Active Incremental Learning of a Contextual Skill Model

Xiaopu Li

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 28.8.2019

Supervisor

Prof. Ville Kyrki

Advisor

M.Sc. Murtaza Hazara

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Xiaopu Li

Title Active Incremental Learning of a Contextual Skill Model

Degree programme Automation and Electrical Engineering

Major Control, Robotics and Autonomous Systems **Code of major** ELEC3025

Supervisor Prof. Ville Kyrki

Advisor M.Sc. Murtaza Hazara

Date 28.8.2019 **Number of pages** 46 **Language** English

Abstract

Contextual skill models enable robot to generalize parameterized skills for a range of task parameters by using regression on several optimal policies. However, the task difficulty and task sequence of learning a contextual skill model is usually neglected. Thus, the learning process is usually time consuming since some tasks might be easier to learn or the knowledge of these tasks might be easier to share with other tasks. In this thesis, we introduce active incremental learning framework for actively learning a contextual skill model based on dynamical movement primitives which are widely used to learn parameterized policies on trajectory level as a dynamical system for robot. The proposed framework will first select a task which maximizes the expected improvement in skill performance over entire task parameters and then optimize the corresponding policy with a fixed number of iterations in policy search. We model the learning rate of policy search for predicting reward improvement over a single iteration. We evaluated the improvement of the skill performance in two tasks, ball-in-a-cup and basketball, with a simulated KUKA robot arm. In both, the results show that active task selection can improve the skill performance continuously over a baseline.

Keywords contextual skill model, active incremental learning, dynamical movement primitives, policy search

Preface

I want to thank Professor Ville Kyrki and my instructor M.Sc. Murtaza Hazara for their great guidance and patience.

Otaniemi, 28.8.2019

Xiaopu Li

Contents

Symbols and abbreviations

Symbols

a	action	Def. 3.2.2
b	damping coefficient	Def. 3.1
D	database	Def. 2.3
$EISP$	expected improvement in skill performance	Def. 2.3
f	forcing function	Def. 3.1
f_{sample}	sampling function	Def. 3.3
$\mathcal{GP}$	Gaussian process	Def. 3.3
$J(t;\beta_J)$	learning rate model with hyper-parameter β_J	Def. 3.4.2
$j(\tau)$	predicted reward of task τ	Def. 2.3
k	kernel	Def. 3.3
k_{oc}	oscillatory coefficient	Def. 3.1
$\mathcal{N}$	Gaussian distribution	Def. 2.3
$P(\tau)$	probability of task τ	Def. 2.3
$R(\tau)$	reward function	Def. 2.3
$R(\tau;\beta_R)$	reward model with hyper-parameter β_R	Def. 3.4.2
$r(\tau;S)$	evaluated performance at task τ with skill model S	Def. 3.4.1
s	state	Def. 3.2.2
SP	skill performance	Def. 2.3
α_y	gain term	Def. 3.1
β_y	gain term	Def. 3.1
Δ	a fixed number of iterations in policy search	Def. 3.4.2
$\Delta R(\tau)$	reward improvement at task τ	Def. 3.4.2
θ	policy parameters	Def. 3.2.2
τ	task parameter	Def. 2.3

Operators

$\int$	integral	Def. 2.3
$\arg\max_\tau$	argument of the maximum	Def. 2.3
$\dot{y}$	the first derivatives of y	Def. 3.1
$\ddot{y}$	the second derivatives of y	Def. 3.1
$\sum_{i=1}^{N}$	sum over index i	Def. 3.1
$exp(x)$	exponential function	Def. 3.1
$\frac{\partial}{\partial t}$	partial derivative with respect to variable t	Def. 3.1

Abbreviations

ACES	Active Contextual Entropy Search
CPS	Contextual Policy Search
CSM	Contextual Skill Model
DMP	Dynamic Movement Primitives
DoF	Degrees of Freedom
EBNN	Explanation-Based Neural Network
EISP	Expected Improvement in Skill Performance
EM	Expectation Maximization
F/T	Force/Torque
GP	Gaussian Process
GPDMP	Global Parametric Dynamic Movement Primitives
GPR	Gaussian Process Regression
KB	Knowledge Base
KBL	Knowledge-Based Learner
LfD	Learning from Demonstration
LWR	Light-Weight Robot
LWRSIM	Light-Weight Robot Simulator
MuJoCo	Multi-Joint dynamics with Contact
PILCO	Probabilistic Inference for Learning Control
PI^2	Policy Improvement with Path Integral
PoWER	Policy Learning by Weighted Exploration with the Returns
RBF	Radial Basis Function
REFS	Relative Entropy Policy Search
RL	Reinforcement Learning
ROS	Robot Operating System
TM	Task Manager

1 Introduction

Human beings have a direct sensory contact with the environment. Therefore, we can construct the perception of the environment by performing actions and receiving the feedback from the environment. Reinforcement learning (RL) is a process in which agents simulate human learning. RL enables an agent to learn from scratch through a trial and error process. In this process, the agent can only take actions and receive feedback of each action by interacting with the dynamic environment. The goal of RL is to learn an optimal policy for the agent and maximize the long term reward. In RL, policy search methods can find the appropriate parameters of a parametrized policy for a given task parameter, therefore these methods are widely used to learn movement skills [1, 2]. However, the learned parametrized policy cannot be adapted to other task parameters directly, which results in different rewards for applying the same policy to different task parameters.

In order to overcome the generalization limitation, contextual skill models (CSMs) are widely used in generalization over a set of task parameters [3, 4, 5, 6, 7]. They first learn policies for a set of task parameters separately and then learn a CSM using regression, which maps task parameters to parametrized policies. However, in order to achieve a good performance, the learning process is usually costly for the agent in high-dimensional control problems. Incremental learning has been proposed to solve this problem which incrementally updates and stores the knowledge with tasks arriving sequentially.

On the other hand, contextual policy search (CPS) learns parametrized policies for a set of task parameters without separating learning according to a given distribution over task parameters [8, 9]. However, CPS does not consider the order of task execution, since the task parameter is given by a fixed distribution, which cannot be changed by the agent during the learning process.

The process of learning a CSM is usually passive and time consuming when the task parameters are selected manually or randomly, since some task parameters might be easier to learn or the knowledge of these task parameters might be easier to share with other task parameters. If we can select the most important task parameters by understanding the task, it will be helpful to the process of learning the model. In order to obtain a better CSM with less effort, active learning has been proposed. Active learning is widely applied in supervised learning to select the most important samples from unlabeled data and achieve a better learning model with lower cost of annotation. In reinforcement learning, active learning has been used based on Gaussian process regression [10]. They introduce a method of actively selecting the task that maximizes the expected improvement in skill performance. However, they do not consider the training difficulty. The learning process will be continued with a task until an optimal or near optimal policy is achieved.

In this thesis, we propose a novel active incremental learning method to actively

select a task step by step based on the current skill performance. Instead of learning a policy until optimal, we will update the skill model and actively select the next task after a fixed number of iterations. Although the current skill model is not optimal, the skill performance of the skill model increases gradually over time and converges in the end.

The remainder of the thesis is divided into five chapters. Chapter 2 introduces the related work of our research. Chapter 3 introduces the research material and methods in detail, including dynamic Movement Primitives, related concepts of reinforcement learning and the active learning algorithm. Chapter 4 introduces the experimental settings and two implemented tasks, ball-in-a-cup and basketball. Chapter 5 presents the results of our research. Chapter 6 summarizes this thesis.

2 Background

This chapter introduces the background of the related work of our research tasks. Sections 2.1 and 2.2 introduce two different methods for learning a contextual skill model (CSM) respectively. Then, Section 2.3 introduces active learning. Finally, incremental learning is presented in Section 2.4.

2.1 Generalization using Regression

Reinforcement learning (RL) is widely applied to learn a parametrized policy for robotic applications. Dynamical movement primitives (DMPs) is a kind of parametrized policy which has been developed to learn complex movements on trajectory level as a dynamical system. A contextual skill model (CSM) can provide parametrized policies for a range of tasks by using regression on limited optimal policies [3, 4, 5, 6, 11, 12]. Calinon et al [3] learned the CSM by using a parametric Gaussian mixture model and trained it using expectation maximization (EM) algorithm. However, the task parameters of the model can only be represented in the form of a coordinate system. Stulp et al. [4] learned parameterizable skills based on a novel DMP formulation, and used task parameters as inputs to the DMP function approximator. Forte et al. [5] utilized Gaussian process regression (GPR) for real-time generalization of example trajectories. Their method allows the robot movement to adapt to the perceived changes in the outside world. Ude et al. [6] applied locally weighted regression to compute the internal parameters of the DMPs in each trial and used GPR for mapping the task parameters to meta-parameters such as goal and duration of DMPs. Da Silva et al. [12] used regression algorithms to deduce a deterministic function which can generalize the learned policy parameters over the entire task parameters. Parametric CSM has also been proposed in [13], [11].

2.2 Generalization using contextual policy search

Policy search is used to find the appropriate parameters for one task with a given policy. Moreover, contextual policy search (CPS) can learn optimal policies for a set of tasks from scratch and has been used to learn a contextual skill model (CSM), which provides policies to tasks that has never been learned, by applying a regression algorithm [9, 8, 14]. In [9, 14], they used model-free approach with CPS to learn a CSM. Kupcsik [8] learned a CSM using model-based policy search with CPS. In both of these approaches the CSM is linear and modeled using a Gaussian function whose hyper-parameters are updated iteratively. However, CPS does not consider the task order, since the task parameter is given by a fixed distribution, which cannot be changed by the agent.

2.3 Active Learning

In reinforcement learning with robotic application, the robot is usually provided with a randomly or manually selected order of task parameters. However, some

tasks might be easier to learn or the knowledge of these tasks might be easier to transferred to similar but more difficult tasks. Therefore, active learning have been proposed to select a task parameter automatically. The goal of active learning is to select the next task parameter which maximizes future expected improvement of skill performance. Active learning has been proposed in [14, 15, 10]. We will elaborate on [10] which is the most relevant to our proposed approach.

In [14], they used heuristic intrinsic reward functions with a non-stationary multi-arm bandit to actively select the next task. In [15], they proposed active contextual entropy search (ACES) which is an information theoretic approach based on Bayesian optimization to minimizing uncertainty about the optimal policy parameters for task parameters without defining a heuristic function.

Da Silva et al. [10] proposed a non-parametric Bayesian approach of skill performance to actively select the next task that maximizes expected improvement in skill performance. They use a Gaussian process (GP) with the spatio-temporal kernel for modelling non-stationary reward $R(\tau)$ function. With the given kernel and current database $D_t = \{(\tau_1, r(\tau_1)), \ldots, (\tau_N, r(\tau_N))\}$, they can compute the Gaussian posterior $P(R_t(\tau)|\tau, D_t)$ over reward for any task parameter which is normally-distributed according to $R_t(\tau) \sim \mathcal{N}(\mu_t(\tau), \sigma_t^2(\tau))$, where $r(\tau_i)$ is the evaluated reward of optimal policy for previous selected task τ_i and t denotes time refers to the order in which tasks are practiced. They define the skill performance

$$SP_t = \int P(\tau)\mu_t(\tau)d\tau, \tag{1}$$

where $P(\tau)$ denotes the probability of task τ occurring. Furthermore, they introduced the Expected Improvement in Skill Performance (EISP) as an acquisition function to predict the expected improvement for a candidate task τ_c

$$EISP_t(\tau_c) = \int P(\tau')(\hat{\mu}_{t+1}(\tau') - \mu_t(\tau'))d\tau', \tag{2}$$

where $\hat{\mu}_{t+1}$ denotes the mean of Gaussian posterior $\hat{R}_{t+1}$ computed by updating $\hat{R}_t$ with the updated database $D_u = D_t \cup \{(\tau_c, j(\tau_c))\}$. $j(\tau_c)$ is the upper endpoint of the 95% confidence interval around the mean of current Gaussian posterior R_t:

$$j(\tau_c) = \mu_t(\tau) + 1.96\sqrt{\sigma_t^2(\tau)}. \tag{3}$$

The task τ^* will be identified by maximizing Equation 2:

$$\tau^* = \arg\max_{\tau} EISP_t(\tau). \tag{4}$$

2.4 Incremental Learning

Most machine learning algorithms are batch Learning, which requires obtaining and learning all the training samples at once. In fact, the number of samples tends to

increase gradually and the information reflected by these samples also changes over time. It is challenging to extract useful information from the ever-increasing samples. If the model is retrained each time when new samples arrive, the demand for time and space will increase rapidly. Thus, the traditional batch learning cannot satisfy this requirement. Incremental learning is an effective way to solve this problem which can incrementally update the knowledge and enhance previous knowledge without having to retrain all the data each time. Incremental learning indicates that the time cost of modifying a trained model is usually less than the cost of retraining a model when new data is added.

