2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 23-27, 2022, Kyoto, Japan

2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-7927-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/IROS47612.2022.9981422

Efficiently Learning Manipulations by Selecting Structured Skill Representations

Mohit Sharma[1] and Oliver Kroemer[1]

Abstract— **A key challenge in learning to perform manipulation tasks is selecting a suitable skill representation. While specific skill representations are often easier to learn, they are often only suitable for a narrow set of tasks. In most prior works, roboticists manually provide the robot with a suitable skill representation to use *e.g.* a neural network or DMPs. By contrast, we propose to allow the robot to select the most appropriate skill representation for the underlying task. Given the large space of skill representations, we utilize a single demonstration to select a small set of potential task-relevant representations. This set is then further refined using reinforcement learning to select the most suitable skill representation. Experiments in both simulation and real world show how our proposed approach leads to improved sample efficiency and enables directly learning on the real robot.**

I. INTRODUCTION

Manipulation skills, or primitives, are crucial building blocks for robotic manipulation as they introduce appropriate task structure and make complex tasks feasible. A key challenge for using manipulation skills is to choose an appropriate skill representation. For instance, we can use monolithic neural-network controllers [1], [2] to generate trajectories or low-level actions for many different manipulation tasks. On the other hand, we can also use more specialized skill representations, *e.g.*, parameterized force-controllers [3], object-centric attractors [4], [5], or Dynamic Movement Primitive (DMP) [6]. While general skill representations may enjoy widespread applicability their over-parameterization can make them unnecessarily difficult to learn. More specialized skill representations often work well in only limited settings, but they can be faster to learn and adapt given their fewer parameters. The robot should therefore ideally use the most specific skill representation that is still suitable for the given task.

In addition to the type of skill representation, skills can also benefit from utilizing representations that exploit the geometric structure of manipulation tasks. Many recent methods have used object-centric keypoints [7], [8], [9] and task-frames [4], [10] to represent this structure of manipulation tasks. The use of geometric structure allows skills to be invariant to certain task-irrelevant object properties, *e.g.* translation invariance when grasping an object. This allows the learned skills to generalize better to a wide variety of different scenarios. However, these object-centric methods often forgo the choice of skill representation and either use one step rigid transforms [7], fixed primitives [9] or manually-defined parameterized controllers [4].

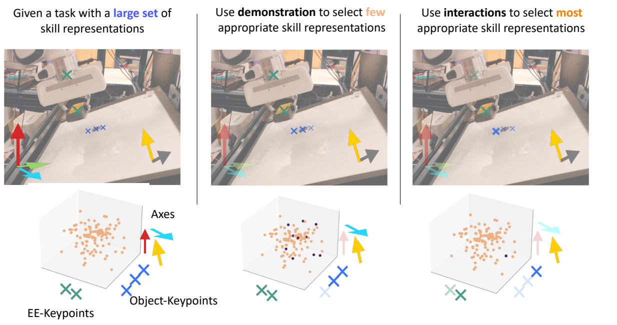
Given a task with a large set of skill representations
Use demonstration to select few appropriate skill representations
Use interactions to select most appropriate skill representations
Axes
EE-Keypoints
Object-Keypoints

Fig. 1: Overview of our approach. We represent object and end-effector keypoints using $\times$ (object keypoints as $\times$, end-effector keypoints as $\times$) and axes as $\uparrow$. *Left*: Large set of skill representations. *Middle*: Use few demonstrations to select a small set of relevant skill representations. *Right:* Use RL to select the most appropriate skill representation for the task. The bottom plots show the likelihood values over different elements of our skill representation. *Bottom Left:* Initially, we have similar likelihoods for all skill representations. *Bottom Middle:* Using a demonstration we select skill representations with high likelihood (shown as black dots). *Bottom Right:* We use interactions to directly learn the most relevant skill representation (black dot) from this imitation reduced set.

In this paper, we present a framework for robots to efficiently learn manipulations by autonomously selecting appropriate skill representations. These representations include both the geometric task structure and the types of controller representations for each axis. Selecting a suitable structured skill representation without any prior knowledge is difficult and inefficient. To overcome this, we propose to use a small set of demonstration data. These demonstrations provide us examples of successful trajectories and thus allow us to quickly exclude skill representations with low likelihood for the provided trajectories.

Given the task demonstration data, we use imitation learning to identify a relatively small sets of distinct skills that con possibly accomplish the task. We learn a high-level policy over the skills, or options, to select between the skills and ultimately sequence them together to perform larger tasks.

Imitation learning (IL) provides a good initialization for skill parameters and rules out many unsuitable skill representations. However, demonstrations may be ambiguous and contain insufficient data to select one specific skill representation and associated parameter values for robust task

[1] Robotics Institute, Carnegie Mellon University, {mohitsharma, okroemer}@cmu.edu

This work was supported by NSF Award No. CMMI-1925130, Office of Naval Research Grant No. N00014-18-1-2775, and ARL grant W911NF-18-2-0218 .

execution. For example, when wiping a horizontal tabletop, a single demonstration cannot indicate that the scrubbing force should be applied in the direction normal to the surface rather than in the vertical direction. To disambiguate the skill representation, the robot would need to observe the wiping of an inclined surface. Rather than rely on an exhaustive set of demonstrations, the robot uses reinforcement learning to refine its skill representation selection and parameter values. This skill refinement is efficient as IL already rules out large numbers of unsuitable skill representations and thus provides a more structured skill space for the agent.

