Simultaneous On-line Discovery and Improvement of Robotic Skill Options

Freek Stulp[1,2], Laura Herlant[1,3], Antoine Hoarau[1,2], Gennaro Raiola[1,2]
[1]Robotics and Computer Vision, École Nationale Supérieure de Techniques Avancées (ENSTA-ParisTech), France
[2]FLOWERS Research Team, INRIA Bordeaux Sud-Ouest, Talence, France
[3]Robotics Institute, Carnegie Mellon University, Pittsburgh, USA

***Abstract*— The regularity of everyday tasks enables us to reuse existing solutions for task variations. For instance, most door-handles require the same basic skill (reach, grasp, turn, pull), but small adaptations of the basic skill are required to adapt to the variations that exist (e.g. levers vs. knobs). We introduce the algorithm "Simultaneous On-line Discovery and Improvement of Robotic Skills" (SODIRS) that is able to autonomously discover and optimize skill options for such task variations. We formalize the problem in a reinforcement learning context, and use the PI$^{\mathrm{BB}}$ algorithm [2] to continually optimize skills with respect to a cost function. SODIRS discovers new subskills, or "skill options", by clustering the costs of trials, and determining whether perceptual features are able to predict which cluster a trial will belong to. This enables SODIRS to build a decision tree, in which the leaves contain skill options for task variations. We demonstrate SODIRS' performance in simulation, as well as on a Meka humanoid robot performing the ball-in-cup task.**

I. Introduction

In most activities of daily living, variations of related tasks are encountered over and over again. Consider, for instance, opening a door; a task which we perform thousands of times per year. Although the basic actions are the same for most door-handles – reach, grasp, rotate and pull – slight variations are needed for different types of handles (lever or knob [1]), and their properties (stiff or compliant).

In this paper, we present "Simultaneous On-line Discovery and Improvement of Robotic Skills" (SODIRS), an algorithm that is able to autonomously learn skill options for task variations through: *Optimization* – Using policy improvement with covariance matrix adaptation to optimize skill options [2]. *Discovery* – Detecting task variations based on clustering the costs of recent trials. *Organization* – Using decision tree learning on cost clusters to determine which perceptual features are relevant for discerning between task variations, and associating skill options with task variations. Key features of SODIRS are that it uses the same data for optimization, exploration and organization, that it learns on-line, incremental and open-ended, and that the number of task variations and skill options need not be specified in advance.

The rest of this paper is structured as follows. We formalize the problem in Section II, and discuss related work in Section III. We present the SODIRS algorithm implementation in Section IV. In our empirical evaluation in Section V, we demonstrate SODIRS' ability to discover and optimize skill options both in simulation, and on a real robot performing the ball-in-cup task [3], as illustrated in Fig. 1. We summarize key features and limitations of SODIRS in Section VI.

This work was supported by INRIA/ADT (Carroman), DIGITEO (PrActIx), and the Intel Science and Technology Center - Embedded Computing.
Contact: `freek.stulp@ensta-paristech.fr`

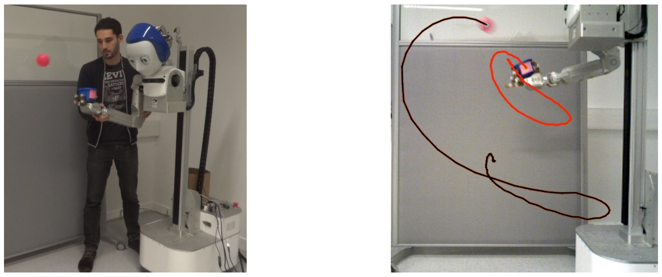
Fig. 1. The ball-in-cup game, where the robot uses momentum generated by its arm motion to pivot the ball at the end of a string into a cup [3]. See also the video attachment. Left: demonstrating the movement to the Meka humanoid robot. Right: A successful trial, showing the tracking of the ball and cup. We use string lengths of 25 and 30cm, which requires the robot to discover and optimize distinct skill options for these task variations.

II. Formalization

We consider episodic reinforcement learning in continuous state/action spaces with parameterized policies $\pi(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of policy parameters. Optimizing policy parameters with respect to a reward/cost function is known as *policy improvement*, or *(direct) policy search*.