Incremental learning, which is also called lifelong learning, is an machine learning framework that incrementally updates the knowledge with tasks arriving sequentially [16, 17, 18, 19]. Thrun [16] described several lifelong learning approaches for supervised learning, such as memory-based learning approaches and explanation-based neural networks (EBNN) learning algorithm. Fei [19] proposed cumulative learning to detect and update new knowledge (classes or concepts) over time without re-training the model. In [18], they sped up model-free policy search by extracting uncertainty in the form of covariance matrices. Chen and Liu [17] introduced lifelong learning in detail (see Fig. 1).

Fig. 1 shows 4 key components of incremental learning: (I) Knowledge Base (KB) which stores the previously learned knowledge for tasks $\tau_1, \tau_2, \cdots, \tau_{N-1}$, (II) Knowledge-Based Learner (KBL) which learns the new task by using the previous knowledge in the KB, (III) a prediction model for application and (IV) a task manager (TM). TM can manage the arriving sequence of the tasks and assign a new task τ_N to the KBL. Then, KBL can use the past knowledge in KB to update a model and also store the new learned knowledge for τ_N to KB.

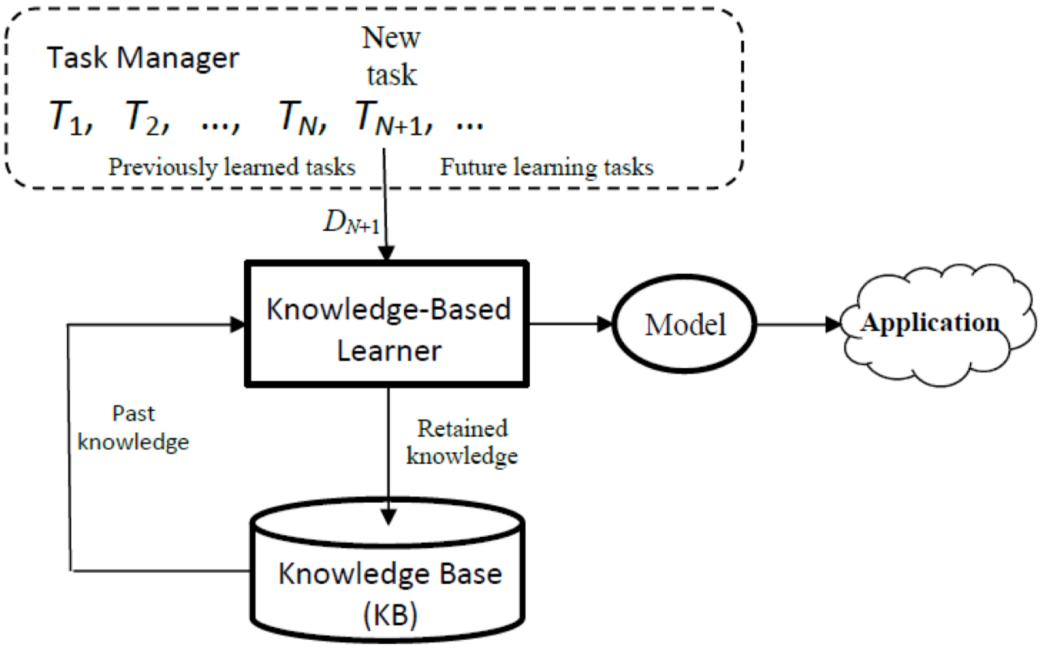

Figure 1: Incremental learning.

3 Research material and methods

This chapter introduces the research material and methods of our research. Section 3.1 introduces dynamic movement primitives (DMP). Then, the basics of reinforcement learning are introduced in Section 3.2. Section 3.3 introduces Gaussian processes. Finally, the framework of active incremental learning is elaborated in Section 3.4.

3.1 Dynamic Movement Primitives

DMPs [20, 21] are applied for trajectory control and planning problem to represent complex movements using a nonlinear dynamical system while ensuring the global stability. DMPs can represent both discrete movements such as reaching and rhythmic movements such as walking. In this section, we only consider the discrete movements.

We start from a mass-spring-damper system. With excitation, the spring always converges to the goal state in a finite amount of time.

$$-k_{oc}x - b\dot{x} = m\ddot{x} \tag{5}$$

where k_{oc} is oscillatory coefficient and b is damping coefficient.

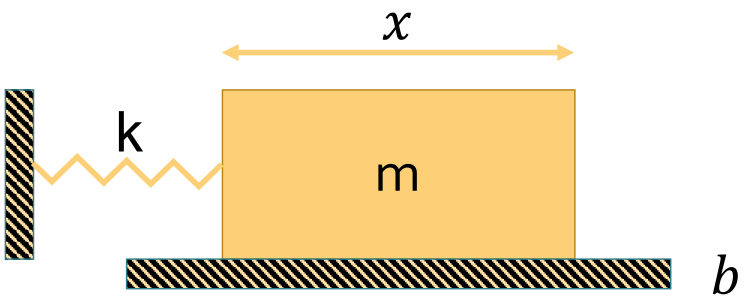

Figure 2: Mass-spring-damper system.

In a DMP, the transformation system is based on a mass-spring-damper system with $m = 1$. A 1-DOF transformation system is given as

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}), \tag{6}$$

where y is the joint or end-effector position, g is the goal state and α_y, β_y are the gain terms which can achieve critical damping with

$$\beta_y = \alpha_y/4. \tag{7}$$

In order to achieve smooth and complex movements, a nonlinear time-dependent forcing term is added to the dynamical system,

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) + f. \tag{8}$$

The forcing function is a weighted sum of the activations of basis functions which can be written as

$$f(t) = \frac{\sum_{i=1}^{N} \Psi^i(t) w_i}{\sum_{i=1}^{N} \Psi^i(t)} \tag{9}$$

where Ψ^i denotes a basis function and w_i is the corresponding weight for the given basis function. Thus, the shape of the forcing function can be changed by changing the weight value and the forcing function can control the motion trajectory of the second-order system.

In [21], they introduce a replacement of time by the first-order linear dynamics in x, called canonical system

$$\tau \dot{x} = -\alpha_x x, \tag{10}$$

where x denotes the phase of the movement, τ is the duration of the movement and α_x is a positive constant. The variable x converges exponentially to zero. Thus, the forcing function can be defined as a function of the canonical system

$$f(x) = \frac{\sum_{i=1}^{N} \Psi_i(x) w_i}{\sum_{i=1}^{N} \Psi_i(x)} (g - y_0) x \tag{11}$$

where y_0 denotes the initial state and Ψ^i is defined as a Gaussian kernel

$$\Psi^i = \exp(-h_i (x - c_i)^2) \tag{12}$$

where c_i and h_i are the center and variance respectively.

In the case of learning a DMP, we first need to compute the accelerations from the desired trajectory y_d, which denotes the time series of desired points in the trajectory,

$$\ddot{y}_d = \frac{\partial}{\partial t} \dot{y}_d = \frac{\partial}{\partial t} \frac{\partial}{\partial t} y_d. \tag{13}$$

Then, we calculate the forcing term

$$f = \ddot{y}_d - \alpha_y (\beta_y (g - y) - \dot{y}). \tag{14}$$

With the desired trajectory, we can compute wights w_i with the following equation

$$\omega_i = \frac{\mathbf{s}^T \mathbf{\Gamma}_i \mathbf{f}_{target}}{\mathbf{s}^T \mathbf{\Gamma}_i \mathbf{s}}, \tag{15}$$

where

$$\mathbf{s} = \begin{bmatrix} x_1(g - y_0) \\ x_2(g - y_0) \\ \cdots \\ x_P(g - y_0) \end{bmatrix} \quad \mathbf{\Gamma}_i \begin{bmatrix} \Psi_i(1) & & & 0 \\ & \Psi_i(2) & & \\ & & \cdots & \\ 0 & & & \Psi_i(P) \end{bmatrix} \quad \mathbf{f}_{target} = \begin{bmatrix} f_{target}(1) \\ f_{target}(2) \\ \cdots \\ f_{target}(P) \end{bmatrix}. \tag{16}$$

Equation (8) formulates the DMP for a 1 degree of freedom (DoF) system. However, the robot, such as KUKA LBR 4+, usually has multiple DoF. In this case, each DoF is formulated with a separate transform system (8) and share the same canonical system (10). The same canonical system can be seen as a global clock coupled with different transform systems. Combined with RL, we can fine-tune the shape parameters of the DMP which are first learned from a human demonstration.

3.2 Reinforcement Learning

Generally, machine learning algorithms can be divided into three categories: supervised learning, unsupervised learning, and reinforcement learning (RL). Supervised learning uses samples which include input-output pairs to learn a function mapping inputs to outputs. Unsupervised learning uses samples which only include inputs to learn relationships between the data. However, RL is to let the agent learn from experience, since the agent can only take actions and receive feedback of each action by interacting with the environment. The goal is to learn a policy π, which selects actions based on the current state, to maximize the returned cumulative reward.

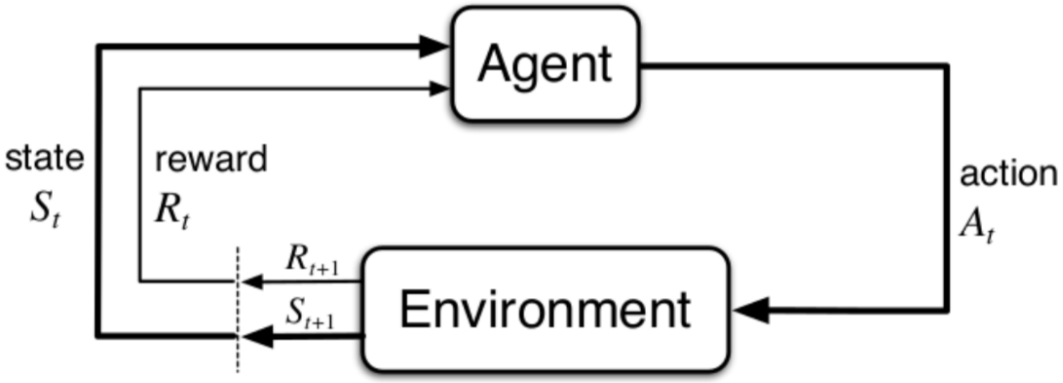

Figure 3: The framework of Reinforcement Learning [22].

The basic components of reinforcement learning include an agent, environment, state, action and reward. In the Fig. 3, the subscripts t and $t+1$ represent time steps to distinguish different states: the state at t and the state at t+1. At time t, the agent receives an observation (state) S_t and a feedback (reward) R_t. Then, an action A_t is selected from a set of available actions and sent to the environment. The time moves to $t+1$ and the environment returns a new state S_{t+1} and reward R_{t+1}. The agent will select the next action according to the new state and reward. This loop will continue until maximizing the long term reward.

In this process, the agent can make trade-offs between exploration and exploitation. Exploration is to select action that have not been executed before, thus exploring more possibilities; exploitation is to select the best action based on current trained

model. The objective is to obtain a policy with the highest long-term return, which can result in short-term return losses. The challenge is that if you exploit too much, the model may fall into a local optimum, and if you explore too much, the model may converge too slowly; this is the exploitation-exploration dilemma.

There are many examples of reinforcement learning, such as the recent famous Alpha Go [23], the robot first defeated the human master on the Go game. The algorithm does not tell the agent how to move, how to make decisions, but to score the behavior of the agent. The agent only needs to remember those high-scoring, low-scoring behaviors, and use the high-scoring behavior to get high scores next time, avoiding low-scoring behavior. This high score behavior is like the correct label in supervised learning.

3.2.1 Policy search

Policy search is a sub-area of reinforcement learning (RL) which aims to find the appropriate parameters for a given policy. It is very suitable for robotics because it can handle high-dimensional states and motion spaces. Policy search is divided into model-based policy search methods such as PILCO [24] and model-free policy search methods such as PoWER [25], PI^2 [26] and REPS [27]. Model-free policy search methods learn an optimal policy directly with several sampled trajectories, which rely on a policy representation such as DMP with less than 100 parameters. Model-based policy search methods first use several sampled trajectories to construct a model of the robot's dynamics, and then use this model to improve the policy. The goal of the policy search is to find the policy parameters θ such that the expected return is maximized. In this thesis, we use a model-free policy search method (PoWER) to learn two tasks: Ball-in-the-Cup and Basketball. The detailed description of PoWER is presented in the next Section 3.2.2.

3.2.2 Policy Learning by Weighted Exploration with the Returns

In this thesis, we use PoWER [25] to refine the shape parameters of the DMP iteratively, which is inspired by Expectation Maximization algorithm and particularly well-suited for trial-based tasks in motor control.

