Robotics: Science and Systems 2021
Held Virtually, July 12–16, 2021

Hierarchical Neural Dynamic Policies

Shikhar Bahl Abhinav Gupta Deepak Pathak
Carnegie Mellon University

***Abstract*—We tackle the problem of generalization to unseen configurations for dynamic tasks in the real world while learning from high-dimensional image input. The family of nonlinear dynamical system-based methods have successfully demonstrated dynamic robot behaviors but have difficulty in generalizing to unseen configurations as well as learning from image inputs. Recent works approach this issue by using deep network policies and reparameterize actions to embed the structure of dynamical systems but still struggle in domains with diverse configurations of image goals, and hence, find it difficult to generalize. In this paper, we address this dichotomy by leveraging embedding the structure of dynamical systems in a hierarchical deep policy learning framework, called Hierarchical Neural Dynamical Policies (H-NDPs). Instead of fitting deep dynamical systems to diverse data directly, H-NDPs form a curriculum by learning local dynamical system-based policies on small regions in state-space and then distill them into a global dynamical system-based policy that operates only from high-dimensional images. H-NDPs additionally provide smooth trajectories, a strong safety benefit in the real world. We perform extensive experiments on dynamic tasks both in the real world (digit writing, scooping, and pouring) and simulation (catching, throwing, picking). We show that H-NDPs are easily integrated with both imitation as well as reinforcement learning setups and achieve state-of-the-art results. Video results are at https://shikharbahl.github.io/hierarchical-ndps/.**

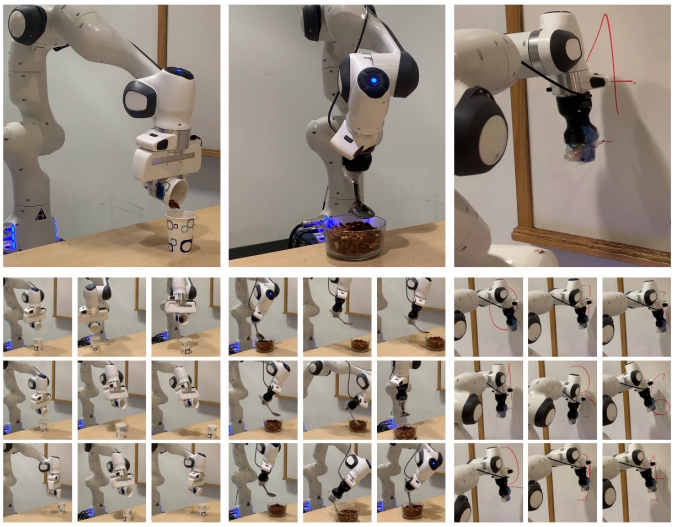


Fig. 1: We present H-NDPs, an efficient real-world robot learning algorithm. Our method is able to perform scooping, writing and pouring from image input only. We are able to generalize across a high amount of diversity, i.e. different object positions, pose, etc. Videos at https://shikharbahl.github.io/hierarchical-ndps/

I. Introduction

Consider the tasks such as pouring liquid or scooping beans as shown in Figure 1. These tasks are dynamic in nature, i.e., they require the robot to continuously apply the right forces and accelerations to act in a reactive manner to changes in the environment. Unlike quasi-static tasks, e.g. pushing or 2D grasping, where it can take arbitrarily long in between each action, the robot needs to reason at the whole trajectory level to execute a swift motion to perform dynamic tasks. For instance, if the robot scoops the beans too slowly they will fall back into the bowl, or if scooped too quickly the beans will be thrown out of the bowl. A common way to address this trajectory-level reasoning is to encode robot movements using nonlinear dynamical systems, like the ones that govern the flow of heat or the movement of planets. This idea is encapsulated by a family of methods known as Dynamic Movement Primitives (DMPs) [33, 12], which can compactly represent basic building block trajectories that are then stitched together to perform complex tasks. DMPs restrict the space of permissible robot movements by constraining the robot's goal and trajectory shape to obey a parametric nonlinear differential equation, consistent with the robot's kinematics and dynamics. This allows DMPs to effectively reason in the space of entire *trajectories*, rather than at the level of individual actions. Consequently, these approaches have led to impressive demos such as pancake flipping [16], dart throwing [15] or playing table tennis [22]. However, this class of approaches suffers from two drawbacks – (a) Generalization: Because of the constraints they impose on trajectories, DMPs do not have the power to represent general movements, and limit the controller to small variations in the initial or goal states. (b) Image-Observations: DMPs have mostly been built on estimating state vectors and struggle with high-dimension input, such as raw images. These shortcomings are in contrast to the flexibility allowed by deep learning methods in terms of generalization to unseen scenarios and scalability to high-dimensional image inputs [18, 28, 20, 13]. However, most of the real robot results using deep learning methods are still limited to pick and place-style quasi-static tasks, as compared to the dynamic tasks achieved by DMP-based methods.

In this paper, our goal is to address the following question: can we build *generalizable* robot policies for dynamic tasks by combining the ability of DMPs to reason in the space of trajectory distributions, with the ability of modern deep robot learning methods to learn from unstructured image data? Recent works [2, 24] address this via Neural Dynamic Policies (NDPs) [2] by embedding structure of a nonlinear dynamical system as a differentiable layer within the policy network and training end-to-end. However, similar to DMPs, these methods still struggle to generalize to unseen state configurations. Why is that? Consider the examples in Figure 1 where we see a large

diversity in the location and pose of objects. Depending on the initial state of the robot and the location of the containers, the joint trajectories will need to be diverse enough in terms of reachability and dynamics, to successfully perform the task of pouring or scooping. Hence, a direct attempt to fit a single policy to all these trajectories poses a big practical challenge in optimization which is further aggravated when the input is a high-dimensional image – which we argue is the case in most interesting real-world robotic problems.

