Learning to Acquire and Adapt Contact-Rich Manipulation Skills with Motion Primitives

Anonymous Author(s)

Affiliation
Address
email

Abstract:
Learning generalizable robotic skills in a data-efficient manner has long been a challenge in the robotics community. While the current state-of-the-art methods using reinforcement learning (RL) show promising performance in acquiring manipulation skills, the algorithms are often data-hungry and hard to generalize. In this paper, we propose a simple yet effective framework to learn and adapt manipulation skills using motion primitives. The core of the method is using blackbox function optimization, optionally leveraging prior experiences similar to the task of interest, to learn/adapt the motion primitive parameters. Human demonstrations are modeled to serve as dense rewards guiding the parameter learning. We validate the effectiveness of the proposed method on peg-in-hole-like tasks. The experimental results show that our proposed framework takes less than 1 hour to acquire the insertion skills and as few as 15 minutes to adapt to an unseen insertion task on a physical robot.

Keywords: Motion Primitives, Contact-Rich Manipulation

1 Introduction

Reinforcement learning (RL) has been widely used to acquire robotic manipulation skills in recent years [1, 2, 3]. However, training an agent to perform certain tasks using RL is typically data-hungry and hard to generalize to novel tasks. This data-inefficiency problem limits the adoption of RL on real robot systems, especially in real-world scenarios.

Motion primitives, due to their flexibility and reliability, serve as a popular skill representation in practical applications[4, 5, 6]. A motion primitive, often realized by hybrid motion/force controllers, in our context, a Cartesian space impedance controller, is characterized by a desired trajectory generator and an exit condition. Taking moving until contact as an example primitive, the robot compliantly moves towards a surface until the sensed force exceeds a pre-defined threshold; a formal definition of motion primitives is deferred to Section 3.2. Despite the light-weighted representation and wide generalizability of the primitives, their parameters are often task-dependent, and the tuning requires domain expertise and significant trial-and-error efforts.

The goal of our work is to develop a data-efficient framework to learn and generalize manipulation skills based on skill primitives. Recently, several research efforts have been devoted to learning primitives for manipulation tasks [7, 8]. However, most works treat primitives as a mid-level controller and use RL to learn the sequence of primitives. While these methods reduce the exploration space by using primitives, they still suffer from the inherent drawbacks of RL: data inefficiency and lack of generalizability. To overcome the above issues, [4] proposed to use a black-box optimizer to search for the optimal primitive parameters in a pre-defined range to minimize the task completion time. However, we observe that the parameter range needs to be narrowly set; otherwise the optimizer spends long time exploring unpromising parameter choices. In addition, the task completion time, though continuous, is a sparse reward, as it is only triggered when the task succeeds. This prevents the optimizer from extracting information from failed task execution trials, which in turn requires a narrow parameter range to be carefully chosen. Motivated from the above limitations,
Figure 1: An overview of our proposed primitive learning and generalization framework. Our framework learns a dense objective function from human demonstrations and applies Bayesian Optimization (BO) to learn the primitive parameters w.r.t. the learned objective function. When generalizing to unseen tasks, we first select similar tasks from the task library based on introduced similarity metric, and then obtain a transferable search space for the new task for BO.

we propose a dense objective function that measures the likelihood of the induced execution trajectory being sampled from the same distribution as the successful task demonstrations. This model based objective function provides dense rewards even when a task execution trial fails, encouraging the optimizer to select parameters inducing execution trajectories more similar to the successful demonstrations, thus navigating the parameter space more efficiently. Furthermore, we propose a generalization method to adapt, as an example, an insertion skill to novel tasks leveraging an introduced task similarity metric, which alleviates the problem of requiring domain expertise to carefully set parameter ranges for novel tasks. In particular, socket (or hole) geometry information is extracted from the insertion tasks and the \( L_1 \) distance between turning functions [9] of two hole shapes is used as the task similarity metric. An overview of our learning and generalization framework is shown in Figure 1. Extensive experiments on 8 different peg-in-hole tasks are conducted to compare our proposed method with baselines. We experimentally demonstrate that our learning and generalization framework can effectively and efficiently i) learn to acquire manipulation skills with about 40 iterations (less than an hour) training on a real robot and ii) generalize the learned manipulation skills to unseen tasks with as few as 15 iterations (less than 15 minutes) on average.

