DDCO: Discovery of Deep Continuous Options for Robot Learning from Demonstrations

Sanjay Krishnan*
EECS Department, UC Berkeley
sanjaykrishnan@berkeley.edu

Roy Fox*
EECS Department, UC Berkeley
royf@berkeley.edu

Ion Stoica
EECS Department, UC Berkeley
istoica@berkeley.edu

Ken Goldberg
EECS and IEOR Departments, UC Berkeley
goldberg@berkeley.edu

Abstract: An option is a short-term skill consisting of a control policy for a specified region of the state space, and a termination condition recognizing leaving that region. In prior work, we proposed an algorithm called Deep Discovery of Options (DDO) to discover options to accelerate reinforcement learning in Atari games. This paper studies an extension to robot imitation learning, called Discovery of Deep Continuous Options (DDCO), where low-level continuous control skills parametrized by deep neural networks are learned from demonstrations. We extend DDO with: (1) a hybrid categorical–continuous distribution model to parametrize high-level policies that can invoke discrete options as well continuous control actions, and (2) a cross-validation method that relaxes DDO's requirement that users specify the number of options to be discovered. We evaluate DDCO in simulation of a 3-link robot in the vertical plane pushing a block with friction and gravity, and in two physical experiments on the da Vinci surgical robot, needle insertion where a needle is grasped and inserted into a silicone tissue phantom, and needle bin picking where needles and pins are grasped from a pile and categorized into bins. In the 3-link arm simulation, results suggest that DDCO can take 3x fewer demonstrations to achieve the same reward compared to a baseline imitation learning approach. In the needle insertion task, DDCO was successful 8/10 times compared to the next most accurate imitation learning baseline 6/10. In the surgical bin picking task, the learned policy successfully grasps a single object in 66 out of 99 attempted grasps, and in all but one case successfully recovered from failed grasps by retrying a second time.

Keywords: Robotics, Imitation, Hierarchy

1 Introduction

An option is a short-term skill consisting of a control policy specialized for a specified region of the state space, and a termination condition recognizing leaving that region [1]. These short-term skills can be parametrized more concisely, e.g., by locally linear approximations to overall nonlinear motions, and these parameters can be substantially easier to learn, given the hierarchical structure. The LfD and motion planning communities have long studied the problem of identifying and learning re-usable motion primitives for low-dimensional observation spaces [2, 3, 4, 5, 6, 7, 8]. A key question is how to extend these results to discover options that map high-dimensional observations to continuous controls and are parametrized by expressive deep neural networks.

Recently, several hierarchical reinforcement learning techniques have been proposed in deep reinforcement learning, which achieve state-of-the-art results on playing Atari games [9, 10, 11]. One of these techniques is the Discovery of Deep Options (DDO) algorithm that takes a policy-gradient approach to train an Abstract Hidden Markov Model [5, 12] via Expectation-Gradient iterations. This

*Equal contribution

1st Conference on Robot Learning (CoRL 2017), Mountain View, United States.

allows DDO to efficiently discover a specified number of *deep options* from a set of demonstration trajectories, and train a policy for the task consisting of a high-level controller that selects from these options and the primitive actions [10].

This paper presents an extension, called Discovery of Deep *Continuous* Options (DDCO), that extends the DDO algorithm to continuous control spaces and robotic imitation learning tasks. This requires two main algorithmic contributions. First, we propose a parametrized nested representation which can be used to represent and train distributions over hybrid output spaces consisting of discrete options and continuous actions. Next, one limitation of DDO is that users must specify the number of options to be discovered. Tuning this parameter online by exhaustive search can be infeasible in robotic settings which rely on physical evaluation. We show that an efficient offline cross-validation step that maximizes the likelihood on a holdout set of demonstrations can automatically select the number of options. Empirically, we show that DDCO learns options from demonstrations provided by human or algorithmic supervisors in simulation with a 3-link robot in the vertical plane pushing a block with friction and gravity, and in two physical experiments on the da Vinci surgical robot, needle insertion and needle bin picking.

2 Related Work

The concept of motion primitives, which are temporally-extended actions on a higher level of abstraction than direct motor control, is well studied [13]. Brooks described this architecture as "layered control" [14], and these ideas helped shape the field of hierarchical reinforcement learning [15, 1, 16]. The robotics community has recognized the need for closed-loop primitives, i.e., primitives that define control policies, and has proposed several control-theoretic models for defining and composing such primitives [2, 3, 4, 7, 17]. Similarly, task segmentation in Learning from Demonstrations considers partitioning a task based on demonstration data, which can be thought of as options [18, 19, 20, 21, 22, 23, 24, 8, 5, 6]. However, all of these prior approaches consider low-dimensional observation spaces, and we look to extend these ideas to primitives that map high-dimensional observations, such as images, to controls. There are also several approaches that consider "demonstration clustering", that is, identifying clusters of already segmented demonstrations [25, 26, 27, 28]. We consider an algorithm that automatically segments and clusters the demonstrations.

Extending similar concepts to image data has been a recent area of interest, mostly in reinforcement and self-supervision settings for simulated domains [29, 11, 30, 10]. We explore leveraging these recent advances for imitation learning in realistic robotic manipulation tasks. We extend one such algorithm from our prior work, DDO [10], which uses a policy-gradient method to train an Abstract Hidden Markov Model [5, 12] via Expectation-Gradient iterations [31, 32]. DDCO builds on DDO and on our prior work in task segmentation (Transition State Clustering [6] and Sequential Windowed Inverse Reinforcement Learning [8]) with two new algorithmic extensions: (1) a hybrid categorical–continuous distribution model to parametrize high-level policies that can invoke discrete options as well continuous control actions, and (2) a cross-validation method that relaxes DDO's requirement that users specify the number of options to be discovered.

3 Background: Imitation Learning and Options

We consider the Imitation Learning (IL) setting, where a control policy is inferred from a dataset of demonstrations of the behavior of a human or algorithmic supervisor. We model the system as discrete-time, having at time t a state s_t in some state space $\mathcal{S}$, such that applying the control a_t in control space $\mathcal{A}$ induces a stochastic state transition $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. The initial state is distributed $s_0 \sim p_0(s_0)$. We assume that the set of demonstrations consists of trajectories, each a sequence of states and controls $\xi = (s_0, a_0, s_1, \ldots, s_T)$ of a given length T.

A flat, non-hierarchical policy $\pi_\theta(a_t|s_t)$ defines the distribution over controls given the state, parametrized by $\theta \in \Theta$. For example, we can maximize the log-likelihood $L[\theta; \xi]$ that the stochastic process induced by the policy π_θ assigns to each demonstration trajectory ξ:

$$L[\theta; \xi] = \log p_0(s_0) + \sum_{t=0}^{T-1} \log(\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)).$$

When $\log \pi_\theta$ is parametrized by a deep neural network, we can perform stochastic gradient descent by sampling a batch of transitions, e.g. one complete trajectory, and computing the gradient

$$\nabla_\theta L[\theta;\xi] = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t).$$

Note that this method can be applied model-free, without any knowledge of the system dynamics p. This paper considers $\pi_\theta(a_t|s_t)$ to be an isotropic Gaussian distribution, with a computed mean $\mu_\theta(s_t)$ and a fixed variance σ^2 (a hyper-parameter that can be set by cross-validation), simplifying the gradient to the standard quadratic loss:

$$\nabla_\theta L[\theta;\xi] = -\sum_{t=0}^{T-1} \nabla_\theta \frac{(\mu_\theta(s_t)-a_t)^2}{2\sigma^2} = \sum_{t=0}^{T-1} \left(\frac{a_t-\mu_\theta(s_t)}{\sigma^2}\right)^\intercal \nabla_\theta \mu_\theta(s_t). \tag{1}$$

An option represents a low-level policy that can be invoked by a high-level policy to perform a certain sub-task [1]. An option h in an options set $\mathcal{H}$ is specified by a control policy $\pi_h(a_t|s_t)$ and a stochastic termination condition $\psi_h(s_t) \in [0,1]$. The high-level policy $\eta(h_t|s_t)$ defines the distribution over options given the state. Once an option h is invoked, physical controls are selected by the option's policy π_h until it terminates. After each physical control is applied and the next state s' is reached, the option h terminates with probability $\psi_h(s')$, and if it does then the high-level policy selects a new option h' with distribution $\eta(h'|s')$. Thus the interaction of the hierarchical control policy $\langle \eta, (\pi_h, \psi_h)_{h\in\mathcal{H}} \rangle$ with the system induces a stochastic process over the states s_t, the options h_t, the controls a_t, and the binary termination indicators b_t.

4 Discovery of Deep Continuous Options (DDCO)

First, we overview our prior work on the Discovery of Deep Options (DDO) algorithm [10], and present its extension to continuous control spaces, as well as a cross-validation method for selecting the number of options to be discovered.