Our solution is to harness the individual strengths of both dynamical systems for movement representations and deep network policies. Instead of directly fitting a single policy on diverse scenarios, we fit dynamical system-based policies first in local regions of the task space and then distill them together into a global one. Our system consists of a library of *local* NDPs [2] and a single *global* NDP. Each local NDP exploits the strengths of DMPs: (a) overfit to operate in small regions of the task space and ensure task success at all times; (b) operate on privileged low-level state information as input. The global NDP is meant to operate on the entire space and only receives raw sensory data as input, e.g., raw images. This global policy is trained to not only maximize the task performance but also to imitate the behavior of the local policies. The key here is that both local and global policies have different objectives: local policies place importance on task success in their local regions, and the global policy places importance on learning from images in a generalizable manner. Owing to this local-to-global structure, we call our framework *Hierarchical Neural Dynamic Policies (H-NDPs)*.

Training H-NDPs for real-world robot learning comes with a practical challenge: there is no guarantee that the local trajectories distilled by the global NDP will indeed be successful when tried even on the same local locations as training unless it is completely overfitting in which case it will not generalize to new locations. Instead of hoping it to just work, we perform multiple iterations of this local-to-global procedure by re-training local NDPs by solving local tasks while being faithful to the global NDPs and then distilling the refined local ones into a new global NDP until convergence, as shown in Figure 2. Such an iterative process is standard practice in general [17, 3, 32] to prevent the distilled network from diverging. However, the added advantage of H-NDPs is that the embedded dynamical system enhances both safety, convergence, and overall performance, as we show in the results section later.

One of the big contributions of this paper is an exhaustive experimental evaluation of H-NDPs and several other baselines on real-world tasks of writing, scooping, and pouring with a robot. Our real-world experiments are conducted in realistic settings with raw high-dimensional images as inputs, with large variations in object positions and goal locations, involving several hundred hours of robot interaction. Finally, we also evaluate complex simulated tasks like throwing, catching, and picking. We show that H-NDPs achieve state-of-the-art performance across all the tasks in reinforcement as well as imitation learning settings.

II. Background

A. Dynamical Systems in Robotics

Dynamical systems have long been used in robotics to represent trajectories and various motion primitives. Such systems operate over an arbitrary robot state (say y, $\dot{y}$ and $\ddot{y}$). Examples of such coordinate systems are joint angles or end-effector positions. Specifically, past work has used a second order dynamical system called Dynamic Movement Primitives (DMPs) [12, 33, 26], derived from Lagrangian Mechanics, to represent robot motions. DMPs are represented by the following:

$$\ddot{y} = \alpha(\beta(g - y) - \dot{y}) + f(x, g), \tag{1}$$

Here, g is a desired goal state and α is a hyperparameter (and $\beta = \frac{\alpha}{4}$, in order for critical damping). The above equation can be broken down into two parts. $\alpha(\beta(g - y) - \dot{y})$ allows for smooth convergence to the goal, making the trajectory physically realizable. $f(x, g)$, a nonlinear forcing function, captures the shape of the trajectory. It is a common practice to use radial basis functions to represent f; the combination of these allows f to represent arbitrary shapes. Traditionally, the weights on these basis functions, w_i, are fit via regression on demonstration trajectories.

$$f(x, g) = \frac{\sum \psi_i w_i}{\sum \psi_i} x(g - y_0), \quad \psi_i = e^{(-h_i(x - c_i)^2)} \tag{2}$$

f decays linearly with x, which is a variable used to replace the time dependency of this ODE. It allows us to arbitrarily stretch or compress time, and sample trajectories of any length from the DMP. x obeys the following first order dynamical system: $\dot{x} = -\alpha_x x$

B. Neural Dynamic Policies

More recently, DMPs have been used in a deep policy learning setup [2, 24]. Neural Dynamic Policies (NDPs) [2] embed the dynamical system described in DMPs inside a deep policy network. Given an input (image or state), s_t, NDPs employ a neural network Φ and output DMP parameters, $w_1, ..., w_n$ (radial basis function weights) and goal state g. These parameters are used by a forward integrator to output a trajectory $\{y_t, \dot{y}_t, \ddot{y}_t\}$ for $t = 1$ to $t = T$. If the robot's action space is not in the same coordinate system as y, then an inverse controller $\Omega(.)$ is used to convert the desired trajectory into a set of actions for the robot to execute in the environment. The forward pass of an NDP involves solving a second order ODE and a pass through the inverse controller. Bahl et al. [2] show that NDPs are fully differentiable and can be efficiently incorporated in RL or Imitation Learning settings, and demonstrate some toy results showing that NDPs can learn from images as well.

III. Method

Consider the real world task of scooping from a bowl. The robot has to both plan a trajectory that will allow it to do the scooping motion properly, and understand any potential

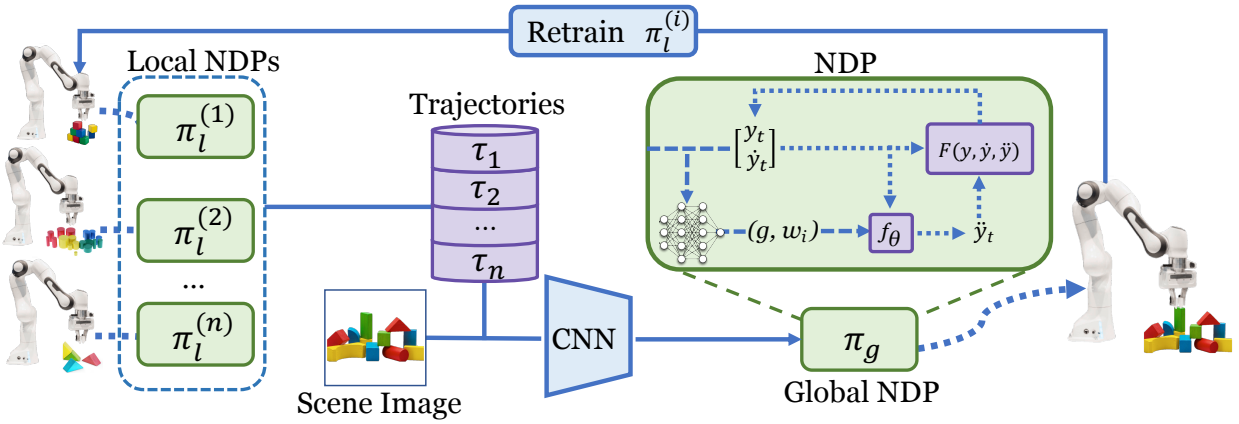


Fig. 2: We train local Neural Dynamic Policies (NDPs) $\pi_l^{(i)}$ on each region i of the task space, from state observations. A global NDP π_g (usually taking in image input I_t) learns to imitate the local experts. We use the global NDP to retrain local NDPs which keeps the NDPs from diverging. These local-to-gloabl interactions happen in an iterative manner. NDPs make a good candidate for capturing such local-to-global interactions due to their shared structure and the fact that they operate over a smooth trajectory space.