We first define a deterministic mean policy

$$\bar{a} = \theta^T \phi(s, t) \tag{17}$$

where θ is the policy parameters from a demonstration and ϕ is the basis function. In each iteration, we perform N roll-outs of the task and choose the best n ($n << N$) roll-outs based on the reward. In each roll-out, random Gaussian noise $\epsilon(s, t)$, which is weighted by the returned accumulated reward, will be added to the shape parameters $\boldsymbol{\theta}$ in order to make model-free policy search possible. That means the roll-out with higher returned reward will contribute more to the updated policy parameters. As a

result, the stochastic policy can be given as

$$a = \theta^T \phi(s,t) + \epsilon(\phi(s,t)). \tag{18}$$

In previous work [28, 29, 30], $\epsilon(\phi(s,t))$ is defined as a state-independent, white Gaussian noise

$$\epsilon(\phi(s,t)) \sim \mathcal{N}(0,\Sigma). \tag{19}$$

However, it has several disadvantages: a large variance increases with the number of time-steps; actions can be perturbed frequently; the system of executing the trajectory can be damaged.

In [31], they propose a state-dependent exploration

$$\epsilon(\phi(s,t)) = \varepsilon_t^T \phi(s,t) \tag{20}$$

where $[\varepsilon_t]_{ij} \sim \mathcal{N}(0,\sigma_{ij}^2)$ and σ_{ij}^2 can be optimized. Then, the resulting policy can be given as

$$a \sim \pi(a_t \mid s_t,t) = \mathcal{N}(a \mid \theta^T \phi(s,t), \hat{\sum}(s,t)) \tag{21}$$

The updated rule is given by

$$\theta_{'} = \theta + E\left\{\sum_{\tilde{t}=t}^{T} \boldsymbol{W}(s_t,t)\hat{Q}^{\pi}(s,a,t)\right\}^{-1} E\left\{\sum_{\tilde{t}=t}^{T} \boldsymbol{W}(s_t,t)\varepsilon_t \hat{Q}^{\pi}(s,a,t)\right\} \tag{22}$$

where $\boldsymbol{W}$ is a constant matrix and $\hat{Q}^{\pi}(s,a,t)$ is the state-action value function [32]. PoWER is presented in Algorithm 1. $\langle\cdot\rangle_{w(\tau)}$ denotes an importance sampler since we always choose the best n from N roll-outs in each iteration.

Algorithm 1 Policy Learning by Weighted Exploration with the Returns (PoWER).

Input: The initial policy parameters $\boldsymbol{\theta}_0$,

1: **repeat**

2: Sample: Perform rollout(s) using $a = (\theta + \varepsilon_t)^T \phi(s,t)$ with $[\varepsilon] \sim N(0,\sigma^2)$ as stochastic policy and collect all $(t, s_t, a_t, s_{t+1}, \varepsilon_t, r_{t+1})$ for $t = \{1, 2, ..., T+1\}$.

3: Estimate: Use unbiased estimate $\hat{Q}^{\pi}(s,a,t) = \sum_{\tilde{t}=t}^{T} r(s_{\tilde{t}}, a_{\tilde{t}}, s_{t\tilde{+}1}, \tilde{t})$.

4: Reweight: Compute importance weights and reweight roll-outs, discard low-importance roll-outs.

5: Update policy using $\theta_{k+1} = \theta_k + \frac{\left\langle \sum_{\tilde{t}=t}^{T} \varepsilon_t \hat{Q}^{\pi}(s,a,t) \right\rangle_{w(\tau)}}{\left\langle \sum_{\tilde{t}=t}^{T} \hat{Q}^{\pi}(s,a,t) \right\rangle_{w(\tau)}}$.

6: **until** Convergence $\theta_{k+1} \approx \theta_k$.

3.3 Gaussian Processes

A Gaussian process (GP) is a non-parametric Bayesian approach which is utilized for optimizing unknown functions based on the observed data [33, 34]. A GP is a set of random variables $\mathbf{y}$ which are arranged according to the continuous variable

$\mathbf{x}$, such as time and space. Each random variable is a Gaussian distribution with a mean μ and a variance σ^2

$$y \sim \mathcal{N}(\mu, \sigma^2). \tag{23}$$

Thus, a GP can be seen as a joint Gaussian distributions over functions, which is defined with a mean function $\mu(\mathbf{x})$ and a covariance function $k(\mathbf{x}, \mathbf{x}^{'})$.

$$f_{sample} \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}^{'})). \tag{24}$$

The covariance function is also called the kernel function which decides the shape of prior and posterior of the GP. An appropriate kernel specifies smoothness of the function and the similarity between function values at different inputs. This means that closer inputs would have higher value of the kernel. One of the popular kernels is the radial basis function kernel (RBF), which is also called the squared exponential kernel

$$k(\mathbf{x}, \mathbf{x}^{'}) = \exp(-\frac{1}{2} d(\frac{\mathbf{x}}{l}, \frac{\mathbf{x}^{'}}{l})^2), \tag{25}$$

where d denotes the Euclidean distance between two 1-D arrays.

The RBF kernel is parametrized by a length-scale parameter l which specifies the width of the kernel for modelling a stationary and smooth function. In this thesis, RBF is used to fit a reward model with limited sample pairs, including inputs, task parameters, and outputs, the corresponding rewards.

Another important kernel is the dot-product kernel which is parameterized by a parameter σ_0^2,

$$k(\mathbf{x}, \mathbf{x}^{'}) = \sigma_0^2 + \mathbf{x} \cdot \mathbf{x}^{'}. \tag{26}$$

When $\sigma_0^2 = 0$, the kernel is homogeneous, otherwise it is inhomogeneous.

The Dot-Product kernel is usually combined with exponentiation,

$$k(\mathbf{x}, \mathbf{x}^{'}) = (\sigma_0^2 + \mathbf{x} \cdot \mathbf{x}^{'})^D. \tag{27}$$

In our work, we use Dot-Product kernel with exponent 2 to fit skill model with limited sample pairs, including inputs, task parameters, and outputs, the corresponding policy parameters.

Assume we have new inputs $\mathbf{x}$ and observed data set $\mathcal{D} = \{(\mathbf{x}_o, \mathbf{y}_o)\}$ which includes n samples $\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$. The objective is to predict the function value $f(\mathbf{x})$ for new inputs. The observation $\mathbf{y}_o$ and function value $f(\mathbf{x})$ follow a joint normal distribution, written as

$$\begin{bmatrix} \mathbf{y}_o \\ f(\mathbf{x}) \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} k(\mathbf{x}_o, \mathbf{x}_o) & k(\mathbf{x}_o, \mathbf{x}) \\ k(\mathbf{x}, \mathbf{x}_o) & k(\mathbf{x}, \mathbf{x}) \end{bmatrix}). \tag{28}$$

Then, we can easily compute the Gaussian posterior for any input based on the observed data with mean and covariance matrix [34],

$$\mu(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_o)[k(\mathbf{x}_o, \mathbf{x}_o)]^{-1} \mathbf{y}_o, \tag{29}$$

$$\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - k(\mathbf{x}, \mathbf{x}_o)[k(\mathbf{x}_o, \mathbf{x}_o)]^{-1} k(\mathbf{x}_o, \mathbf{x}). \tag{30}$$

In this thesis, we consider a Gaussian Process to represent the contextual skill model, which can generalize skills over a range of task parameters; and reward model which allows agent to predict reward for one task without executing the skill.

Figure 4 shows an example of the GP prior and posterior with a radial basis function kernel. Figure 5 shows an example of the GP prior and posterior using Dot-Product kernel with exponent 2. Colorful lines indicate 10 samples from the GP. Red dots indicate empirical observations. The black line marks the current mean of the GP. The gray filling area indicates standard deviation of the GP.

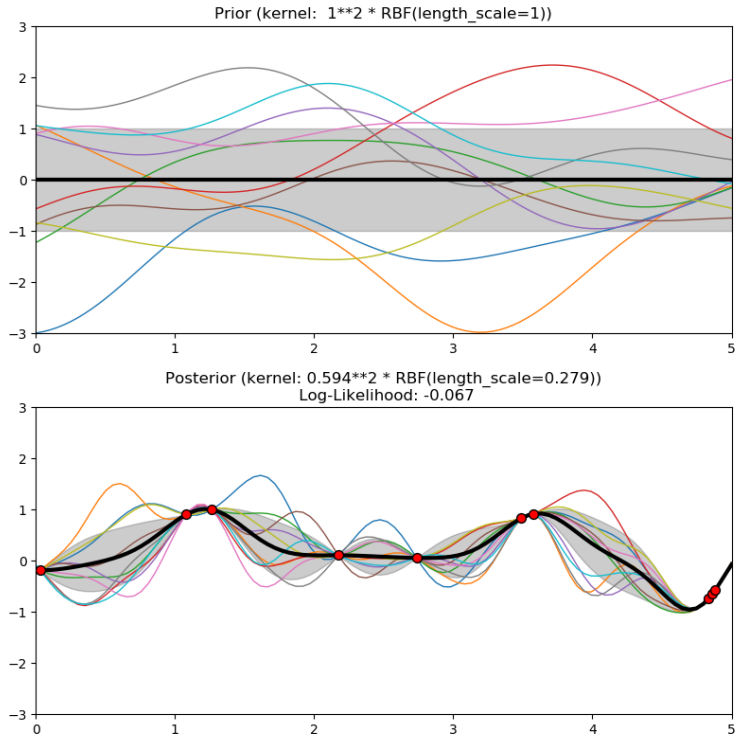

Figure 4: The prior and posterior of a GPR with RBF kernel [35].

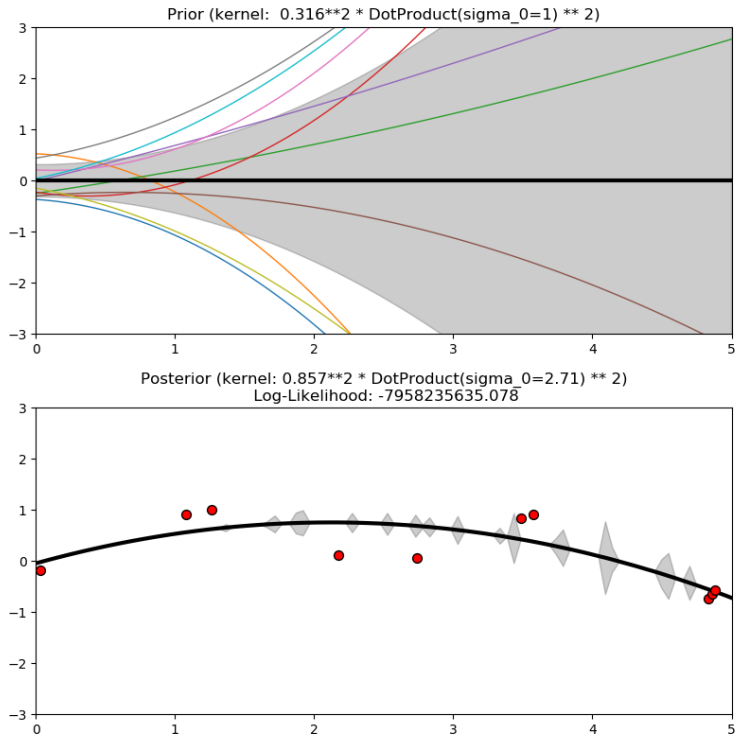

Figure 5: The prior and posterior of a GPR with Dot-Product kernel [35].

3.4 Active Incremental Learning

3.4.1 Problem Definition

We will have a database $\mathcal{D}_t = \{(\boldsymbol{\tau}_i, \boldsymbol{\theta}_{\boldsymbol{\tau}_i}) | i = 1 \dots N\}$ at time t with a set of tasks $\boldsymbol{\tau}_i$ and their corresponding policies $\boldsymbol{\theta}_{\boldsymbol{\tau}_i}$. We assume that tasks will be added to the database sequentially. A contextual skill model share knowledge over a range of tasks in $\mathcal{D}_t$ by fitting a regression model mapping task parameters to policy parameters. We can generalize policy parameters for any task parameters using the skill model, instead of learning policy for each task.