4.1 Discovery of Deep Options (DDO)

DDO is a policy-gradient algorithm that discovers deep options by fitting their parameters to maximize the likelihood of a set of demonstration trajectories. We denote by θ the vector of all trainable parameters used for η and for π_h and ψ_h of each option $h \in \mathcal{H}$. We wish to find the $\theta \in \Theta$ that maximizes the log-likelihood of generating each demonstration trajectory $\xi = (s_0, a_0, s_1, \ldots, s_T)$. The challenge is that this log-likelihood depends on the latent variables in the stochastic process, namely the options and the termination indicators $\zeta = (b_0, h_0, b_1, h_1, \ldots, h_{T-1})$. DDO employs the Expectation Gradient trick [31, 32]:

$$\nabla_\theta L[\theta;\xi] = \mathbb{E}_\theta[\nabla_\theta \log \mathbb{P}_\theta(\zeta,\xi)|\xi],$$

where $\mathbb{P}_\theta(\zeta,\xi)$ is the joint probability of the latent and observable variables, given by

$$\mathbb{P}_\theta(\zeta,\xi) = p_0(s_0)\delta_{b_0=1}\eta(h_0|s_0) \prod_{t=1}^{T-1} \mathbb{P}_\theta(b_t,h_t|h_{t-1},s_t) \prod_{t=0}^{T-1} \pi_{h_t}(a_t|s_t)p(s_{t+1}|s_t,a_t),$$

where in the latent transition $\mathbb{P}_\theta(b_t,h_t|h_{t-1},s_t)$ we have with probability $\psi_{h_{t-1}}(s_t)$ that $b_t = 1$ and h_t is drawn from $\eta(\cdot|s_t)$, and otherwise that $b_t = 0$ and h_t is unchanged, i.e.

$$\begin{aligned}
\mathbb{P}_\theta(b_t{=}1,h_t|h_{t-1},s_t) &= \psi_{h_{t-1}}(s_t)\eta(h_t|s_t) \\
\mathbb{P}_\theta(b_t{=}0,h_t|h_{t-1},s_t) &= (1-\psi_{h_{t-1}}(s_t))\delta_{h_t=h_{t-1}}.
\end{aligned}$$

Expectation-Gradient follows an alternating optimization scheme similar to Expectation-Maximization (EM), with an E-step and a G-step. The E-step computes the marginal posteriors

$$u_t(h) = \mathbb{P}_\theta(h_t{=}h|\xi); \qquad v_t(h) = \mathbb{P}_\theta(b_t{=}1,h_t{=}h|\xi); \qquad w_t(h) = \mathbb{P}_\theta(h_t{=}h,b_{t+1}{=}0|\xi)$$

using a forward-backward algorithm similar to Baum-Welch [33], and the G-step uses these posteriors to compute the log-likelihood gradient

$$\nabla_\theta L[\theta;\xi] = \sum_{h\in\mathcal{H}}\left(\sum_{t=0}^{T-1}\left(v_t(h)\nabla_\theta \log\eta(h|s_t) + u_t(h)\nabla_\theta \log\pi_h(a_t|s_t)\right)\right. \tag{2}$$
$$\left. + \sum_{t=0}^{T-2}\left((u_t(h) - w_t(h))\nabla_\theta \log\psi_h(s_{t+1}) + w_t(h)\nabla_\theta \log(1-\psi_h(s_{t+1}))\right)\right).$$

The DDO algorithm is derived in detail in [10]. The gradient computed above can then be used in any stochastic gradient descent algorithm. In its original formulation, DDO is applied to finite action spaces, and $\log\pi_h$ is the log-probability of a categorical distribution.

4.2 Hybrid Categorical–Continuous Distribution Model

To apply DDO to continuous control spaces, we can try to compute the log-density of the continuous distribution $\pi_h(a_t|s_t)$, which in the case of isotropic Gaussian distributions is proportional to the square distance of a_t from the mean control $\mu_h(s_t)$, as in (1). We allow the high-level policy to solve some sub-tasks directly, without invoking an option, by augmenting its output space to include both physical control and options, $\mathcal{A}\cup\mathcal{H}$. In states that are not clustered into any option, the high-level policy can choose the physical control to apply in every step. In continuous control spaces, this introduces the challenge that $\eta(h_t|s_t)$ is now a hybrid categorical–continuous distribution. We denote by $\eta(h^c|s) = 1 - \sum_{h\in\mathcal{H}}\eta(h|s)$ the probability that the high level applies physical control, and by $\mu_\eta(s)$ the mean control in that event. Then the continuous part of the distribution has the unnormalized density $\eta(a|s) = \eta(h^c|s)\mathcal{N}(a;\mu_\eta(s),\sigma^2)$.

This allows us to represent and train the hybrid distribution with the following neural network architecture. With $k = |\mathcal{H}|$, the output layer represents a categorical distribution with log-probabilities $y_0,\ldots,y_k$, as well as the parameter of the control distribution μ_η. The relevant gradient terms in (2) are, as before, $v_t(h)\nabla_\theta y_h(s_t$, and now additionally the complement term $v_t(h^c)\nabla y_0(s_t)$, with the probability $v_t(h^c) = \mathbb{P}_\theta(b_t{=}1) - \sum_{h\in\mathcal{H}} v_t(h)$ that the high level applies physical control in time-step t; as well as the control-gradient term $v_t(h^c)\left(\frac{a_t-\mu_\eta(s_t)}{\sigma^2}\right)^\intercal\nabla_\theta\mu_\eta(s_t)$, similarly to (1).

4.3 Cross-Validation For Parameter Tuning

We explored whether it is possible to tune the number of options offline. DDCO is based on a maximum-likelihood formulation, which describes the likelihood that the observed demonstrations are generated by a hierarchy parametrized by θ. However, the model expressiveness is strictly increasing in k, causing the optimal training likelihood to increase even beyond the point where the model overfits to the demonstrations and fails to generalize to unseen states. We therefore adopt a cross-validation technique that holds out 10% of the demonstration trajectories for each of 10 folds, trains on the remaining data, and validates the trained model on the held out data. We select the value of k that achieves the highest average log-likelihood over the 10 folds, suggesting that training such a hierarchical model generalizes well. We train the final policy over the entire data.

5 Results

5.1 Box2D Simulation: 2D Surface Pushing with Friction and Gravity

In the first experiment, we simulate a 3-link robot arm in Box2D (Figure 1). This arm consists of three links of lengths 5 units, 5 units, and 3 units, connected by ideal revolute joints. The arm is controlled by setting the values of the joint angular velocities $\dot\phi_1,\dot\phi_2,\dot\phi_3$. In the environment, there is a box that lies on a flat surface with uniform friction. The objective is to push this box without toppling it until it rests in a randomly chosen goal position. After this goal state is reached, the goal position is regenerated randomly. The task is for the robot to push the box to as many goals as possible in 2000 time-steps. Our algorithmic supervisor runs the RRT Connect motion planner of the Open Motion Planning Library, ompl, at each time-step planning to reach the goal. Due to the geometry of the configuration space and the task, there are two classes of trajectories that are generated, when the goal is to the left or right of the arm. Details of the environment and motion planner are included in the supplementary material (**SM 1.1**).

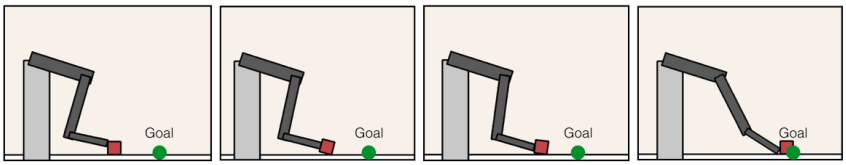

Figure 1: A 3-link robot arm has to push a box along a surface with friction to a randomly chosen goal state to the box's left or right without toppling it. Due to the collision and contact constraints of this task, it has two geometrically distinct pushing skills, backhand and forehand, for goals on the left and on the right.

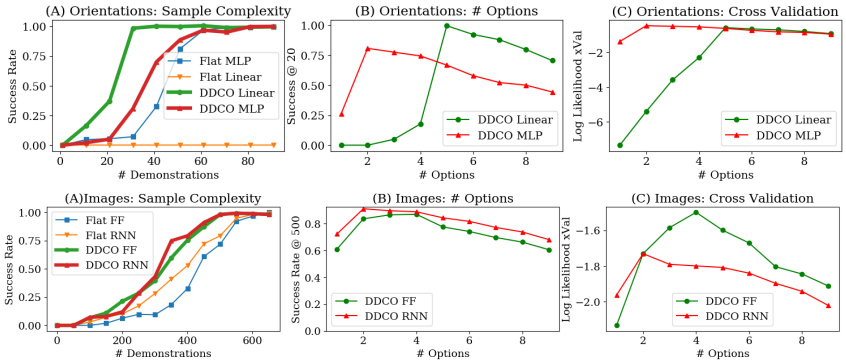

Figure 2: 2D Surface Pushing from low-dimensional observations and image observations. (A) The sample complexity of different policy representations, (B) The success rate for different numbers of discovered options, (C) The peak in the reward correlates over the number of options with the 10-fold cross-validated log-likelihood.

