Dual REPS: A Generalization of Relative Entropy Policy Search Exploiting Bad Experiences

Adrià Colomé, and Carme Torras

***Abstract*—Policy Search (PS) algorithms are nowadays widely used for their simplicity and effectiveness in finding solutions for robotic problems. However, most current PS algorithms derive policies by statistically fitting the data from the best experiments only. This means that those experiments yielding a poor performance are usually discarded or given too little influence on the policy update. In this paper, we propose a generalization of the Relative Entropy Policy Search (REPS) algorithm that takes bad experiences into consideration when computing a policy. The proposed approach, named Dual REPS (DREPS) following the philosophical interpretation of the duality between good and bad, finds clusters of experimental data yielding a poor behavior and adds them to the optimization problem as a repulsive constraint. Thus, considering there is a duality between good and bad data samples, both are taken into account in the stochastic search for a policy. Additionally, a cluster with the best samples may be included as an attractor to enforce faster convergence to a single optimal solution in multi-modal problems. We first tested our proposed approach in a simulated Reinforcement Learning (RL) setting and found that DREPS considerably speeds up the learning process, especially during the early optimization steps and in cases where other approaches get trapped in between several alternative maxima. Further experiments in which a real robot had to learn a task with a multi-modal reward function confirm the advantages of our proposed approach with respect to REPS.**

***Index Terms*—Reinforcement Learning, Policy Search, Relative Entropy Policy Search (REPS), Low-reward data reuse.**

I. Introduction

The goal of Reinforcement Learning (RL) algorithms [1] is to find a policy π that, in each situation, provides the best action to execute so as to maximize reward. For this purpose, RL algorithms iteratively explore a search space through a series of experiments to progressively improve the policy. While approaches as deep learning have been gaining popularity in the last years, RL is still better suited to robotic applications due to the limited amount of data required [2]. Some forms of deep reinforcement learning are currently being explored, but so far their application still requires many data samples [3].

Policy Search (PS) [4], [5] is a variant of RL very suitable for the cases where the policy can be parameterized, a common situation in robot motion applications [6]. The search for a good policy is therefore performed directly in the parameter space characterizing the policy. Finding the best policy becomes a stochastic optimization problem. PS methods can be gradient-based, as in the case of Policy Improvement with Path Integrals [7]–[9], or gradient-free, as in Relative Entropy Policy Search (REPS) [10], [11]. Both types of PS (and others in literature [6]) try to optimize the policy parameters $\boldsymbol{\theta}$, so that an expected reward $\mathbf{J}(\boldsymbol{\theta})$ is maximal. After each trajectory reproduction, namely rollout, the reward/cost function is evaluated and used to search for a set of parameters that improves the performance over the initial movement.

However, these ideas have resulted in algorithms that require several rollouts to find a proper policy update. In addition, most PS algorithms compute an associated weight for each trajectory reward, to later obtain a new policy given the parameters and weights for each rollout. This ends up being an elitist way of updating the policy, which just takes good rollouts into consideration, usually discarding experiments with bad outcomes.

Along this work, we present a generalization of REPS, called Dual REPS (DREPS), where we take into account the duality of good and bad experiences in our proposed policy update. To this end, we will be using REPS as a reference PS algorithm. Formally, REPS [10], [11] finds the policy π^* that maximizes the expected reward for a given task. The REPS algorithm uses Kullback-Liebler (KL) divergence [12], which is a non-symmetric indicator of the difference between two probability distributions p, q over a random variable x:

$$\mathrm{KL}(p\|q) = \int p(x)\log\frac{p(x)}{q(x)}dx. \tag{1}$$

In this work, we will represent the policies by probability distributions over a set of parameters. A policy $\pi(\boldsymbol{\theta})$ can then be represented by a normal distribution with mean $\boldsymbol{\mu}_\omega$ and covariance $\boldsymbol{\Sigma}_\omega$, generating samples $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$. Given the previous policy $q(\boldsymbol{\theta})$, REPS obtains the new policy $\pi(\boldsymbol{\theta})$ by adding a KL-divergence bound ϵ between the newly obtained policy and the previous one to the optimization of the expected reward. The bound on the KL-divergence limits the variation on the new policy and prevents the PS algorithm from being too greedy. Such too greedy algorithms may cause severe problems in some robotics applications, where a drastic change in the policy could result in an unpredictable, dangerous behavior of the robot. The new policy π^* is then computed as the solution of:

$$\begin{gathered}\pi^* = \mathrm{argmax}_\pi \int \pi(\boldsymbol{\theta})\mathbf{R}(\boldsymbol{\theta})d\boldsymbol{\theta} \\ \text{s.t. } \epsilon \geq \mathrm{KL}(\pi(\boldsymbol{\theta})\|q(\boldsymbol{\theta})) \\ 1 = \int \pi(\boldsymbol{\theta})d\boldsymbol{\theta}\end{gathered} \tag{2}$$

where $\boldsymbol{\theta}$ are the parameters, $\mathbf{R}(\boldsymbol{\theta})$ is their associated reward, and $\pi(\boldsymbol{\theta})$ is a probability distribution over the parameter space.

Solving the constrained optimization problem (2) provides a solution of the form [10]

$$\pi^* \propto q(\boldsymbol{\theta})\exp(-\mathbf{R}(\boldsymbol{\theta})/\eta), \tag{3}$$

where η is the Lagrange multiplier for the KL bound. Given the value of η and the rewards, the exponential term in (3) acts as a weight to be used with the samples $\boldsymbol{\theta}_k$ in order to obtain the new policy, usually with a Gaussian Weighted Maximum Likelihood Estimation (WMLE). However, it has been shown [13] that the ordering of the KL arguments used in REPS - $\mathrm{KL}(\pi(\boldsymbol{\theta})\|q(\boldsymbol{\theta}))$ instead of $\mathrm{KL}(q(\boldsymbol{\theta})\|\pi(\boldsymbol{\theta}))$ - has an averaging-between-solutions behavior, sometimes producing non-optimal solutions due to competition between two or more close local optima. Such KL ordering generates a solution with a high probability where data presents a high reward, therefore averaging between modes in Gaussian distributions. Using the reverse ordering would help to find a single solution faster, as such approach would find a solution with low probability values where data has low reward [13], therefore focusing on only one mode. However, there is no closed REPS solution using $\mathrm{KL}(q\|\pi)$ instead of $\mathrm{KL}(\pi\|q)$. While the former KL ordering seems more appropriate for most RL applications, its analytic unsolvability makes its application impractical. Along this paper we will show that our proposed DREPS algorithm avoids the aforementioned averaging-between-solutions behavior, exhibiting a good ability to escape from getting trapped in between local maxima.