If we ignore the costs at individual time steps r_t, and only use the return of an episode $R = \sum_{t=1}^{T} r_t$, policy improvement is equivalent to black-box optimization [2], where the black-box cost function $J\colon \Theta \mapsto \mathbb{R}$ takes $\boldsymbol{\theta}$ as an input, and returns the scalar return of the episode R, as in (1). Each evaluation of J thus corresponds to one episode, or *rollout*. Policy parameters that minimize this cost function are known as optimal policy parameters $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

$$J\colon \Theta \mapsto \mathbb{R} \qquad \text{BBO Cost function } J(\boldsymbol{\theta}) = \textstyle\sum_{t=1}^{T} r_t \tag{1}$$

A. Task Variations and Parameterizable Skills

Different tasks require different cost functions. For example, when turning a door handle, we may measure how far the handle was turned, assigning lowest cost to a full successful turn. But this cost function will be of little use for learning how to press buttons. The same cost function, however, can often be applied to *variations* of the same task. For instance, for both a door knob and lever, the cost function is the same (how far was the knob/lever turned), but it will return different costs for the same action, because fully turning the knob requires different actions from fully turning the lever. We summarize task variations in the task parameter vector $\boldsymbol{q}$. For the door-handles example, the task parameter space would be $\mathbf{Q} = \{\textsc{Lever}, \textsc{Knob}\}$. To reuse the same cost function for variations of a task, we add a task parameter vector $\boldsymbol{q}$ to the cost function, because the cost depends on both the policy *and* task parameters:

$$J\colon \Theta \times \boldsymbol{Q} \mapsto \mathbb{R} \qquad \text{Parameterized cost function } J(\boldsymbol{\theta}, \boldsymbol{q}) \qquad (2)$$

If a cost function depends on task parameters (2), then, to be optimal, different policy parameters are required for different task parameters. Policies therefore *must* take these task parameters into account in order to be optimal. To this end, *parameterized skills* calculate the policy parameters based on the task parameters in a separate step:

Step 1 (before episode)

$$\Pi : \boldsymbol{Q} \mapsto \Theta \qquad \text{Policy parameter function } \Pi(\boldsymbol{\theta}|\boldsymbol{q}) \qquad (3)$$

Step 2 (during episode)

$$\pi : S \times \Theta \mapsto A \qquad \text{Policy } \pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{\theta}) \qquad (4)$$

Now there are two policies at different levels of abstraction. The upper-level *policy parameter function* Π, which computes the appropriate policy parameter vector for a given task parameter vector *before* an episode, and the lower-level *policy* π, which computes the appropriate actions *during* the episode (terminology taken from [4], [5]).

In the most general case, the policy parameter function Π maps to the continuous policy parameter space [4]–[6], i.e. its output is $\boldsymbol{\theta} \in \Theta$. Alternatively, Π maps only to a discrete number of possible policy parameter vectors, i.e. $\boldsymbol{\theta}_o \in \{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \ldots, \boldsymbol{\theta}^O\}$ for O possible *options*, as defined by Daniel et al. [7]. SODIRS addresses the problem of autonomously discovering and optimizing discrete skill options $\boldsymbol{\theta}_{o=1:O}$ for an unknown number of discrete task variations.

B. Task Parameters and Features

So far, we have assumed that the task parameter space $\boldsymbol{Q}$ is provided to the robot, as is the case in [4]–[6]. However, we want the robot to autonomously distinguish between different tasks. To do so, we extract a set of features from the observations of the robot, and the policy parameter function determines the appropriate policy parameters given only these features:

$$\Pi : F \mapsto \Theta \qquad \text{Policy parameter function } \Pi(\boldsymbol{\theta}|\boldsymbol{f}) \qquad (5)$$

The features may be an estimation of the state or task parameters itself, but they need not be. Our aim here is to be general, and allow the algorithm to determine which features are relevant.

Because SODIRS uses perceptual features $\boldsymbol{f}$ to distinguish between task variations, it is not necessary to explicitly represent $\boldsymbol{q}$. For instance, we vary the length of the string in the ball-in-cup task, but do not directly provide this information to the robot. Furthermore, the exact same cost function is used for all task variations. The output of that cost function clearly depends on whether a short or long string is used, but this difference simply arises from interaction with the world. So although task parameters $\boldsymbol{q}$ help to formalize the problem, it is not explicitly represented in the cost function or by the robot, and need not be provided by the user.

C. Problem Statement

With this formalization, SODIRS addresses the questions:

- *How do we determine the optimal policy parameters $\boldsymbol{\theta}^* = \arg\min J(\boldsymbol{\theta})$, given the cost function J?* We implement this with the PI$^{\text{BB}}$ algorithm, an evolution strategy with covariance matrix adaptation (Section IV-A).
- *When should a novel skill option $\boldsymbol{\theta}_{O+1}$ be added to the already available set of options $\{\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^O\}$?* We implement this decision by clustering in cost space (with DBSCAN [11]). Cost clusters imply task variations, which require different skill options (Section IV-B).
- *How do we determine which skill option should be used for which task variation?* To do so, we incrementally learn decision trees, which implement the policy parameter function $\Pi(\boldsymbol{\theta}_o|\boldsymbol{f})$. The current task variation is thus derived only from the features, and task variations and skill options are organized hierarchically in a tree (Section IV-C).