We evaluate the performance $r(\boldsymbol{\tau}; S)$ of the skill models. Then, we define skill performance as the expected performance of the skill over a set of tasks

$$SP(S_t) = \int P(\boldsymbol{\tau}) r(\boldsymbol{\tau}; S_t) d\boldsymbol{\tau}, \tag{31}$$

where $P(\boldsymbol{\tau})$ denotes the probability of the task $\boldsymbol{\tau}$ occurring. We assume that the tasks occur with the same probability. Thus, we can rewrite (31) into

$$SP(S_t) = \frac{1}{\boldsymbol{\tau}_{max} - \boldsymbol{\tau}_{min}} \int_{\boldsymbol{\tau}_{min}}^{\boldsymbol{\tau}_{max}} r(\boldsymbol{\tau}; S_t) d\boldsymbol{\tau}. \tag{32}$$

Furthermore, we can define the expected improvement in skill performance ($EISP$)

which corresponds to the expected skill performance definition (2) considered in [10]

$$\begin{aligned} EISP(\boldsymbol{\tau}_c) &= SP(S_{t+1}) - SP(S_t) \\ &= \frac{1}{\boldsymbol{\tau}_{max} - \boldsymbol{\tau}_{min}} \int_{\boldsymbol{\tau}_{min}}^{\boldsymbol{\tau}_{max}} r(\boldsymbol{\tau}; S_{t+1}) - r(\boldsymbol{\tau}; S_t) d\boldsymbol{\tau}, \end{aligned} \tag{33}$$

where S_{t+1} represents the updated skill model with updated database $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(\boldsymbol{\tau}_c, \boldsymbol{\theta}_{\tau_c})\}$.

Using the $EISP$, a task will be selected according to $\tau^* = \arg\max_\tau EISP_t(\tau)$.

3.4.2 Algorithm

We propose a novel active incremental learning method to actively select a task step by step based on the current performance which is evaluated with current skill model. In order to predict the improvement in skill performance, we introduce a learning rate model $J(t)$ to predict reward improvement over a single iteration by recording total rewards after every update step t of policy search until achieving the optimal rewards R^*. Furthermore, we assume that the learning rate model $J(t)$ does not depend on task parameters, and rewards achieved by the skill model are similar for similar task parameters. This consistency is then modeled with current reward model $R(\boldsymbol{\tau})$. Using the learning rate model $J(t)$ and reward model $R(\boldsymbol{\tau})$, we can evaluate the expected improvement in skill performance ($EISP$) for any task.

The first step of active incremental learning (see Algorithm 2) is initialization, including choosing the initial task parameter $\boldsymbol{\tau}_0$, optimizing policy for $\boldsymbol{\tau}_0$, learning the learning rate model $J(t; \beta_J)$, initializing the database $\mathcal{D}$ and estimating skill model $S(\boldsymbol{\tau})$ with $\mathcal{D}$ in lines 1-5. We utilize an exponential family to represent the learning rate model

$$J(t; \beta_J) = -\exp(-a(t-b)) + c \tag{34}$$

where $\beta_J = \{a, b, c\}$ denotes the hyper-parameters of the learning rate model. We assume that the learning rate models are the same for different task parameters within a certain range (see Section 5.2).

The hyper-parameters are estimated using *curve_fit* function in *scipy.optimize* which uses non-linear least squares to fit learning rate function to data. The data includes the independent variable t and the dependent data J collected when optimizing policy parameters $\boldsymbol{\theta}_{\boldsymbol{\tau}_0}$ for the initial task parameter $\boldsymbol{\tau}_0$. After that, we estimate the skill model $S(\boldsymbol{\tau})$ using the database $\mathcal{D}$ containing the initial sample $(\boldsymbol{\tau}_0, \boldsymbol{\theta}^*_{\boldsymbol{\tau}_0})$.

Then, we update the skill model $S(\boldsymbol{\tau})$ iteratively in lines 7-16. In order to build the reward model $R(\boldsymbol{\tau})$ over task parameters, we first estimate the reward $r(\boldsymbol{\tau}; S_t)$ with the current skill model $S_t(\boldsymbol{\tau})$ for each task parameters $\boldsymbol{\tau} \in \boldsymbol{\tau}_{eval}$ in line 7. Then, the reward model

$$R(\boldsymbol{\tau}; \beta_R) \sim \mathcal{GP}(\boldsymbol{\tau}, \beta_R), \tag{35}$$

is estimated by a Gaussian process (GP) with the training samples $\{(\boldsymbol{\tau}, r(\boldsymbol{\tau}))\}$. β_R is the hyper-parameter of GP which can be optimized by maximizing evidence function [36] in line 8.

With the learning rate model $J(t;\beta_J)$ and reward model $R(\boldsymbol{\tau})$, we evaluate $EISP$ for each candidate task parameter in lines 9-12. The most important step is to predict the reward improvement $\Delta R(\boldsymbol{\tau}_c)$ using the learning rate model $J(t;\beta_J)$. We first calculate the time step t_c for the candidate task parameter $\boldsymbol{\tau}_c$ according to the inverse of the learning rate function

$$t_c = J^{-1}(R(\boldsymbol{\tau}_c;\beta_J)). \tag{36}$$

Then, we calculate the reward improvement $\Delta R(\boldsymbol{\tau}_c)$

$$\Delta R(\boldsymbol{\tau}_c) = J(t_c + \Delta;\beta_J) - R(\boldsymbol{\tau}_c;\beta_R), \tag{37}$$

where Δ denotes a fixed number of iterations in policy search. When we calculate the estimated reward $r(\boldsymbol{\tau}; S_{t+1})$ after Δ iterations across all task parameters (38), we assume that the closer the task is to the candidate task, the more the reward increases, but not greater than the reward improvement of the candidate task. Thus, we predict the expect reward $r(\boldsymbol{\tau}, S_{t+1})$ after Δ iterations by

$$r(\boldsymbol{\tau}; S_{t+1}) = R(\boldsymbol{\tau};\beta_R) + \Delta R(\boldsymbol{\tau}) \times \exp(c_d \left\|\boldsymbol{\tau} - \boldsymbol{\tau_c}\right\|^2), \tag{38}$$

where S_{t+1} denotes the expected skill model with expected $\mathcal{D} = \mathcal{D} \cup (\boldsymbol{\tau}_c, \boldsymbol{\theta}_{\boldsymbol{\tau}_c})$ and c_d is a constant controlling the similarity across tasks. In our experiments, c_d is -0.1. Now we can calculate $EISP$ (33) by

$$EISP = \frac{1}{\boldsymbol{\tau}_{max} - \boldsymbol{\tau}_{min}} \sum_{\boldsymbol{\tau}=\boldsymbol{\tau}_{min}}^{\boldsymbol{\tau}_{max}} r(\boldsymbol{\tau}; S_{t+1}) - r(\boldsymbol{\tau}; S_t). \tag{39}$$

The most promising task τ^* is selected by maximizing $EISP$ in line 13

$$\tau^* = \arg\max_{\tau} EISP(\tau). \tag{40}$$

$EISP$ can be seen as a quantitative representation of the contributions of different task parameters to improve the overall skill performance. Some task parameters may contribute more to the overall skill performance. Thus, computing and comparing the $EISP$ for different task parameters can help us find the most promising task parameter without executing it.

We run PoWER for Δ iterations for τ^* in order to update the corresponding policy parameters $\boldsymbol{\theta}_{\tau^*}$ in line 14 and update skill model with updated database $D = D \cup \{(\tau^*, \theta_{\tau^*})\}$ in lines 15-16. We will continue to update the skill model $S(\boldsymbol{\tau})$ iteratively by repeating lines 7-16 until $S(\boldsymbol{\tau})$ can provides successful skills for all $\boldsymbol{\tau} \in \boldsymbol{\tau}_{eval}$.

Algorithm 2 Active Incremental Learning of a CSM $S(\boldsymbol{\tau})$

Input: $\boldsymbol{\tau} = \{\boldsymbol{\tau}_i \mid 1 \leq i \leq n\}$, $\boldsymbol{\tau}_{eval} = \{\boldsymbol{\tau}_j \mid 1 \leq j \leq k\}$
Output: Skill model $S(\tau)$.

Initialization :
Choose initial task parameter $\boldsymbol{\tau}_0$.
Optimize policy for $\boldsymbol{\tau}_0$ using RL to determine $\boldsymbol{\theta}^*_{\boldsymbol{\tau}_0}$.
Estimate parameters β_J for learning rate model $J(t; \beta_J)$.
Initialize database of policies $\mathcal{D} = \{(\boldsymbol{\tau}_0, \boldsymbol{\theta}^*_{\boldsymbol{\tau}_0})\}$.
Estimate skill model $S(\boldsymbol{\tau})$ with $\mathcal{D}$.
repeat
  Evaluate $r(\boldsymbol{\tau})$ for $\boldsymbol{\tau} \in \boldsymbol{\tau}_{eval}$.
  Estimate parameters β_R for reward model $R(\boldsymbol{\tau}; \beta_R)$ using $r(\boldsymbol{\tau})$.
  for each $\boldsymbol{\tau}_c \in \boldsymbol{\tau}_{eval}$ **do**
    Predict reward improvement $\Delta R(\boldsymbol{\tau}_c)$ using (37).
    Evaluate $EISP(\boldsymbol{\tau}_c)$ using (33) and (38).
  end for
  Choose next task $\tau^* = \arg\max_\tau EISP(\tau)$.
  Optimize policy for Δ iterations for τ^* to determine θ_{τ^*}.
  Update $D = D \cup \{(\tau^*, \theta_{\tau^*})\}$.
  Re-estimate $S(\tau)$ with D.
until S provides success for all $\boldsymbol{\tau} \in \boldsymbol{\tau}_{eval}$.
return Skill model $S(\tau)$.

4 Experiment

This chapter introduces the experimental setting and research tasks. Section 4.1 introduces the setting, including the robot arm, robot operating system (ROS) and simulation software. Then, the two tasks, ball-in-a-cup and basketball, is introduced in section 4.2. We implemented active incremental learning algorithm on the two tasks and compared with a baseline, in order to achieve successful skills for a range of task parameters.

4.1 Setting

4.1.1 The KUKA LWR4+ Robot Arm

We evaluate the proposed active incremental learning framework on KUKA LWR4+ (see Fig. 6) which has a payload capacity of 7 kg and low weight of 16 kg. KUKA LWR4+ is a light-weight 7 degrees of freedom robotic arm, which enable the robot to reach any position in its working area. Moreover, each joint is provided with integrated Force/Torque (F/T) sensors which are independent from each other. These F/T sensors make robot suitable for process monitoring which can detect collisions in real time. Due to these characteristics, KUKA LWR4+ has high sensitivity, flexibility and safety.

Figure 6: KUKA LWR4+.

Fig. 7 shows the zero configuration of KUKA LWR4+, indicating positive and negative directions of joint rotations. In order to reduce the search space during reinforcement learning, only some joints are movable, and the rest joints will be fixed at home position. For the ball-in-a-cup experiment, all the joints have been moved. For the basketball experiment, joints A2, A3, A5 are movable.

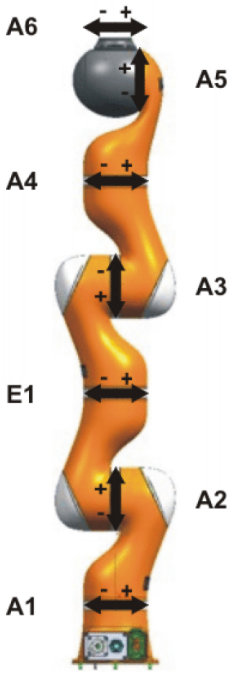

Figure 7: Frontal View of KUKA LWR4+.

4.1.2 Robot Operating System

Robot Operating System (ROS) is an open source system with a set of software libraries and tools for developing robotics applications. The primary goal of ROS is to increase the rate of code reuse in the field of robotics research. A robot control system usually consists of many nodes and a super administrator ROS Master. A node is the minimum processing unit of ROS and a node is usually responsible for a single function of the robot. A ROS master can manage all nodes in the system so that nodes can communicate with each other by means of publish/subscribe. For example, a sensor on the robot can act as a node of the ROS, which can publish the message obtained by the sensor to a given topic. The published message can be subscribed by other nodes that are interested in this kind of message types. Note that the message is passed directly from the publishing node to the subscribing node without going through the ROS Master. Fig. 8 shows the basic structure of the topic communication.

Service is another communication method of a node, using a request-reply communication model. Such a communication model does not have frequent message delivery which is suitable for one-to-one communication. A service consists of two parts, Client and Server. The client sends the request messages and waits for reply. After the server finishes processing the request and replies, the client will continue to execute.