The contributions of our work is summarized as follows:

1. We designed a manipulation policy composed of learnable motion primitives leveraging impedance control; rather than using sparse objective functions, we propose to learn a dense objective function from task demonstrations by modeling the execution trajectories.
2. We put forward a transfer learning method which retrieves similar tasks from a task library leveraging task meta information and adapts the previously learned parameters to the novel task with few interactions.
3. We collected 8 peg-in-hole tasks with diverse geometry shapes and clearance. Experimental results demonstrate that our method can acquire and adapt the insertion skills at practically acceptable time cost, 1 hour and 15 minutes respectively, while achieving higher task success rate than baselines.

2 Related Work

2.1 Learning Robotic Assembly Skills

Robotic assembly tasks, e.g., peg insertion, have been studied for decades and are still one of the most popular and challenging topics in the robotics community [10]. Recently, many works have focused on developing learning algorithms for assembly tasks, among which deep reinforcement learning (RL) methods are most common. For example, [11] proposed a self-supervised learning method to learn a neural representation from sensory input and used the learned representation as the input of deep RL. Schoettler et al. [12] combines a simple P-controller with a RL policy to reduce exploration space of the RL training. In [13], a reward learning approach is proposed to learn rewards from the high dimensional sensor input, and the reward is used to train a model-free RL
policy. There are also some works that combine RL and learning from demonstrations (LfD) to address the data inefficiency issue [14, 15]. However, the impractical amount of robot-environment interactions required by deep RL algorithms, and the domain expertise needed to adapt to different insertion tasks, limit their adoption in real-world, particularly industrial scenarios. Instead, motion primitives, often implemented with hybrid motion/force or impedance controllers, are often used for insertion tasks in practice. [4] makes the first attempt to learn the primitives via black-box optimizers, minimizing the task completion time. This algorithm work effectively if primitive parameter ranges are narrowly set; otherwise, due to the sparse reward choice, it takes a large amount of robot-environment interactions for the optimizer to escape the parameter region leading to unsuccessful executions. Thus, building on the framework of [4], we propose a learned dense objective function from demonstration, and develop a generalization method to avoid training from scratch for unseen tasks.

2.2 Generalizing Robotic Manipulation Skills to Unseen Tasks

Transferring the existing policies to novel tasks is extensively studied in the field of robotic manipulation recently. It is especially important for RL based approaches as most RL algorithms consider learning a task in simulation first and transferring the policy to the real world. One common approach is domain randomization, in which a variety of tasks are trained in simulation in order to capture the task distribution [16, 17]. Meta RL has gained significant attraction in recent years [18, 19], where the experience within a family of tasks can be adapted to a new task in that family. However, for motion primitive learning methods, often optimized with gradient-free parameter search methods [4, 6], there haven’t been efforts to utilize such prior experience on similar tasks. To the best of our knowledge, our work is one of the first attempts to encode the prior relevant task experience and reduce the parameter exploration space during primitive learning on a novel task.

3 Proposed Method

In this section, we detail our primitive learning and generalization framework for manipulation tasks. Compared against traditional RL methods, our framework is more data-efficient and interpretable. We deploy the proposed method on a wide range of peg-in-hole tasks for illustration, though our framework can also be adopted to other manipulation tasks.