Observing Positions and Orientations: First, we consider the case where we observe the full low-dimensional state of the system: three joint angles, the box's position and orientation, and the position of the goal state. We compare our hierarchical policies with two flat, non-hierarchical policies. One baseline policy we consider is an under-parametrized linear policy which is not expressive enough to approximate the supervisor policy well. The other baseline policy is a multi-layer perceptron (MLP) policy. We train each of these policies via Behavior Cloning (BC), i.e. by maximizing the likelihood each gives to the set of demonstrations. As expected, the flat linear policy is unsuccessful at the task for any number of observed demonstrations (Figure 2 Top-A). The MLP policy, on the other hand, can achieve the maximum reward when trained on 60 demonstrations or more.

We apply DDCO and learn two 2-level hierarchical policies, one with linear low-level options, and the other with MLP low-level options of the same architecture used for the flat policy. In both cases, the termination conditions are parametrized by a logistic regression from the state to the termination probability, and the high-level policy is a logistic regression from the state to an option selection. For the linear hierarchy we set DDCO to discover 5 options, and for the MLP hierarchy we discover 2 options. The MLP hierarchical policy can achieve the maximum reward with 30 demonstrations, and is therefore 2x more sample-efficient than its flat counterpart (Figure 2 Top A). Details of the parametrization are included in the Supplementary Material (**SM 1.1**).

We also vary the number of options discovered by DDCO, and plot the reward obtained by the resulting policy (Figure 2 Top-B). While the performance is certainly sensitive to the number of options, we find that the benefit of having sufficiently many options is only diminished gradually with each additional option beyond the optimum. Importantly, the peak in the cross-validated log-likelihood corresponds to the number of options that achieves the maximum reward (Figure 2 Top-C). This allows us to use cross-validation to select the number of options without having to evaluate the policy by rolling it out in the environment.

Observing Images: Next, we consider the case where the sensor inputs are 640x480 images of the scene. The low-dimensional state is still fully observable in these images, however these features are not observed explicitly, and must be extracted by the control policy. We consider two neural network architectures to represent the policy: a convolutional layer followed by either a fully connected layer

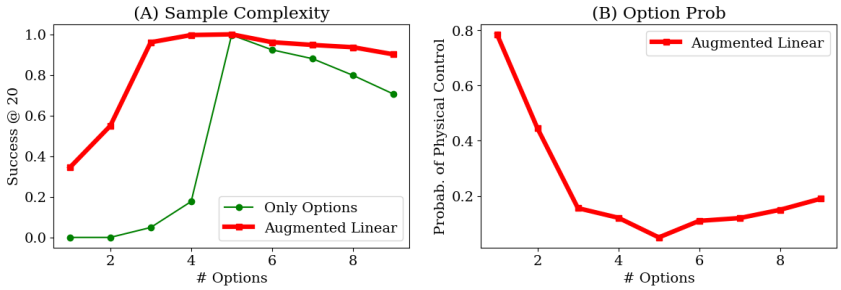

Figure 3: (A) The sample complexity of augmenting the high-level control space v.s. selecting only options. (B) The fraction of high-level selections that are of physical control. As the number of options increases the frequency of selecting a physical control first goes down, then up again as options overfit.

or an LSTM, respectively forming a feed-forward (FF) network and a recurrent network (RNN). We use these architectures both for flat policies and for low-level options in hierarchical policies. For the hierarchical policies, as in the previous experiment, we consider a high-level policy that only selects options. For the FF policy we discover 4 options, and for the RNN policy we discover 2 options. The details of the parametrization are included in the Supplementary Material (**SM 1.2**).

The hierarchical policies require 30% fewer demonstrations than the flat policies to achieve the maximum reward (Figure 2 Bottom A). Figure 2 Bottom-B and Figure 2 Bottom-C show how the success rate and cross-validated log-likelihood vary with the number of options. As for the low-dimensional inputs, the success rate curve is correlated with the cross-validated log-likelihood. We can rely on this to select the number of options offline without rolling out the learned hierarchical policy.

Control Space Augmentation: We test two different architectures for the output layer of the high-level policy: either a softmax categorical distribution selecting an option, or the hybrid categorial–continuous distribution output described in Section 4.2. The low-level policies are linear.

Figure 3 describes the empirical estimation, through policy rollouts, of the success rate as a function of the number of options, and of the fraction of time that the high-level policy applies physical control. When the options are too few to provide skills that are useful throughout the state space, physical controls can be selected instead to compensate in states where no option would perform well. This is indicated by physical control being selected with greater frequency. As more options allow a better coverage of the state space, the high-level policy selects physical controls less often, allowing it to generalize better by focusing on correct option selection. With too many discovered options, each option is trained from less data on average, making some options overfit and become less useful to the high-level policy. In the Supplementary Material (**SM 1.5**), we perform the same experiment for the image-based case, with similar results.

5.2 Physical Experiment 1: Needle Insertion

We consider a needle orientation and insertion task on the da Vinci Research Kit (DVRK) surgical robot (Figure 4). In this task, the robot must grasp a surgical needle, reorient it in parallel to a tissue phantom, and insert the needle into the tissue phantom. The task is successful if the needle is inserted into a 1cm diameter target region on the phantom. Small changes in the needle's initial orientation can lead to large changes in the in-gripper pose of the needle due to deflection. The state is the current 6-DoF pose of the robot gripper, and algorithmically extracted visual features that describe the estimated pose of the needle. These features are derived from an image segmentation that masks the needle from the background and fits an ellipsoid to the resulting pixels. The principal axis of this 2D ellipsoid is a proxy for the pose of the needle. The task runs for a fixed 15 time-steps, and the policy must set the joint angles of the robot at each time-step.

The needle's deflection coupled with the inaccurate kinematics of the DVRK make it challenging to plan trajectories to insert the needle properly. A visual servoing policy needs to be trained that can both grasp the needle in the correct position, as well as reorient the gripper in the correct direction after grasping. To collect demonstrations, we programmed an initial open-loop control policy, interrupted

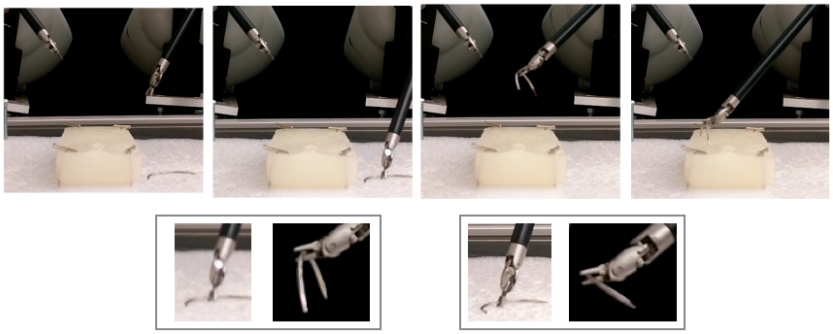

Figure 4: A needle orienting and insertion task. The robot must grasp a surgical needle, reorient it in parallel to a tissue phantom, and insert the needle.

the robot via keyboard input when adjustment was needed, and kinesthetically adjusted the pose of the gripper. We collected 100 such demonstrations.

We evaluated the following alternatives: (1) a single flat MLP policy with continuous output, (2) a flat policy consisting of 15 distinct MLP networks, one for each time-step, (3) a hierarchical policy with 5 options trained with DDCO. We considered a hierarchy where the high-level policy is a MLP with a softmax output that selects the appropriate option, and each option is parametrized by a distinct MLP with continuous outputs. DDCO learns 5 options, two of which roughly correspond to the visual servoing for grasping and lifting the needle, and the other three handle three different types of reorientation. For 100 demonstrations, the hierarchical policy learned with DDCO has a 45% higher log-likelihood measured in cross-validation than the flat policy, and a 24% higher log-likelihood than the per-timestep policy. Training results, hyper-parameter tuning, interpretation of the options learned, and details of the parametrization are included in the Supplementary Material (**SM 1.3**).

