eppner, S. Levine, and P. Abbeel, "Learning dexterous manipulation for a soft robotic hand from human demonstrations," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 3786–3793, 2016.

[79] G. S. Sawicki, K. E. Gordon, and D. P. Ferris, "Powered lower limb orthoses: applications in motor adaptation and rehabilitation," in *IEEE/RAS-EMBS international conference on rehabilitation robotics (ICORR)*, pp. 206–211, 2005.

[80] N. G. Tsagarakis and D. G. Caldwell, "Development and control of a 'soft-actuated' exoskeleton for use in physiotherapy and training," *Autonomous robots*, vol. 15, no. 1, pp. 21–33, 2003.

[81] C. Chou and B. Hannaford, "Measurement and modeling of mckibben pneumatic artificial muscles," *IEEE transactions on robotics and automation*, vol. 12, no. 1, pp. 90–102, 1996.

[82] F. Daerden and D. Lefeber, "Pneumatic artificial muscles: actuators for robotics and automation," *European journal of mechanical and environmental engineering*, vol. 47, no. 1, pp. 11–21, 2002.

[83] K. Kogiso, K. Sawano, T. Itto, and K. Sugimoto, "Identification procedure for mckibben pneumatic artificial muscle systems," in *2012 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 3714–3721, 2012.

[84] M. Fliess and C. Join, "Model-free control," *International journal of control*, vol. 86, no. 12, pp. 2228–2252, 2013.

[85] P.-A. Gédouin, E. Delaleau, J.-M. Bourgeot, C. Join, S. A. Chirani, and S. Calloch, "Experimental comparison of classical pid and model-free control: position control of a shape memory alloy active spring," *Control engineering practice*, vol. 19, no. 5, pp. 433–441, 2011.

[86] L. dos Santos Coelho, M. W. Pessôa, R. R. Sumar, and A. A. R. Coelho, "Model-free adaptive control design using evolutionary-neural compensator," *Expert systems with applications*, vol. 37, no. 1, pp. 499–508, 2010.

[87] V. Milanés, J. Villagrá, J. Pérez, and C. González, "Low-speed longitudinal controllers for mass-produced cars: A comparative study," *IEEE transactions on industrial electronics*, vol. 59, no. 1, pp. 620–628, 2012.

[88] 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*, vol. 3, p. 5, Kobe, 2009.

[89] A. J. Smola and B. Schölkopf, *Learning with kernels*. GMD-Forschungszentrum Informationstechnik, 1998.

[90] X. Xu, T. Xie, D. Hu, and X. Lu, "Kernel least-squares temporal difference learning," *International journal of information technology*, vol. 11, no. 9, pp. 54–63, 2005.

[91] X. Xu, D. Hu, and X. Lu, "Kernel-based least squares policy iteration for reinforcement learning," *IEEE transactions on neural networks*, vol. 18, no. 4, pp. 973–992, 2007.

[92] T. Jung and D. Polani, "Kernelizing LSPE (λ)," in *IEEE international symposium on approximate dynamic programming and reinforcement learning*, pp. 338–345, 2007.

[93] G. Taylor and R. Parr, "Kernelized value function approximation for reinforcement learning," in *Annual international conference on machine learning (ACM)*, pp. 1017–1024, 2009.

[94] J. Mercer, "Functions of positive and negative type, and their connection with the theory of integral equations," *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, vol. 209, pp. 415–446, 1909.

[95] V. Vovk, "Kernel ridge regression," in *Empirical inference*, pp. 105–116, Springer, 2013.

[96] L. Buşoniu, A. Lazaric, M. Ghavamzadeh, R. Munos, R. Babuška, and B. De Schutter, "Least-squares methods for policy iteration," in *Reinforcement learning*, pp. 75–109, 2012.

[97] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of machine learning research (JMLR)*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[98] L. Van Der Maaten, "Accelerating t-SNE using tree-based algorithms," *Journal of machine learning research (JMLR)*, vol. 15, no. 1, pp. 3221–3245, 2014.

[99] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Hierarchical relative entropy policy search," *Journal of machine learning research (JMLR)*, vol. 17, no. 1, pp. 3190–3239, 2016.