3.1 Cartesian Space Impedance Control and the State-Action Space

Impedance control is used to render the robot as a mass-spring-damping system following the dynamics below,

\[ M(\ddot{x} - \ddot{x}_d) + D(\dot{x} - \dot{x}_d) + K(x - x_d) = -F_{ext} \]  

(1)

where \( M, D, K \) are the desired mass, damping, and stiffness matrices, and \( F_{ext} \) denotes the external wrench. \( \ddot{x}_d, \dot{x}_d, x_d \) are the desired Cartesian acceleration, velocity, and position of the end-effector, and \( \ddot{x}, \dot{x}, x \) are the current values correspondingly. We assume a small velocity in our tasks and set \( \ddot{x}, \dot{x}, x \) to 0, thus arriving at this control law,

\[ \tau = J(q)\dot{F} \]

\[ F = -K(x - x_d) - D\dot{x} + g(q) \]

(2)

where \( \tau \) is the control torque, \( F \) is the control wrench, \( J(q) \) is the Jacobian, and \( g(q) \) is the gravity compensation force.

Throughout a manipulation task, we would like to design a desired trajectory and a variable impedance to guide the robot movement. In favor of stability and ease of learning, we use a diagonal stiffness matrix \( K = \text{Diag}[K_x, K_y, K_z, K_{roll}, K_{pitch}, K_{yaw}] \), and, for simplicity, the damping matrix \( D \) is scaled such that the system is critically damped.

In summary, our manipulation policy output, \( a_t \in A \), fed to the impedance controller defined above, is composed of a desired end-effector pose \( x_d \) and the diagonal elements of the stiffness matrix \( k = \{K_x, K_y, K_z, K_{roll}, K_{pitch}, K_{yaw}\} \). The input to the policy, \( s_t \in S \), consists of end-effector pose \( x_t \) and the sensed wrench \( \bar{f}_t \), and is extensible to more modalities such as RGB and depth images.
3.2 Manipulation Policy with Motion Primitives

In this section, we provide a detailed design on our manipulation policy, which entails a state machine with state-dependent motion primitives. Each motion primitive $P_m$ associated with the $m$-th state defines a desired trajectory $f_{m}(x_{enter}, T)$, an exit condition checker $h_{m}(\cdot) : S \rightarrow \{0,1\}$, and a 6-dimensional stiffness vector $k_m$. $\theta_m$ contains all the learnable parameters in the primitive $P_m$. $x_{enter}$ denotes the end-effector pose upon entering the $m$-th state. $T$ contains the task information such as the 6 DOF poses of the peg and the hole; often, the hole pose defines the task frame of the motion primitives.

In the following, we formally describe the 4 motion primitives used in the peg-in-hole tasks, as shown in Figure 2.

**Free space alignment.** The end-effector moves to an initial alignment pose.

\[
\theta_1 = u(x_{enter}, x_{target}), \quad h_{\theta_1}(s_t) = \mathbb{I}[\|x_t - x_{target}\|_2 < \sigma], \quad k_1 = k_{max},
\]

where $\mathbb{I}[\cdot]$ is an indicator function mapping the evaluated condition to $\{0,1\}$. $u(\cdot, \cdot)$ generates a linearly interpolated motion profile from the first to the second pose provided. The target end-effector pose $x_{target}$ is extracted from the task information $T$ as $x_{target} = T_{hole}^base \cdot T_{pec}^hole \cdot T_{pec}^ee$.

**Move until contact.** The end-effector moves towards the hole until the peg is in contact with the peg top surface.

\[
\theta_2 = u(x_{enter}, x_{enter} - [0 0 \delta 0 0 0]^T), \quad h_{\theta_2}(s_t) = \mathbb{I}[f_{t,z} > \eta], \quad k_2 = k_{max}.
\]

$\delta$ is the desired displacement along z-axis in the task frame, $f_{t,z}$ is the sensed force along z-axis at time $t$, and $\eta$ is the exit force threshold. Therefore the parameters defining this 2-nd primitive consists of $\theta_2 = \{\delta, \eta\}$.