We ran preliminary trials to confirm that the trained options can be executed on the robot. For each of the methods, we report the success rate in 10 trials, i.e. the fraction of trials in which the needle was successfully grasped and inserted in the target region. All of the techniques had comparable accuracy in trials where they successfully grasped and inserted the needle into the 1cm diameter target region. The algorithmic open-loop policy only succeeded 2/10 times. Surprisingly, Behavior Cloning (BC) did not do much better than the open-loop policy, succeeding only 3/10 times. Per-timestep BC was far more successful (6/10). Finally, the hierarchical policy learned with DDCO succeeded 8/10 times. On 10 trials it was successful 5 times more than the direct BC approach and 2 times more than the per-timestep BC approach. While not statistically significant, our preliminary results suggest that hierarchical imitation learning is also beneficial in terms of task success, in addition to improving model generalization and interpretability. Detailed discussion of the particular failure modes is included in the Supplementary Material (**SM 1.3**).

5.3 Physical Experiment 2: Surgical Bin Picking

In this task, the robot is given a foam bin with a pile of 5–8 needles of three different types, each 1–3mm in diameter. The robot must extract needles of a specified type and place them in an "accept" cup, while placing all other needles in a "reject" cup. The task is successful if the entire foam bin is cleared into the correct cups.

In initial trials, the kinematics of the DVRK were not precise enough for grasping needles. We then realized that visual servoing is needed, which requires learning. However, even with visual servoing, failures are common, and we would like to also learn automatic recovery behaviors. To define the state space for this task, we first generate binary images from overhead stereo images, and apply a color-based segmentation to identify the needles (the image input). Then, we use a classifier trained in advance on 40 hand-labeled images to identify and provide a candidate grasp point, specified by position and direction in image space (the grasp input). Additionally, the 6 DoF robot gripper pose and the open-closed state of the gripper are observed (the kin input). The state space of the robot is (image, grasp, kin), and the control space is the 6 joint angles and the gripper angle.

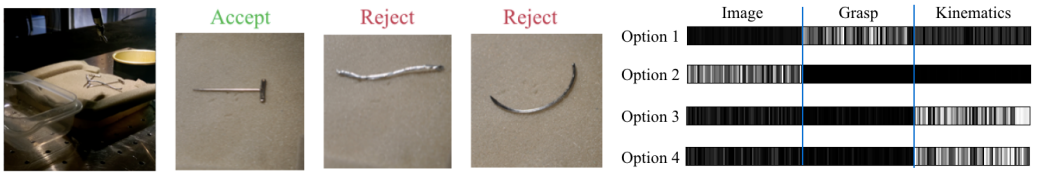

Figure 5: (Left) Surgical Bin Picking: the robot is given a foam bin with a pile of 5–8 needles of three different types, each 1–3mm in diameter. The robot must extract needles of a specified type and place them in an "accept" cup, while placing all other needles in a "reject" cup. (Right) For each of the 4 options, we plot how heavily the different inputs are weighted (image, grasp, or kin) in computing the option's action. Nonzero values of the ReLU units are marked in white and indicate input relevance.

Each sequence of grasp, lift, move, and drop operations is implemented in 10 control steps of joint angle positions. As in the previous task, we programmed an initial open-loop control policy, interrupted the robot via keyboard input when adjustment was needed, and kinesthetically adjusted the pose of the gripper. We collected 60 such demonstrations, in each fully clearing a pile of 3–8 needles from the bin, for a total of 450 individual grasps. Details of the state space and the demonstration protocol are included in the Supplementary Material (**SM 1.4**).

We apply DDCO to the collected demonstrations with the policy architecture described in **SM 1.4**. In this network, there are three inputs: a binary image, a candidate grasp, and kinematics (kin). These inputs are processed by distinct branches of the network before being aggregated into a single feature layer. Using the cross-validation technique described in Section 4.3, we selected the number of options to be $k = 4$.

We plot the average activations of the feature layer for each low-level option (Figure 5). While this is only a coarse analysis, it gives some indication of which inputs (image, grasp, or kin) are relevant to the policy and termination. We see that the options are clearly specialized. The first option has a strong dependence only on the grasp candidate, the second option attends almost exclusively to the image, while the last two options rely mostly on kin. Details of the architecture and additional interpretations of the learned options are included in the Supplementary Material (**SM 1.4**).

Finally, we evaluate the success of the learned hierarchical policy in 10 full trials, according to 4 different success criteria. First, the overall task success is measured by the success rate of fully clearing the bin without failing 4 consecutive grasping attempts. Second, we measure the success rate of picking exactly one needle in individual grasp attempts. Third, we measure the success rate of appropriately reacting to grasps, by dropping the load and retrying unsuccessful grasps, and not dropping successful grasps. Fourth, we measured the success rate of needle categorization.

In 10 trials, 7/10 were successful. The main failure mode was unsuccessful grasping due to picking either no needles or multiple needles. As the piles were cleared and became sparser, the robot's grasping policy became somewhat brittle. The grasp success rate was 66% on 99 attempted grasps. In contrast, we rarely observed failures at the other aspects of the task, reaching 97% successful recovery on 34 failed grasps. The Supplementary Material (**SM 1.4**) describes a full breakdown of the failure modes, as well as an additional experiment of policy generalization to unseen objects.

6 Conclusion

This paper extends our prior work on the DDO algorithm, which was designed for discrete action spaces, to continuous robot control problems, and proposes new methods for addressing issues of hybrid discrete–continuous action parametrization and hyper-parameter tuning of the number of options. Results suggest that imposing a hierarchical structure with DDCO on the trained policy has the potential to significantly reduce the number of demonstrations needed for learning in real and simulated tasks. DDCO facilitates imitation learning for long duration, multi-step tasks such as bin picking which involves grasping, categorizing, and recovery behaviors. We believe that this hierarchical structure can also facilitate generalization in multi-task learning settings, and hope to explore that in further detail in future work. In future work, we also hope to explore heterogeneous option parametrizations, where some options are learned and some are derived from analytic formulae.

Acknowledgments

This research was performed at the AUTOLAB at UC Berkeley and the Real-Time Intelligent Secure Execution (RISE) Lab in affiliation with the Berkeley AI Research (BAIR) Lab and the CITRIS "People and Robots" (CPAR) Initiative. This research is supported in part by DHS Award HSHQDC-16-3-00083, NSF CISE Expeditions Award CCF-1139158, by the Scalable Collaborative Human-Robot Learning (SCHooL) Project NSF National Robotics Initiative Award 1734633, and donations from Alibaba, Amazon, Ant Financial, CapitalOne, Ericsson, GE, Google, Huawei, Intel, IBM, Microsoft, Scotiabank, VMware, Siemens, Cisco, Autodesk, Toyota Research, Samsung, Knapp, and Loccioni Inc. We also acknowledge a major equipment grant from Intuitive Surgical and by generous donations from Andy Chou and Susan and Deepak Lim. We thank our colleagues who provided helpful feedback and suggestions, in particular Jeff Mahler and Michael Laskey.

References

[1] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *AI*, 112(1-2):181–211, 1999. doi:10.1016/S0004-3702(99)00052-1.

[2] A. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Neural Information Processing Systems (NIPS)*, pages 1523–1530, 2002.

[3] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *IEEE ICRA*, 2009.

[4] S. Manschitz, J. Kober, M. Gienger, and J. Peters. Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations. *Robotics and Autonomous Systems*, 2015.

[5] H. H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden Markov model. *JAIR*, 17:451–499, 2002.

[6] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. Goldberg. Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. In *International Symposium of Robotics Research. Springer STAR*, 2015.

[7] C. Daniel, G. Neumann, and J. Peters. Hierarchical relative entropy policy search. In *AISTATS*, pages 273–281, 2012.

[8] S. Krishnan, A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg. Swirl: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards.

[9] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. *arXiv preprint arXiv:1609.05140*, 2016.

[10] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

[11] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

[12] C. Daniel, H. Van Hoof, J. Peters, and G. Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3):337–357, 2016.

[13] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial intelligence*, 3:251–288, 1972.

[14] R. Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986.

[15] R. E. Parr. *Hierarchical control and learning for Markov decision processes*. PhD thesis, UNIVERSITY of CALIFORNIA at BERKELEY, 1998.

[16] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003. doi:10.1023/A:1022140919877.

[17] R. Lioutikov, G. Neumann, G. Maeda, and J. Peters. Probabilistic segmentation applied to an assembly task. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 533–540. IEEE, 2015.

[18] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *NIPS*, pages 271–278, 1992.

[19] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *ICML*, volume 2, pages 243–250, 2002.

[20] J. Z. Kolter, P. Abbeel, and A. Y. Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. In *NIPS*, volume 20, 2007.

[21] G. Konidaris and A. G. Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900, 2007.

[22] S. Niekum, S. Osentoski, G. Konidaris, and A. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2012.

[23] S. Calinon. Skills learning in robots by interaction with users and environment. In *IEEE Int. Conf. on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2014.