III. Related Work

SODIRS determines when different task contexts require different behaviors (the options $\boldsymbol{\theta}_o$), and also which features $\mathbf{f}$ are relevant to making this distinction. This is similar to *perceptual de-aliasing*, which was considered in the context of reinforcement learning by Chrisman [13]. Rather than extending the state of a discrete MDP to counter partial observability, SODIRS selects immediately observable features that are relevant to distinguishing between task variations.

In "Reinforcement Learning of Visual Classes (RLVC)" [14], similar ideas to those in [13] are applied to much more challenging input spaces, consisting of raw input images. Here, the temporal-difference error is used to decide when different visual classes require different actions. RLVC operates on discrete action spaces and "is unlikely to scale up to interesting robotic tasks without support from simulation" [14]. SODIRS and RLVC are thus complementary, in that SODIRS considers continuous action spaces and on-line performance on physical robot systems, whereas RLVC uses more challenging feature spaces.

Daniel et al. [7] use Hierarchical Relative Entropy Policy Search (HiREPS), very similar to PI$^{\text{BB}}$, with hierarchical policy representations. Rather than building trees starting from a single initial skill, HiREPS chooses between a discrete number of conditional distributions over policy parameter vectors. This number is initially chosen to be larger than the number of possible optima, and redundant options are deleted as the algorithm runs, with care taken to avoid premature deletion. Calinon et al. [15] uses a Gaussian Mixture Model to estimate the structure of the cost function J, mapping the cost as a probability density function that depends on the policy parameters. Exploration noise is decayed manually, rather than automating it with covariance matrix adaptation. The key difference between the algorithms in [7], [15] and SODIRS is that the former consider multiple skill options that are able to solve the *same* task, whereas SODIRS learns multiple options for *multiple* task variations, without requiring the user to specify the task parameter space.

IV. The SODIRS Algorithm

We now describe the implementation of skill optimization, cost clustering and decision tree learning in SODIRS.

A. Skill Optimization through Policy Improvement with PI$^{\mathrm{BB}}$

The optimization algorithm we use is PI$^{\mathrm{BB}}$, short for "Policy Improvement with Black-Box optimization" [2]. The PI$^{\mathrm{BB}}$ algorithm is explained and visualized in Fig. 2, where the cost function is simply $J(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||$, i.e. the distance to the origin. The algorithm alternates between an exploration phase and a parameter update phase.

1) Exploration: K policy parameter vectors $\boldsymbol{\theta}_{k=1\dots K}$ are sampled from a normal distribution with mean $\boldsymbol{\theta}_\mu$ and covariance matrix $\boldsymbol{\Sigma}$ (6). The cost function J is then evaluated for each of these vectors; in policy improvement this corresponds to executing the policy, and computing the return by summing over the costs at each time step (7). The K resulting rollouts are together called an *epoch*.

$$\boldsymbol{\theta}_{k=1\dots K} \sim \mathcal{N}(\boldsymbol{\theta}_\mu, \boldsymbol{\Sigma}) \qquad \text{sample} \qquad (6)$$

$$\forall k \;\; J_k = J(\boldsymbol{\theta}_k) \qquad \text{cost function} \qquad (7)$$

$$\forall k \;\; P_k = e^{\left(\frac{-h(J_k - \min(\mathbf{J}))}{\max(\mathbf{J}) - \min(\mathbf{J})}\right)} \qquad \text{cost-to-weight} \qquad (8)$$

("lower cost $\Rightarrow$ higher weight")

$$\boldsymbol{\theta}_\mu^{new} = \sum_{k=1}^{K} P_k \boldsymbol{\theta}_k \qquad \text{weighted averaging} \qquad (9)$$

$$\boldsymbol{\Sigma}^{new} = \sum_{k=1}^{K} P_k (\boldsymbol{\theta}_k - \boldsymbol{\theta}_\mu)(\boldsymbol{\theta}_k - \boldsymbol{\theta}_\mu)^\intercal \qquad \text{weighted averaging} \qquad (10)$$

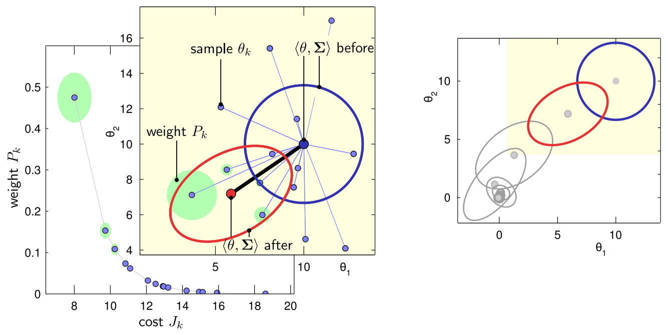

Fig. 2. Visualization of PI$^{\mathrm{BB}}$. Left: distributions in parameter space before (blue) and after (red) updating. The image in the background shows the mapping from costs J_k to weights P_k. Right: Iterative updating towards the minimum $\boldsymbol{\theta}^*$, which lies at the origin. The covariance matrix shrinks, once the mean of the distribution ($\boldsymbol{\theta}_\mu$) is at the optimum ($\boldsymbol{\theta}^*$).