For our proposed DREPS algorithm to fit in the current trends of RL, we assume that policies are encoded as multivariate normal random distributions. Such encoding, as well as representing the clustered samples by Gaussian distributions, allows us to solve the resulting equations analytically and obtain a closed-form solution.

In the following section, we will detail how to build a clustered data structure for DREPS, followed by the algorithm's derivation, which is

The authors are with the Inst. de Robòtica i Inf. Ind., CSIC-UPC, Barcelona, Spain. e-mail: [acolome@iri.upc.edu, torras@iri.upc.edu].

This work is partially funded by Spanish project RoboInstruct (TIN2014-58178-R) and CSIC project MANIPlus (201350E102). Adrià Colomé is also supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant (AP2010-1989).

done similarly to REPS and other information-theoretic Policy Search approaches [14].

II. The Dual REPS Algorithm

The idea behind Dual REPS (DREPS) is to use clustered badly-performing data samples as a repulsive field and good-performing ones as an attractor within the policy search algorithm. For this purpose, we assume that we can cluster the data from a set of experiments and encode such information as a constraint in the optimization problem. The natural way of adding this new constraint has been to set a minimum/maximum KL-divergence between the bad/good data clusters and the new policy. This will act as a repulsive/attractive field from these clusters in the policy update.

In this section, we will first explain how to select and fit these data clusters, which will be labeled *bad/good data*, to later present the whole mathematical derivation of the proposed DREPS algorithm. The clusterization presented in this paper is an attempt at finding a reliable way of clustering parameter data with aggregated rewards which could be used in the DREPS algorithm. While such proposed clustering has proved effective, we note that it is just a tool to obtain the input needed for the DREPS algorithm itself. The clusters are fitted with Gaussian distributions that are then used for the policy search update. Throughout this section, we will use C_{low} as the set of low-performing data clusters, with its cardinal represented as $|C_{low}|$, and C_{high} as the set of $|C_{high}|$ high-performing data clusters. Additionally, for simplicity of notation we will use $C = C_{low} \cup C_{high}$ with cardinal $c = |C|$.

A. Clustering

The low-performance and high-performance samples may be spread out in the parameter space. Therefore, we have opted for grouping the samples with a bad/good reward in several clusters, so as to represent them by Gaussians to be included in the policy update. In such clustering, we assume the reward function is smooth almost everywhere in a mathematical measure theory sense, as well as a good repeatability of the reward value wrt. variability in the policy parameters. If the repeatability is low, i.e., the reward obtained in several runs with the same parameters differs significantly, such noise might be transferred to the clustering algorithm.

In order to obtain such clusters, we decided to use *K-means* clustering, considering both the sample vectors and the rewards generated by them. Hence, we append a transformed reward $f(r_k)$, with $r_k \triangleq \mathbf{R}(\boldsymbol{\theta}_k)$, to every sample $\boldsymbol{\theta}_k \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$, $k = 1, ..., N$, N being the number of rollouts per policy update. Then, we use the vectors $\left[\boldsymbol{\theta}_k^T - \boldsymbol{\mu}_\omega^T, f(r_k)\right]^T$ as input to a *K-means* clustering with a given fixed number of clusters. The usage of such $f(r_k)$ is to properly scale the relative importance between rewards and parameters in the clustering. Otherwise, either one or the other could have a too-large influence, as seen in Fig. 1. We used the *K-means* clustering algorithm in MATLAB. However, such implementation [15] initializes the cluster centers by randomly selecting a number of points from the dataset. This resulted in a non-deterministic way of clustering points, so the clusters are initialized using the algorithm proposed in [18] instead.

We perform two independent clustering processes on the same data, using two different reward transformations, to obtain the low- and high-performance clusters. The reason for these two transformations is to increase the importance of the reward dimension in the clustering (amplifying either the low or high values), while maintaining the proportionality in the samples. Note that, as we are clustering the data twice with the last component being different, some clusters in C_{high} might overlap with those in C_{low}.

1) Obtaining the high-performance clusters: Given the vector of rewards $\mathbf{r} = \{r_1, ..., r_N\}$, we define:

$$f(r_k) = \rho \cdot \sqrt{\text{tr}(\boldsymbol{\Sigma}_\omega)} \frac{r_k - \min(\mathbf{r})}{\max(\mathbf{r}) - \min(\mathbf{r}) + 10^{-9}}, \tag{4}$$

where ρ is a relative importance weight, which will keep the sample reward importance proportional to the sample variability during the clustering part of DREPS. This transformation of the reward firstly normalizes the values to $[0, 1]$ and then scales such values to a similar magnitude to that of the parameter variance. Otherwise, as mentioned earlier, undesired clusterization like those displayed in Fig. 1 (left and middle) could be obtained, where data are clustered by reward only or by parameters only. A value of $\rho = 10/D$, D being the number of parameters, has been used throughout this paper. Once the data has been prepared, we run the *K-means* clustering algorithm and obtain a cluster label for each sample, indicating to which cluster it has been assigned. Taking the average transformed reward for each of the clusters, we separate them into two groups (using a $1-$dimensional *K-means* clustering with 2 clusters). The group with the best average rewards will be the clusters we will consider as *high-performance clusters* and we will gather them in the set C_{high}. We may define a maximum number of low-performing clusters, consider only a single cluster, or let the algorithm choose the number of high-performance clusters.

2) Obtaining the low-performance clusters: The only difference with the just described clustering process is that here we use $1/\max(r_k, 10^{-9})$ instead of r_k in (4). In this way, we obtain a set of clusters C_{low} with low-performance data. The use of the inverse of r_k helps to discriminate better the low-performing samples in order to obtain the repulsive clusters. The supplementary material includes a video comparing the proposed 2-step clustering using the inverse of the reward and 1-step clustering without using such inverse.

Next, we fit each cluster in C with a Normal distribution $g_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ using their associated transformed rewards as weights in an WMLE to obtain the resulting parametrizations for each cluster $i \in C$, $\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$. Given the reward function in Fig. 2, Fig. 3 shows an example of the classification of the sample points with *K-means* clustering, while Fig. 4 displays their associated rewards, and the Normal distributions resulting from the WMLE using a total of 3 clusters, 2 of them being considered as having a low performance. No high-performance cluster is used as attractor in this example. The plot shows the effectiveness of the clustering algorithm at detecting poorly performing areas in the policy space. Moreover, an experimental test of the robustness of the proposed clustering wrt. sampling variability has been included in the supplementary material. Algorithm 1 summarizes the process of computing the clusters given the data samples.