[24] G. Konidaris, S. Kuindersma, R. A. Grupen, and A. G. Barto. Robot learning from demonstration by constructing skill trees. *IJRR*, 31(3):360–375, 2012. doi:10.1177/0278364911428653.

[25] R. Fox, M. Moshkovitz, and N. Tishby. Principled option learning in Markov decision processes. In *UAI*, 2016.

[26] K. Hausman, Y. Chebotar, S. Schaal, G. Sukhatme, and J. Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. *arXiv preprint arXiv:1705.10479*, 2017.

[27] Z. Wang, J. Merel, S. Reed, G. Wayne, N. de Freitas, and N. Heess. Robust imitation of diverse behaviors. *arXiv preprint arXiv:1707.02747*, 2017.

[28] M. Wulfmeier, I. Posner, and P. Abbeel. Mutual alignment transfer learning. *arXiv preprint arXiv:1707.07907*, 2017.

[29] A. Jonsson and V. Gómez. Hierarchical linearly-solvable Markov decision problems. *arXiv preprint arXiv:1603.03267*, 2016.

[30] C. Florensa, Y. Duan, and P. Abbeel. Cstochastic neural networks for hierarchical reinforcement learning. In *ICLR*, 2017.

[31] R. Salakhutdinov, S. Roweis, and Z. Ghahramani. Optimization with EM and expectation-conjugate-gradient. In *ICML*, pages 672–679, 2003.

[32] G. McLachlan and T. Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.

[33] L. E. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

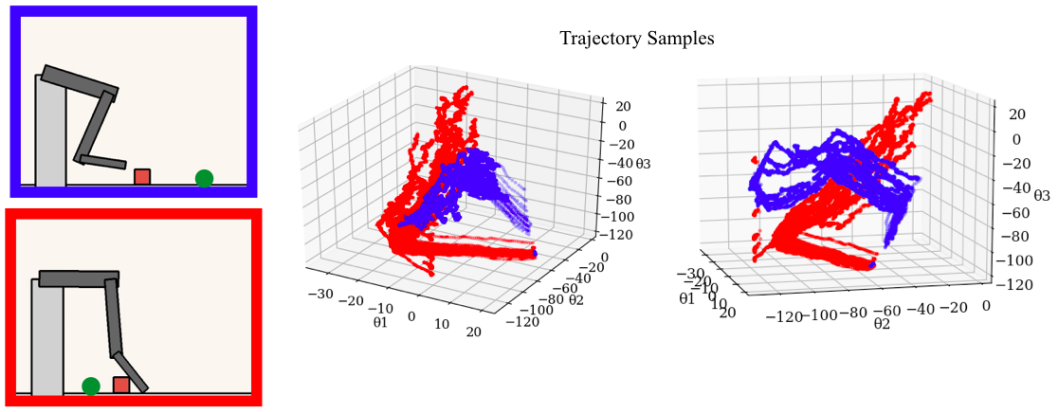

Figure 6: Due to the collision and contact constraints of the 2D Surface Pushing task, it leads to two geometrically distinct pushing skills, backhand and forehand, for goals on the left and on the right. Right: 50 sampled trajectories from a motion-planning supervisor.

Supplement: Experimental Details

1.1 Box2D Simulation: 2D Surface Pushing with Friction

Algorithmic Supervisor: Our algorithmic supervisor runs an RRT Connect motion planner (from the Open Motion Planning Library ompl) at each time-step planning till the goal. The motion planning algorithm contains a single important hyper-parameter which is the maximum length of branches to add to the tree. We set this at 0.1 units (for a 20 x 20 unit space).

Task Geometry: We designed this experiment to illustrate how options can lead to more concise representations since they can specialize in different regions of the state-spaces. Due to the geometry of the configuration space and the task, there are two classes of trajectories that are generated, when the goal is to the left or right of the arm. The 50 sampled trajectories plotted in joint angle space in Figure 6 are clearly separated into two distinct skills, backhand and forehand. Furthermore, these skills can be implemented by a locally affine policies in the joint angle space.

Multi-Layer Perceptron Flat Policy: One of the baseline policies is a multi-layer perceptron (MLP) policy which has a single ReLU hidden layer of 64 nodes. This policy is implemented in Tensorflow and is trained with an ADAM optimizer with learning rate $1e-5$.

DDCO Policy 1: In the first policy trained by DDCO, we have a logistic regression meta-policy that selects from one of k linear sub-policies. The linear sub-policies execute until a termination condition determined again by a logistic regression. This policy is implemented in Tensorflow and is trained with an ADAM optimizer with learning rate $1e-5$. For the linear hierarchy, we set DDCO to discover 5 options, which is tuned using the cross-validation method described in the paper.

DDCO Policy 2: In the second policy trained by DDCO, we have a logistic regression meta-policy that selects from one of k multi-layer perceptron sub-policies. As with the flat policy, it has a single ReLU hidden layer of 64 nodes. The MLP sub-policies execute until a termination condition determined again by a logistic regression. This policy is implemented in Tensorflow and is trained with an ADAM optimizer with learning rate $1e-5$. For the MLP hierarchy, we set DDCO to discover 2 options, which is tuned using the cross-validation method described in the paper.

1.2 Image-Based Surface Pushing with Friction

Next, we consider the case where the sensor inputs are 640x480 images of the scene.

DDCO FF Policy: First, we consider a neural network with a convolutional layer with 64 5x5 filters followed by a fully connected layer forming a feed-forward (FF) network. The high-level model only

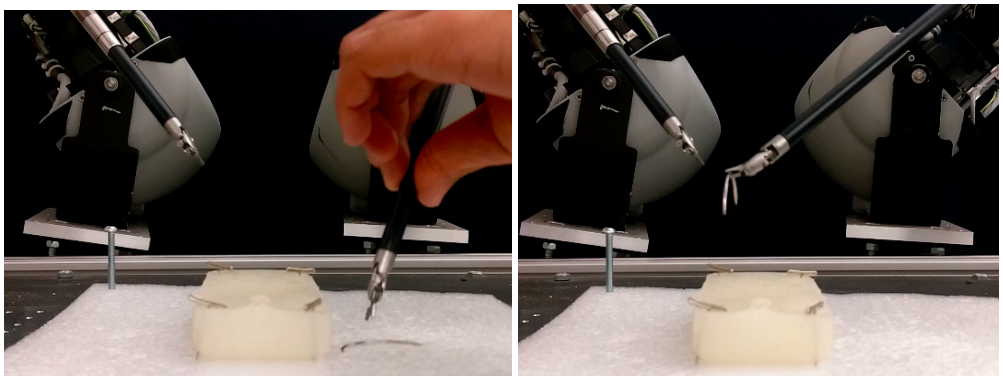

Figure 7: To collect demonstrations, we programmed an initial open-loop control policy. We observed the policy execute and interrupted the robot via keyboard input when adjustment was needed.

selects options and is parametrized by the same general architecture with an additional softmax layer after the fully connected layer. This means that the meta-control policy is a two-layer convolutional network whose output is a softmax categorical distribution over options. This policy is implemented in Tensorflow and is trained with an ADAM optimizer with learning rate $1e-5$. We used $k=2$ options for the FF policy.

DDCO RNN Policy: Next, we consider a neural network with a convolutional layer with 64 5x5 filters followed by an LSTM layer forming a recurrent network (RNN). The high-level model only selects options and is parametrized by the same general architecture with an additional softmax layer after the LSTM. This policy is implemented in Tensorflow and is trained with a Momentum optimizer with learning rate $1e-4$ and momentum $4e-3$. We used $k=4$ options for the RNN policy.

1.3 Physical Experiment 1: Needle Insertion

Task Description: The robot must grasp a 1mm diameter surgical needle, re-orient it parallel to a tissue phantom, and insert the needle into a tissue phantom. The task is successful if the needle is inserted into a 1 cm diameter target region on the phantom. In this task, the state-space is the current 6-DoF pose of the robot gripper and visual features that describe the estimated pose of the needle. These features are derived from an image segmentation that masks the needle from the background and fits an ellipsoid to the resulting pixels. The principal axis of this 2D ellipsoid is a proxy for the pose of the needle. The task runs for a fixed 15 time-steps and the policy must set the joint angles of the robot at each time-step.

Robot Parameters: The challenge is that the curved needle is sensitive to the way that it is grasped. Small changes in the needle's initial orientation can lead to large changes to the in-gripper pose of the needle due to deflection. This deflection coupled with the inaccurate kinematics of the DVRK leads to very different trajectories to insert the needle properly.