We evaluate our proposed framework on both simulated and real robot experiments. We use multiple tasks from Robosuite [11], as well as real world surface-wiping and toolbox-opening tasks. Our evaluations show that our framework is able to make efficient use of the demonstrations to quickly identify a small set of possible skill representations which can then be refined and learned through interactions directly on the real system. Please see supplementary materials, with additional results and experiment videos, at `https://sites.google.com/view/efficient-manipulation`.

II. Related Work

Manipulation Skills can be represented in many different ways. Early work on manipulation skill representation used both geometric and mechanical features of the task to define manipulation primitives such as position, force primitives or visual servoing primitives [12], [13]. Contact states have also been used to define manipulation skills [14]. While early skills were often manually parameterized, recent skills have become more versatile utilizing learned parameters and preconditions [15], [16], [17], [18], [19], [20]. Some of the widely used skill representations include, Dynamic Movement Primitives (DMPs) [6], [21], [22], Neural Networks [23], [24], [25], Task-Parameterized GMMs (TP-GMMs) [26], [27]. In contrast to the above approaches, we do not propose a new skill representation but instead we utilize these different skill representations and learn to select the most appropriate skill-representation for the underlying task.

Task Frames: Manipulation skills inherently involve object interactions, and objects are usually defined by their task axes or task frames. Task frames are used ubiquitously in robotics. Their widespread usage within manipulation is also termed as Task-Frame Formalism (TFF) [28], [29], [30]. Previous works have used task frames to define task-motion constraints for compliant robot motions [31], [32], [33], programming by demonstrations [34], as well as inferring relevant task frame using demonstrations [35], [36], [37].

More recently task frames have been used (although under different names) to provide meaningful generalization in various task settings [7], [4], [10], [8]. Among these works the closest to our approach is [4], [10]. The authors of [4] define *oriented keypoints* which are task-specific keypoints with orientation (i.e. axes), thus analogous to task frames. These oriented keypoints are used in conjunction with manually defined controllers to solve different manipulation tasks.

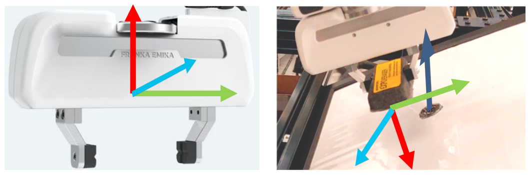

Fig. 2: *Left:* Palm keypoint with an arbitrary frame. *Right:* Tool keypoint with EE frame and surface normal (black arrow) for black mark on the board. The other in-plane axes for the black mark are much more arbitrary.

Alternatively, [10] uses keypoints and task-axes to define a large set of controllers which are used to solve manipulation tasks using RL. By contrast, our approach tries to unify [4] and [10], *i.e.*, we want our robot to be able to learn a task directly in the real world without having to manually parameterize everything. To avoid this manual specification we use a small set of task demonstrations and a geometric skill structure that is closely related to [4], [10].

III. Preliminaries: Skill Fundamentals

We use a compositional approach towards skill representation. Our skill representation is grounded in the geometry of the given task and its associated environment. We utilize a task-frame based formulation to utilize this geometric representation. Task-frame formalism (TFF) [32], [28], [29] allows us to associate 3D frames with task relevant objects in the environment. Instead of relying on full 3D task-frames we decompose each task-frame into a set of task-axes and only use the relevant axes for each task. This formulation is important in scenarios where frames or axes on different objects are required for control, *e.g.* in a wiping task we only need the surface normal of the surface being wiped while the target location is the marker to be wiped (Figure 2 Right).

Skill Representation: Each skill in our formulation consists of a set of one or more task-axes controllers (TACs). Overall, each task-axes controller (TAC) is parameterized by an object keypoint $(k \in K)$ that acts as the anchor for the controller, an end-effector keypoint $(e \in E)$ which indicates which part of the end-effector will interact with the object, a task-axes $(u \in U)$ that denotes the motion axes, and a controller with type $(c \in T)$ and corresponding parameters (θ) (*e.g.* DMP weights, or control-points for splines), which is used to generate the motion. Table I lists these elements and we describe each of them below. We use uppercase symbols K, E, U, T to denote sets of end-effector and object keypoints, axes, and controllers. Each element of these sets is indexed using lowercase symbols, *i.e.*, k, e, u, c respectively. Figure 3 visualizes how TACs are combined together to achieve skill representations.

A. Object and End-Effector Keypoints

We utilize keypoint-based representations to ground the task-axes controllers in the task environment. In this work, we assume a fixed set of object keypoints, denoted as K, which can be tracked in the environment. It is possible to predict these keypoints directly from the image input or the

TAC Element	Values
Object-Keypoint	Keypoints on object surface, center of object, object edges or corners
End-Effector Keypoints	End-effector center, Points on fingers, Palm points, Palm edges, Tool Keypoint
Task Axes	Global axes, Surface normals, Object axes, Linear axes to goal keypoints
Controller Types	Min-Jerk controllers, Force controllers, Alignment controllers, DMPs, Neural Networks

TABLE I: Different elements of task-axes controllers along with the possible values each element can take.

3D point cloud representations. However, this often requires pre-training to learn features based on local object geometry [38], [8]. Further, in some scenarios using object-part centric approaches [39] may also be useful, *e.g.* grasping an airplane model from its tip. All of the above approaches can provide suitable keypoint representations for TACs.