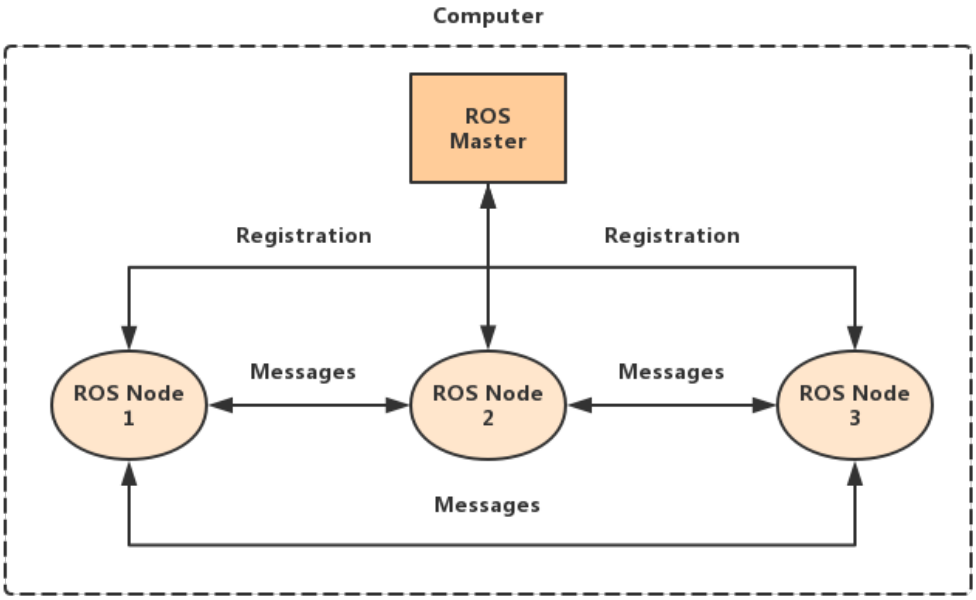

Figure 8: Structure of the topic communication.

4.1.3 Learning a Robotic Skill in a Simulation Software

In order to learn a task, the robot usually requires repeating experiments for that task. Thus, in real robot implementations, the learning process is time-consuming. Moreover, it causes wear and tear on the robot due to the high repeatition. Thus, we perform the proposed framework in a simulation software named LWRSIM [37]. It uses MuJoCo [38], an advanced physics simulation developed by Emo Todorov for Roboti LLC, to simulate physical contacts between the object. The simulation runs faster than real-time without physical damage.

4.2 Tasks

4.2.1 Ball in a Cup

The ball-in-a-cup game needs a cup, a string, and a ball. The cup is attached to the end-effector of KUKA LBR 4+; the string is attached to the bottom of the cup and the ball hangs vertically under the cup by the string (see Fig. 9). The objective of the game is to move the cup in order to get the ball in the cup. Since the length of the string can be observed, changed and evaluated easily, we chose the string length as the task parameter τ, which was varied within $\tau \in \{29\text{ cm}, 30\text{ cm}, \ldots, 43\text{ cm}\}$. In order to succeed in this game, the movement requires a complex trajectory. Moreover, the change in string length can result into a significant change in the dynamics of the

task which requires a complex change in the motion. Thus, we can easily evaluate the generalization capability of a CSM with ball-in-a-cup game. Since the movement was restricted to a plane, the trajectories along y and z direction were encoded using separate DMPs [39]. Each DMP requires 20 kernels, thus in total $N = 40$ parameters were needed to be fine-tuned by the PoWER algorithm for a task parameter.

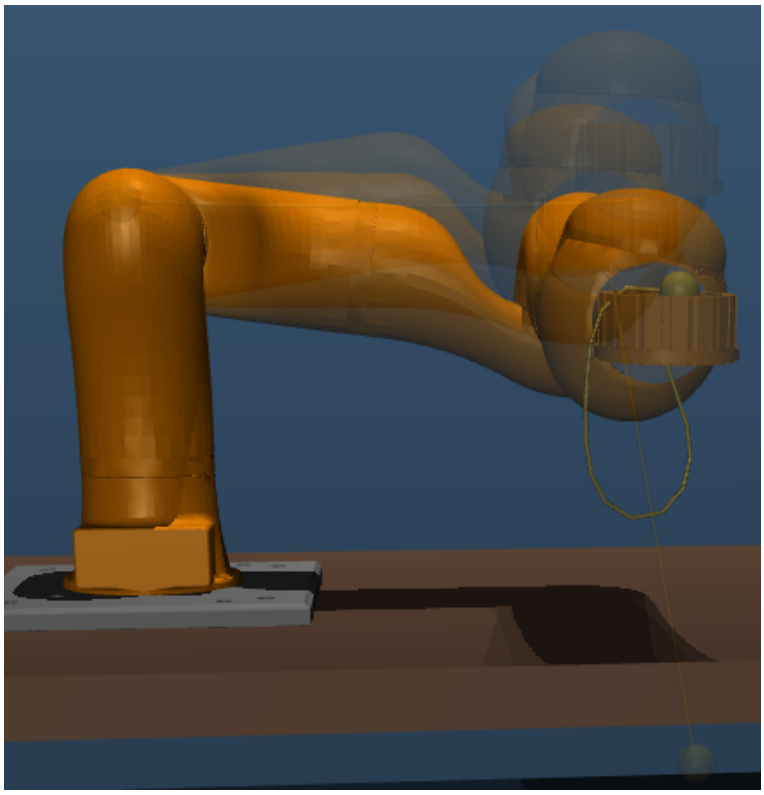

Figure 9: Learning ball-in-a-cup skill using KUKA LBR 4+ in MuJoCo.

4.2.2 Basketball

The basketball game requires a ball holder, a basket, and a ball. The ball holder is attached to the end-effector of KUKA LBR 4+ and the basket is located at a certain distance in front of the robot (see Fig. 10). The objective of the game is to throw the ball into the basket. We chose the distance between the basket and the base of the robot as the task parameter τ, which was varied within $\tau \in \{120 \text{ cm}, 130 \text{ cm}, \ldots, 240 \text{ cm}\}$. Identical to the ball-in-a-cup game, the movement was restricted to a plane, thus only joints 2, 3, and 6 were used; the rest of the joints were kept fixed. Each DMP requires 20 kernels, thus in total $N = 60$ parameters were needed to be fine-tuned by the PoWER algorithm for a task parameter.

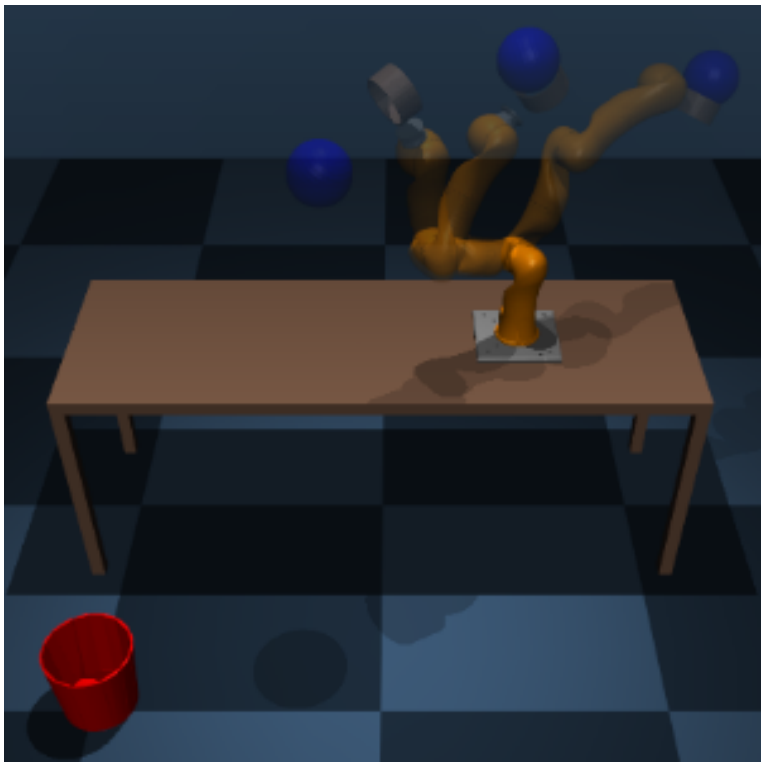

Figure 10: Learning basketball skill using KUKA LBR 4+ in MuJoCo.

5 Results

This chapter presents the experiment results. Section 5.1 represents the results of contextual skill model learned in different situations. Then, the results of learning rate model and reward model are presented in Section 5.2 and Section 5.3 respectively. Finally, Section 5.4 shows the results of skill performance over time for ball in a cup and basketball tasks.

5.1 Contextual Skill Model

In order to map the task parameters to policy parameters, we used Gaussian process regression (GPR) to learn a contextual skill model. In this case, we selected dot-product kernel with exponent 2, which can be seen as a quadratic kernel. This model has been used in simulation to real world transfer [40], called global parametric dynamic movement primitives (GPDMP). It has been shown that the generalization capabilities with GPDMP performed better than the linear CSM [11] or local models using model selection [39]. Besides that, we compare the performance of GPDMP and GPR with RBF kernel in Section 5.1.1.

5.1.1 Learning CSM with RBF and GPDMP

In order to determine a suitable kernel for the CSM, we compared the skill performance of GPR with RBF kernel vs. GPDMP.

We selected 3 optimal policies with string length $\{30cm, 35cm, 41cm\}$ to learn CSM with GPDMP and RBF kernel respectively, and used 6 optimal policies with string length $\{30cm, 32cm, 35cm, 37cm, 39cm, 41cm\}$ to learn the CSM with RBF kernel. Then, We evaluated the skill performance for these 3 CSMs. Figure 11 shows that the CSM learned with GPDMP has the best skill performance (the average reward $\bar{r}_{GPDMP3}$ is 0.615). The CSM learned with RBF kernel from 6 samples (the average reward $\bar{r}_{RBF6}$ is 0.475) has better skill performance than the CSM learned with RBF kernel from 3 samples (the average reward $\bar{r}_{RBF3}$ is 0.405). Thus, GPDMP is more suitable in our experiments.

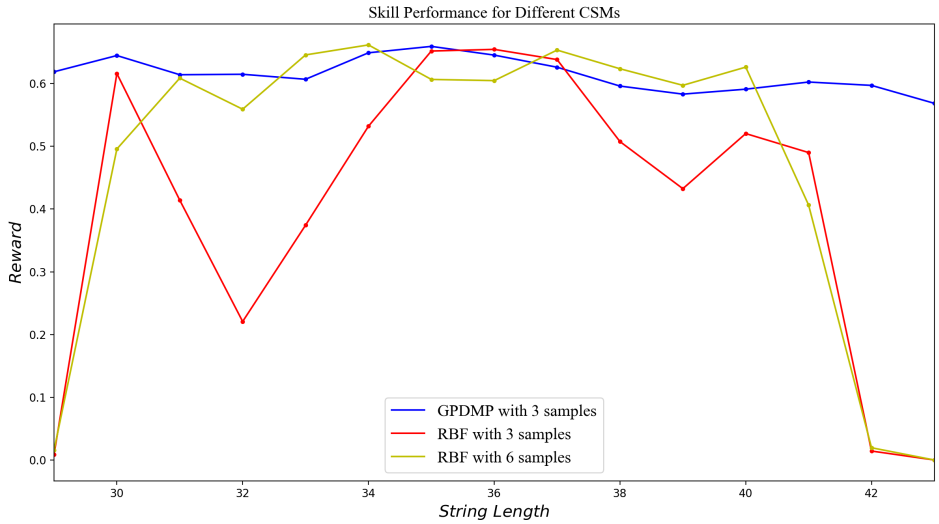

Figure 11: Skill Performance for Different CSMs.

We compared 3 CSMs which are learned with RBF using 3 samples, RBF using 6 samples and GPDMP using 6 samples (see Fig. 12 and Fig. 13). In the figures, the x axis represents the task parameter, and y axis represents the corresponding policy parameter. The figures show great differences that exist in the edge region of the fitting range, since RBF can cause edge effects when fitting functions, which means that the fitting effect is not ideal in the edge region of the fitting range. The reason is that the data points at the edge are only trained using one side of the data, which inevitably leads to loss of precision. According to the skill performance in Fig. 11, the CSM learned with RBF kernel from more samples has better skill performance, since the shape of the CSM is more similar to the CSM learned with GPDMP, which is a quadratic curve.

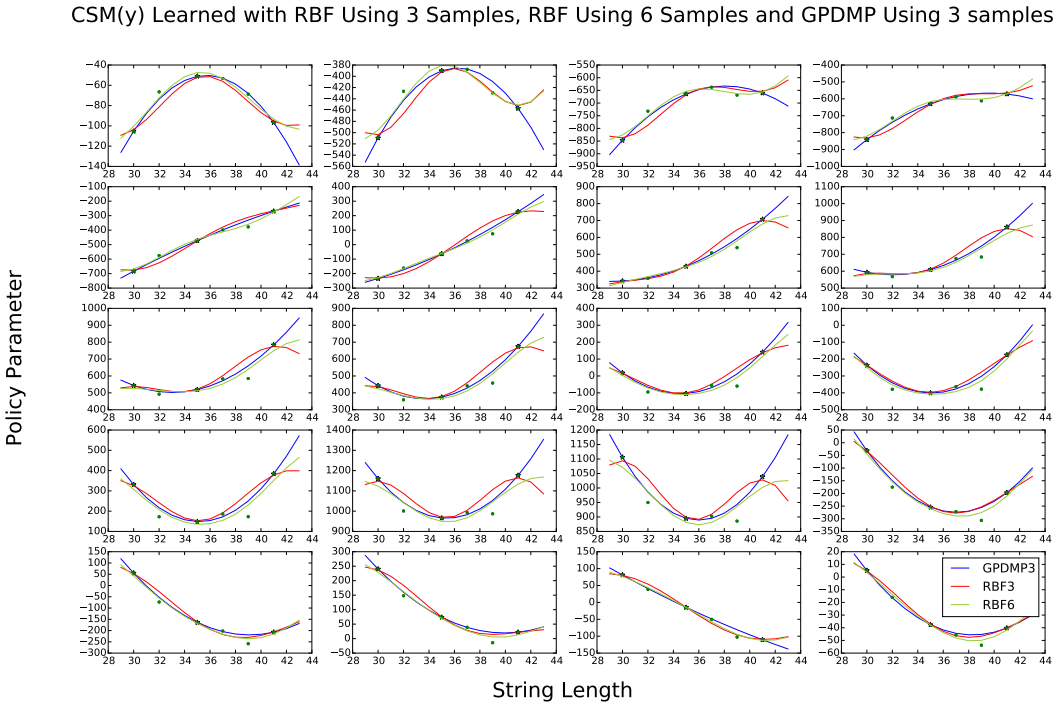

Figure 12: CSMs(y) learned with RBF using 3 samples, RBF using 6 samples and GPDMP using 3 samples.