The robotic setup includes a stereo endoscope camera located 650 mm above the 10cm x 10 cm workspace. After registration, the dvrk has an RMSE kinematic error of 3.3 mm, and for reference, a gripper width of 1 cm. In some regions of the state-space this error is even higher, with a 75% percentile error of 4.7 mm. The learning in this task couples a visual servoing policy to grasp the needle with the decision of which direction to orient the gripper after grasping.

Demonstration Protocol: To collect demonstrations, we programmed an initial open-loop control policy. This policy traced out the basic desired robot motion avoiding collisions and respecting joint limits, and grasping at where it believed the needle was and an open-loop strategy to pin the needle in the phantom. This was implemented by 15 joint angle way points which were interpolated by a motion planner. We observed the policy execute and interrupted the robot via keyboard input when adjustment was needed. This interruption triggered a clutching mechanism and we could kinesthetically adjusted the joints of the robot and pose of the gripper (but not the open-close state). The adjustment was recorded as a delta in joint angle space which was propagated through the rest of

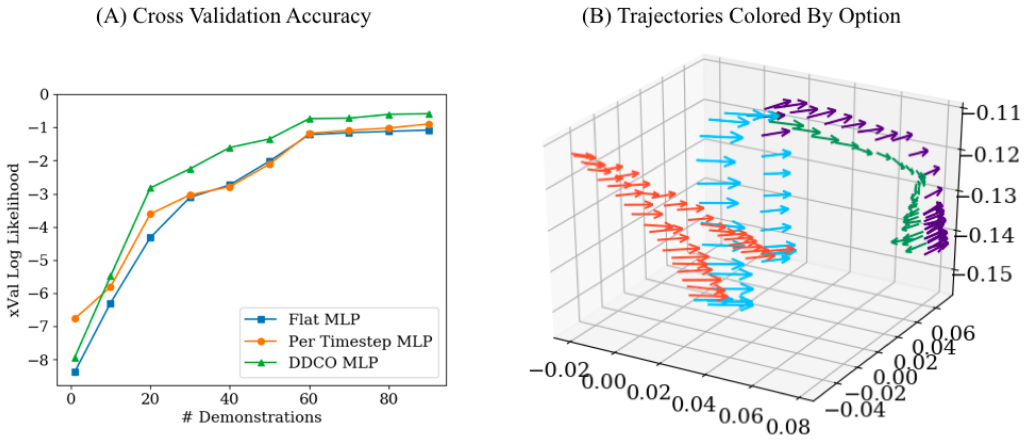

Figure 8: (A) We plot the cross validation likelihood of the different methods as a function of the number of demonstrations. (B) We visualize two representative trajectories (position and gripper orientation) color coded by the most likely option applied at that timestep. We find that the two trajectories have the same first two options but then differ in the final step due to the re-orientation of the gripper before insertion.

the trajectory. We collected 100 such demonstrations and images of these adjustements are visualized in image (Figure 7).

Learning the Parameters: Figure 8A plots the cross-validation log-likelihood as a function of the number of demonstrations. We find that the hierarchical model has a higher likelihood than the alternatives—meaning that it more accurately explains the observed data and generalizes better to held out data. At some points, the relative difference is over 30%. It, additionally, provides some interpretability to the learned policy. Figure 8B visualizes two representative trajectories. We color code the trajectory based on the option active at each state (estimated by DDCO). The algorithm separates each trajectory into 3 segments: needle grasping, needle lifting, and needle orienting. The two trajectories have the same first two options but differ in the orientation step. One of the trajectories has to rotate in a different direction to orient the needle before insertion.

Flat Policy: One of the baseline policies is a multi-layer perceptron (MLP) policy which has a single ReLU hidden layer of 64 nodes. This policy is implemented in Tensorflow and is trained with an ADAM optimizer with learning rate $1e-5$.

Per-Timestep Policy: Next, we consider a degenerate case of options where each policy executes for a single-timestep. We train 15 distinct multi-layer perceptron (MLP) policies each of which has a single ReLU hidden layer of 64 nodes. Thes policies are implemented in Tensorflow and are trained with an ADAM optimizer with learning rate $1e-5$.

DDCO Policy: DDCO trains a hierarchical policy with 5 options. We considered a hierarchy where the meta policy is a multilayer perceptron with a softmax output that selects the appropriate option, and the options are parametrized by another multilayer perceptron with continuous outputs. Each of the MLP policies has a single ReLU hidden layer of 64 nodes. Thes policies are implemented in Tensorflow and are trained with an ADAM optimizer with learning rate $1e-5$.

Execution: For each of the methods, we execute ten trials and report the success rate (successfully grasped and inserted the needle in the target region), and the accuracy. The results are described in aggregate in the table below:

	Overall Success	Grasp Success	Insertion Success	Insertion Accuracy
Open Loop	2/10	2/10	0/0	7 ± 1 mm
Behavioral Cloning	3/10	6/10	3/6	6 ± 2 mm
Per Timestep	6/10	7/10	6/7	5 ± 1 mm
DDCO	8/10	10/10	8/10	5 ± 2 mm

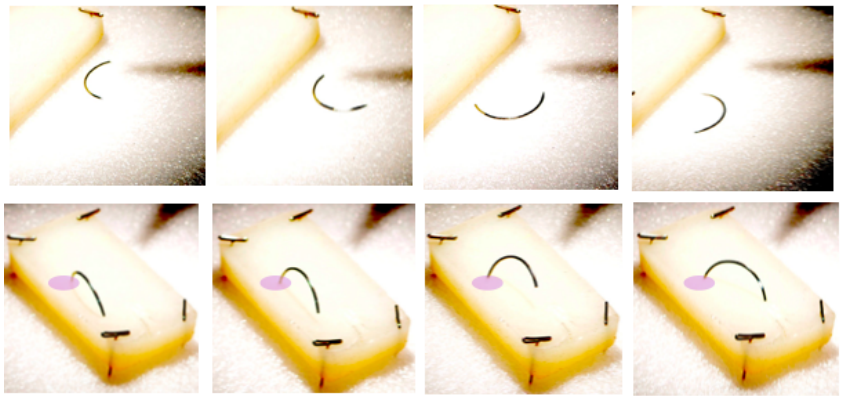

Figure 9: Illustration of the needle orientation and insertion task. Above are images illustrating the variance in the initial state, below are corresponding final states after executing DDCO.

1.4 Physical Experiment 2: Surgical Bin Picking

Task Description: We consider a task with a more complicated high-level structure. In this task, the robot is given a foam bin with 3 different types of needles (1mm-3mm in diameter) lying in a pile (5-8 needles in experiments). The robot must extract a particular type of needle and place it in the accept cup and place all others in a reject cup. The task is successful if the entire foam bin is cleared.

Figure 11 shows representive objects for this task. We consider three types of "needles": dissection pins, suturing needles, and wires. Dissection pins are placed in the accept cup and the other two are placed in the reject cup.

Robot and State-Space Parameters: As in the previous task, the task requires learning because the kinematics of the dvrk are such that the precision needed for grasping needles requires visual servoing. However, even with visual servoing, failures are common due to the pile (grasps of 2, 3, 4 objects). We would like to automatically learn recovery behaviors. In our robotic setup, there is an overhead endoscopic stereo camera, and it is located 650mm above the workspace.

To define the state-space for this task, we first generate binary images from the stereo images and apply a color-based segmentation to identify the needles (we call this feature image). Then, we use a classifier derived from 40 hand-labeled images to identify possible grasp points to sample a candidate

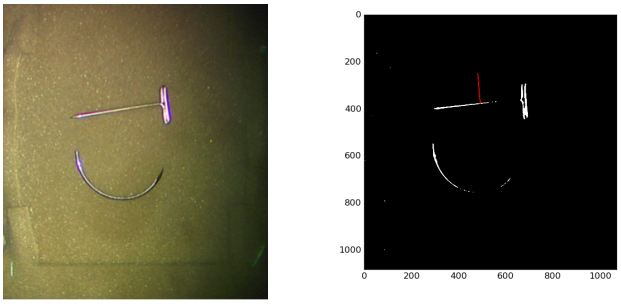

Figure 10: The endoscope image and a corresponding binary mask with a selected grasp. The arrow corresponds to the orientation of the gripper along the grasp axis.

Figure 11: There are two bins, one accept and one reject bin. In the accept bin, we place dissection pins and place the suturing needles and the wires in the other.