2) Parameter Update: Given the scalar cost of each rollout, the costs are then converted into weights with an exponential mapping, which assigns higher weights to samples with lower costs (this function is visualized in the lower left corner of Fig. 2). Then, the mean is computed by taking the weighted average over the samples (9). Because of the mapping from cost to weights, low-cost samples contribute more to the new mean then high-cost samples, and the mean $\boldsymbol{\theta}_\mu$ (on average) moves closer to $\boldsymbol{\theta}^*$. The same weighting is used for the covariance matrix update (10). Fig. 2 (left) visualizes such an update of the distribution. Updating the distribution is iterated until the costs converge, or a fixed number of iterations has been completed, as visualized in Fig. 2 (right).

3) Covariance Matrix Update: Covariance matrix adaptation automatically adapts the exploration so as to generate more samples in the direction of the minimum. One important property is that the covariance matrix, and thus exploration, goes towards zero if the current mean of the distribution is at the minimum, i.e. $\boldsymbol{\theta}_\mu \approx \boldsymbol{\theta}^*$. This adaptive exploration can be observed in the right graph in Fig. 2.

4) Illustrative example: Before turning to task/skill hierarchies, we illustrate the solution idea with a simple problem. The left plot in Fig. 3 illustrates the task: passing through a via-point at a specific time. The trajectory is generated by a Dynamical Movement Primitive (DMP) [10], which is parameterized by the policy parameters $\boldsymbol{\theta}$. In this example, $\boldsymbol{\theta}$ is only 2-D, for ease of visualization. Variation in the policy parameters leads to variations in the trajectory, as depicted in Fig. 3 (left). The DMP starts at $y_0 = 0$, and converges towards the goal $g = 1$. The aim is to pass through the viapoint $y^v = 0.75$ at the time step 20. The cost function penalizes the distance to the viapoint at $t = 20$, as well as the acceleration at each time step: $J = \delta(t-20) \cdot ||y_t - y^v|| + \sum_{t=1}^{T} 10^{-5} \ddot{y}_t{}^2$. The top row in Fig. 3 summarizes the result of optimizing the policy parameters with covariance matrix adaptation for $y^v = 0.75$.

B. Skill Option Discovery through Cost Clustering

The optimization in the top row of Fig. 3 is done on one task only, i.e. for the viapoint at $y^v = 0.75$. If we now provide tasks with viapoints selected equiprobably from $y^v = \{0.25, 0.75, 0.80\}$, we get learning as in Fig. 3. Here, we see that the distribution $\langle \boldsymbol{\theta}_\mu, \boldsymbol{\Sigma} \rangle$ still converges, i.e. the cost is stable, and the covariance matrix has shrunk to the solution for $y^v = 0.75$. In the learning curve however, we see that the average costs over all $K = 20$ rollouts in an epoch are still high, because this solution only generates low costs when $y^v = 0.75$. In fact, in the individual samples, we now clearly see three clusters, one for each task variation/viapoint.

Given the last L rollouts, SODIRS attempts to find such clusters, potentially leading to "low cost" and "high cost" clusters. If clustering succeeds, SODIRS then attempts to find a classifier that correctly assigns each of the L rollouts to the two clusters. The classification is based on a decision boundary in one of the perceptual features. The key idea is that *each cost cluster corresponds to one task variation*, each requiring a different skill option. And if a certain feature enables us to distinguish between clusters in past rollouts, it will also enable us to *distinguish between task variations in future rollouts*, allowing us *to apply the appropriate skill option* to future tasks.