randomness, for instance if the bowl changes locations. A single policy likely will have a lot of trouble with this. We address such challenges by presenting Hierarchical Neural Dynamic Policies (H-NDPs). H-NDPs use a local-to-global learning scheme which makes it much easier for the agent to learn how to handle diversity in the task and deal with raw image inputs. We leverage structure provided by NDPs [2] for policy learning, allowing our hierarchical policies to operate in a shared space, and thus leads to smoother trajectories and more sample efficient learning. In this section, we describe how this setup works, both in the reinforcement and imitation learning settings.

A. *Hierarchical Neural Dynamical Policies*

We break down policy learning for a given task into two components: local controllers which operate from exact state observations, and a global policy which learns from raw sensory observations, for example robot poses and images. Both policies are NDP; the policy networks have an embedded dynamical system as a layer. Directly optimizing the global policy for the full task can be difficult, since dynamical systems can easily overfit to a single trajectory. Let π be an NDP and let $\phi(s;\theta)$ be the deep network inside the NDP, parameterized by weights θ. $\phi(s;\theta)$ outputs the DMP parameters which are used by the forward integrator F to solve the differential equation described in Equation 1. τ, the output of $\pi(.|s)$, is $F(\phi(s;\theta))$.

We divide the task space into M different regions. We train local NDPs $\pi_l^{(i)}$ on each region i (R_i) of the task space. For example, for a task like scooping, each bowl location would be its own region, and we would train a single NDP to solve the task for that specific bowl location. For the rest of this section, let the local policy $\pi_l^{(i)}$ be parameterized by network weights by $\theta_l^{(i)}$. For task i, we will compute loss on the NDP output, $\mathcal{L}_i(\theta_l^{(i)})$ and optimize with respect to $\theta_l^{(i)}$. This loss can be any differentiable loss based on the policy output. In the next two sections, we will describe what $\mathcal{L}_i$ is in the case of RL and imitiation learning.

Once the local policies are trained, the global NDP π_g parameterized by network weights θ_g, learns to imitate the local policies. This makes it easier for the global policy to understand the difference between high level task details and low level task optimization. The global NDP conditioned on the current observation, s_t, learns to clone the behavior of the local NDPs in using the loss: $\mathcal{L}_{\text{BC}} = \sum_i ||F(\phi(s_t;\theta_g)) - F(\phi(s_t;\theta_l^{(i)}))||_2$. There is no guarantee, however, that a single iteration of behavior cloning will work. In practice, an iterative process is standard. Therefore, we fine-tune π_g on the loss function for the full task (union of all task regions R_i), $\mathcal{L}_g(\theta_g)$. In summary, the overall global NDP training loss is:

$$\mathcal{L}_{\text{global}} = \mathcal{L}_{\text{BC}} + \mathcal{L}_g(\theta_g) \tag{3}$$

We would like to minimize the amount of human supervision, thus we do not want to create more task spaces but need more data to train the policy. One possibility is to retrain the local NDPs and collect more data, However this could easily lead to divergence. Thus we use the NDP π_g to reduce divergence by adding $\alpha_i D_{KL}(\pi_l^{(i)}||\pi_g)$ to the local NDP task loss $\mathcal{L}_i(\theta_l^{(i)})$. α_i is a hyperparameter for the weight of this extra loss term. We collect more data from the local experts and repeat the above steps until convergence. We call this process *iterative refinement*. This structure allows the local experts to adjust their outputs based on what is easier for the global policy to learn. NDPs make a good candidate for such a learning scheme due to their shared structure and the fact that they both operate over a smooth trajectory space, leading to much more efficient learning. Hence, the overall loss for the ith local NDPs is:

$$\mathcal{L}_{\text{local}} = \mathcal{L}_i(\theta_l^{(i)}) + \alpha_i D_{KL}(\pi_l^{(i)}||\pi_g) \tag{4}$$

We provide a detailed description of H-NDPs in Algorithm 1.

The general idea of local-to-global learning has been widely studied in machine learning for generalization in complex domains [37, 3, 32]. For robot learning in particular, local-to-global structure has been exploited by for imitation learning by Levine and Koltun [17] and for RL by Ghosh et al. [10], Teh

et al. [38]. In contrast to the black-box policy networks used in these works, our main contribution is to embed the structure of a nonlinear dynamical system within the network. NDPs make the interactions between global and local policies a lot more efficient, since both operate in the same DMP parameter space. This allows generalization to new configurations for dynamic tasks, a strong advantage of our method. We now discuss how to apply H-NDPs to both imitation and RL settings in the following subsections.

B. H-NDPs for Imitation Learning

In the imitation learning setting, we train the global NDP (π_g) via visual inputs and the local NDPs ($\pi_l^{(i)}$) are trained via supervised learning to imitate kinesthetic demonstrations. We start with a single demonstration for each R_i. Let this demonstration be $\tau_{\text{demo}}^{(i)}$. Therefore the local NDP loss $\mathcal{L}_i$ (described in Equation 4) for the IL case:

$$\mathcal{L}_i(\theta_l^{(i)}) = ||F(\phi(s_t;\theta_l^{(i)})) - \tau_{\text{demo}}^{(i)}||^2 \quad (5)$$

For simplicity, both local and global NDPs are set to be Guassian with a fixed variance. The KL-divergence in the extra loss term to the local NDP loss (described in Equation 4) therefore simply becomes:

$$D_{KL}(\pi_l^{(i)}||\pi_g) = \alpha_i||F(\phi(s_t^{(i)};\theta_l^{(i)})) - F(\phi(s_t^{(i)};\theta_g))||_2 \quad (6)$$