In addition to the object keypoints we also utilize end-effector keypoints E that ground the end-effector interaction with the object. Figure 2 visualizes the palm end-effector keypoints. Overall, we use the end-effector center, finger keypoints, tool keypoint or palm keypoints. Although for most interactions the middle keypoint of the end-effector suffices, for many scenarios utilizing the fingers of the robot or the palm surface may also be relevant.

B. Task Axes

Each TAC is associated with some axis $u \in U$ along which it acts ($u \in \mathbb{R}^3$). While prior works often assume full 3D frames associated with each keypoint [4], [40], we decompose these frames into separate task axes. This is beneficial since in many scenarios only some axes of the full 3D reference frame are useful. For instance, in the peg insertion task or wiping tasks in [4] it is only the Z-axes of the hole or the surface normal of the board which are important, while the other axes can be more arbitrary *e.g.* any two axes in the plane normal to the hole will suffice for circular peg insertion. Based on this, we note that for most manipulation tasks a common set of axes can be used. These include the global X, Y, Z axes, the surface normals on objects, object-axes (which can be found from object-pose if known), linear axes from ee-keypoints to anchor keypoints (that are estimated as $k - e$, where k is an object keypoint and e is the end-effector keypoint), and joint-axes which can be used to define motion constraints.

C. Controller Types and Parameters

Each TAC is further associated with a feedback-based controller that generates the robot motion. These controllers can combine trajectory generators and feedback controllers together to generate the final robot motion. These TAC controllers can utilize the object and end-effector keypoints discussed above. However, they have their own set of parameters c_θ for motion generation. Table I lists the possible set of controllers.

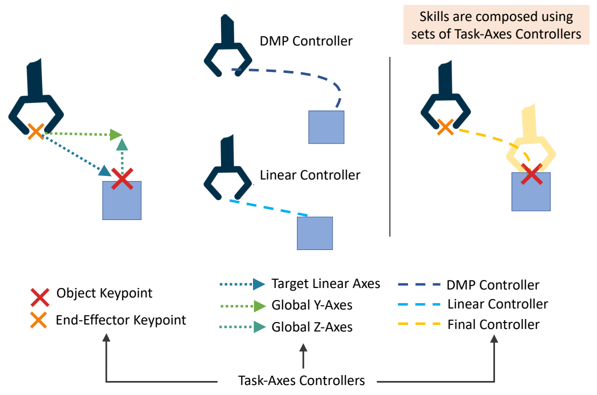

Fig. 3: Skill composition using different TAC representations.

Min-Jerk controllers are parameterized with the goal keypoint $k \in K$ and an offset $k_o \in \mathbb{R}^3$ with respect to these keypoints. Thus, the final target for the given end-effector keypoint is $k + k_o$, and these controllers act along the provided task-axes. They are further parameterized by the duration to reach the target.

Force controllers are parameterized by a desired force target $f_t \in [f_{\min}, f_{\max}]$ and act along the given task-axes.

Dynamic Movement Primitive based controllers are parameterized with goal location (*i.e.* the keypoint k and keypoint offset k_o). DMPs can be used to imitate arbitrary smooth motions by learning appropriate shape parameters $w \in \mathbb{R}^N$, (where N are the number of basis vectors) used to represent the forcing function [6], [21].

Alignment controllers are used to align some axes of the end-effector with some task-axes such as the surface normal of the whiteboard to wipe. These controllers can additionally be parameterized with (θ, ϕ) which specify the remaining 2-DOF for the 3-DOF orientation space.

Neural Network controllers can be parameterized with respect to goal locations (which are inferred from keypoints and offsets). The parameters for neural network controllers consist of all the weight and bias parameters in their layers. Given different TACs we compose them together using null-space projections similar to [5].

IV. Learning and Selecting Skills using Demonstrations

Taking into account the possible set of object and end-effector keypoints, axes and controller types we get a combinatorial explosion for different possible TACs to choose from. Trying to learn the appropriate TACs and their parameters from this large set is prohibitively expensive. To avoid this issue, we use a small set of demonstrations (≤ 5) to narrow down the possible set of valid skills for the given task, and provide a multi-modal space of skills for the agent to explore.

A. Task Segmentation:

Before learning appropriate TACs we segment a given task into multiple subtasks. This segmentation breaks down long horizon problems and is often necessary given the non-linearity around contact mode changes. We use change point detection to find contact mode transitions and use them to decompose the given task demonstration into multiple subtasks. For each subtask we now aim to find a subset of relevant TACs which are composed using null-space projections. TACs for all subtasks are then composed over time to accomplish the given task.

B. Computing Posteriors over Skill Representations

Given demonstrations for each subtask we would now like to infer a distribution over all sets of TACs to identify a small set of valid controllers that could be used to perform the task. For this we use a likelihood based approach. We denote by $q(k, e, u, c, c_\theta \in \mathbb{R}^N)$ some prior distribution over TACs. Given the demonstration data X we would like to find the posterior distribution over these parameters $p(k, e, u, c, c_\theta|X)$, where

$$p(k, e, u, c, c_\theta|X) \propto p(X|k, e, u, c, c_\theta)q(k, e, u, c, c_\theta). \quad (1)$$

Since the set of possible keypoints and controllers can be very large, we use some general priors to efficiently estimate the posterior distribution in (1). This set of priors is applicable across a large variety of manipulation tasks.