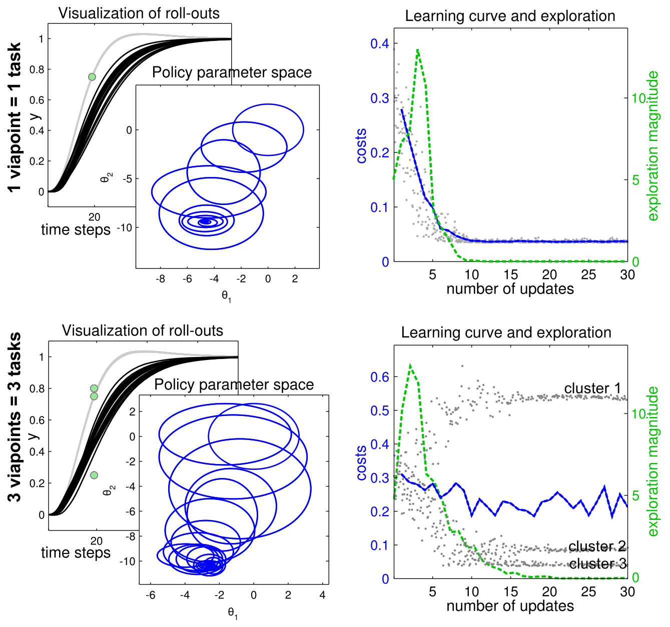

Fig. 3. Learning progress for 1 viapoint ($y^v = 0.75$, top) or 3 viapoints ($y^v = \{0.25, 0.75, 0.80\}$, bottom) Left: Samples trajectories in an epoch before (black) and after (gray) optimization, as well as the search in parameter space, visualized by error ellipses representing the distributions $\langle \boldsymbol{\theta}_\mu, \boldsymbol{\Sigma} \rangle$. Right: Learning curve, depicting costs of individual samples (gray dots) and their average per epoch (thick line). The dashed line represents the exploration magnitude as learning progresses. Exploration magnitude is the largest eigenvalue of $\boldsymbol{\Sigma}$. It converges towards 0 once the optimum (or one of the optima) is found. For the 3 viapoints (bottom row), low costs are achieved for $y^v = 0.75$, but this solution leads to higher costs for $y^v = \{0.25, 0.80\}$ (the three clusters).

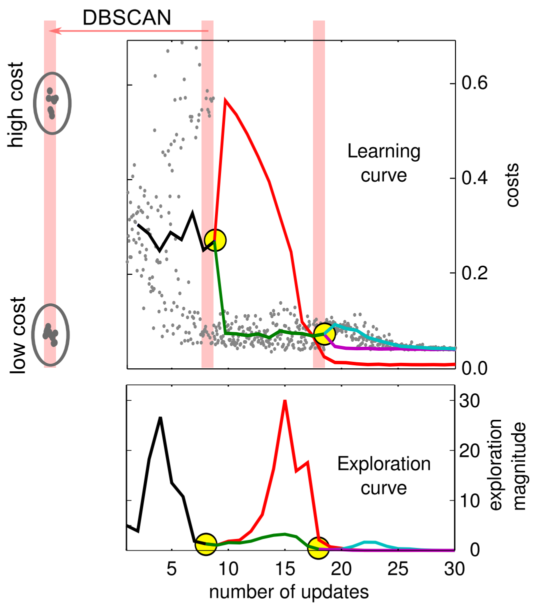

Fig. 4. Optimization of the viapoint task with variations $y^v = \{0.25, 0.75, 0.80\}$ with SODIRS, which discovers new skill options (at updates 8 and 18) and optimizes them individually. Top: costs of each individual rollout (dots) and average costs per epoch (lines) as learning progresses, with a visualization of cost clustering with DBSCAN. Bottom: exploration magnitude as learning progresses. Different colors highlight different skill options, which are optimized in parallel.

Clustering itself is done with the clustering algorithm "density-based spatial clustering of applications with noise (DBSCAN)" [11] on the costs of the last L rollouts. L is usually chosen to be higher than K (the number of rollouts in an epoch, as used for one update in PI$^{\text{BB}}$). Advantages of DBSCAN over, for instance, k-means is that the number of clusters need not be specified in advance, it deals well with noisy data, and is able to find non-linearly separable clusters. DBSCAN does suffer from the curse of dimensionality, but this is not an issue for our application, as we consider only the cost space, which is 1D. DBSCAN takes one parameter, which specifies the minimum number of points P required to form a cluster.

DBSCAN converts the continuous cost space into a set of clusters, each representing a different class (e.g. 'high cost', 'low cost'). In our viapoint example, DBSCAN determines that two clusters have arisen at update 8, based on the costs of the $L = 20$ rollouts preceding this update. In Fig. 4, these 20 rollouts are inside by the pink band in the top graph. The clusters that DBSCAN detects (with $P = 0.4L$) are to the left of this graph.

C. *Skill Option Hierarchies through Decision Tree Learning*

The next step is to determine which perceptual features are able to predict to which class the cost will belong (e.g. "high cost" or "low cost"). This is implemented by a decision stump [12], which is a decision tree of depth 1. In the viapoint example, the decision stump that best predicts which cluster will arise is if $y^v < 0.5$ then the cost will be in the low cluster, else in the high one (Here, y^v is provided as a feature $y^v \in \mathbf{f}$. Section V contains more interesting examples).