Here, let Where $s_t^{(i)}$ be the observation received by the agent while performing task i. Naively using a constant α might make the local NDPs worse. For instance, at the beginning of training, the global NDP may not be successful for every task region. Instead, we deploy the trained global policy to collect $F(\phi(s_t^{(i)};\theta_g))$, a trajectory for every local region i. We set $\alpha_i = 1$ only if $F(\phi(s_t^{(i)};\theta_g))$ is judged successful by a human. Otherwise, we set it to 0. Finally, $\mathcal{L}_g(\theta_g)$, the loss function for the global NDP is simply the imitation learning loss on the original demonstrations:

$$\mathcal{L}_g(\theta_g) = \sum_i ||F(\phi(s_t^{(i)};\theta_g)) - \tau_{\text{demo}}^{(i)}||^2 \quad (7)$$

C. H-NDPs for Reinforcement Learning

In the RL framework, the objective is to learn a policy $\pi(a_t|s_t)$ that maximizes the sum of expected rewards $R_t = \mathbb{E}[\sum_{i=t}^{T} \gamma^i R(s_i, a_i, s_{i+1})]$. It can be difficult for RL policies to work in highly dynamic environments, especially when there is a high amount of stochasticity or variation in the task.

In the RL setting, similarly to the imitation learning setting, we split the task into i regions. Each local NDP is an RL policy and we use $\mathcal{L}_i = J_i$, where J_i is the surrogate policy gradient loss from an off-the-shelf policy optimization algorithm. Specifically, we use the loss from Proximal Policy Optimization (PPO [34]), and update the parameters of $\pi_i^{(l)}$ with respect to ∇J_i. Similarly, π_g is optimized via PPO on the loss $\mathcal{L}_g = J_g$. To avoid divergence of the global NDP, we compute the KL divergence from the global NDP to each local NDP, as in Equation 4, and add to the local NDP training loss, $\mathcal{L}_i$. Since we use Gaussian policies with learned variance, then we have that: $D_{KL}(\pi_l^{(i)}||\pi_g) = \log\frac{\sigma_g}{\sigma_i} + \frac{\sigma_i^2 + (\mu_i - \mu_g)^2}{2\sigma_g^2} - \frac{1}{2}$. Here the output of $\pi_i^{(l)}$ is $\mathcal{N}(\mu_i, \sigma_i^2)$ and that of π_g is $\mathcal{N}(\mu_g, \sigma_g^2)$. In practice, we found that setting α_i to either 0 or a very low value worked much better. An explanation for this phenomenon is that NDPs already contain more structure via the embedded dynamical system and hence do need KL-divergence constraint. The DMP parameter space these policies are in is already a lot more meaningful than the general neural network space.

Algorithm 1 Training H-NDPs

Require: NDP Policy randomly initialized global policy π_g with weights θ_g, M local regions R_i, for each region i a local NDP $\pi_l^{(i)}$ and corresponding NN weights $\theta_l^{(i)}$, initialize empty $\mathcal{D}$

for $1, 2, \ldots$ iterations **do**
  for $i = 1 \ldots m$ **do**
    Run policy $\pi_l^{(i)}$ on environment R_i for H steps
    Collect trajectory $F(\phi(.;\theta_l^{(i)}))$ and store into $\mathcal{D}$
    Compute $\mathcal{L}_{\text{local}} = \mathcal{L}_i(\theta_l^{(i)}) + \alpha_i D_{KL}(\pi_l^{(i)}||\pi_g)$
    $\theta_l^{(i)} \leftarrow \theta_l^{(i)} - \eta\nabla_{\theta_l^{(i)}}\mathcal{L}_{\text{local}}$ (until convergence)
  end for
  Compute $\mathcal{L}_{\text{BC}} = \sum_{i=1}^{i=M} ||F(\phi(s_t;\theta_g)) - F(\phi(s_t;\theta_l^{(i)}))||_2$
  Compute loss $\mathcal{L}_{\text{global}} = \mathcal{L}_{\text{BC}} + \mathcal{L}_g(\theta_g)$
  $\theta_g \leftarrow \theta_g - \eta\nabla_{\theta_g}\mathcal{L}_{\text{global}}$ (until convergence)
end for

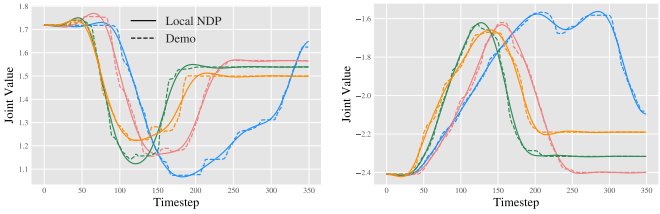


Fig. 3: Visualizations of the original demonstrations and the trained local NDP on selected joints, for the real-world scooping task. The x-axis is the timestep and y is the joint value. Each curve representations a different demonstration. We can see that local NDPs can efficiently capture the desired motion in a smooth manner.

D. Advantages of H-NDPs for Real-World Robotics

Due to their search over a physically smooth space, H-NDPs provide safer and more efficient learning. This can be a benefit in the real world, where hardware setups can be brittle and exploration can be dangerous. In fact, in Figure 3 we show the trajectory for multiple joints of sampled demonstrations and the output of the corresponding fitted local NDP, which learns a much smoother version of the demonstration.

Secondly, since H-NDPs operate at a trajectory level, the policy π is only used every k steps. Hence fewer forward passes need to be taken by the policy network. With large networks and computationally expensive hardware (such as robot controllers and cameras), more forward passes can actually be an impediment to the learning algorithm, as it cannot execute the task at a high enough frequency. Additionally, we can also sample trajectories at arbitrary lengths, and can therefore output a more compact trajectory if needed.