**Search.** The robot searches for the location of the hole while keeping contact with the hole until the peg and the hole are perfectly aligned. After empirical comparisons with alternatives including the commonly used spiral search, we choose the Lissajous curve as the searching pattern, which gives the most reliable performance. While searching for the translation alignment, the peg simultaneously rotates along the z-axis to address the yaw orientation error. The roll and pitch orientation errors are expected to be corrected by the robot being compliant to the environment with the learned stiffness.

\[
\theta_3(t) = x_{enter} + \begin{bmatrix}
    \frac{A \sin(2\pi a t/T)}{B \sin(2\pi b t/T)} \\
    -\gamma \\
    0 \\
    0 \\
    \varphi \sin(2\pi \frac{t}{T})
\end{bmatrix}, \quad h_{\theta_3}(s_t) = \mathbb{I}[x_{enter,z} - x_{t,z} > \zeta], \quad k_3 = k_{search},
\]

where $a = 7, b = 6$ are the Lissajous numbers selected and $T$ is the cycle period in Lissajous search, $\varphi$ is the maximum tolerated yaw error of the estimated hole pose, set as 6 degree in our experiments. The learnable parameters of this primitive are $\theta_3 = \{A, B, \frac{a}{T}, \frac{b}{T}, \gamma, \zeta, k_{search}\}$.

![Figure 2: An illustrative figure of the motion primitives designed for peg-in-hole tasks. We show the start and the end states of the robot for each primitive.](image)
**Insertion.** The peg is inserted into the hole in a compliant manner.

\[ f_{\theta_4} = u(x_{\text{enter}}, x_{\text{exit}} - [0 0 0 0 0]^T), \quad h_{\theta_4} = I[\text{success condition}], \quad k_4 = k_{\text{insertion}} \]

where the success condition is provided by the task information \( T \), e.g., \( ||x_t - x_{\text{success}}||_2 < \epsilon \). The primitive parameters to learn are \( \theta_4 = \{\lambda, k_{\text{insertion}}\} \).

### 3.3 Learning Primitive Parameters

In this section, we illustrate how to learn the primitive parameters \( \Theta = \{\theta_1, \theta_2, \theta_3, \theta_4\} \). The core idea is using Bayesian Optimization (BO) to optimize a task-relevant objective function \( J(\cdot) \). While a similar idea has been explored in [4], the objective function used is simply the measured task execution time. The major drawback of this objective function is that the objective signal is sparse and can only be triggered when the task is successfully executed. This makes the optimizer difficult to find a feasible region initially, especially when the primitive parameter space is large. Motivated by this, we propose a dense objective function that measures the likelihood of the induced execution trajectory being sampled from the distribution of successful task demonstrations \( \mathcal{E}_D = \{\xi_i\} (i = 1, 2, ..., M) \). Assuming the trajectories are Markovian, a trajectory rollout \( \xi = [x_0, x_1, ..., x_{n-1}] \) is modeled as:

\[
\begin{align*}
  p(\xi; \Theta) &= p(x_0) \prod_{i=1}^{n-1} p(x_i | x_0, ..., x_{i-1}) = p(x_0) \prod_{i=1}^{n-1} p(x_i | x_{i-1}). 
\end{align*}
\]

In order to learn \( p(x_i | x_{i-1}) \) from demonstrations, we first use a Guassian Mixture Model (GMM) to model the joint probability as

\[
\begin{align*}
  p(\left[ \begin{array}{c} x_i \\ x_{i-1} \end{array} \right]) &= \sum_{j=1}^{K} \phi_j \mathcal{N}(\mu_j, \Sigma_j), \quad \text{where } \sum_{j=1}^{K} \phi_j = 1, \text{ and } K \\
  \text{is the number of GMM clusters.}
\end{align*}
\]