Each cost cluster corresponds to a different discrete task. Each discrete task requires a skill option. Successful cost clustering and decision stump learning implies that the current skill used to solve the task must be split into multiple skill options, one for each cost cluster.

Therefore, the final step is to add subskills, e.g. 'Option 1' which is executed with parameters $\boldsymbol{\theta}_1$ when $y^v < 0.5$, and the 'Option 2' ($\boldsymbol{\theta}_2$) when $y^v \geq 0.5$. The policy parameters of the subskills are initialized to the policy parameters of their superskill. During optimization, tasks for which $y^v < 0.5$ are now used to optimize *only* 'Option 1'. These two skill options are now optimized individually, as is visualized in Fig. 4. These also highlight that another split occurs at update 18, for the task pair $y^v = \{0.75, 0.80\}$, and we thus have three skill options corresponding to the three task variations, and the policy parameter function is as follows:

$$\Pi(\boldsymbol{\theta}_o|\boldsymbol{q}) = \begin{cases} \text{if } y^v \leq 0.5: & \langle \boldsymbol{\theta}_\mu^1, \boldsymbol{\Sigma}^1 \rangle \\ \text{if } y^v > 0.5 & \begin{cases} \text{if } y^v \leq 0.775: & \langle \boldsymbol{\theta}_\mu^{2.1}, \boldsymbol{\Sigma}^{2.1} \rangle \\ \text{if } y^v > 0.775: & \langle \boldsymbol{\theta}_\mu^{2.2}, \boldsymbol{\Sigma}^{2.2} \rangle \end{cases} \end{cases}$$

With this tree of skill options, three parallel optimizations are running, with one option for each task variation. When SODIRS encounters a task for which for instance $y^v = 0.75$, it will take a sample from $\mathcal{N}(\boldsymbol{\theta}_\mu^{2.1}, \boldsymbol{\Sigma}^{2.1})$, and, if K samples have been evaluated, update only $\boldsymbol{\theta}_\mu^{2.1}$ and $\boldsymbol{\Sigma}^{2.1}$, not the parameters of the skill options in the other leaves.

Thus, execution and updating are specific to a single option, corresponding to one leaf in the tree.

V. Experimental Evaluation

A. Petanque (simulation)

In this task, a 7-DOF articulated arm, simulated in Simulation Lab, learns to throw objects towards a goal position, as depicted in Fig. 5. Three objects of the same weight, but with different shapes are thrown. **Task parameters.** The different shapes lead to different friction coefficients (set manually in simulation), which requires different throws for the three objects, if they are to reach the goal location. **Cost function.** The cost function is the squared distance to the goal location when the object hits the ground.

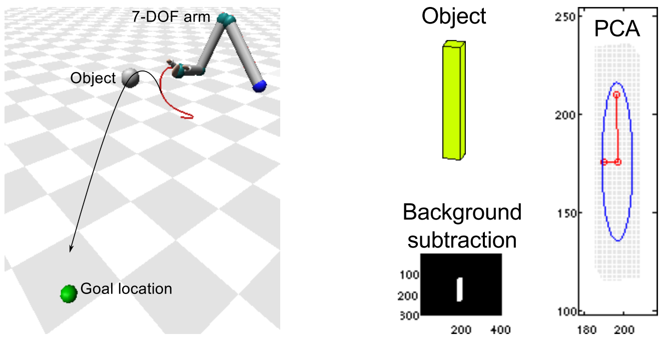

Fig. 5. The petanque task.

Features. A snapshot of the object to be thrown is rendered. After background subtraction, the 2D eigenvectors and eigenvalues of the locations of the pixels belonging to the object, the number of pixels, and the average hue of the pixels form the 6D feature vector. This is a highly idealized, noise-free feature extraction process. Our focus in this experiment is not on feature extraction itself (which may be tailored to the task at hand), but rather on demonstrating how SODIRS is able to extract relevant features from a larger set. **Algorithm parameters.** The movement is represented by a 7-D DMP with 5 basis functions per dimension, which generates desired joint trajectories. For PI^{BB}, the parameters $h = 10$, the number of rollouts per epoch is $K = 15$, the window size for cost clustering is $L = 30$, and DBSCAN's parameter is set as $P = 0.25L$. **Evaluation of Covariance Matrix Updating.** With the parameter settings above, we ran four experiments: (1) PI^{BB} with Covariance Matrix Adaptation (CMA), with an initial exploration magnitude $\lambda\mathbf{I}$, with $\lambda = 25$. (2-4) PI^{BB} with constant exploration, and $\lambda = \{25, 5, 1\}$.