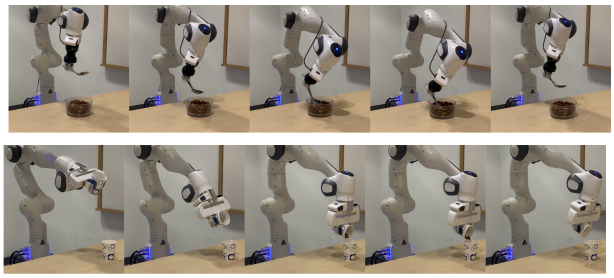


Fig. 4: Sample trajectories for Scooping (top) and Pouring (bottom) tasks, on the Franka Panda robot

IV. Experimental and Implementation Setup

A. Real Robot Tasks Setup

For all the real world tasks (scooping, pouring and writing), we use visual inputs for the global NDP and state inputs for the local NDP. We use a Franka Panda 7 DoF robot, controlled by joint angle control. We use the robot control code from Zhang et al. [43]. We run the controller at about 50 Hz for both local and global policies. We are in fact bottlenecked by the frequency of the image capture and processing software. Note that a neural network policy which needs image inputs every timestep would be significantly slower (5-10Hz). An H-NDP predicts one trajectory of k steps, thus needs only one forward pass per k steps allowing for a 50Hz controller. In all of our experiments, we use $k = 350$. To mimic real world conditions, we vary goal locations and intentionally change the scene a little bit (slightly shift the robot, camera or the object in the robots hand). For ease of use, we utilize the same initial robot joint positions. In each of the real world tasks, a human decides whether a trial is successful or failed. More details of each task can be found in the supplementary material.

B. Simulated Tasks Setup

We also perform simulation experiments on dynamic tasks, inspired from tasks performed by Ghosh et al. [10]. The simulated robot is a 6 DoF Kinova Jaco, which we control in joint angle space. All tasks are simulated in the MuJoCo [39] framework. Throwing involves first grabbing then throwing a cube inside a target box. To add diversity to the task, we vary the location of the goal box. These constitute different regions of the task space. For picking, the goal is to grasp a cube and lift it as highly possible. Here the diversity comes from varying the starting position of the block. Finally, we perform the task of catching a ball being launched in the air. The goal is for the robot to catch the ball and keep it in its hand till the end of the episode. Here, we randomize the starting location of the ball. Images of these tasks can be seen in Figure 5.

C. Policy Architecture and Network Pretraining

In the RL setting, we use the same architecture as Bahl et al. [2] (2 hidden layers of 100 neurons). In the imitation learning setting, we use a very small fully connected neural network (one hidden layer with 40 neurons) for our local policies, and a similar Convolutional Neural Network (CNN) architecture to that of GPS [18]. We also use a spatial softmax

(a) Throwing (b) Picking (c) Catching

Fig. 5: Reinforcement Learning environments in MuJoCo [39].

layer [18], which for each channel c $f_{cx} = \sum_{ij} s_{cij} x_{ij}$ and $f_{cy} = \sum_{ij} s_{cij} y_{ij}$, where i, j are pixel coordinates, and s_{cij} is the spatial softmax function for pixel a_{cij}. We then concatenate robot joint poses to this network, and pass them through two fully connected layers and output desired joint angles.

In order to provide the global policy with basic visual features, we pretrain the network to predict object pose data (this form of pretraining is common in robotics, e.g. in Levine et al. [18]). For our scooping and pouring task, we provide a similar form of pretraining, although with approximate poses. We do not use any AR markers for estimation. Instead we sample and move the robot to a position, place the object there and capture a training image. This naturally leads to imprecision in the training data, but is more realistic. For the digit writing task, we pretrain on an MNIST-like classification task, where we use a few digits written on a board by a human.

V. Results: H-NDPs for Imitation Learning

We evaluate H-NDPs on three real world tasks for imitation learning from images: Digit Writing, Scooping and Pouring. Videos at https://shikharbahl.github.io/hierarchical-ndps/ and in supplementary. One of the main focus of this work is thorough scientific evaluation in the real world itself. This experimentation involved hundreds of hours of interaction that took several weeks on hardware to complete the real-world evaluation shown in Table I and Figure 6. We clearly separate training and testing scenarios for each of the tasks and describe them in the following subsections as well as in the supplementary.

The goal of this empirical study is to answer following scientific questions across all the tasks:

- How much does the structure of dynamical systems contribute to the performance of H-NDPs?
- How much does iterative refinement of global policy contribute to the performance of H-NDPs?
- How much does the local-to-global structure contribute to the performance of H-NDPs?

We attempt to answer these questions by running baseline methods. Firstly, to understand the importance of dynamical systems in H-NDPs, we run comparisons against a method that uses iterative refinement as well as well as a local-to-global structure, but with fully connected neural network layers instead of embedded dynamical systems. This method is our implementation of GPS [18]. For every other baseline, we also design a version of that baseline with only fully connected layers, calling it vanilla NN. Secondly, to address the question of the effect of iterative refinement, we train H-NDPs and

GPS for only one iteration. To have a fair comparison, we train these baselines with 5x more demonstrations so as to provide effectively the same amount of data. However, note that these baselines have same number of interactions *but have 5x more supervision* because H-NDPs does not need more expert demonstrations after the first iteration. Finally, we test the importance of the local-to-global structure by introducing baselines that do not use it. NDP is the global policy that just learns from demonstrations. which is very similar to the method from Bahl et al. [2]. Vanilla NN is the fully connected counterpart of NDP.

A. Task 1: Digit Writing on the Whiteboard from Image Input

The goal in this task is for the robot to draw a digit on a whiteboard, given an image of the digit (ranging from 0 to 9). A dry-erase marker is attached to the robot hand. We collect 10 kinesthetic demonstrations (one for each digit 0-9) for training. We keep 10 digit images held-out which are not shown to the robot during training on which we compute the "test" success. We pretrain the global policy network using the procedure discussed in Section IV-C.

Role of Dynamical System Structure: In Table I, we can see that H-NDPs achieve the highest test success rate, 80%. Our approach outperforms the GPS baseline by a large margin. We show a sample of the test results in Figure 7. The picture on the left is what the robot sees at test time, and the on the right is the final output. Compared to all the other methods, H-NDPs have the smoothest and most accurate result. Figure 7c shows a much smoother output by our method compared to that of GPS (Figure 7e). The major difference between the implementation of the two approaches is that GPS uses fully connected layers instead of dynamical-system based layers. Interestingly, when comparing all other baselines (no local-to-global structure, no iterative refinement, etc) the dynamical system based methods (in Table I these are H-NDPs and NDP) all outperform their fully connected counterparts. This clearly indicates that the role of dynamical system-based structure is crucial for writing.