B. DREPS Derivation

Given the information provided by the clustering in the previous section, we will use the computed clusters, represented as Gaussian distributions $g_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, $i \in C$, as repulsive or attractive data for the optimization problem, which now becomes:

$$\begin{aligned} \pi^* = \text{argmax}_\pi \textstyle\int \pi(\boldsymbol{\theta}) \mathbf{R}(\boldsymbol{\theta}) d\boldsymbol{\theta} \\ \text{s.t. } \epsilon \geq = \text{KL}(\pi \| q) \\ 1 = \textstyle\int \pi(\boldsymbol{\theta}) d\boldsymbol{\theta} \\ \xi \leq \text{KL}(\pi \| g_i) \ , \ i \in C_{low} \\ \text{KL}(\pi \| g_i) \leq \chi \ , \ i \in C_{high}, \end{aligned} \tag{5}$$

where ϵ is the bound on the KL-divergence for the REPS algorithm, and ξ, χ are the minimum and maximum KL-divergence we want to have between the new policy and the precomputed low-performance and high-performance clusters, respectively. Note that the condition

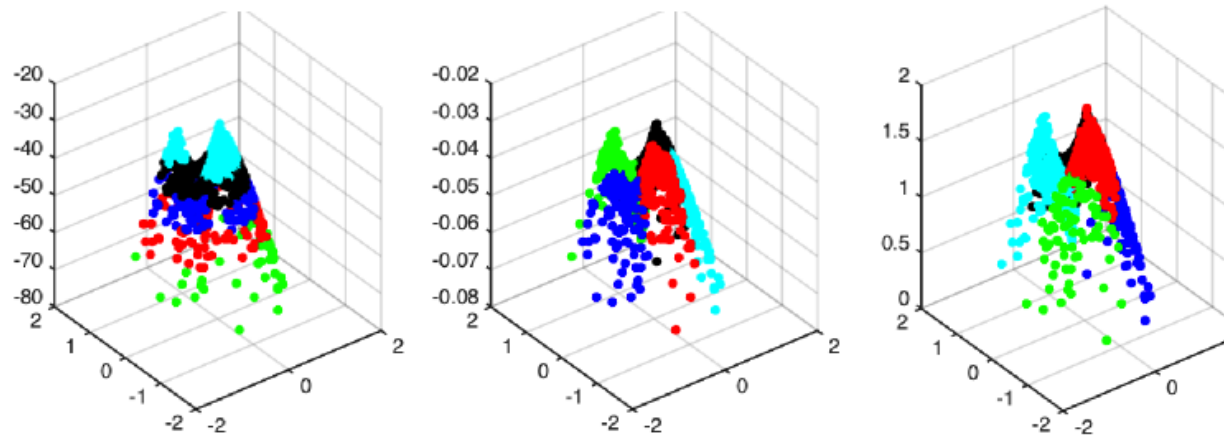

Fig. 1. Clustering variants with different reward treatments of the 3-dimensional data points consisting of two random variables and the transformed reward without (and with) the proposed rescaling in Eq.(4). Left plot: Clustering using unnormalized r_k with a too large scale wrt. parameter variability. On the contrary, the center plot shows the result of using $f(r_k)$ with a too low weight wrt. parameter variability. The right plot shows the proposed relative weighting in Eq.(4).

Algorithm 1 *K-means* clustering for DREPS

Input:

Sample vector $\boldsymbol{\theta}_k$, rewards r_k, $\forall k = 1, ..., N$

Number of dual clusters c

1: Transform the rewards to more discriminating values $f(r_k)$ with (4).
2: Perform standard *K-means* clustering with $[\boldsymbol{\theta}_k, f(r_k)]$ and obtain c clusters.
3: Compute the average transformed reward $\overline{f(r)}_i$ for each cluster $i = 1..c$.
4: Choose the clusters with the best average transformed reward, manually or with a 2-cluster *K-means* approach, and assign them to C_{high}.
5: Transform the Rewards by using $1/r_k$ instead of r_k in (4) to obtain $f(1/r_k)$.
6: Perform standard K-means clustering with $[\boldsymbol{\theta}_k, f(1/r_k)]$ and obtain c clusters.
7: Choose the clusters with the highest average inverse reward, manually or with a 2-cluster *K-means* approach, and assign them to C_{low}.
8: **for** $i \in C$ **do**
9: Compute μ_i, Σ_i with reward-Weighted Maximum Likelihood Expectation (WMLE) using points assigned to cluster i, using the transformed rewards as weights.
10: **end for**

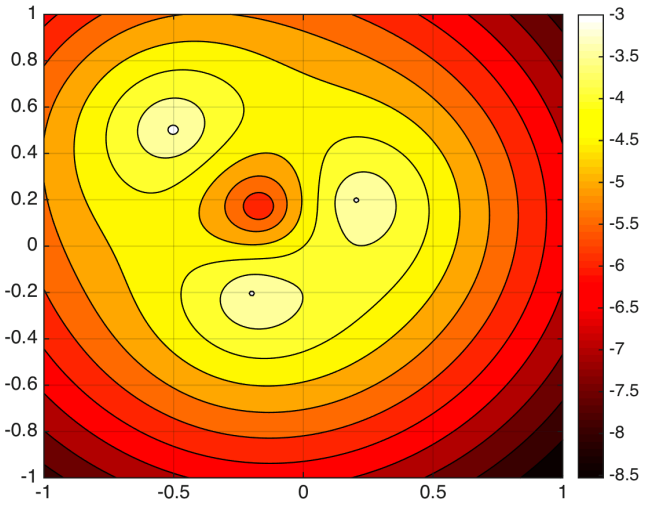

Fig. 2. Reward function used as a clustering illustrative example and in the experimental section (see (12)).

$\epsilon \leq \mathrm{KL}(\pi \| q)$ could be included in the C_{high} restriction. However, we decided to keep it separate to make clear that here the KL-divergence is not with respect to a cluster, but with respect to the previous policy parameters and will always be mantained, while C_{high} may be an empty set and represents a more local influence. The solution of (5) can be found analytically by using Lagrange multipliers and has the form

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})^{\frac{\eta}{\eta+\omega-\nu}} \prod_i g_i(\boldsymbol{\theta})^{\frac{\omega_i-\nu_i}{\eta+\omega-\nu}} \exp\left(\frac{\mathbf{R}(\boldsymbol{\theta})}{\eta+\omega-\nu}\right), \quad (6)$$

where η is the Lagrange multiplier of the first KL constraint in (5) and $\nu = \sum \nu_i, \omega = \sum \omega_i$ are the multipliers for the other KL constraints. These variables can be found by minimizing the dual function of the optimization problem (see the appendix for its derivation). Note that in this paper we will use ν and $\boldsymbol{\nu} = [\nu_1, ..., \nu_{|C_{low}|}]$. Analogously, we will be using ω and $\boldsymbol{\omega}$. Hence, the optimal Lagrange multipliers are the ones obtained by solving:

$$\{\eta^*, \boldsymbol{\nu}^*, \boldsymbol{\omega}^*\} = \mathrm{argmin}_{\eta,\boldsymbol{\nu},\boldsymbol{\omega}} h(\eta, \boldsymbol{\nu}, \boldsymbol{\omega}), \quad (7)$$