Results. Fig. 6 visualizes the results for one of the learning sessions with covariance matrix adaptation. It visualizes the landing locations of the objects (variance in locations is indirectly related to the exploration magnitude in policy parameter space), the two splits that are made, and the convergence of the three skills towards the minimum.

Table I shows the average ± standard deviation over 5 learning sessions for each of the four experiments. For CMA, we see that the first/second split occurs on average after 90/267 rollouts. After on average 162 rollouts, the costs of

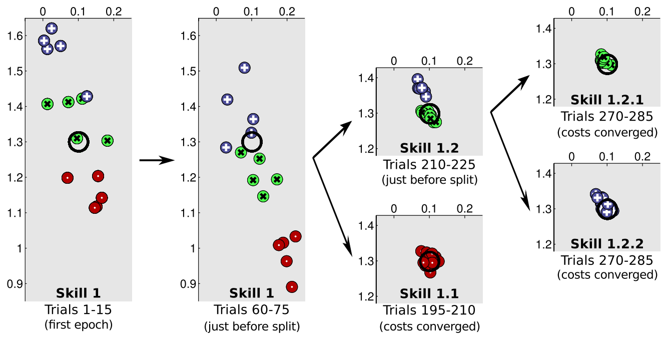

Fig. 6. Example learning session for the petanque task. The 2D landing positions of the objects on the floor at different phases of learning. 15 rollouts within one epoch are shown. The goal position is indicated by the black circle. Note that the actual search space is in the 35=7·5 dimensional policy parameter space; these different 2D landing positions arise from variations in the end-effector trajectories due to the search in this 35D space.

all three skill options has dropped below 10. For constant exploration with an exploration magnitude of $\lambda = 25$ or 5, the first split is after 71/100 rollouts, but the second split does not occur. This is because the variance in the cost due to exploration is higher than that caused by the different tasks; therefore, cost clusters do not arise. With an exploration magnitude of $\lambda = 1$ both splits *do* occur, but very late (after 245/323 rollouts respectively). This is because this low exploration converges only very slowly toward the minimum. After 600 rollouts (when the learning sessions were stopped), not all of the skill options have a cost lower than 10.

Summary. SODIRS consistently learns different skill options for task variations. High exploration is required for fast convergence, and low exploration is required for cost clustering and skill option discovery. Covariance matrix adaptation is the only method that can switch between these two different requirements.

	CMA	Constant		
Exploration→	λ^{init}=25	λ=25	λ=5	λ=1
First split	90 ±30	71±8	100±46	245±77
Second split	267±61	X	X	323±95
Avg. Cost < 10	162±29	168±22	225±20	X

TABLE I
Petanque task results (values represent number of rollouts)

B. Ball-in-cup (Meka humanoid)

In this experiment, the Meka humanoid robot learns to perform the ball-in-cup game, using momentum generated by its arm motion to lift the ball at the end of a string into a cup [3], as illustrated in Fig. 1. The position of the ball and cup are tracked with a Kinect, mounted from the side, perpendicular to the floor. **Task parameters.** The task parameter q is the length of the string, which is either 25 or 30cm (the demonstration is provided only for the 25cm string). **Cost function.** The cost of a rollout is 0 if the ball lands in the cup, and 10 when it hits the cup. Otherwise, it is the horizontal distance of the ball to the cup, when the ball is moving downward and at the same height as the cup. This distance is specified in pixels (typically between 30-120). Note that due to tracking inaccuracies and dynamics

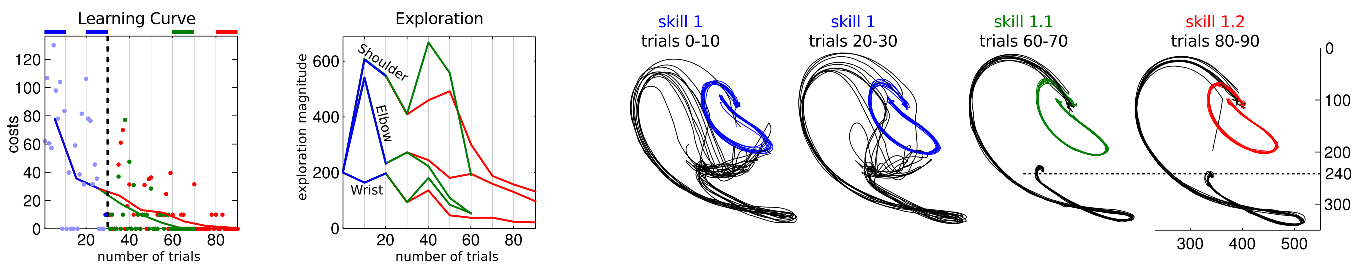

Fig. 7. Left: Learning curves, showing the costs of all rollouts (dots) and their average per epoch (line). The split (dashed line) occurs after the 3rd update, i.e. after 30 rollouts. Center: Magnitude of exploration for three joints as learning progresses. Right: The 10 rollouts in an epoch at various stages of learning (epochs shown are highlighted by horizontal bars above the learning curve)