Role of Iterative Refinement: We can see that in Table I that performance of H-NDPs drops without iterative refinement (with 5x more supervision performance still drops to about 50%). From Figure 6b, it is clear that our method benefits from iterative refinement, as the test and train success rates increase. Interestingly, at test time our method was able to capture the "4" digit better than at training time. Despite a drop in performance, H-NDPs without iterative refinement still outperforms almost all other baselines.

Role of Local-to-Global: H-NDPs clearly outperform methods that do not employ a local-to-global structure. Both the NDP and Vanilla NN baselines perform significantly worse. This is true for methods that use 1x the demos as H-NDPs as well 5x. In fact, methods that use 1x the demos tended to fit to one demonstration and ignore the rest; i.e. Vanilla NN only output 8's for all ten digits.

	#Demos	#Iter	Writing	Scooping	Pouring
No local-to-global structure:					
NDP [2]	1x	1	0.2	0.2	0.0
Vanilla NN	1x	1	0.1	0.0	0.0
No local-to-global structure with 5x Demos:					
NDP [2]	5x	1	0.5	0.3	0.0
Vanilla NN	5x	1	0.1	0.0	0.0
Local-to-global but no iterative refinement:					
GPS [18]	5x	1	0.1	0.0	0.0
H-NDPs (ours)	5x	1	0.4	0.3	0.0
Both local-to-global and iterative refinement:					
GPS [18]	1x	5	0.3	0.0	0.2
H-NDPs (ours)	1x	5	**0.8**	**0.6**	**0.3**

TABLE I: Final results on the three real world tasks. We average the test success rate normalized to $[0-1]$ over 10 trials on held-out testing images/locations. We compare against vanilla NDP [2], vanilla NN imitation, and we replace NDPs in our method with vanilla neural networks (a similar method to GPS [18]). We can see that our method outperforms all the baselines substantially.

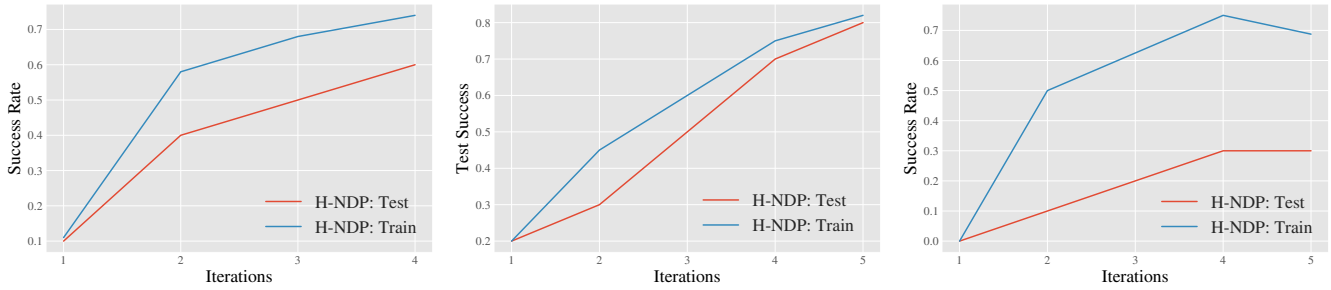
B. Task 2: Scooping from Image Input

In the scooping task, the robot has a spoon attached to its effector and its goal is to scoop almonds from a bowl. We vary the bowl locations, and the robot must infer from only from raw images where it should scoop. We have 18 distinct locations on the table for training and 10 distinct locations kept held-out for testing, shown in Figure 1 in the supplementary. We collect kinesthetic demonstrations on training locations. We provide the pretraining discussed in Section IV-C. Figure 4 shows a picture of this setup.

Role of Dynamical System Structure: Similarly to the writing task, it is clear from Table I that H-NDPs drastically outperforms all baselines without dynamical system-based structure (GPS, vanilla NN, etc). In fact, all such baselines ended up executing the mean trajectory. For example, wherever the bowl would be placed, the networks would output the same trajectory towards the center of table. This clearly shows that trajectory level prediction is hard for traditional networks, and that dynamical systems are important for this task.

Role of Iterative Refinement: In Figure 6a, we see that both training and test success rates for H-NDPs go up as we perform more iterations of refinement. For H-NDPs, iterative refinement doubles the performance. However, even without iterative refinement, H-NDPs still obtains a 30% success rate, the highest of the baselines.

Role of Local-to-Global: We can see that H-NDPs obtains a higher success rate at test time than any of the baselines that do not use the local-to-global structure (1x and 5x demos both). The performance gain from 5x to 1x demonstrations is not very high. This indicates just adding more data is not as important as the global-to-local framework. While H-NDP's success rate is 60%, even in case of failure it would go close to the bowl but not actually scoop any almonds out.



(a) Scooping (b) Digit Writing (c) Pouring

Fig. 6: Success rate for the three real-world tasks across iterations. Note that more iterations of the H-NDP method in fact does help in learning, both in the train and test (held-out/unseen) scenarios.

(a) Writing Task Setup (b) Input (c) Ours (d) Local NDP (e) GPS

Fig. 7: Images showing the writing task setup. (a) shows the robot setup we used, a 7 DoF robot with a marker placed inside its end-effector, controlled via joint angles. (b) shows the input image (unseen at training time). (c) shows the output of our method, (d) shows the output of the local controller and (e) shows the final output of GPS [18]. We can see that our method produces a smooth and correct-looking 4.

C. Task 3: Pouring from Image Input

We perform experiments on a real world visual pouring task. The robot starts with a cup of almonds in its gripper, and must pour the almonds into a target cup, without any falling out. Just like scooping, the global policy must act from camera input only, and the target cup moves around to different locations. We collect kinesthetic demonstrations for 16 training locations and keep 10 held-out locations for testing as in the case of scooping, shown in the supplementary (Figure 1). Pretraining is discussed in Section IV-C and the task setup in Figure 4.