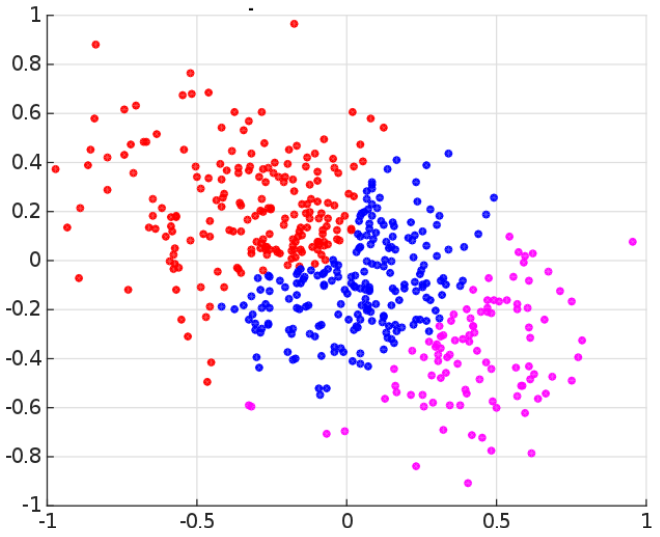

Fig. 3. Result of classification with *K-means* clustering of the 3-dimensional data points consisting of two random variables and the transformed reward for the reward function in Fig. 2.

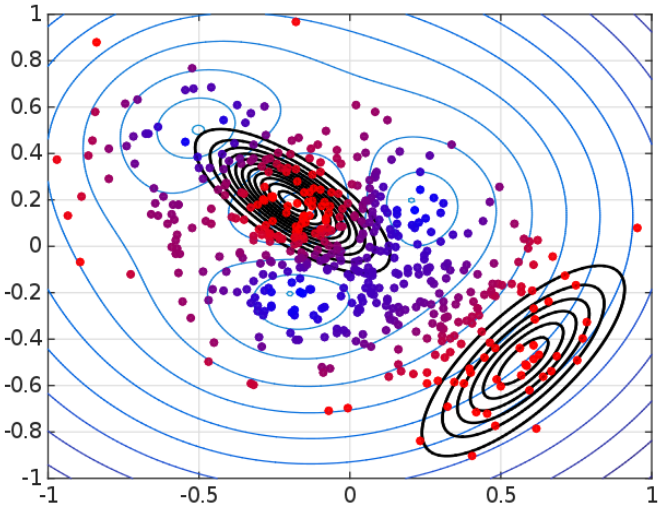

Fig. 4. Result of applying Algorithm 1 to the data points color coded from red (low reward) to blue (high reward). After clustering using the transformed reward as displayed in Fig. 3, the two lowest-performing clusters are fitted with Gaussians using WMLE and are here shown in black.

with $h(\eta, \boldsymbol{\nu}, \boldsymbol{\omega})$ derived in the appendix:

$$h(\eta,\boldsymbol{\nu},\boldsymbol{\omega}) = \eta\epsilon + \sum_{i\in C_{high}} (\omega_i\chi) - \sum_{i\in C_{low}} (\nu_i\xi) + \\ +(\eta+\omega-\nu)\log\left[\frac{1}{N}\sum_{k=1}^{N} q_{(k)}^{\frac{\nu-\omega}{\eta+\omega-\nu}} \prod_{i\in C} g_{i\,(k)}^{\frac{\omega_i-\nu_i}{\eta+\omega-\nu}} \exp\left(\frac{r_k}{\eta+\omega-\nu}\right)\right]. \quad (8)$$

Thus, once the Lagrange multipliers $\eta, \boldsymbol{\nu}, \boldsymbol{\omega}$ have been found, one can update the policy by using WMLE, with the weights d_k for each rollout coming from the samples and the solution form shown in (6):

$$d_k = q_{(k)}^{\frac{\nu-\omega}{\eta+\omega-\nu}} \prod_{i\in C} g_{i\,(k)}^{\frac{\omega_i-\nu_i}{\eta+\omega-\nu}} \exp\left(\frac{r_k}{\eta+\omega-\nu}\right). \quad (9)$$

Algorithm 2 Dual Relative Entropy Policy Search (DREPS)

Input:
Parameters ϵ, ξ, χ, and rollouts per update N
Previous policy $q(\boldsymbol{\theta})$

1: **for** $k = 1..N$ **do** Perform an experiment using $\boldsymbol{\theta}_k$, a sample from the policy $q(\boldsymbol{\theta})$. Compute reward r_k.
2: **end for**
3: Perform both steps of *K-means* clustering as defined in Sec. II-A and obtain $g_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, for $i \in C$
4: Compute the probabilities of each rollout for the previous policy $q(\boldsymbol{\theta}_k)$ and the dual policies $g_{i(k)} = g_i(\boldsymbol{\theta}_k)$ for $k = 1..N$ and $i \in C$.
5: Perform optimization to find $\eta, \boldsymbol{\nu}, \boldsymbol{\omega}$ with the dual function in (7).
6: Find weights d_k for each rollout k as in (9).
7: Perform WMLE with the obtained weights d_k and parameter vectors $\boldsymbol{\theta}_k$ to find the new policy π.

In Algorithm 2 we summarize the DREPS algorithm for clarity.

Note that, for $\boldsymbol{\nu} = 0$ and $\boldsymbol{\omega} = 0$, the effect of the clustered data would be none and the algorithm should behave exactly as REPS. Indeed, for $\boldsymbol{\nu} = 0$ and $\boldsymbol{\omega} = 0$ the solution in (6) becomes:

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})\exp\left(\frac{\mathbf{R}(\boldsymbol{\theta})}{\eta}\right), \quad (10)$$

and the dual function to optimize is

$$h(\eta) = \eta\epsilon + \eta\log\left[\frac{1}{N}\sum_{k=1}^{N}\exp\left(\frac{r_k}{\eta}\right)\right], \quad (11)$$

which are the REPS solution and the dual function, respectively. Thus, setting the influence of the dual policies g_i to zero, we can see that our proposed algorithm reduces to REPS and, therefore, it is a generalization of REPS. An experiment with a real robot described at the end of Section III-B experimentally confirms this theoretical remark in unimodal problems.