not captured in the demonstration, merely reproducing the demonstration does not lead to successful task execution, i.e. we have costs of 87 ± 4 and 107 ± 2 (over 5 rollouts for the two string lengths 25/30cm). **Features.** The feature vector $\mathbf{f}$ contains the 2D position of the ball and hand (in pixels) before the movement starts. These positions could have been transformed into a robot-centric Cartesian frame of reference, but we have deliberately used the raw pixel values to demonstrate that the algorithm is agnostic about the semantics of the features. **DMP representation.** The movement is represented as a 3D DMP, which generates desired joint trajectories for the shoulder, elbow and wrist joints. The other joints are fixed, which leads to an end-effector trajectory approximately in a 2D plane. The DMP parameters $\boldsymbol{\theta}$ are initialized through locally weighted regression (10 basis functions per dimension) on a demonstrated movement [10], as shown in Fig. 1 (left). **Algorithm parameters.** PI$^{\text{BB}}$ is parameterized as in Section V-A, except for the window size for cost clustering ($L = 20$), and the number of rollouts in an epoch ($K = 10$, 5 with the short string, and 5 with the long string). For exploration, the covariance matrix for each joint is initially set to $\Sigma = 200 \cdot \mathbf{I}$.

Results. Fig. 7 summarizes the results. During the first 30 rollouts, there is one skill ('skill 1'), whose cost drops over time after each update. Clustering the costs is possible after 37 updates. The feature that best predicts the two cost clusters is the height of the ball, which corresponds to our intuition, since the initial height before movement is higher if the string is shorter. The decision boundary for this feature is 240 pixels, as visualized in the right two graphs in Fig. 7. After the split, the exploration in the three joints mostly decreases, enabling a 'perfect' epoch in which the robot is successful in every trial for skill 1.1 in rollouts 60-70, and a 'near perfect' epoch for skill 1.2 in rollouts 80-90.

VI. Conclusion

SODIRS autonomously discovers, organizes, and determines the applicability of skill options for task variations. The key idea is to cluster the costs of recent rollouts, and to determine decision boundaries on perceptual features that predict which cluster a rollout belongs to. This leads to a hierarchical structure in which skill options are applied to and optimized separately for distinct task variations. Covariance matrix adaptation is a key ingredient for SODIRS to work, because it automatically switches between high exploration (for quick convergence) and low exploration (necessary for predictable cost clusters to arise).

The main open parameters of the algorithm are the number of rollouts in an epoch K and the window for cost clustering L. The main limitation of the current implementation is the decision stump, which we expect may fail when faced with high-dimensional, redundant feature spaces, as in [14]. We are currently replacing the decision stump with more robust classification algorithms.

References

[1] Canada's war on doorknobs. The Economist, Apr 19th, 2014.

[2] F. Stulp and O. Sigaud, "Robot skill learning: From reinforcement learning to evolution strategies," *Paladyn. Journal of Behavioral Robotics*, vol. 4, no. 1, pp. 49–61, September 2013.

[3] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[4] B. da Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," in *Proc. of the 29th Int'l Conf. on Machine Learning*, 2012.

[5] A. Kupcsik, M. Deisenroth, J. Peters, and G. Neumann, "Data-efficient generalization of robot skills with contextual policy search," in *AAAI*, 2013.

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

[7] C. Daniel, G. Neumann, and J. Peters, "Learning concurrent motor skills in versatile solution spaces," in *Proceedings of the International Conference on Robot Systems (IROS)*, 2012.

[8] F. Stulp, "Adaptive exploration for continual reinforcement learning," in *International Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[9] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[10] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal, "Dynamical Movement Primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, 2013.

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Int'l Conference on Knowledge Discovery and Data Mining (KDD)*, 1996, pp. 226–231.

[12] W. I. Ai and P. Langley, "Induction of one-level decision trees," in *Proceedings of the Ninth International Conference on Machine Learning*. Morgan Kaufmann, 1992, pp. 233–240.

[13] L. Chrisman, "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.

[14] J. Piater, S. Jodogne, R. Detry, D. Kraft, N. Krüger, O. Kroemer, and J. Peters, "Learning visual representations for perception-action systems," *Int. J. Rob. Res.*, vol. 30, no. 3, pp. 294–307, Mar. 2011.

[15] S. Calinon, P. Kormushev, and D. G. Caldwell, "Compliant skills acquisition and multi-optima policy search with em-based reinforcement learning," *Robotics and Autonomous Systems*, vol. 61, no. 4, 2013.