Proximity and Alignment priors: We first note that the geometric properties for the TACs, i.e, the object and end-effector keypoints and task axes can be separated from the feedback controllers. This is because given a set of values for these geometric properties multiple different controllers can be used to generate the robot motion. More specifically, the object keypoint parameter is used as an anchor for the controller the end-effector keypoint is used as the origin for the controller, and the task-axes projects the motion onto the appropriate direction. Given these parameters different feedback controllers can be used to generate the motion. We utilize this property of geometric properties to rewrite (1) as,

$$p\left(k, e, u, c, c_\theta|X\right) = p\left(c, c_\theta|X, k, e, u\right)p\left(k, e, u|X\right). \quad (2)$$

We estimate the second quantity in this product of posteriors $p\left(k, e, u|X\right)$ using proximity and alignment priors. A proximity prior allows us to find the most relevant keypoints based on the idea that relevant object and end-effector keypoints should be close to each other at the end of the interaction. Similarly, for the alignment prior we project the motion onto the relevant task axes and find proximity along this axes. Denoting the end-effector keypoint trajectory as $X^e_{1:T}$ (we use the subscript to denote time) we transform it to the appropriate object-keypoint and axes using

$$X^{e|k,u}_{1:T} = -\left(uu^T\right)\left(X^e_{1:T} - k\right), \quad (3)$$

where we use the notation $X^{e|k,u}_{1:T}$ to denote the trajectory for end-effector keypoint e with object-keypoint k as origin and along task-axes u.

Given this projected trajectory we use heuristics to estimate the likelihood for the given geometric parameters *i.e.*, $P(X|k, e, u)$. We use simple distance-based heuristic based on the last position of the demonstration trajectory,

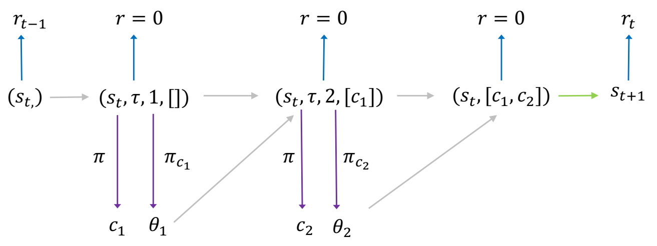

Fig. 4: The expanded-MDP formulation for parameterized actions used in our approach.

$$p(X|k, e, u) = \exp\left(-\frac{||X^{e|k,u}_T - \hat{k}||_2}{\eta}\right) \quad (4)$$

where $\hat{k}$ is the 3D location for the given keypoint k and η is the temperature parameter. Using a low temperature value focuses on the closest keypoints alone while a larger value may use multiple keypoints. We note that other heuristics which look at the convergence behaviors of demonstrations can also be used [35], [41], [36].

Additionally, to estimate the full posterior $p(k, e, u|X)$ we can incorporate an informed prior $q(k, e, u)$,

$$p(k, e, u|X) = \frac{\exp\left(-||X^{e|k,u}_T - \hat{k}||_2/\eta\right) + q(k, e, u)}{\sum_{k',e',u'} \exp\left(-||X^{e'|k',u'}_T - \hat{k'}||_2/\eta\right) + q(k', e', u')}$$

These priors encapsulate task knowledge to guide the learning process. For instance, for prehensile manipulation a relevant prior is to use the middle EE-keypoint, while for tool-based tasks, when the hand is already grasping an object, the prior can focus on tool keypoints. Previous approaches to task-frame selection, and the more recent keypoint based approaches, use similar priors on end-effector keypoints [4].

C. Controller Selection and Learning

To estimate $p(c, c_\theta|X, k, e, u)$, we need to calculate the likelihood $p(X, k, e, u|c, c_\theta)$. For this we we use a set of criterions for the different controller types listed in Table I. We note that as before we project the trajectory data onto the particular axes similar to (3). We use force-controller only in scenarios where the observed force in the projected force-data is beyond a threshold. We use a threshold of 2N in simulation and slightly larger 6N in real world given noisy force-torque measurements in real world. The force controller is parameterized by a force-value which is directly inferred from the demonstration.

For linear controllers (min-jerk) we use a simple linear regression on the non-zero velocity part of the trajectory to verify if the linear motion does not underfit the observed trajectory. We parameterize a linear controller with the time required to reach the target. On the other hand, DMP and Neural-Network (NN) controllers can fit arbitrary trajectories and thus we can always learn parameters for these controllers that correspond to the observed demonstration trajectories.

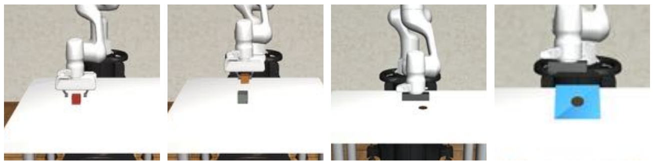
Fig. 5: Sample tasks from robosuite [11], from left to right – Lift, Stack, Wipe, Wipe with slanted board.

However, given limited training data NN controllers can easily overfit to the trajectories and may not generalize well. We provide more details on controller criterions in the appendix on the project website.

V. Refining Skills from Interactions

We use a threshold on $p(k, e, u, c, c_\theta|X)$ to restrict the valid TACs to a small set. However, this may not result in one specific set of TAC composition for each subtask. Since in some scenarios there may exist ambiguity in the provided demonstrations, *e.g.*, for wiping task the normal of the surface may align with the global Z-axes. Also, sometimes multiple TACs may fit the observed trajectory equally well. For instance, a close to linear motion will have high likelihood for both linear and DMP controllers. Finally, we may also want to finetune TAC parameters, such as force value for force controllers, weights for DMPs or NNs, or time for linear controllers.