III. Experiments

In this section, we present two experimental setups to assess the performance of our proposed algorithm, especially in multi-modal problems: First, a 2-D example of a multi-modal reward function, and second, a multi-modal real robotic problem.

A. Multi-modal 2D reward function

To evaluate how the proposed algorithm performs, we built an example task in which the policy is to sample points $\boldsymbol{\theta}_k$, $k = 1, ..., N$ in a 2-dimensional space and, for each sample, evaluate a reward function r_k consisting in a high reward at three given points $\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \boldsymbol{\psi}_3$ and very low reward in between:

$$r_k = 10\|\sum_{i=1..3}\left(\frac{\boldsymbol{\psi}_i}{3}\right) - \boldsymbol{\theta}_k\| - 5\sum_{i=1..3}\|\boldsymbol{\psi}_i - \boldsymbol{\theta}_k\| \quad (12)$$

The reward function is displayed in Fig. 2, where one can note that there are 3 possible candidates for an optimal solution.

To find the optimal point on the plane, we initialize the policy with $\boldsymbol{\mu}_\omega = \mathbf{0}$ and $\boldsymbol{\Sigma}_\omega = \mathbf{I}$, and 100 samples are evaluated for every policy update, reusing up to 400 previous samples. When using REPS with a KL bound of $\epsilon = 0.5$ for this optimization problem, we noticed that the learning curve had a plateau in most cases (see Fig. 5), corresponding to the algorithm averaging the rewards of two of the optimal points (see Fig. 6). The REPS algorithm keeps obtaining samples near both candidates and cannot improve the policy further

until significantly more samples get closer to one of the candidates than the other, moving the policy towards one of the solutions. As a result, the more rollouts per policy update used, the more likely REPS is to stay longer in such plateau.

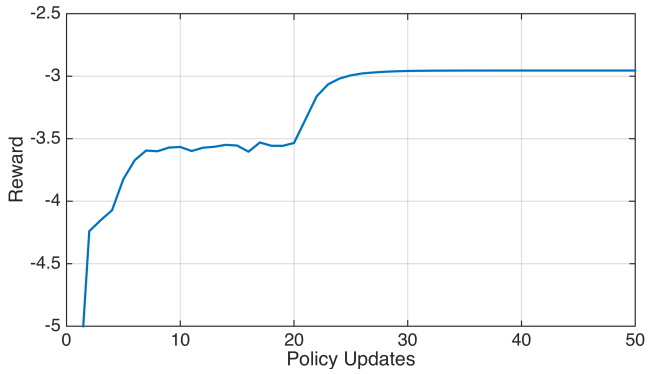

Fig. 5. Learning curve of the REPS algorithm for the 2D optimization example. The algorithm gets stuck in a plateau averaging between two solutions (see Fig. 6) between the 7th and 21th policy updates.

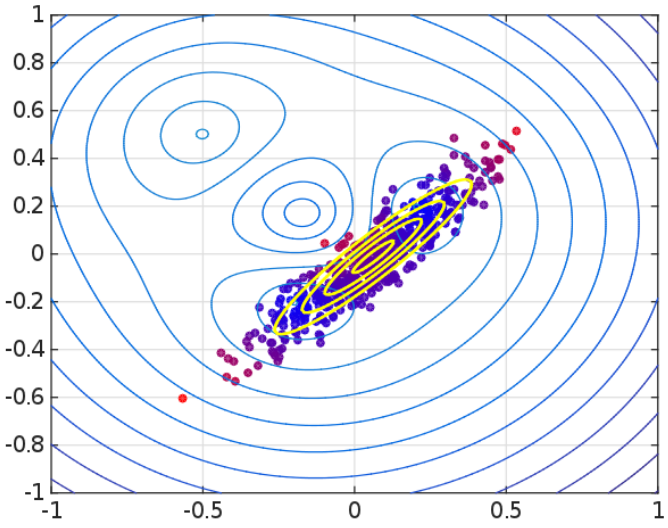

Fig. 6. Gaussian policy resulting from applying REPS to the multi-modal 2D optimization example. As shown here, REPS averages between two solutions, and keeps doing so for several iterations, as can be seen in the video included in the supplementary material.

If, instead of REPS, we use our proposed approach DREPS, the effect of the repulsive Gaussian in the middle allows the algorithm to quickly avoid this plateau and keep on with the optimization. We compared the performance of a REPS algorithm (REPS), a dual REPS algorithm with 3 repulsive Gaussian, and none attractive (nDREPS), and the full DREPS algorithm with one attractive Gaussian and up to 4 repulsive ones (DREPS). In the latter case, we let the algorithm itself decide how many repulsive clusters it would use with a 2-cluster *K-means*, as explained in Sec.II-A, i.e.: $|C_{low}| \leq 4$, $|C_{high}| = 1$, and parameters $\xi = 5$, $\chi = 2\epsilon$. We performed 50 learning experiments for both REPS and DREPS, and the results are displayed in Fig. 7, where we can see that nDREPS performs better than REPS, but both are outperformed by the full DREPS. A video comparing the evolution of REPS vs. DREPS can be found in the supplementary material. The scalability of the proposed approach has been assessed using

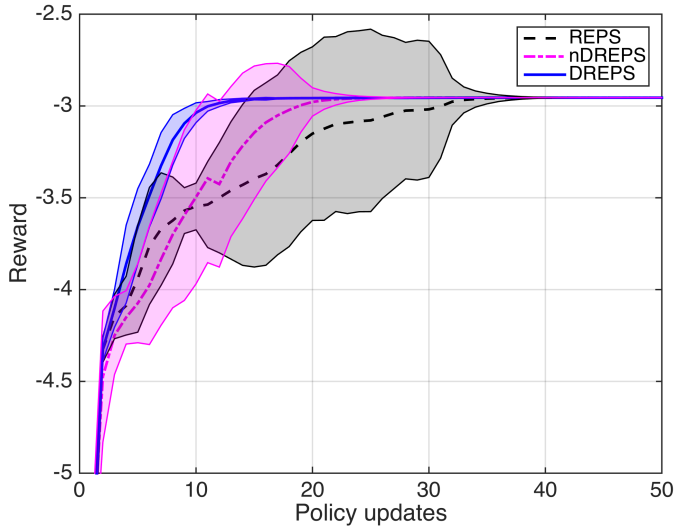

Fig. 7. The learning curves for the multi-modal 2D optimization example, averaged for 50 experiments each (mean and 2-standard deviations are plotted), show the advantage of using the DREPS algorithm.

the same problem in a larger-dimensional parameter space, and the results have been included in the supplementary material as well.