Role of Dynamical System Structure: This task is inherently more difficult than the others, possibly due to the size of the cups and a lot more accuracy is needed for a successful pour. H-NDPs, however, still outperforms every other baseline, including GPS (30% vs 20%). During testing we observed that in the failed trials robot would go close to the cup but miss it marginally. On the other hand, GPS completely missed the target most of the time. The other vanilla NN baselines which do not perform iterative refinement would actually produce infeasible and dangerous behavior. This shows that dynamical system is important for the pouring task.

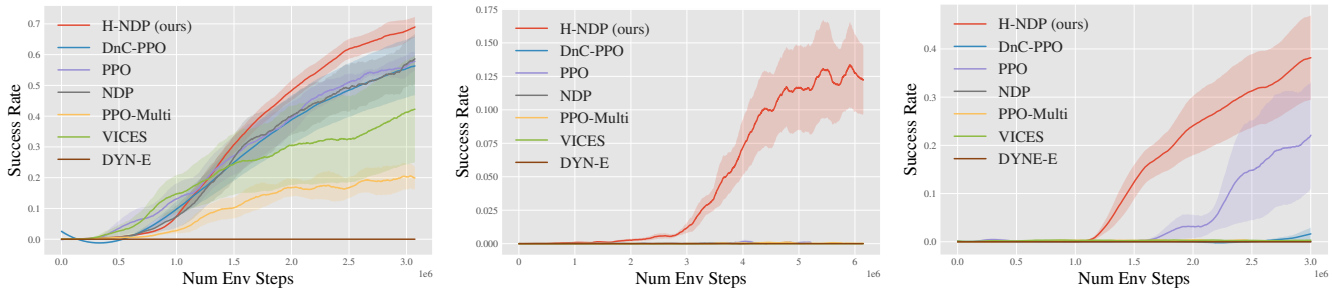
Role of Iterative Refinement: In Table I, we can see that all of the baselines without iterative refinement have 0 success. On the other hand, Figure 6c shows that H-NDPs also starts with a 0% success rate, but improves via iterative refinement. Additionally, most of the baselines produced the same trajectory for every input. This shows that iterative refinement is very helpful, especially in more challenging tasks.

Role of Local-to-Global: All the methods that do not use the local-to-global framework have a success rate of 0%. Both methods that do use the local-to-global structure (as well as iterative refinement), H-NDPs and GPS, are the only ones that achieve any success. Therefore, local-to-global structure is in fact important for the pouring task.

VI. Results: H-NDPs for Reinforcement Learning

In the RL setting, we compare our method against several competing methods. We firstly test a similar method to Ghosh et al. [10] which we call PPO-DnC. This is very similar to the original method, however it uses PPO as a base algorithm, so that it can be compared apples-to-apples with H-NDP. In another baseline, we run the NDP algorithm [2], equivalent to training the global NDP only. Additionally, we run our base RL algorithm, vanilla PPO [34] as a comparison as well. To have a fair comparison, we also consider the baselines used by Bahl et al. [2], in fact using their provided code. We compare against the multi-action PPO from Bahl et al. [2], Variable Impedance Control in End-Effector Space (VICES) [21] and Dynamics-Aware Embeddings (DYN-E) [42]. The latter two methods provide alternative parameterizations for action space: in VICES [21] the policy directly outputs parameters for a PID controller, and in DYN-E [42] an action-based encoder is learnt from environment interaction.

We can see in the RL results in Figure 8. We present the result of 3 random seeds run on the same codebase. We plot the success rate versus the number of environment steps taken. H-NDP, our method, outperforms all the baselines discussed above either in terms of sample efficiency or absolute performance. This difference is especially stark for the catching task (Figure 8b), since the randomness is the starting position



(a) Throwing (b) Catching (c) Picking

Fig. 8: Success rate for the three simulated RL tasks: throwing, catching and picking. Note that all these tasks are stochastic. Our method (red) outperforms all the baselines.

of the ball, and even a small perturbation can have a large effect on the trajectory. H-NDPs are able to capture this high level variation quite well, while the baselines cannot. In the other two tasks, shown in Figure 8c and Figure 8a, H-NDPs are still more sample efficient and have a better final performance, even though the baselines get a relatively higher performance compared to that of catching. This is likely due to the fact that randomness in throwing and picking isn't as drastic as catching. However, overall, we can clearly see that H-NDPs provide a strong performance boost, likely due to the embedded dynamical system which allows for smooth trajectories and efficient distillation of knowledge.

VII. Related Work

Robot Learning for Dynamic Tasks Robot learning methods have been successful for many real-world tasks. However, these tasks have been performed in a controlled, and quasi-static setting where the robot can take arbitrarily long gap between subsequent actions [19, 1, 13, 28]. The real world, however, is a lot more dynamic. When humans perform daily actions, like cutting vegetables, they think at a trajectory level and not at a discrete action level. To this end, seminal work in robotics has proposed using dynamical systems to model actions and trajectories, in a continuous space. Dynamic Movement Primitives (DMPs) [12, 33, 29] have been widely used to perform diverse, dynamic tasks such as table tennis [22], panckake flipping [16] or tether-ball [25]. They are able to model smooth, natural motions, and have in fact been used to inspire many policy learning schemes [8, 5, 4, 40, 11, 7]. More recent work [2, 24, 30, 6] has shown DMPs can be incorporated in a differentiable, end-to-end deep learning setting, which is an attribute that H-NDPs leverage.

Hierarchical Frameworks for robot learning While DMPs have been used in previous works for building hierarchical policies [9, 36, 14, 27], these have mostly been constrained to discrete primitives [9, 27] and relatively simple settings from a perception standpoint. Previous works have also attempted to share knowledge between DMPs; for example Rückert and d'Avella [31] leverages shared basis functions for controlling multidimensional systems. To our knowledge, our work is the first to use a hierarchical local-to-global structure using DMPs.