We use RL to get the final prioritized set of TACs for each subtask. We initialize the agent's action space with the TACs inferred from the demonstrations including their parameters. We use an expanded MDP formulation where at each step we select some TAC as well as its parameters. Figure 4 visualizes a few steps of the expanded-MDP formulation. Our state consists of $(s_t, \tau, p, [c_0, \cdots])$, where s_t denotes the environment state, τ is the task-segment index (1-hot), p is the priority of the current controller being selected (1-hot) and a one-hot list of previously selected controllers. We take 1 step in the environment for each task-segment (τ), thus the total MDP horizon is the number of task segments times the number of controllers being selected. The multiple selected TACs are combined using null space projection and the composed controller is executed in the environment.

To train our policy we use the parameterized action MDP (PAMDP) formulation [42], [43]. In particular we use an approach similar to [44] to optimize our policy (we provide details in the appendix). Specifically, we use one network π that selects the next TAC controller to execute while the parameters of each TAC are found separately using π_{c_i}.

VI. Experimental Results

With our experiments we aim to answer the following questions: 1) Can our proposed approach extract relevant skill information and discard irrelevant TACs using the demonstrations? 2) How efficiently can we use the reduced TACs for overall task completion? 3) Does our approach allow real-world learning on the robot?

Setup: To evaluate our approach we use three different tasks from robosuite [11] – Lift, Stack, and Wipe (Figure 5).

Task	Obj-Keypoint	EE-Keypoint	Axes	Controllers
Lift	top-surface	EE-tip	Global, Linear, Surface Normal	DMP, MinJerk
Stack	top-surface	EE-tip, object's tip	Global, Linear, Surface Normal	DMP, MinJerk
Wipe	marker	Tool-tip	Global, Linear, Surface Normal	DMP, MinJerk, Force, Alignment

TABLE II: Skill parameters extracted from demonstrations.

Each of these tasks are of varying difficulty and evaluate different aspects of our approach. For the Lift task we do not make any changes to the robosuite env. While for Stack we make one change, *i.e.*, we ensure that the blocks to be stacked will have a minimum distance of $3cm$ between them. This ensures that the demonstration trajectory environment and the train environments align with each other. Finally, for the Wipe task we make two changes. First, we reduce the number of markers to only 1, this makes it easy for us to test our approach in the real world, since we can use a simple vision system to detect this marker. Second, we stochastically add a slanted board (Figure 5 right) and place the marker on this board instead of the table surface. This requires the agent to better generalize the wiping task.

Demonstrations: For all tasks we use a single demonstration collected through a spacemouse in simulation and kineshetically in the real world.

Compared Approaches: We compare our approach against multiple different methods on each of the above tasks.

1) **BC with DMPs:** We compare our approach against a behavior-cloning (BC) baseline. Given that we only use one expert trajectory we use DMPs for BC instead of neural networks, since the latter would require a much larger number of expert trajectories for generalization.
2) **BC + RL with DMPs:** In addition to behavior cloned DMPs, we also compare against an approach in which we finetune the learned DMPs with RL.
3) **RL:** We also compare against an RL based approach. For this, we tried using both on-policy *i.e.* Proximal Policy Optimization (PPO) [45] as well as off-policy Soft-Actor Critic (SAC) [46] approaches. However, the PPO based approach did not make any progress within sufficient amount of time, hence we show results only for SAC. Additionally, for SAC we tried using the single expert demonstration by adding it to the replay buffer, however this did not yield any advantage.
4) **TACs:** We evaluate our proposed approach wherein all TAC elements including keypoints, ee-keypoints, axes, controller types are first selected from demonstration using IL and then refined using RL.

A. Metrics and Training

We use success ratio as the metric to quantitatively compare the approaches, since reward as a metric may not

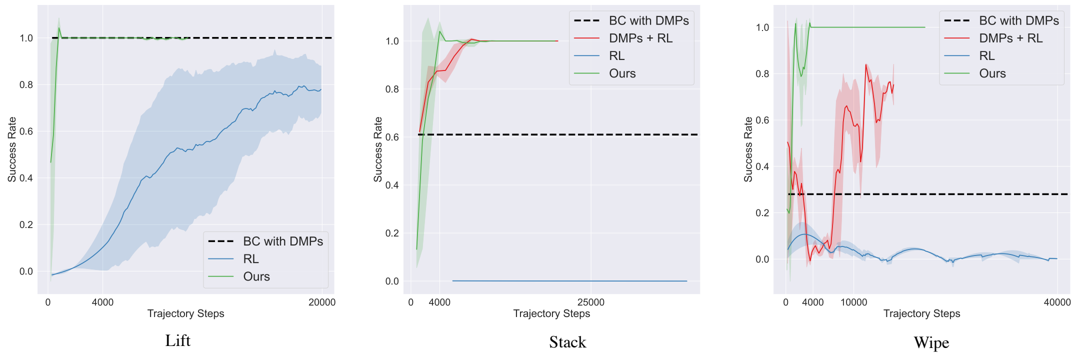

Fig. 6: Success Ratio for the different tasks – Lift, Stack, and Wipe. All tasks in simulation are run for 4 seeds.

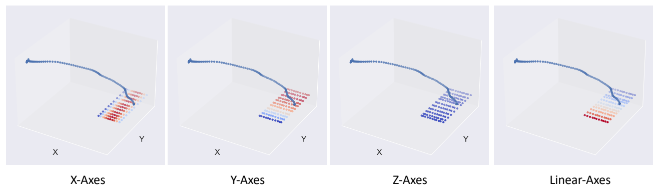

Fig. 7: Heatmap of object-keypoint likelihood for multiple different task-axes u and end-effector gripper center k.