B. Real robot multi-modal problem

As a second experiment, we programmed a Barrett WAM robot so that its end-effector would follow a straightline trajectory with fixed orientation (facing down) and fixed z component, from a starting position towards a goal position. Two bottles were added on the way as seen in Fig. 8 and, using RL, the robot had to adapt the trajectory to an *S-shaped* motion that would not knock down any of the bottles. The trajectory was encoded as a Dynamic Movement Primitive [19] initialized to a straightline with 10 Gaussians per DoF equally spaced in time, to a total of 20 parameters, representing the linear multipliers of such fixed Gaussians, to characterize the x and y components of the trajectory. The same implementation of DREPS as in the simulated experiment, with identical algorithmic parameters, was used.

Regarding the reward function to optimize, an initial approach was taken with strong penalizing terms for knocking down the bottles and the length of the trajectory:

$$\mathbf{R} = -2N_{bottlesdown} - 0.15L_{trajectory}, \tag{13}$$

where $N_{bottlesdown}$ is the number of bottles knocked down (0, 1 or 2) and $L_{trajectory}$ is the trajectory length in meters. The relative weights of these terms were set to 2 and 0.15, respectively, giving a higher importance to task accomplishment in the reward function, as usually done in literature [6].

With this setting, DREPS outperformed REPS in convergence velocity, as we can see in Fig. 9. Obviously, all the final solutions obtained with both REPS and DREPS were of the two left-most cases of Fig. 10, which was to be expected given the reward function in (13), but they were not the desired solutions. For this reason, we added a term penalizing the fact that the robot would not cross between the bottles. To do so, we evaluated in which side of the line drawn in Fig. 8 the arm was when passing by each of the two bottles, and a term was added to the reward function penalizing when it was on the same side. Note that this term allows for symmetric

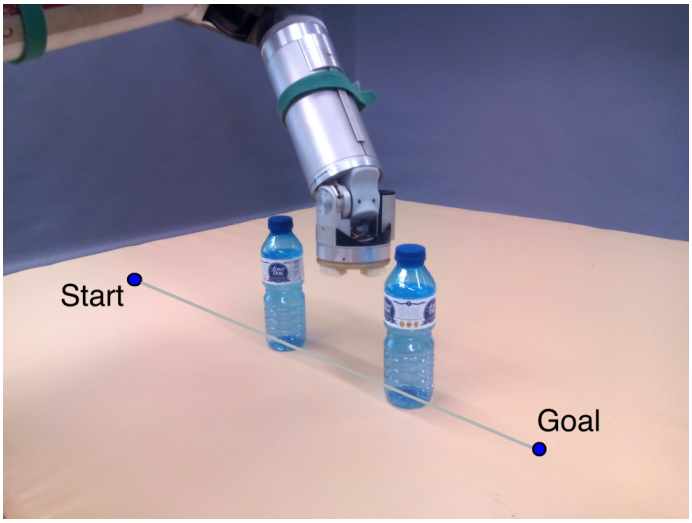

Fig. 8. Experimental setup for a real robot experiment where the robot must learn to perform an S-shaped motion between the two bottles.

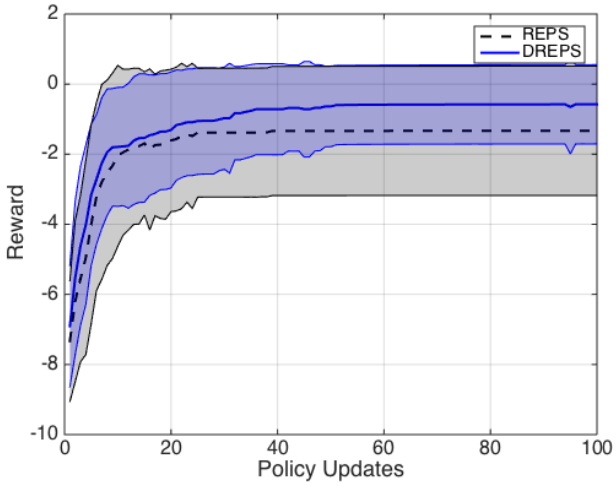

Fig. 11. Learning curves for REPS and DREPS with the reward function in (14). Policy updates were calculated after every 50 rollouts.

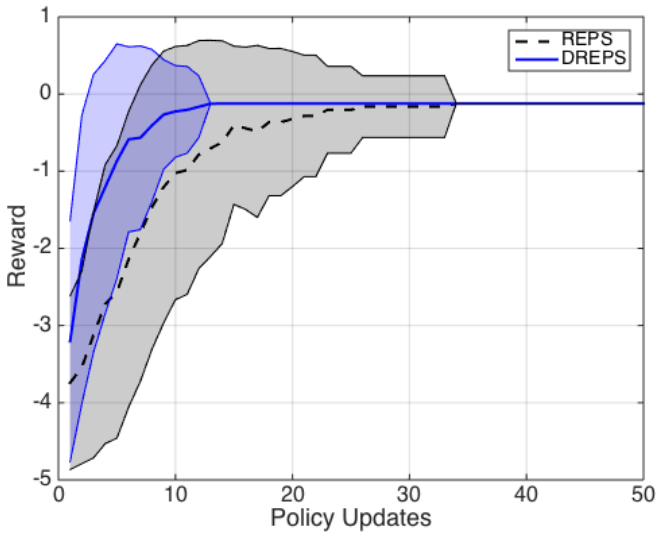

Fig. 9. Learning curves for REPS and DREPS with the reward function in (13). Policy updates were calculated after every 50 rollouts. The advantage of using DREPS is even more evident in the learning curves plotting the mean and the Standard Error of the Mean (SEM) with 95% confidence intervals included in the supplementary material.

solutions as the two right-most ones, plotted in blue, in Fig. 10. The new reward function would then be:

$$\mathbf{R} = -2N_{bottlesdown} - 0.15L_{trajectory} - 4\mathbf{I}_{cross}, \qquad (14)$$

where $\mathbf{I}_{cross}$ indicates whether the robot did or did not cross between the bottles ($\mathbf{I}_{cross} = 1$ in case the robot did not cross, and $\mathbf{I}_{cross} = 0$ in case it crossed). The relative weight of such added term was set to 4 to have the same negative effect in the reward function as if knocking down the two bottles.

We also performed 50 simulated experiments of 100 policy updates with 50 rollouts each and the learning curves with the mean and 2-standard deviations can be seen in Fig. 11. REPS obtained a satisfactory solution in 35 out of 50 experiments, while DREPS solved the problem correctly in 47 of them. Moreover, the average reward for the unsatisfactory solutions obtained by REPS was -4.12, while the average reward for the unsatisfactory solutions obtained by DREPS was -2.80. This is due to the fact that REPS is more likely to prematurely converge to one of the two left-most solutions in Fig. 10, while DREPS's repulsive term pushes the solutions away from those, which actually yield a lower reward than the magenta solutions in the middle of Fig. 10.