We further represent the Gaussian mean \( \mu_j \) and variance \( \Sigma_j \) as: \( \mu_j = \begin{bmatrix} \mu_j^1 \\ \mu_j^2 \end{bmatrix} \), \( \Sigma_j = \begin{bmatrix} \Sigma_{j1}^{11} & \Sigma_{j1}^{12} \\ \Sigma_{j2}^{12} & \Sigma_{j2}^{22} \end{bmatrix} \). We can then derive the conditional probability

\[
\begin{align*}
  p(x_i | x_{i-1}) &= \sum_{j=1}^{K} \phi_j \mathcal{N}(\mu_j, \Sigma_j), \quad \text{where}
  \\
  \mu_j &= \mu_j^1 + \Sigma_{j2}^{22} (\Sigma_j^{22})^{-1} (x_{i-1} - \mu_j^2), \\
  \Sigma_j &= \Sigma_{j1}^{11} - \Sigma_{j2}^{12} (\Sigma_j^{22})^{-1} \Sigma_{j2}^{21}.
\end{align*}
\]

After obtaining the analytical form of \( J(\xi) = \log p(\xi; \Theta) \), we use BO to solve \( \Theta^* = \arg\max_{\Theta} J(\Theta) \).

**Expected Improvement (EI) is used as the acquisition function, and we run BO for \( N \) training iterations.** The learned parameter \( \Theta^* \) that achieves maximum \( J(\Theta) \) during \( N \) training iterations is selected as the optimal primitive configuration.

### 3.4 Task Generalization

In this section, we detail our method on how to leverage prior experience when adapting to a novel insertion task, in particular, how to adapt previously learned peg-in-hole policies to different tasks with unseen hole shapes. Our adaptation procedure is composed of two core steps: measuring task similarities and transferring similar task policies to unseen shapes.

#### 3.4.1 Measuring task similarity

Given an insertion skill library, i.e., a set of learned peg insertion policies for different shapes, \( \mathcal{M} = \{\pi_1(\Theta_1), \pi_2(\Theta_2), ..., \pi_n(\Theta_n)\} \) and an unseen shape, our goal is to first identify which subset of the \( n \) tasks are most relevant to the new task. While there is a diverse range of auxiliary task information that can be used to measure task similarity, here we define the task similarity as the similarity between the hole cross-section contours. This assumption is based on the intuition that similar hole shapes would induce similar policies for insertion. For example, the insertion policies for a square hole and a rectangle hole are likely to be similar, and the optimal policy for a round hole might still work for a hexadecagon hole. The similarity between a shape pair is measured by the \( L_1 \) distance between the two shapes’ turning functions [9].
Turning functions are a commonly used representation in shape matching, which represent the angle between the counter-clockwise tangent and the x-axis as a function of the travel distance along a normalized polygonal contour. Two example turning functions are shown in Figure 3. After obtaining the shape distances of the unseen shape and each shape in the task library, we choose top $L$ shapes that are closest to the unseen shape as the similar shapes. The policies of the similar shapes are then used as input for transfer learning detailed below in Section 3.4.2.

### 3.4.2 Adapting policies to unseen shapes

Given a novel task, our goal is to efficiently adapt the already learned manipulation policies of the most similar shapes. We build upon BO with hyperparameter transfer [20]. Unlike many works framing BO transfer learning as a multi-task learning problem, here we attempt to learn the search space of BO from the similar task policies and apply it to learning for the new task.

Specifically, let $T = \{T_1, T_2, ..., T_t\}$ denotes the task set of different hole shapes we selected as described in Section 3.4.1, and $F = \{J_1, J_2, ..., J_t\}$ denotes the corresponding objective functions for each task. All the objective functions are initially defined on a common search space $X \subseteq \mathbb{R}^{|\Theta|}$, and it’s assumed that we already obtained the optimal policies for the $t$ tasks $\{\pi_1(\Theta^*_1), \pi_2(\Theta^*_2), ..., \pi_t(\Theta^*_t)\}$ ($\Theta^*_i \in X$). Given a unseen task $T_{t+1}$, we aim to learn a new search space $\tilde{X} \subseteq X$ from the previous tasks to expedite the new task learning process. We define the new search space as $\tilde{X} = \{\Theta \in \mathbb{R}^{|\Theta|} | l \leq \Theta \leq u\}$, where $l, u$ are the lower and upper bounds. It was proved in [20] that the new search space can be obtained by solving the constrained optimization problem:

$$
\min_{l,u} \frac{1}{2} \|u - l\|^2_2 \text{ such that for } 1 \leq i \leq t, \ l \leq \Theta^*_i \leq u.
$$