Hierarchical frameworks are popular for deep imitation learning setups. One prominent example is Guided Policy Search (GPS [17, 18]) which uses a bottom-up approach by learning "expert" local controllers from state observations and then distill them into a image-based policy. While H-NDPs uses a similar bottom-up framework, we employ the structure of dynamical systems within our policy architecture, allowing us to perform more dynamic tasks than GPS. Furthermore, our local controllers are learnt from a few $(10-15)$ demonstrations, which are a lot easier to obtain than assuming fully observable environment and hand-engineering required for GPS.

Hierarchical learning has also long been explored in the context of RL from both top-down [41] as well as bottom-up [35, 23] perspective. Ghosh et al. [10] and Teh et al. [38] propose a hierarchical local-to-global framework to perform more complex, diverse tasks. Ghosh et al. [10] takes advantage of local RL policies trained via policy gradients and a global policy which imitates the former. The local-to-global interactions between neural networks can lead to suboptimal behavior, especially for more difficult dynamic tasks. On the other hand, the local-to-global interactions taking place within H-NDPs are in a much more structured space (the space of physically plausible trajectories), leading to a stronger performance by H-NDPs.

VIII. Conclusion

The main contributions of this paper are:

- We propose Hierarchical Neural Dynamic Policies (H-NDPs) that embed the structure of dynamical systems in a hierarchical framework for end-to-end policy learning. H-NDPs facilitate reasoning at the trajectory level while learning from high-dimensional image inputs.
- We show that H-NDPs are easily integrated with standard imitation as well as reinforcement learning and achieve state-of-the-art performance across dynamic tasks in both the real world and simulation.
- We perform thorough scientific evaluation of held-out generalization on real-world tasks involving several hundred hours of robot interaction.

In this paper, we consider generalization only across different configurations of same objects and leave the generalization across different objects types for future work.

Acknowledgments

We thank Vikash Kumar for fruitful discussions and grateful to Aravind Sivakumar, Russell Mendonca, Sudeep Dasari for comments on early drafts of this paper. The work was supported by NSF IIS-2024594 and AG was supported by ONR YIP.

References

[1] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *NIPS*, 2016. 8

[2] Shikhar Bahl, Mustafa Mukadam, Abhinav Gupta, and Deepak Pathak. Neural dynamic policies for end-to-end sensorimotor learning. In *NeurIPS*, 2020. 1, 2, 3, 5, 6, 7, 8

[3] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *SIGKDD*, 2006. 2, 3

[4] Sylvain Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 2016. 8

[5] Sylvain Calinon, Irene Sardellitti, and Darwin G. Caldwell. Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies. *IROS*, 2010. 8

[6] Nutan Chen, Maximilian Karl, and Patrick Van Der Smagt. Dynamic movement primitives in latent space of time-dependent variational autoencoders. In *International Conference on Humanoid Robots (Humanoids)*, 2016. 8

[7] Ching-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan Ratliff. Rmpflow: A computational graph for automatic motion policy generation. *Algorithmic Foundations of Robotics XIII*, 2020. 8

[8] Adam Conkey and Tucker Hermans. Active learning of probabilistic movement primitives. *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, 2019. 8

[9] Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 2016. 8

[10] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017. 3, 5, 7, 8

[11] Yanlong Huang, Leonel Rozo, João Silvério, and Darwin G Caldwell. Kernelized movement primitives. *The International Journal of Robotics Research*, 2019. 8

[12] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 2013. 1, 2, 8

[13] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018. 1, 8

[14] Jens Kober and Jan Peters. Learning motor primitives for robotics. In *ICRA*, 2009. 8

[15] Jens Kober, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust robot movements to new situations. *RSS*, 2011. 1

[16] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *IROS*, 2010. 1, 8

[17] Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML*, 2013. 2, 3, 8

[18] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 2016. 1, 5, 6, 7, 8

[19] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with large-scale data collection. In *ISER*, 2016. 8

[20] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017. 1

[21] Roberto Martin-Martin, Michelle A. Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. *IROS*, 2019. 7

[22] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 2013. 1, 8

[23] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018. 8

[24] Rok Pahic, Andrej Gams, Aleš Ude, and Jun Morimoto. Deep encoder-decoder networks for mapping raw images to dynamic movement primitives. *ICRA*, 2018. 1, 2, 8

[25] Simone Parisi, Hany Abdulsamad, Alexandros Paraschos, Christian Daniel, and Jan Peters. Reinforcement learning vs human programming in tetherball robot games. In *IROS*, 2015. 8

[26] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *ICRA*, 2009. 2

[27] Peter Pastor, Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, and Stefan Schaal. Skill learning and task outcome prediction for manipulation. In *ICRA*, 2011. 8

[28] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *ICRA*, 2016. 1, 8

[29] M. Prada, A. Remazeilles, A. Koene, and S. Endo. Dynamic movement primitives for human-robot interaction: Comparison with human behavioral observation. In *International Conference on Intelligent Robots and Systems*, 2013. 8

[30] Nathan D Ratliff, Jan Issac, Daniel Kappler, Stan Birchfield, and Dieter Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018. 8

[31] Elmar Rückert and Andrea d'Avella. Learned parametrized dynamic movement primitives with shared synergies for controlling robotic and musculoskeletal systems. *Frontiers in computational neuroscience*, 2013. 8

[32] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015. 2, 3

[33] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*. Springer, 2006. 1, 2, 8

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017. 4, 7

[35] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, 2002. 8

[36] F. Stulp, E. A. Theodorou, and S. Schaal. Reinforcement learning with sequences of motion primitives for robust manipulation. *Transactions on Robotics*, 2012. 8

[37] Ron Sun, Edward Merrill, and Todd Peterson. From implicit skills to explicit knowledge: A bottom-up model of skill learning. *Cognitive science*, 2001. 3

[38] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017. 4, 8

[39] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *The IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012. 5

[40] Aleš Ude, Andrej Gams, Tamim Asfour, and Jun Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *Transactions on Robotics*, 2010. 8

[41] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *ICML*, 2017. 8

[42] William Whitney, Rajat Agarwal, Kyunghyun Cho, and Abhinav Gupta. Dynamics-aware embeddings. *arXiv preprint arXiv:1908.09357*, 2019. 7

[43] Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-interface and frankapy. *arXiv preprint arXiv:2011.02398*, 2020. 5