Regarding the computational time of the clustering, Alg. 1 took an average of $0.52s$ in an $i5$-$2400S$ CPU at 2.50GHz for the bottle experiment, clustering 50 samples of dimension 20. Such additional computational cost, together with the cost of Alg. 2, makes our approach more CPU-demanding than REPS. However, real robot motion is more time demanding and costly than such computational time every N robot motions. In particular, looking at Fig. 9, such increment on computing time results in a better learning curve, thus requiring less real-robot experiments. A video comparing REPS and DREPS in the bottle avoiding task can be found in the supplementary material, together with the execution on the real robot of the final trajectory found by DREPS.

C. Real robot uni-modal problem

Moreover, we tested the performance of DREPS on a uni-modal reward task, namely drawing a circle, in which the same 7-DoF WAM robot had to improve an initial motion towards a 3D circle-tracking motion. We kinesthetically taught a 7-DoF WAM arm to follow a circular trajectory in the Cartesian space with its wrist. The circle best fitting the initial trajectory, which was very inaccurate, was computed and a cost function consisting in a point-to-point deviation from that circle, plus an acceleration-penalizing term, was considered. We fitted the taught trajectory with a Dynamic Movement Primitive (DMP). 12 Gaussians equally spaced in time were used for each DoF, as the trajectory to be learned was a complex 20-second movement. This generated a set of 84 parameters, representing the linear multipliers of such Gaussians. After applying both REPS and DREPS algorithms, the outcome after 50 learning experiments was very similar, which could be expected due to the uni-modality of the problem. The learning curves for both REPS and DREPS can be seen in Fig. 12. As theoretically anticipated in Section II-B, DREPS displayed the same behavior as REPS in terms of learning speed and resulting trajectories, as also shown in the supplementary material.

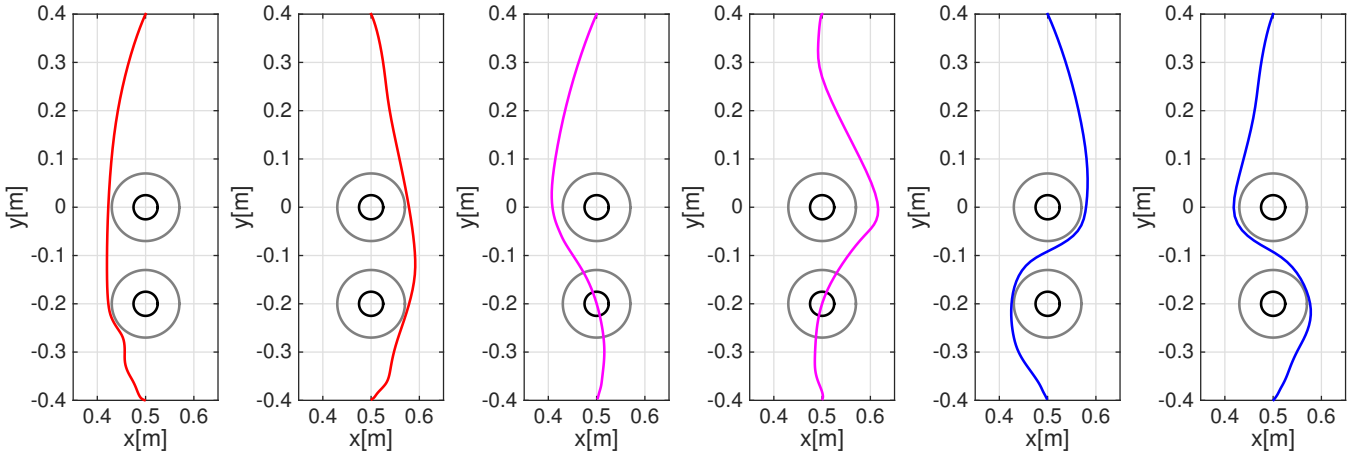

Fig. 10. Schematic visualization of some representative trajectories obtained for the real-robot experiment using the reward in (14). The bottles (in black) have been expanded with a safety threshold corresponding to the width of the arm's end effector (gray). The left-most plots show trajectories with a low reward (< -4) due to the robot not crossing in-between the bottles. The center plots with trajectories in magenta show solutions with a reward between $(-3, -2)$, while examples of quasi-optimal trajectories are shown in blue on the right-most plots, corresponding to a reward (> -1).

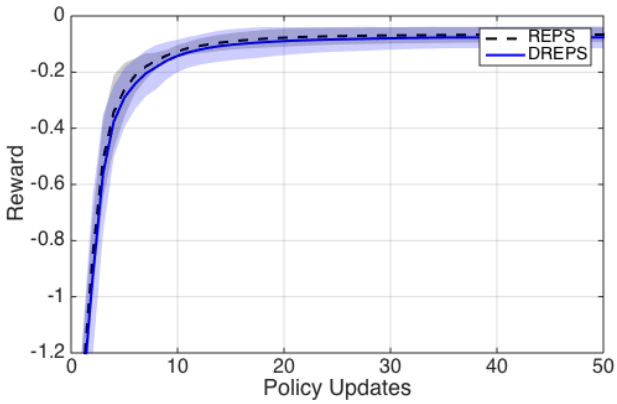

Fig. 12. Learning curves for REPS and DREPS for a simulated uni-modal problem with real robot data. Both algorithms show the same performance as expected for a uni-modal task.

IV. Conclusions

In this work, we developed a generalization of the Policy Search (PS) algorithm known as Relative Entropy Policy Search. Such generalization, which is equal to REPS if the clusterization is omitted, considers the possibility of using both bad experiences to have a repulsive effect, and best data to encourage approaching the best-performing areas. This helps to influence the solution away from bad data collected during sampling/experimentation. While the performance of REPS and DREPS is similar in purely convex problems, our algorithm shows to be effective at preventing the loss of time in plateaus by other algorithms, as seen in the learning curve in Fig. 7, without the need of using a multi-modal solution as in Hierarchical REPS [11].

Clusterization prior to the application of DREPS as presented in this paper has proved effective, but future work includes a deeper study of this topic, testing if other clustering algorithms or approaches to obtain the attractive and repulsive clusters could yield better results.

The proposed algorithm, while showing a very similar behaviour to REPS in uni-modal problems, is very suitable in cases where there is a multi-modal solution to the problem, but the user only needs a single solution. Multi-modal PS approaches would need more samples in order to fit the different possible solutions, while DREPS focuses on a single solution and refines it faster. We have first assessed the benefits of using DREPS using a synthetic multi-modal reward function. Then, experiments in a real robot setup have been performed, using DMPs to parametrize robot motion [17] in a task with a multi-modal reward function, and the results confirm the better performance of DREPS versus REPS.