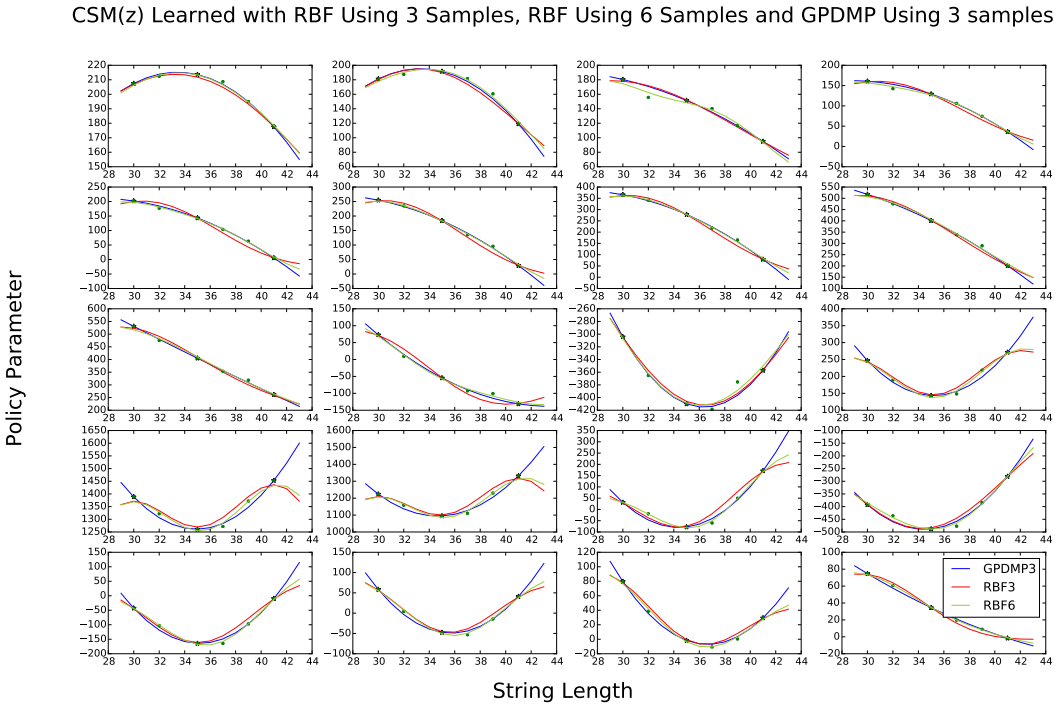

Figure 13: CSMs(z) learned with RBF using 3 samples, RBF using 6 samples and GPDMP using 3 samples.

5.1.2 Learning CSM with Random task order

In order to study whether the task order contributes to the learning speed, we did three experiments where the task orders were generated randomly and learned CSM with all optimal policies until skill performance converges. It should be noted that the convergence of the skill performance does not mean that the final CSM is optimal. In Fig. 14, experiment 1 (blue line) converges fastest and achieves a successful skill model with the task order $\{35cm, 32cm, 37cm, 41cm, 39cm, 30cm\}$ after only 13 policy updates. However, experiments 2 (orange line) with the task order $\{35cm, 32cm, 29cm, 33cm, 33cm, 33cm, 40cm, 34cm, 37cm, 39cm, 41cm, 42cm\}$ and 3 (green line) with the task order $\{35cm, 30cm, 33cm, 30cm, 30cm, 38cm, 39cm, 38cm, 32cm, 37cm, 40cm, 43cm\}$ converge slowly and cannot achieve a successful skill model at the end. In this case, we can easily achieve a successful skill model with a good task order, which means that the learning speed depends to some extent on the task order. On the other hand, the result also shows that more samples and longer learning times do not mean that we can learn a better skill model.

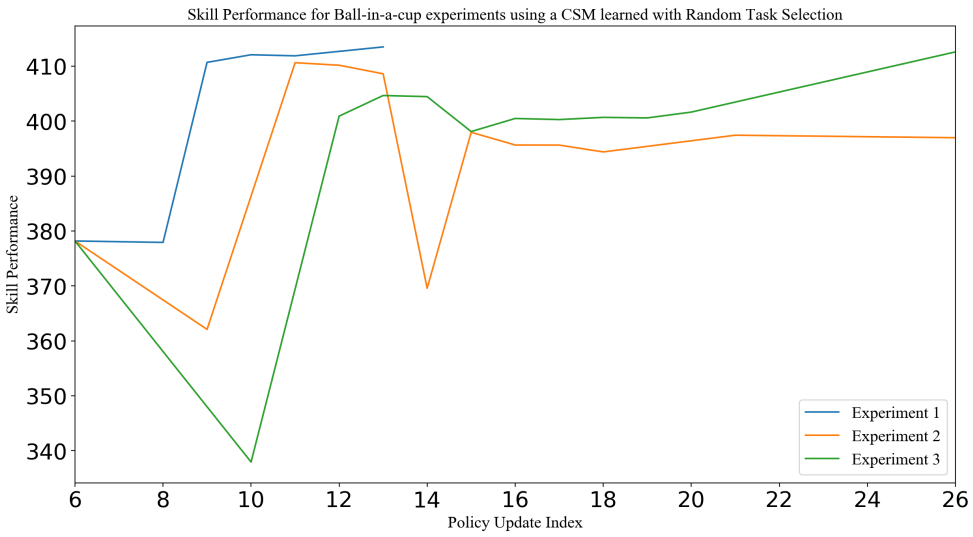

Figure 14: Skill performance for ball in a cup experiments using a CSM learned with random task order.

5.1.3 Learning CSM with spatiotemporal kernel

In [10], Da Silva et al. proposed a non-parametric Bayesian approach of skill performance which uses a GP with a spatio-temporal kernel to accommodate the non-stationary skill performance. They actively selected each task that maximizes expected improvement in skill performance, then learned each task until the policy was optimal, and used all optimal policies to train a CSM. In order to compare with the results of our approach, we performed experiments with the aforementioned

approach and learned CSMs with GPDMP in the ball-in-a-cup task.

The spatio-temporal kernel is comprised of two kernels, one for measuring the similarity between tasks and one for measuring the similarity between time. With this kernel, we got two inconsistent results (see Fig. 15) even though the task order has been the same, which is $\{35, 38, 36, 34, 37, 39, 33, 31, 32, 40, 42, 41, 30, 43, 29\}$.

The learning process started with close task parameters $\{35, 38, 36\}$, since the algorithm tends to select a task parameter which is close to the already learned task parameters. However, the two CSMs learned with the same three samples $\{35, 38, 36\}$ are very different (see Fig. 16 and Fig. 17), which explains to some extent why the results of continuous updated skill performance with the same task order are quite different (see Fig. 15). The reason behind this is that the current CSM is always updated with the previous CSM. When the CSM is learned with bad samples in the initial stages, the model would tend to the wrong direction, resulting in an unsuccessful CSM in the end. Thus, this active learning approach is useful when the task parameters are irregular and non-smooth or has local underlying regularities.

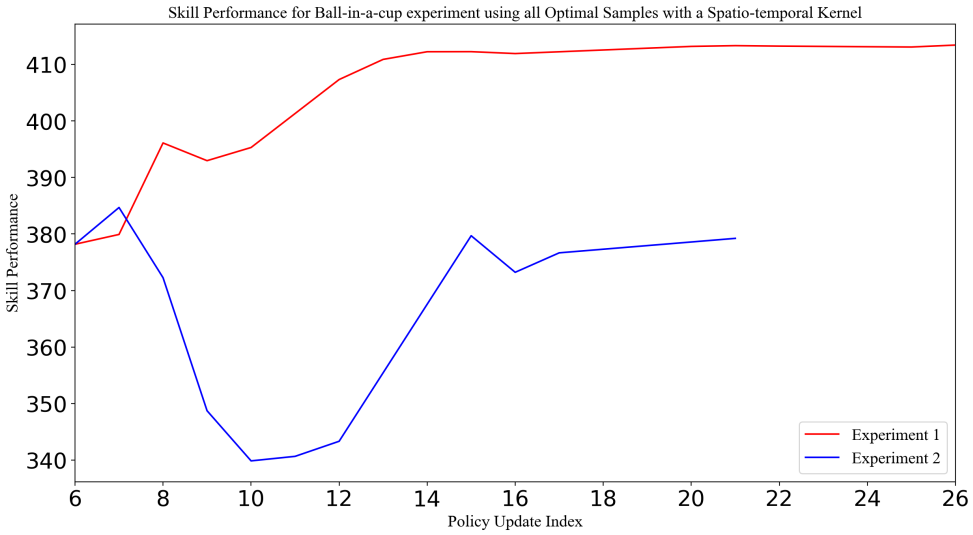

Figure 15: Skill performance for ball-in-a-cup experiment using all optimal samples with a spatiotemporal kernel.

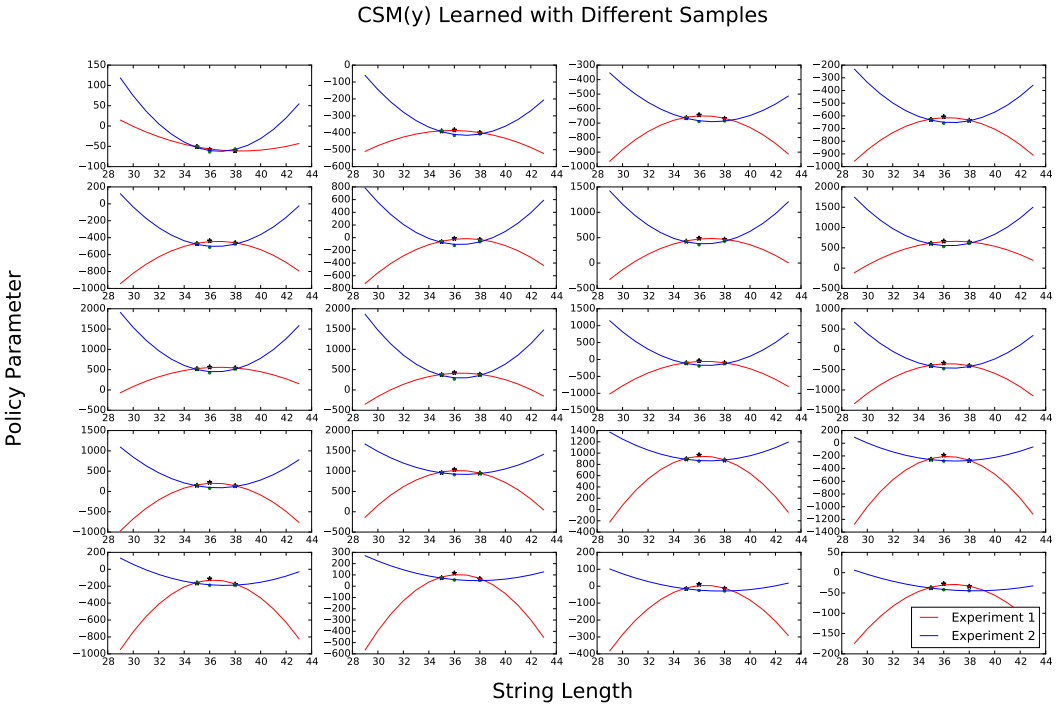

Figure 16: CSMs with different samples.