[100] S. Levine and V. Koltun, "Guided policy search," in *International conference on machine learning (ICML)*, pp. 1–9, 2013.

[101] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning (ICML)*, pp. 387–395, 2014.

[102] X. Xu, C. Liu, and D. Hu, "Continuous-action reinforcement learning with fast policy search and adaptive basis function selection," *Soft computing-A fusion of foundations, methodologies and applications*, vol. 15, no. 6, pp. 1055–1070, 2011.

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

[104] K. Fukumizu, A. Gretton, G. R. Lanckriet, B. Schölkopf, and B. K. Sriperumbudur, "Kernel choice and classifiability for RKHS embeddings of probability distributions," in *Neural information processing systems (NIPS)*, pp. 1750–1758, 2009.

[105] L. Song, K. Fukumizu, and A. Gretton, "Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models," *IEEE signal processing magazine*, vol. 30, no. 4, pp. 98–111, 2013.

[106] G. Lever, J. Shawe-Taylor, R. Stafford, and C. Szepesvári, "Compressed conditional mean embeddings for model-based reinforcement learning.," in

Association for the advancement of artificial intelligence (AAAI), pp. 1779–1787, 2016.

[107] B. Dai, N. He, Y. Pan, B. Boots, and L. Song, "Learning from conditional distributions via dual embeddings," in *Artificial intelligence and statistics (AISTATS)*, pp. 1458–1467, 2017.

[108] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *IEEE international conference on robotics and automation (ICRA)*, pp. 3876–3881, 2014.

[109] 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.

[110] H. Ben Amor, G. Neumann, S. Kamthe, O. Kroemer, and J. Peters, "Interaction primitives for human-robot cooperation tasks," in *IEEE international conference on robotics and automation (ICRA)*, pp. 2831–2837, 2014.

[111] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.

[112] T. Matsubara, S.-H. Hyon, and J. Morimoto, "Learning parametric dynamic movement primitives from multiple demonstrations," *Neural networks*, vol. 24, no. 5, pp. 493–500, 2011.

Publication List

Journal

[1] Yunduan Cui, Takamitsu Matsubara and Kenji Sugimoto, "Pneumatic artificial muscle-driven robot control using local update reinforcement learning," Advanced Robotics, vol. 31, No. 8, pp. 397-412, 2017.

[2] Yunduan Cui, Takamitsu Matsubara and Kenji Sugimoto, "Kernel Dynamic Policy Programming: Applicable Reinforcement Learning on Robot Systems with High Dimensional States," Neural Networks, vol. 94, pp. 13–23, 2017.

International Conference

[1] Yunduan Cui, Takamitsu Matsubara and Kenji Sugimoto, "An Empirical Comparison of Approximate Dynamic Policy Programming and LSPI for Simple Robot Motor Control Problems," SICE Annual Conference, pp. 798-803, China, 2015.

[2] Yunduan Cui, Takamitsu Matsubara and Kenji Sugimoto, "Local Update Dynamic Policy Programming in reinforcement learning of pneumatic artificial muscle-driven humanoid hand control," IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp. 1083-1089, Korea, 2015.

[3] Yunduan Cui, Takamitsu Matsubara and Kenji Sugimoto, "Kernel Dynamic Policy Programming: Practical Reinforcement Learning for High-dimensional Robots," IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp. 662-667, Mexico, 2016. [Best Oral Paper Award]

[4] Yunduan Cui, James Poon, Takamitsu Matsubara, Jaime Valls Miro, Kenji Sugimoto and Kimitoshi Yamazaki, "Environment-adaptive Interaction Primitives for Human-Robot Motor Skill Learning," 2016 IEEE-RAS International Conference on Humanoid Robots (Humanoids),pp. 711-717, Mexico, 2016.

[5] James Poon, Yunduan Cui, Jaime Valls Miro, Takamitsu Matsubara and Kenji Sugimoto, "Local Driving Assistance from Demonstration for Mobility Aids,"

IEEE International Conference on Robotics and Automation (ICRA), pp. 5935-5941, Singapore, 2017.

[6] 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), 2017. (accepted)