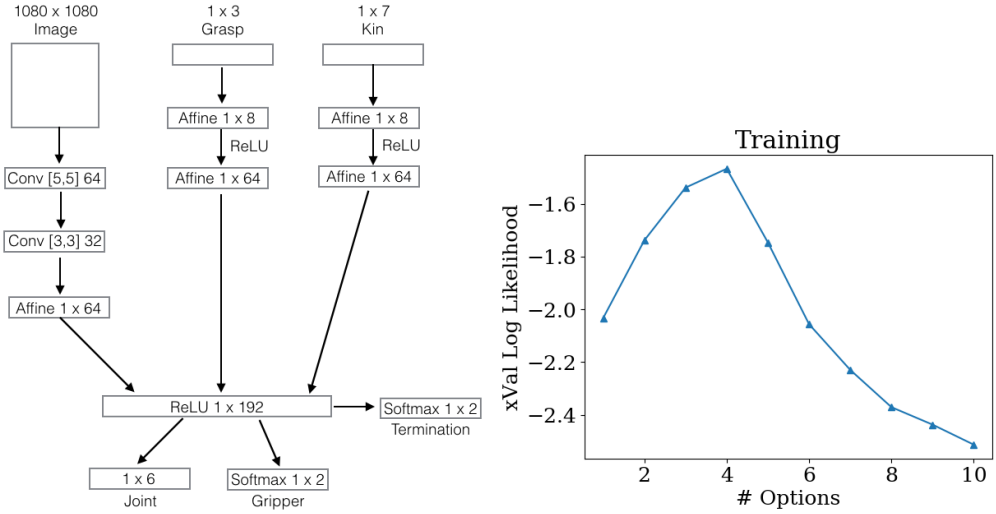

Figure 12: We use the above neural network to parametrize each of the four options learned by DDCO. In this network, there are three inputs the a binary image, a candidate grasp, and kinematics. These inputs are processed through individual neural net branches and aggregated into a single output layer. This output layer sets the joint angles of the robot and the gripper state (open or closed). Option termination is also determined from this output.

grasp (left pixel value, right pixel value, and direction) (we call this feature grasp). These features are visualized in Figure 10. Additionally, there is the 6 DoF robot gripper pose and the open-close state of the gripper (we call this feature kin). The state-space of the robot is (kin, image, grasp), and the action space for the robot is 6 joint angles and the gripper angle. Each grasp, lift, move, and drop operation consists of 10 time steps of joint angle positions. The motion between the joint angles is performed using a SLURP-based motion planner.

Demonstration Protocol: As in the previous task, to collect demonstrations, we start with a hard-coded open-loop policy. We roll this policy out and interrupt the policy when we anticipate a failure. Then, we kinesthetically adjust the pose of the dvrk and it continues. We collected 60 such demonstrations of fully clearing the bin filled with 3 to 8 needles each–corresponding to 450 individual grasps. We also introduced a key that allows the robot to stop in place and drop it current grasped needle. Recovery behaviors were triggered when the robot grasps no objects or more than one object. Due to the kinesthetic corrections, a very high percentage of the attempted grasps (94%) grasped at least one object. Of the successful grasps, when 5 objects are in the pile 32% grasps picked up 2 objects, 14% picked up 3 objects, and 0.5% picked up 4. In recovery, the gripper is opened and the epsiode ends leaving the arm in place. The next grasping trial starts from this point.

Policy Parametrization: We apply DDCO to the collected demonstrations with the policy parametrization described in Figure 12. In this network, there are three inputs the a binary image, a candidate grasp, and kinematics. These inputs are processed through individual neural net branches and aggregated into a single output layer. This output layer sets the joint angles of the

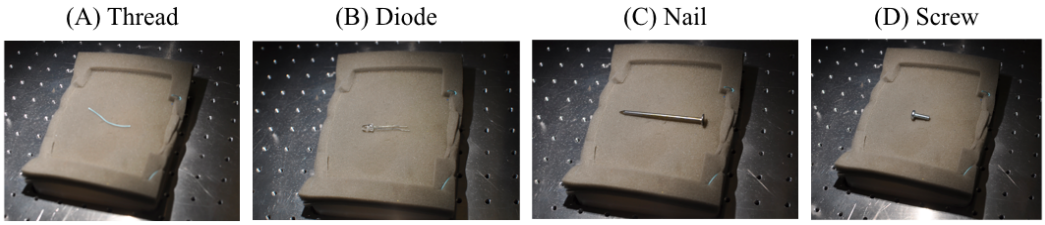

Figure 13: We evaluated the generalization of the learned policy on a small set of unseen objects. This was to understand what features of the object binary mask is used to determine behaviors.

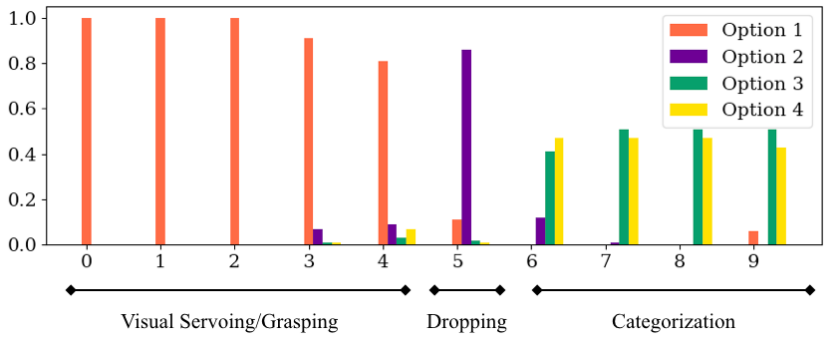

Figure 14: We plot the time distribution of the options selected by the high-level policy. The x-axis represents time, and the bars represent the probability mass assigned to each option. We find that the structure of the option aligns with key phases in the task such as servoing to the needle, grasping it, and categorizing.

robot and the gripper state (open or closed). Option termination is also determined from this output. Using the cross-validation technique described in the paper, we identify 4 options (Figure 12). The high-level policy has the same parametrization but after the output layer there is a softmax operation that selects a lower-level option.

Coordinate Systems: We also experimented with different action space representations to see if that had an effect on single object grasps and categorizations. We trained alternative policies with the collected dataset where instead of predicting joint angles, we alternatively predicted 6 DoF end-effector poses with a binary gripper open-close state , and end-effector poses represented a rotation/translation matrix with a binary gripper open-close state. We found that the joint angle representation was the most effective. In particular, we found that for the grasping part of the task, a policy that controlled the robot in terms of tooltip poses was unreliable.

	Items	Successful Grasp	Successful Recovery	Successful Categorizations
Joint Angle	8	7/8	2/2	7/7
Tooltip Pose	8	3/8	5/5	3/3
Rotation	8	2/8	0/8	0

Interpreting Learned Options: We additionally analyzed the learned options to see if there was an interpretable structure. We examined the collected demonstrations and looked at the segmentation structure. We average over the 60 trials the probability for the high-level policy to choose each option in each of first 10 time-steps during training (Figure 14). We find that the options indeed cluster visited states and segment them in alignment with key phases in the task, such as servoing to the needle, grasping it, dropping it if necessary, and categorizing it into the accept cup.

Full Break Down of Experimental Results: In 10 trials, 7 out of 10 were successful (Table 1.4). The main failure mode was unsuccessful grasping (defined as either no needles or multiple needles). As the piles were cleared and became sparser, the robot's grasping policy became somewhat brittle. The grasp success rate was 66% on 99 attempted grasps. In contrast, we rarely observed failures at

the other aspects of the task. Of the grasps that failed, nearly 34% were due to grasping multiple objects.

Trial #	# of Needles	# Needles Cleared	Grasping	Recovery	Categorization
1	6	**6**	6/9	3/3	6/6
2	8	6	7/12	5/6	6/7
3	7	**7**	7/8	1/1	7/7
4	6	**6**	6/10	4/4	6/6
5	7	6	6/11	5/5	6/6
6	8	**8**	8/13	5/5	8/8
7	6	5	5/9	4/4	5/5
8	7	**7**	7/10	3/3	7/7
9	8	**8**	8/10	2/2	8/8
10	6	**6**	6/7	1/1	6/6
Totals	69	Success: 70%	Success: 66%	Success: 97%	Success: 98%

Generalization to unseen objects: We also evaluated the learned policy on a few unseen objects (but similarly sized and colored) to show that there is some level of generalization in the learning. We tried out four novel objects and evaluated what the learned policy did for each (Figure 13). For each of the novel objects we tried out 10 grasps in random locations and orientations in the bin (without any others). We evaluated the grasp success and whether it was categorized consistently (i.e., does the learned policy consistently think it is a pin or a needle).

We found that the diode was consistently grasped and categorized as a dissection pin. We conjecture this is because of its head and thin metallic wires. On the other hand, the screw and the thread were categorized in the reject cup. For 8/10 of the successful grasps, the nail was categorized as a failure mode. We conjucuture that since it is the large object it looks similar to the two object grasps seen in the demonstrations.