This new search space is then utilized for policy training of this unseen shape task, following the procedure described in Section 3.3.

### 4 Experimental Results

We aim to investigate the effectiveness of our primitive learning and generalization framework by answering two questions: 1) if the dense objective function proposed in Section 3.3 expedites the primitive learning process and improves policy performance, and 2) if the generalization algorithm described in Section 3.4 is effective when transferring to an unseen shape. An insertion task library of 8 different peg-hole pairs is constructed, including 6 representative 3D-printed geometry shapes (round, triangle, parallelogram, rectangle, hexadecagon, ellipse) and two common industrial connectors (RJ45, waterproof). Examples of the pegs and holes are shown in Figure 4.

### 4.1 Experimental Setup

Our experiment setup is shown in Figure 1, where a 6-DoF FANUC Mate 200iD robot is used throughout our experiments. The robot is equipped with an ATI Mini45 Force/Torque sensor. The clearances for all 3D printed peg-hole pairs are 1mm; the waterproof and RJ45 are unaltered off-the-shelf connectors. To mimic the pose estimation error during industrial
deployments, a uniform perturbation error of +/- 5mm in translation and +/- 6 degree in orientation is applied along each dimension. The controller takes the manipulation policy output at 10 Hz and computes the torque command streamed to the robot at 1000Hz. All the learnable parameters of the policy and their initial range are listed in Table 1.

Two metrics are used to evaluate the effectiveness and efficiency of the approach, i.e., the number of iterations the robot takes to accomplish the first successful insertion during training (denoted as number of iterations), measuring the efficiency of the algorithm, and the success rate of the optimal policy after a fixed number of iterations (denoted as success rate), quantifying the algorithm effectiveness.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Move until</th>
<th>Search</th>
<th>Insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>max</td>
</tr>
<tr>
<td>x1</td>
<td>0</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>x2</td>
<td>0</td>
<td>10</td>
<td>0.06</td>
</tr>
<tr>
<td>δ1</td>
<td>1</td>
<td>0.02</td>
<td>0.04</td>
</tr>
<tr>
<td>δ2</td>
<td>0</td>
<td>0.02</td>
<td>0.04</td>
</tr>
<tr>
<td>H</td>
<td>2/10</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>t1</td>
<td>0/60</td>
<td>20/10</td>
<td>600<strong>1-1</strong></td>
</tr>
<tr>
<td>t2</td>
<td>0/60</td>
<td>0.02</td>
<td>40<strong>1-1</strong></td>
</tr>
<tr>
<td>γ</td>
<td>0</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>λ</td>
<td>0</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>k_search</td>
<td>0</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>k_insertion</td>
<td>0</td>
<td>0</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 1: Learnable parameters and corresponding range in the motion primitives.

![Figure 5](image-url) A sample trajectory during training on a triangle peg-in-hole task. The corresponding primitive parameters are reasonable but the insertion narrowly failed.

### 4.2 Learning Primitive Parameters

In this experiment, we would like to examine if the learned objective function expedites the primitive learning process. Specifically, we applied our learned objective function from demonstration (LfD) for primitive parameter optimization as described in Section 3.3, and compare the results against primitive optimization by minimizing the measured task execution time (T ime) [4]. When learning the objective function, the number of GMM clusters is set as K = 25.