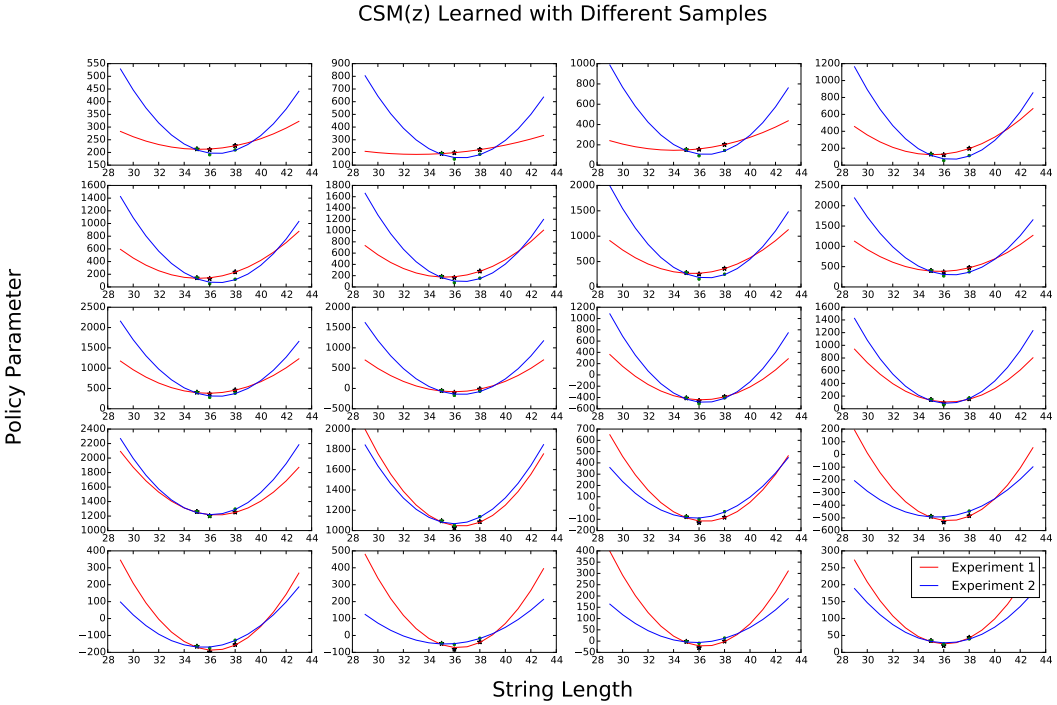

Figure 17: CSMs with different samples.

5.2 Learning Rate Model

Instead of predicting the reward with the upper endpoint of the 95% confidence interval around the mean of GP, we proposed a learning rate model $J(t)$ to predict reward improvement if we continue optimizing the corresponding policy parameters for Δ policy updates in policy search. In order to model the learning rate, we did an experiment where we learned ball-in-a-cup game for different task parameters with the same initial policy parameters using PoWER. Then, we recorded each reward value after each policy update of policy search until the reward converges.

5.2.1 Reward function with exponential

As the reward function we chose,

$$R = \exp(-150 \cdot d^2) \exp(-0.0005 \sum v) \tag{41}$$

where d is the distance between ball and the center of the cup, v is the velocities of the robot joints. We first built learning rate model with logistic regression

$$J(t; \beta_J) = \frac{d}{c + \exp(-a(t-b))}, \tag{42}$$

where $\beta_J = \{a, b, c, d\}$ is the hyper-parameters of the learning rate model. Logistic regression is a better choice than Gaussian process regression, since the reward will continue to improve during policy search and finally converge.

In order to simplify the learning rate model, we fixed the value of c and d. In this case, d is the highest reward which is achieved by implementing PoWER on the initial task parameter. Then, the updated model can be rewritten into

$$J(t; \beta_J) = \frac{0.66}{1 + \exp(-a(t-b))}. \tag{43}$$

Fig. 18 shows the learning rate model with logistic regression for different task parameters. Different colors indicate different task parameters. The colorful points are rewards we recorded during the learning process. All learning processes start from the same human demonstration.

The result shows the logistic regression fits the observed data well. However, the initial rewards with the same initial policy are the same 0 for different task parameters, while the initial recorded distances between the ball and target for different task parameters are not the same with the same initial reward. Thus, the learning rate model cannot show the difference of the initial situation for different task parameters.

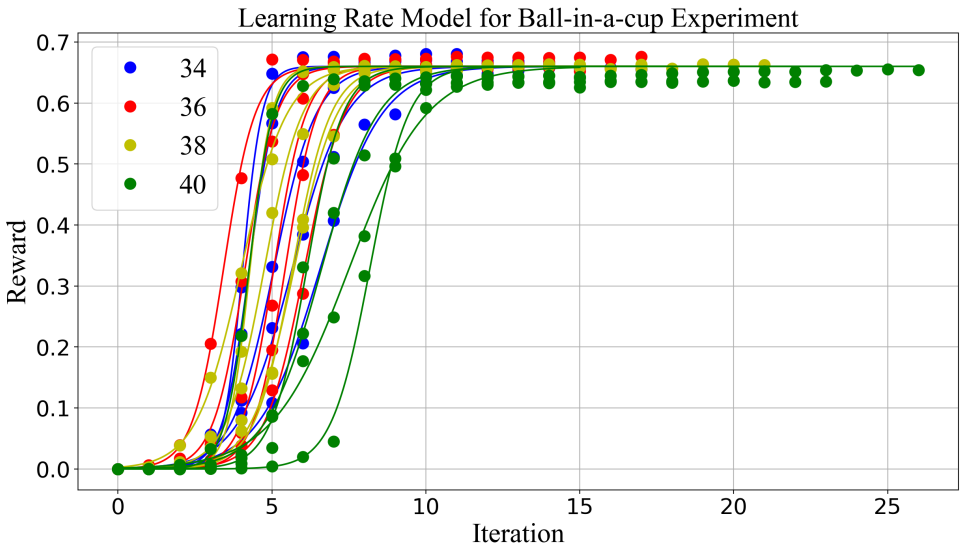

Figure 18: Learning rate model for ball-in-a-cup experiment with different task parameters.

5.2.2 Reward function without exponential

In order to solve the problem we mentioned above, we changed the reward function into

$$R = -150 \cdot d^2 - 0.0005 \sum v. \tag{44}$$

The learning rate model should be changed correspondingly,

$$J(t; \beta_J) = -\exp(-a(t-b)). \tag{45}$$

We used an exponential model to fit the learning rate model, and the result for ball-in-a-cup game are shown in Fig. 19. It can be observed that the recorded initial rewards are quite different with the same initial policy for different task parameters. On the other hand, these curves are in perfect alignment, which means that we can overlap them by translation, indicating that the learning rate is independent of the task parameters. The improved learning rate model is better than the previous one since it can show the difference of the initial situation very well. We observed similar results in basketball experiment (see Fig. 20).

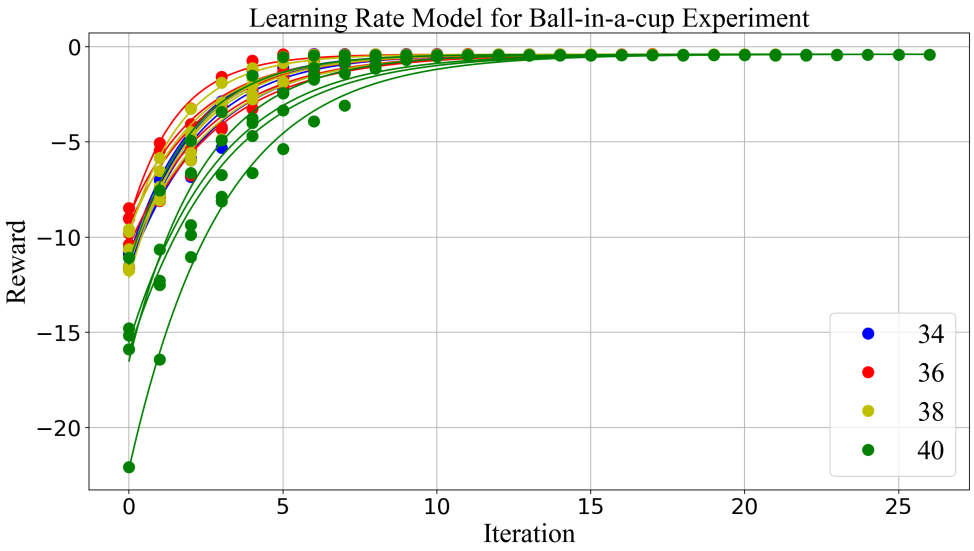

Figure 19: Learning rate model for ball-in-a-cup experiment with different task parameters.

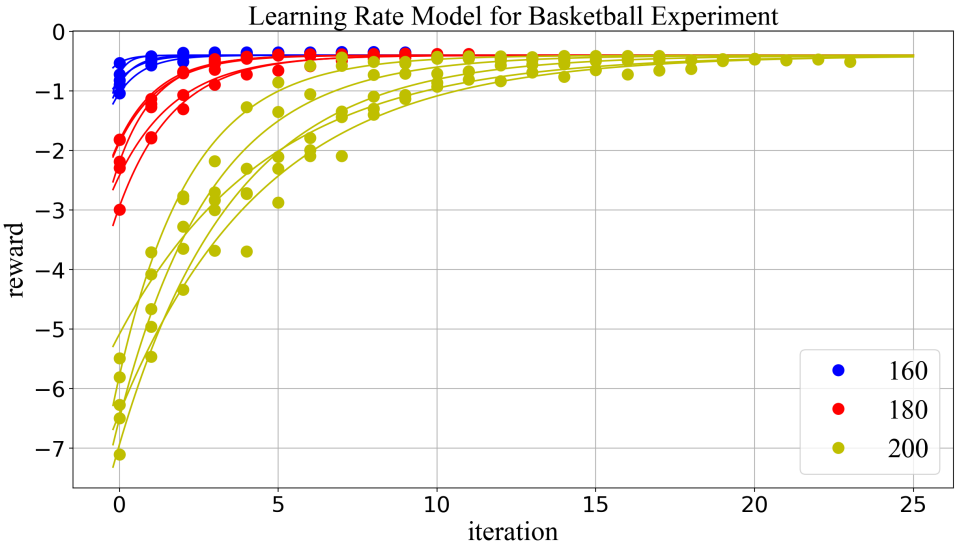

Figure 20: Learning rate model for basketball experiment with different task parameters.

5.3 Reward Model

During the learning process, different task parameters will have different task difficulty. The reward model built only with optimal rewards cannot indicate the effort taken

for different task parameters. In order to model the reward $R(\tau)$ across all task parameters with different task difficulty, we evaluated the reward for each task parameter by executing the corresponding policy from the current skill model. Then, we built the reward model with Gaussian process (GP)

$$R(\boldsymbol{\tau}; \beta_R) \sim \mathcal{GP}(\boldsymbol{\tau}, \beta_R). \tag{46}$$

Fig. 21 shows an example of the reward model. Blue stars mark the observed rewards for each task parameters based on the current skill model, while the blue curve indicates the reward model using the observed rewards. Red stars mark the predicted reward after 2 iterations in policy search for each task parameter using learning rate model, while the red curve indicates the predicted reward model if we continue optimizing the corresponding policy parameters for the next task parameter $\tau = 43$ (yellow star).

We updated the reward model $R(\tau)$, predicted the improvement in skill performance after each Δ iterations of RL, and then computed the expected improvement in skill performance across a range of task parameters for each candidate. However, this reward model is only useful when the learning rate can be predicted or modeled. The intermediate policies have not yet converged to optima during the learning process, thus we cannot to predict the reward after Δ iterations of RL without learning rate model of reward function.

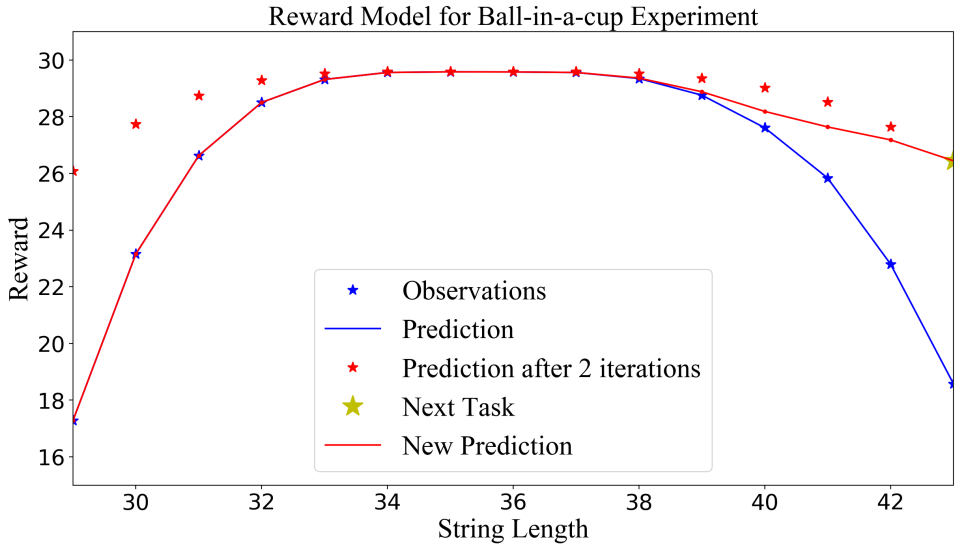

Figure 21: Reward model for ball-in-a-cup experiment.