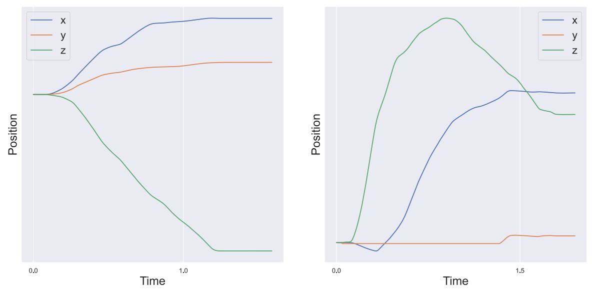

Fig. 8: Segmented trajecories for Stack task before (Left) and after (Right) grasping the smaller cube. Note that both of these trajectories are in the EE-frame of the robot.

be consistent across different baselines due to their different horizons and overall steps. Also importantly, we plot the success-ratio metric against the trajectory steps instead of environment steps. This is because in trajectory based approaches (such as DMPs or ours) a single environment step corresponds to multiple simulator steps. Thus, we also divide the RL steps by the maximum number of trajectory steps. For the RL baseline we use the code provided by the robosuite-benchmark. We use the same hyperparameter settings provided by them and run their code on the slightly modified environments. Hyperparameters and further details are in the appendix on the project website.

B. *Simulation Results - Demonstrations*

Table II lists the set of TACs that are learned from the demonstrations. For all tasks we use a prior which is concentrated on the EE-tip (for Wipe we use the eraser tip) instead of EE-fingers or palm keypoints, hence we get the EE-tip as the the extracted EE-keypoint. For lift and stack we use multiple object keypoints on each object's top surface. We visualize these keypoints for the Lift task through the likelihood values shown in Figure 7. Figure 7 visualizes a heatmap for the likelihood of all different object-keypoints and multiple different axes (Global XYZ and Linear axes). Since for the final controllers multiple task-axes (*e.g.* X, Y, Z-axes or Z and Linear-Axes) need to be combined, we only select (k, e, u) tuples for which the posterior values across all the possible axes is large. Our formulation reduces to task-frame projections when all axes are orthogonal but is also applicable in scenarios when they are not orthogonal.

We now look at how the demonstration trajectories allow us to get the possible set of controllers. For this we use the Stack task and plot its segmented trajectories in Figure 8. These segments are extracted using change point detection and consist of reach and grasp (Left) and then place (Right). For DMPs we get high likelihood (i.e. low fit error) for each axis and hence we add separate DMPs for all axes to the relevant set of controllers. For min-jerk controllers we get high likelihood for all axes except the Z-axes for segment 1. This can be seen from the Z-axes plot in Figure 8 (Right), which cannot be represented by a linear (min-jerk) trajectory that reaches directly to the goal. Although it is indeed possible to break down this Z-trajectory into multiple linear components, such decompositions are beyond the scope of our current work. Finally, for the Stack task there are no significant orientation or force changes in the demonstration, hence, the robot discards the alignment and force controllers.

C. *Simulation Results - Interactions*

Given the reduced set of skill representations we now discuss the results of choosing (and refining) the most appropriate controllers using RL. Figure 6 plots the success ratio for each of the environment across all approaches. This success ratio is obtained using the deterministic policy by running it for 50 episodes intermittently during training. Also, as noted in Section-VI-A the X-axes in Figure 6 refer to the trajectory steps and not the environment steps.

From Figure 6 we can see that for the simplest task, Lift, the DMPs learned from demonstrations directly suffice to complete the task. By comparison, for our approach, since there exists multiple similar TACs initially, a small amount of exploration is required to arrive at the optimal solution. For the Stack task (Figure 6 Middle)), learned DMPs achieve around 60% success rate and do not generalize perfectly to all test scenarios. These scenarios include when the objects are

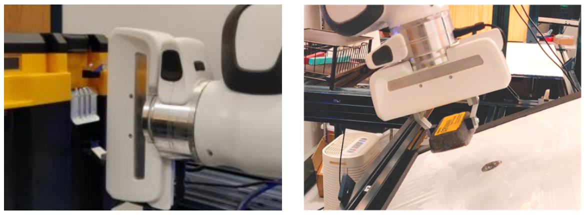
Fig. 9: Real world tasks *Left:* Open-Toolbox, *Right:* Wipe.

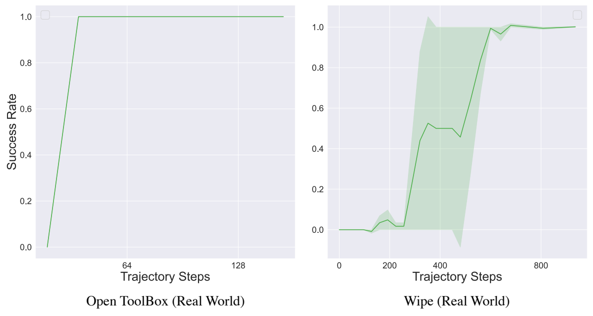

Fig. 10: Success Ratio for real world tasks, run on 2 seeds.

placed in very different configurations than in the provided expert demonstrations and consequently the robot often ends up hitting the other object instead of moving above it. We provide qualitative results of this failure on our website. Alternatively, using these learned DMPs with our TAC skills or performing RL with these DMPs does learn to solve the task quickly and robustly. However, our approach is slightly more sample efficient since the high-level policy chooses to use the linear min-jerk controller for placing instead of a DMP controller for the XY axes. This linear controller is only parameterized by its velocity and hence is faster to optimize than a 10-dimensional DMP. This result signifies the advantage of our approach wherein the higher level policy is able to choose the most appropriate controller to optimize.