We conducted the experiments on all 8 peg-hole pairs described previously. For each experiment, we run BO for 40 iterations. Within each iteration, the current policy is executed twice with independently sampled hole poses. The average objective of the two trials is used as the final objective value consumed by the BO step. The optimal policy is selected as the policy at the BO step when the optimal objective value is achieved, and evaluated by being executed 20 trials with independently sampled hole poses. As shown in Figure 6, LfD outperforms T ime on almost all the insertion tasks in both number of iterations and success rate. In some tasks, e.g., parallelogram and RJ45, T ime cannot find proper parameters to achieve a single successful trial within 40 iterations, while LfD successfully navigates through the parameter space and accomplishes successful trials for all of the tasks. This validates the learned dense objective function, i.e., a likelihood function measuring trajectory similarity w.r.t. the demonstrations, provides richer information for primitive learning than sparse signals like task success and completion time. To better illustrate the advantage of our learned objective function from demonstration, we show a sample trajectory during training in Figure 5. The search trace thoroughly covers the hole area and the corresponding parameters should be a reasonable candidate for the desired primitive configuration. However, the insertion did not succeed due to stochastic noise, and the optimizer discourages exploring the search space around this candidate in T ime. By contrast, this candidate gets assigned a high objective value in LfD since its induced trajectory has a likelihood evaluated by our learned model from demonstration.

### 4.3 Generalizing to Unseen Shapes

We now examine how our generalization method described in Section 3.4 performs when transferring to unseen shapes. We consider the leave-one-out cross validation experiment setting, i.e., when presented 1 of the 8 tasks as the task of interest, the learning algorithm has access to all interaction data during policy learning on the other 7 tasks. Two sets of experiments are conducted. First, we apply the full method described in Section 3.4 (Fu11) to learn a reduced search space using the 3 most similar tasks, within which the primitive parameters are optimized.
Figure 6: Experimental results for primitive learning and generalization. Time: primitive learning using task execution time as objective function, LfD: primitive learning using learned objective function as described in Section 3.3, NoSim: primitive generalization without measuring task similarities, Full: full generalization method as described in Section 3.4. * represents no successful trials is found during the learning process. The results indicate the effectiveness and efficiency of our learning and generalization framework.

<table>
<thead>
<tr>
<th>Shape</th>
<th>Time</th>
<th>LfD</th>
<th>NoSim</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>success rate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of iterations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7: The similarity distances between all insertion tasks.

Compared with LfD where parameters are optimized over the full space, Full reached a comparable or better success rate, meanwhile achieved the first task completion with significantly lower number of iterations. Second, we consider learning the search space without measuring the task similarities (NoSim), i.e., the new search space is obtained using all the other tasks instead of only the similar ones. As seen in Figure 6, Full outperforms NoSim consistently on all insertion tasks, indicating the significance of finding similar tasks using the task geometry information before learning the new search space. The similarity distances between all insertion tasks are shown in Figure 7 to qualitatively demonstrate the effectiveness of using the task geometry information. For example, given an unseen waterproof insertion task, the triangle, rectangle and parallelogram shapes are selected with the smallest similarity distances. With such knowledge encoded, in Full, a narrower search space is obtained compared to NoSim, thus leading to better performance by focusing the BO budgets in the parameter subspace with higher chance of task success.

5 Discussion

We propose a data-efficient framework for learning and generalizing manipulation skills with motion primitives. Extensive peg-in-hole experiments on 8 different shapes are conducted to demonstrate the advantages of our method. The results show that our framework enables a physical robot to learn peg-in-hole manipulation skills and to adapt the learned skills to unseen tasks at low time cost.

One limitation of the proposed generalization method is that the skill transfer is only beneficial across a narrow task family, e.g., peg-in-hole with different shapes. One future direction is generalizing the skills across more diverse tasks [10, 21]. Besides, our current framework assumes an already designed primitive sequence composing the manipulation policy. Learning the primitive sequence from demonstration or self-supervisedly is another path to explore.
References