	Grasps	Successful	Drop	Categorize Accept	Categorize Reject
Thread	10	10	1	0	9
Diode	10	10	0	10	0
Nail	10	8	8	0	0
Screw	10	4	0	0	4

1.5 Action Augmentation: Image-based Pushing

We run the same experiment as described in the paper but on the image state-space instead of the low dimensional state. The sensor inputs are 640x480 images of the scene and the task is the Box2D pushing task.

DDCO FF Policy: First, we consider a neural network with a convolutional layer with 64 5x5 filters followed by a fully connected layer forming a feed-forward (FF) network. The high-level model only selects options and is parametrized by the same general architecture with an additional softmax layer after the fully connected layer. This means that the meta-control policy is a two-layer convolutional network whose output is a softmax categorical distribution over options. This policy is implemented in Tensorflow and is trained with an ADAM optimizer with learning rate $1e-5$. We used $k=2$ options for the FF policy.

DDCO RNN Policy: Next, we consider a neural network with a convolutional layer with 64 5x5 filters followed by an LSTM layer forming a recurrent network (RNN). The high-level model only selects options and is parametrized by the same general architecture with an additional softmax layer after the LSTM. This policy is implemented in Tensorflow and is trained with a Momentum optimizer with learning rate $1e-4$ and momentum $4e-3$. We used $k=4$ options for the RNN policy.

We test two different architectures for the output layer of this high-level policy: either a softmax categorical distribution selecting an option, or the hybrid softmax/continuous distribution output described in Section 4.2. Figure 15 describes the empirical estimation, through policy rollouts, of the success rate as a function of the number of options, and of the fraction of time that the high-level policy applies physical control. When the options are too few to provide skills that are useful throughout the state space, physical controls can be selected instead to compensate in states where no option would perform well.

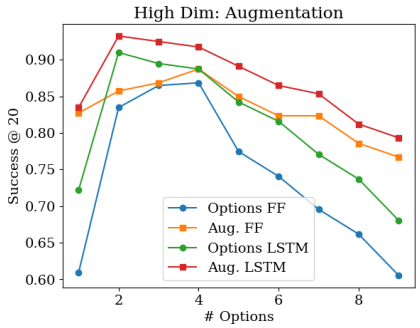

Figure 15: Augmenting the action space instead of selecting only options with the meta-policy leads attenuates the losses when selecting too few or too many options.

Supplement: Additional Experiments on Algorithm Stability

We also ran a number of experiments to evaluate the stability of the solutions found with DDCO. In other words, given different initializations, how consistent are the learned options in terms of quality and structure.

2.6 Instability Modes

When real perceptual data is used, if all of the low-level policies initialize randomly the forward-backward estimates needed for the Expectation-Gradient will be very poorly conditioned where there is an extremely low likelihood assigned to any particular observation. We encountered the following degenerate cases in DDCO:

High-Fitting: When the high-level policies can invoke both actions and options, one degenerate solution is "high fitting"–where the meta-policy disregards all of the options.

Imbalance: If the options are highly expressive, another degenerate case is when the options are imbalanced, i.e., one option dominates over a large portion of the state-space.

Redundancy: Options that essentially perform the same sub-task might be learned.

We find that these concerns can be greatly mitigated with more intelligent initialization and layer-wise training.

2.7 Vector Quantization For Initialization

One challenge with DDCO is initialization. When real perceptual data is used, if all of the low-level policies initialize randomly the forward-backward estimates needed for the Expectation-Gradient will be poorly conditioned where there is an extremely low likelihood assigned to any particular observation. The EG algorithm relies on a segment-cluster-imitate loop, where initial policy guesses are used to segment the data based on which policy best explains the given time-step, then the segments are clustered, and the policies are updated. In a continuous control space, a randomly initialized policy may not explain any of the observed data well. This means the small differences in initialization can lead to large changes in the learned hierarchy.

We found that a necessary pre-processing step was a variant of vector quantization, originally proposed for problems in speech recognition. We first cluster the state observations using a k-means clustering and train k behavioral cloning policies for each of the clusters. We use these k policies as the initialization for the EG iterations. Unlike the random initialization, this means that the initial low level policies will demonstrate some preference for actions in different parts of the state-space. We set k to be the same as the k set for the number of options, and use the same optimization parameters.

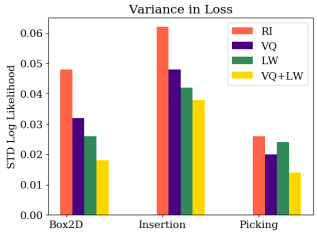

Figure 16: This experiment measures the variance in log likelihood over 10 different runs of DDCO with different training and initialization strategies. Vector Quantization (VQ) and Layer-Wise training (LW) help stabilize the solution.

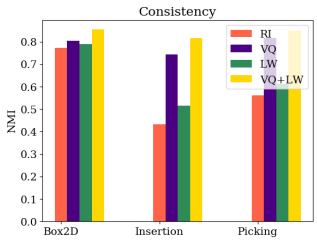

Figure 17: We measure the average normalized mutual information between multiple runs of DDCO. This measures the consistency of the solutions across intializations.

2.8 Layer-wise Hierachy Training

We also found that layer-wise training of the hierarchy greatly reduced the likelihood of a degenerate solution. While, at least in principle, one can train When the meta-policy is very expressive and the options are initialized poorly, sometimes the learned solution can degenerate to excessively using the meta-policy (high-fitting). We can avoid this problem by using a simplified parametrization for the meta-control policy η_d used when discovering the low-level options. For example, we can fix a uniform meta-control policy that chooses each option with probability $1/k$. Now, once these low-level options are discovered, then we can augment the action space with the options and train the meta-policy.

This same algorithm can recursively proceed to deep hierarchies from the lowest level upward: level-d options can invoke already-discovered lower-level options; and are discovered in the context of a simplified level-d meta-control policy, decoupled from higher-level complexity. Perhaps counter-intuitively, this layer-wise training does not sacrifice too much during option discovery, and in fact, initial results seem to indicate that it improves the stability of the algorithm. An informative meta-control policy would serve as a prior on the assignment of demonstration segments to the options that generated them, but with sufficient data this assignment can also be inferred from the low-level model, purely based on the likelihood of each segment to be generated by each option.

2.9 Results: Stability of DDCO

In the first experiment, we measure the variance in log likelihood over 10 different runs of DDCO with different training and initialization strategies (Figure 16). We use the entire datasets presented in the paper and use the FF architecture for the Box2D experiments. Vector Quantization (VQ) and Layer-Wise training (LW) help stabilize the solution and greatly reduce the variance of the algorithm. This reduction in variance is over 50% in the Box2D experiments, and a little more modest for the real data.

In the next experiment, we measure the consistency of the solutions found by DDCO (Figure 17). For each of the demonstration trajectories, we annotate the time-step by the most likely option. One can view this annotation as a hard clustering of the state-action tuples. Then, we measure the average normalized mutual information (NMI) between all pairs of the 10 different runs of DDCO. NMI is a measure of how aligned two clusterings are between 0 and 1, where 1 indicates perfect alignment. As

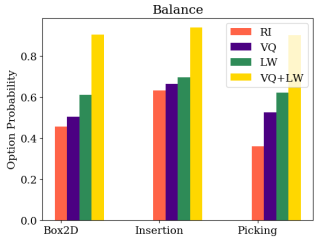

Figure 18: We measure probability that an option is selected by the high-level policy.

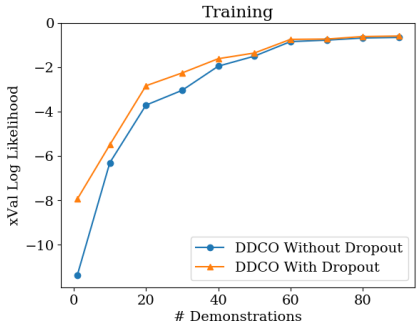

Figure 19: Dropout seems to have a substantial effect on improving cross-validation accuracy on a hold out set.

with the likelihood, Vector Quantization (VQ) and Layer-Wise training (LW) significantly improve the consistency of the algorithm.

In the last experiment, we measure the symptoms of the "high-fitting" problem (Figure 18). We plot the probability that the high-level policy selects an option. In some sense, this measures how much control the high-level policy delegates to the options. Surprisingly, VQ and LW have an impact on this. Hierarchies trained with VQ and LW have a greater reliance on the options.

2.10 Results: Dropout

For the real datasets, we leverage a technique called dropout, which has been widely used in neural network training to prevent overfitting. Dropout is a technique that randomly removes a unit from the network along with all its connections. We found that this technique improved performance when the datasets were small. We set the dropout parameter to 0.5 and measured the performance on a hold out set. We ran an experiment on the needle insertion task dataset, where we measured the cross-validation accuracy on held out data with and without dropout (Figure 19). Dropout seems to have a substantial effect on improving cross-validation accuracy on a hold out set.