Finally, the Wipe task Figure 6 (Right) requires a more fine-grained contact interaction than previous tasks. We find that the learned DMPs perform comparatively worse (around 30%). This is because a sufficient amount of downward force needs to be generated to wipe the marker which is hard to learn from position trajectories alone. Training these DMPs with RL we find that although they can complete the task, the resultant policy can be very unsafe (see video on project website). Hence, we add an additional constraint to force the RL trajectory to have the final EE-location within a safe threshold $(0.12m, 0.04m, 0.04m)$ of the final demonstration location. Even with this change the DMP controllers are still less sample efficient and plateau early in comparison to our approach, which instead uses a force controller to generate the appropriate amount of force to wipe the marker.

D. Real World Results

We also evaluate our approach in the real-world on two different tasks – Open ToolBox and Wipe (Figure 9). We use these task to show that our approach can learn a policy from scratch directly in the real world (given a *single* demonstration).

Setup: Figure 9 shows out task setups. For the Open-ToolBox task we use a common toolbox and learn to open its locks. For Wipe, we use a setup similar to our simulation, wherein a single black mark needs to be erased. We use OpenCV's blob detector to get the marker positions. For both tasks we change the object orientations i.e. the toolbox and whiteboard orientations arbitrarily and use the covariance of point cloud around the keypoint positions to get their surface normal. We do not utilize any other instrumentation for state-estimation. We use the open-source FrankaPy [47] to run our code, and and use the same set of controllers as used in simulation.

We use one demonstration to reduce the initial set of TACs. We use the same set of TAC parameters as shown in Table II except that we do not use DMP controllers during the interaction with the toolbox or whiteboard. We do this purely for safety reasons as is evident from the unsafe nature of updating learned DMPs for Wipe in simulation. During RL training we provide a reward of 1 to the agent for reaching close to the target object, and an additional reward of 4 for completing the task. Additionally, for the open-toolbox task we also provide an additional reward of 1 if the agent correctly aligns with the toolbox. We do this to evaluate how much of an effect dense rewards have on the learning process. In all other scenarios for both tasks we provide 0 reward.

Figure 10 plots the success ratio for both tasks, using a deterministic policy which runs for 5 episodes intermittently during training. As seen above, for both tasks we are quickly able to learn the appropriate controller sequence and parameter values in real world settings. However, the robot learns the Open-ToolBox task much faster than the Wipe task. This is because of two reasons: First, for the former we provide better shaped rewards (additional reward for alignment with tool lock). Second, the low level parameters inferred from the demonstrations for this task (e.g. force parameter) is sufficient for task completion, hence the agent requires less exploration to successfully complete the task.

While we do not observe any intra-seed variance for the Open-ToolBox task there exists some variance for Wipe. This is due to the high level policy (for one seed) not using the appropriate alignment and force-controller for performing the wiping task. Instead, initially, the policy incorrectly aligns with the whiteboard, and once the correct alignment is learned the policy only chooses the force controller after more exploration. However, once learned both of the policies are able to accomplish the task successfully.

VII. Conclusion

In this work, we propose to learn the most appropriate skill representation for a given manipulation task. We use a compositional skill representation, which is grounded in the robot's environment. Given the large number of such environment grounded skills, we use a small set of demonstrations to discard a large number of irrelevant skill representations. The agent then uses the remaining set of representations to explore the environment and learns to select the most appropriate skill representation and corresponding parameters.

References

[1] O. Kroemer, S. Niekum, and G. D. Konidaris, “A review of robot learning for manipulation: Challenges, representations, and algorithms,” *Journal of machine learning research*, vol. 22, no. 30, 2021.

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

[3] M. H. Raibert and J. J. Craig, “Hybrid position/force control of manipulators,” 1981.

[4] W. Gao and R. Tedrake, “kpam 2.0: Feedback control for category-level robotic manipulation,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2962–2969, 2021.

[5] M. Sharma, J. Liang, J. Zhao, A. LaGrassa, and O. Kroemer, “Learning to compose hierarchical object-centric controllers for robotic manipulation,” *arXiv preprint arXiv:2011.04627*, 2020.

[6] 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.

[7] L. Manuelli, W. Gao, P. Florence, and R. Tedrake, “kpam: Keypoint affordances for category-level robotic manipulation,” *arXiv preprint arXiv:1903.06684*, 2019.

[8] P. R. Florence, L. Manuelli, and R. Tedrake, “Dense object nets: Learning dense visual object descriptors by and for robotic manipulation,” *arXiv preprint arXiv:1806.08756*, 2018.

[9] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani *et al.*, “Transporter networks: Rearranging the visual world for robotic manipulation,” *arXiv preprint arXiv:2010.14406*, 2020.

[10] M. Sharma and O. Kroemer, “Generalizing object-centric task-axes controllers using keypoints,” *arXiv preprint arXiv:2103.10524*, 2021.

[11] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín, “robosuite: A modular simulation framework and benchmark for robot learning,” in *arXiv preprint arXiv:2009.12293*, 2020.

[12] J. D. Morrow and P. K. Khosla, “Manipulation task primitives for composing robot skills,” in *Proceedings of International Conference on Robotics and Automation*, vol. 4, 1997, pp. 3354–3359 vol.4.