5.4 Active incremental learning

We applied the active incremental learning algorithm (see Algorithm 2) to actively select a task step by step for learning ball-in-a-cup and basketball games. In order

to compare the skill performance, we used random task order as the baseline. For each game, we performed both active and random task selection 5 times.

For ball-in-a-cup game, we chose the string length $\tau_0 = 35$ cm as the initial task parameter. For basketball game, we chose the distance of the basket from the base of the robot $\tau_0 = 180$ cm as the initial task parameter. We used PoWER [25] to train the initial task parameter and achieved an optimized policy after 6 policy updates for the ball-in-a-cup, and after 5 policy updates for the basketball. We run PoWER for $\Delta = 2$ iterations for the selected task in order to update the corresponding policy parameters in each policy search iteration.

We compared the skill performance SP over time for ball-in-a-cup (see Figs. 22) and basketball (see Figs. 23). The blue curve denotes the proposed active incremental learning method and the grey curve is the baseline (random learning), error bars denoting 1 standard deviation. Generally, the skill performance improves over time for both methods. However, active learning method learned faster on average and had smaller variance. With the active learning, the CSM could provide successful zero-shot generalization for the entire range of task parameters after 20 policy updates for ball-in-a-cup and 23 policy updates for basketball. The success rate for the random learning after the same number of policy updates was 75% for ball-in-a-cup and 80% for basketball.

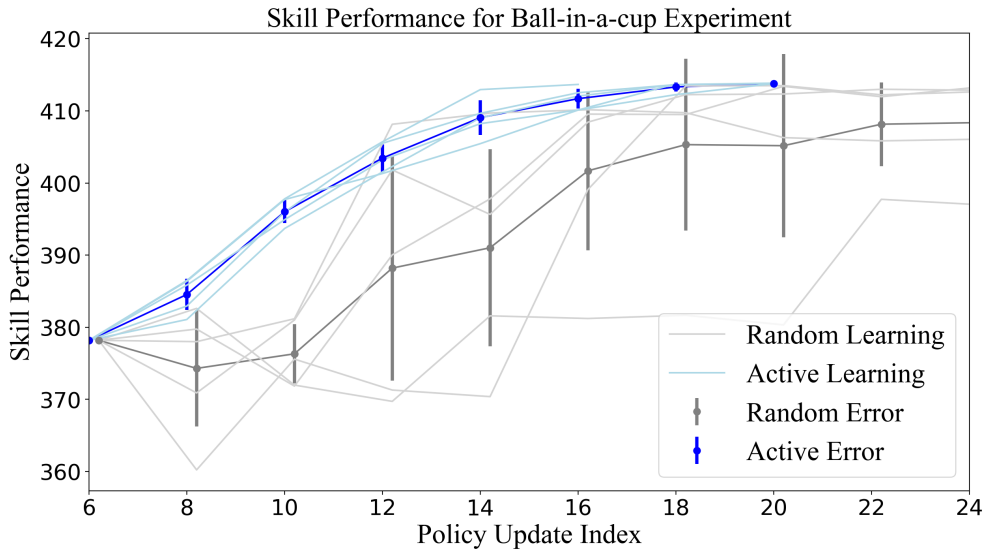

Figure 22: Skill performance on ball-in-a-cup skill: active (in blue) versus random task selection (in grey).

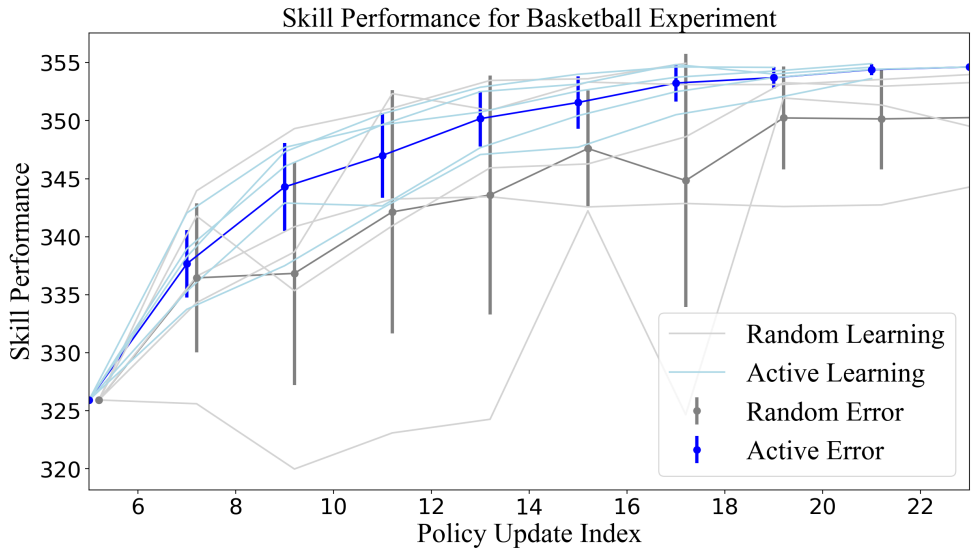

Figure 23: Skill performance on basketball skill: active (in blue) versus random task selection (in grey).

For both games, the average skill performance of random learning also improves continuously which benefits from incremental learning. However, the result shows that incremental learning combined with active learning could improve the skill performance faster and more consistently, achieving the predictable improvement behavior.

5.5 Conclusion

We choose ball-in-a-cup and basketball games as our research tasks, because the task parameters can be observed, evaluated easily and the task space is regular and continuous, which is beneficial to model the CSM with DMP. However, in fact, the task parameter is not easy to determine in more complex RL problems and the task might be inherently difficult to learn.

In [10], Da silva et al. considered equal task difficulty for all candidate task parameters. However, this assumption is not clear and it is challenging to model a task difficulty in a general case. The task difficulty can be indicated by reward, reward uncertainty or learning time. In this case, we proposed to stop learning the selected task and evaluate the skill performance for all candidate task parameters after fixed learning time of RL until the CSM can achieve successful skills for all task parameters. Compared to random task selection, the results show that active learning had smaller variance in skill performance, led to almost consistent results and achieved successful CSM in both ball-in-a-cup and basketball games. However, this active incremental learning algorithm is useful when the learning rate of reward can be modeled easily, otherwise $EISP$ cannot be predicted.

6 Summary

The goal of this thesis was to learn a contextual skill model (CSM) which maps task parameter to policy parameters using active incremental learning (see Algorithm 2). The learned CSM shared information across task parameters and provided successful policies for the entire range of task parameters at the end. The task order was actively selected by maximizing the expected improvement of skill performance over all task parameters. For both games, the result indicated that actively selecting the task order could improve skill performance significantly.

Instead of directly learning a CSM with several optimal policies, we updated the skill model iteratively. At each update stage, the next task parameter was actively selected for the CSM and the intermediate CSM provided a better initial policy of the selected task parameter than a human demonstration for the policy search. Then, the intermediate CSM was updated with the intermediate policies which may not have been optimized, since the selected task parameter is only optimized with a fixed number of policy updates. In our experiments, we divided the task space into three regions and always chose the most recent sample from each region to fit the CSM. We compared the performance of using all samples and partial samples. The result showed that the latter had a better performance, since the outdated samples which have not been updated for a long time may not contribute to improve the CSM.

In order to predict the reward improvement after a fix number of policy updates, we introduced the learning rate model which is determined at the initialization stage. In our experiments, we found the shape of the learning rate model is similar for different task parameters, which means the model is independent of task parameters, although this is not true for most reinforcement learning problems due to high stochasticity, where the learning rate model cannot be easily modeled or need to add uncertainties to the parameters. In our experiment, the learning rate model could also be parametrized with respect to task parameters, which require more data to estimate at the beginning. Therefore, the task-independent learning rate model was a better choice in our experiments.

In this thesis, the framework is agnostic to the type of policy representation, contextual skill model, and policy search. The main assumption behind the framework is that the reward function converges exponentially over time and the learning rate of reward is independent of task parameters. However, the framework was only tested in a simulated environment. Therefore, future studies should focus on implementing the application of the framework on a real robot and addressing the challenge of how to model the learning rate for more complex reinforcement learning problems.

References

[1] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot motor skill coordination with em-based reinforcement learning," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3232–3237, IEEE, 2010.

[2] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[3] S. Calinon, T. Alizadeh, and D. G. Caldwell, "On improving the extrapolation capability of task-parameterized movement models," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 610–616, IEEE, 2013.

[4] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning compact parameterized skills with a single regression," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 417–422, IEEE, 2013.

[5] D. Forte, A. Gams, J. Morimoto, and A. Ude, "On-line motion synthesis and adaptation using a trajectory database," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1327–1339, 2012.

[6] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[7] B. Nemec, R. Vuga, and A. Ude, "Efficient sensorimotor learning from multiple demonstrations," *Advanced Robotics*, vol. 27, no. 13, pp. 1023–1031, 2013.

[8] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-efficient generalization of robot skills with contextual policy search," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[9] G. Neumann *et al.*, "Variational inference for policy search in changing situations," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pp. 817–824, 2011.

[10] B. Da Silva, G. Konidaris, and A. Barto, "Active learning of parameterized skills," in *International Conference on Machine Learning*, pp. 1737–1745, 2014.

[11] J. Lundell, M. Hazara, and V. Kyrki, "Generalizing movement primitives to new situations," in *Conference Towards Autonomous Robotic Systems*, pp. 16–31, Springer, 2017.

[12] B. Da Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," *arXiv preprint arXiv:1206.6398*, 2012.

[13] 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.

[14] A. Fabisch and J. H. Metzen, "Active contextual policy search," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3371–3399, 2014.

[15] J. H. Metzen, "Active contextual entropy search," *arXiv preprint arXiv:1511.04211*, 2015.

[16] S. Thrun, "Is learning the n-th thing any easier than learning the first?," in *Advances in neural information processing systems*, pp. 640–646, 1996.

[17] Z. Chen and B. Liu, "Lifelong machine learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 10, no. 3, pp. 1–145, 2016.

[18] M. Hazara and V. Kyrki, "Speeding up incremental learning using data efficient guided exploration," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, May 2018.

[19] G. Fei, S. Wang, and B. Liu, "Learning cumulatively to become more knowledgeable," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1565–1574, ACM, 2016.

[20] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 2, pp. 1398–1403, IEEE, 2002.

[21] 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.

[22] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[23] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, "Google deep mind's alphago," *OR/MS Today*, vol. 43, no. 5, pp. 24–29, 2016.

[24] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

[25] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, pp. 849–856, 2009.

[26] M. Hazara and V. Kyrki, "Reinforcement learning for improving imitated in-contact skills," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 194–201, Nov 2016.

[27] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *AAAI*, pp. 1607–1612, Atlanta, 2010.

[28] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[29] F. Guenter, M. Hersch, S. Calinon, and A. Billard, "Reinforcement learning for imitating constrained reaching movements," *Advanced Robotics*, vol. 21, no. 13, pp. 1521–1544, 2007.

[30] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *Proceedings of the 24th international conference on Machine learning*, pp. 745–750, ACM, 2007.

[31] T. Rückstieß, M. Felder, and J. Schmidhuber, "State-dependent exploration for policy gradient methods," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 234–249, Springer, 2008.

[32] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*, vol. 135. MIT press Cambridge, 1998.

[33] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*, vol. 1. MIT press Cambridge, 2006.

[34] E. Schulz, M. Speekenbrink, and A. Krause, "A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions," *Journal of Mathematical Psychology*, vol. 85, pp. 1–16, 2018.

[35] J. H. Metzen, "Illustration of prior and posterior gaussian process for different kernels." https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_prior_posterior.html#sphx-glr-auto-examples-gaussian-process-plot-gpr-prior-posterior-py.

[36] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*, pp. 63–71, Springer, 2003.

[37] S. Vaara, T. Hirvola, K. Voutilainen, and J. Antonenko, "Simulating robot skills: lwrsim." https://wiki.aalto.fi/display/AEEproject/Simulating+robot+skills.

[38] Roboti, "Mujoco: advanced physics simulation." http://www.mujoco.org.

[39] M. Hazara and V. Kyrki, "Model selection for incremental learning of generalizable movement primitives," in *18th IEEE International Conference on Advanced Robotics (ICAR 2017), Hong Kong*, 2017.

[40] M. Hazara and V. Kyrki, "Transferring generalizable motor primitives from simulation to real world," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2172–2179, 2019.