[13] S. Hutchinson, G. D. Hager, and P. I. Corke, “A tutorial on visual servo control,” *IEEE transactions on robotics and automation*, vol. 12, no. 5, pp. 651–670, 1996.

[14] D. Henrich, T. Ogasawara, and H. Worn, “Manipulating deformable linear objects-contact states and point contacts,” in *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning (ISATP’99)(Cat. No. 99TH8470)*. IEEE, 1999, pp. 198–204.

[15] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, “A survey of robot manipulation in contact,” *arXiv preprint arXiv:2112.01942*, 2021.

[16] L. Johannsmeier, M. Gerchow, and S. Haddadin, “A framework for robot manipulation: Skill formalism, meta learning and adaptive control,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5844–5850.

[17] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, “Learning compositional models of robot skills for task and motion planning,” *The International Journal of Robotics Research*, vol. 40, no. 6-7, pp. 866–894, 2021.

[18] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, “From skills to symbols: Learning symbolic representations for abstract high-level planning,” *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[19] M. Sharma and O. Kroemer, “Relational learning for skill preconditions,” in *Conference on Robot Learning*. PMLR, 2021, pp. 845–861.

[20] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, “Integrated task and motion planning,” *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[21] O. Kroemer and G. S. Sukhatme, “Learning relevant features for manipulation skills using meta-level priors,” *arXiv preprint arXiv:1605.04439*, 2016.

[22] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Using probabilistic movement primitives in robotics,” *Autonomous Robots*, vol. 42, no. 3, pp. 529–551, 2018.

[23] L. P. Kaelbling, “Learning to achieve goals,” in *IJCAI*, vol. 2. Citeseer, 1993, pp. 1094–8.

[24] T. Schaul, D. Horgan, K. Gregor, and D. Silver, “Universal value function approximators,” in *International conference on machine learning*. PMLR, 2015, pp. 1312–1320.

[25] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg, “Learning by playing solving sparse reward tasks from scratch,” in *International conference on machine learning*. PMLR, 2018, pp. 4344–4353.

[26] S. Calinon, D. Bruno, and D. G. Caldwell, “A task-parameterized probabilistic model with minimal intervention control,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3339–3344.

[27] S. Calinon, “A tutorial on task-parameterized movement learning and retrieval,” *Intelligent service robotics*, vol. 9, no. 1, pp. 1–29, 2016.

[28] H. Bruyninckx and J. De Schutter, “Specification of force-controlled actions in the” task frame formalism”-a synthesis,” *IEEE transactions on robotics and automation*, vol. 12, no. 4, pp. 581–589, 1996.

[29] T. Kroger, B. Finkemeyer, and F. M. Wahl, “A task frame formalism for practical implementations,” in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*, vol. 5. IEEE, 2004, pp. 5218–5223.

[30] D. H. Ballard, “Task frames in robot manipulation.” in *AAAI*, vol. 19, 1984, p. 109.

[31] M. T. Mason, “Compliance and force control for computer controlled manipulators,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 6, pp. 418–432, 1981.

[32] J. De Schutter and H. Van Brussel, “Compliant robot motion i. a formalism for specifying compliant motion tasks,” *The International Journal of Robotics Research*, vol. 7, no. 4, pp. 3–17, 1988.

[33] ——, “A methodology for specifying and controlling compliant robot motion,” in *1986 25th IEEE Conference on Decision and Control*. IEEE, 1986, pp. 1871–1876.

[34] T. Kröger, B. Finkemeyer, U. Thomas, and F. M. Wahl, “Compliant motion programming: The task frame formalism revisited,” *Mechatronics & Robotics, Aachen, Germany*, 2004.

[35] L. Peternel, L. Rozo, D. Caldwell, and A. Ajoudani, “A method for derivation of robot task-frame control authority from repeated sensory observations,” *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 719–726, 2017.

[36] J. Kober, M. Gienger, and J. J. Steil, “Learning movement primitives for force interaction tasks,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3192–3199.

[37] L. Armesto, J. Moura, V. Ivan, M. S. Erden, A. Sala, and S. Vijayakumar, “Constraint-aware learning of policies by demonstration,” *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1673–1689, 2018.

[38] T. Schmidt, R. Newcombe, and D. Fox, “Self-supervised visual descriptor learning for dense correspondence,” *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 420–427, 2016.

[39] B. Deng, K. Genova, S. Yazdani, S. Bouaziz, G. Hinton, and A. Tagliasacchi, “Cvxnet: Learnable convex decomposition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 31–44.

[40] Y. Huang, J. Silvério, L. Rozo, and D. G. Caldwell, “Generalized task-parameterized skill learning,” in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 5667–5474.

[41] F. J. Abu-Dakka, B. Nemec, J. A. Jørgensen, T. R. Savarimuthu, N. Krüger, and A. Ude, “Adaptation of manipulation skills in physical contact with the environment to reference force profiles,” *Autonomous Robots*, vol. 39, no. 2, pp. 199–217, 2015.

[42] W. Masson, P. Ranchod, and G. Konidaris, “Reinforcement learning with parameterized actions,” 2015.

[43] M. Hausknecht and P. Stone, “Deep reinforcement learning in parameterized action space,” *arXiv preprint arXiv:1511.04143*, 2015.

[44] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu, “Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space,” *arXiv preprint arXiv:1810.06394*, 2018.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[46] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[47] K. Zhang, M. Sharma, J. Liang, and O. Kroemer, “A modular robotic arm control stack for research: Franka-interface and frankapy,” *arXiv preprint arXiv:2011.02398*, 2020.