Appendix A
Derivation of the Dual Objective Function

Given the optimization problem (5), we can compute the lagrangian as:

$$\begin{aligned}\mathcal{L} = &\int \pi(\boldsymbol{\theta})\mathbf{R}(\boldsymbol{\theta})d\boldsymbol{\theta} + \eta\left(\epsilon - \int \pi(\boldsymbol{\theta})\log\tfrac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}d\boldsymbol{\theta}\right)\\ &+ \sum_{i\in C_{low}} \nu_i\left(\int \pi(\boldsymbol{\theta})\log\frac{\pi(\boldsymbol{\theta})}{g_i(\boldsymbol{\theta})}d\boldsymbol{\theta} - \xi\right)\\ &\sum_{i\in C_{high}} \omega_i\left(\chi - \int \pi(\boldsymbol{\theta})\log\frac{\pi(\boldsymbol{\theta})}{g_i(\boldsymbol{\theta})}d\boldsymbol{\theta}\right) + \lambda\left(\int \pi(\boldsymbol{\theta})d\boldsymbol{\theta} - 1\right)\end{aligned} \tag{15}$$

where η, ν_i,ω_i, $\forall i$ are positive. Differentiating with respect to $\pi(\boldsymbol{\theta})$ (and omitting $\boldsymbol{\theta}$ for simplicity) we obtain

$$\begin{aligned}\tfrac{\partial\mathcal{L}}{\partial\pi} = &\mathbf{R} - \eta(\log\pi - \log q + 1) + \lambda\\ &+ \sum_{i\in C_{low}} \nu_i(\log\pi - \log g_i + 1)\\ &- \sum_{i\in C_{high}} \omega_i(\log\pi - \log g_i + 1)\end{aligned} \tag{16}$$

which, setting $\frac{\partial\mathcal{L}}{\partial\pi} = 0$ and isolating $\log\pi$ becomes:

$$\begin{aligned}\log\pi = &\tfrac{\mathbf{R}}{\eta+\omega-\nu} + \tfrac{\eta\log q}{\eta+\omega-\nu} + \tfrac{\sum_{i\in C_{high}}\omega_i\log g_i}{\eta+\omega-\nu}\\ &- \tfrac{\sum_{i\in C_{low}}\nu_i\log g_i}{\eta-\nu} - \tfrac{\eta+\omega+\lambda-\nu}{\eta+\omega-\nu}\end{aligned} \tag{17}$$

with $\nu = \sum_{i\in C}\nu_i$, and setting $Z = \exp\left(\frac{\eta+\omega+\lambda-\nu}{\eta+\omega-\nu}\right)$, we obtain

$$\pi = Z^{-1}q^{\eta/(\eta+\omega-\nu)}\prod_{i\in C} g_i^{\omega_i-\nu_i/(\eta+\omega-\nu)}\exp\left(\frac{\mathbf{R}}{\eta+\omega-\nu}\right) \tag{18}$$

where, given that $1 = \int \pi(\boldsymbol{\theta}) d\boldsymbol{\theta}$,

$$Z = \int_{\boldsymbol{\theta}} q^{\eta/(\eta+\omega-\nu)} \prod_{i \in C} g_i^{\omega_i - \nu_i/(\eta+\omega-\nu)} \exp\left(\frac{\mathbf{R}}{\eta+\omega-\nu}\right) d\boldsymbol{\theta}. \tag{19}$$

Now, reinserting (18) into (15), we obtain a dual function for the lagrangian problem:

$$h(\eta, \boldsymbol{\nu}, \boldsymbol{\omega}) = \eta\epsilon + \sum_{i \in C_{high}} (\omega_i \chi) - \sum_{i \in C_{low}} (\nu_i \xi) + \lambda + \eta + \omega - \nu \tag{20}$$

where, isolating $\lambda + \eta + \omega - \nu$ from Z in equation (19) and inserting it into (20):

$$h(\eta, \boldsymbol{\nu}, \boldsymbol{\omega}) = \eta\epsilon + \sum_{i \in C_{high}} (\omega_i \chi) - \sum_{i \in C_{low}} (\nu_i \xi) + (\eta + \omega - \nu) \log \int_{\boldsymbol{\theta}} q^{\eta/(\eta+\omega-\nu)} \prod_{i \in C} g_i^{\omega_i - \nu_i/(\eta+\omega-\nu)} \exp\left(\tfrac{\mathbf{R}}{\eta+\omega-\nu}\right). \tag{21}$$

We can now replace the integral over a sum of samples to obtain the dual objective function:

$$h(\eta, \boldsymbol{\nu}, \boldsymbol{\omega}) = \eta\epsilon + \sum_{i \in C_{high}} (\omega_i \chi) - \sum_{i \in C_{low}} (\nu_i \xi) + (\eta + \omega - \nu) \log \left[\frac{1}{N} \sum_{k=1}^{N} q_{(k)}^{\frac{\nu-\omega}{\eta+\omega-\nu}} \prod_{i \in C} g_{i(k)}^{\frac{\omega_i - \nu_i}{\eta+\omega-\nu}} \exp\left(\frac{r_k}{\eta+\omega-\nu}\right) \right]. \tag{22}$$

This dual function can be evaluated provided we can compute the probability of a given trajectory for both the previous policy q and the dual policies g_i. These can be computed from the direct policy evaluation or, in cases where the outcome is a sequence of states, by multiplying the transition probabilities for all the timesteps of a sequence. For numerical stability reasons, we recommend to directly compute the log-probability of such normal distribution.

From the mathematical perspective, there is no guarantee that this problem will always be convex for ν, thus in order to minimize the dual function h, we set a minimum value for ν_i in the active-set optimization of the dual function, it being an indicator of the minimum influence we want the dual policies to have. If g_i are defined by fitting a normal distribution given some clustered samples with their associated rewards (assuming rewards are negative, and closer to zero is considered better), we can, for example, set $\nu_i = \nu \frac{1}{\sqrt{|\overline{R}_i|}}$, with $\nu = \sum_i \nu_i$ and $\overline{R}_i$ the average reward for the i-th cluster.

Additionally, in some circumstances the solution provided by the solver might not be fully respecting the ϵ bound on the KL-divergence. This comes from trying to find a probability distribution with a min/max dissimilarity with respect to other distributions, which could then become a set of restrictions impossible to comply with. For that reason, the KL-divergence of the solution found was evaluated after the policy update, and in case $KL(\pi \| q) > \epsilon$, the gradient of the KL of the solutions found with respect to ν_i, χ and ξ was iteratively obtained, performing gradient descent on these parameters until a suitable solution within the KL-divergence bound was found. In order to perform such gradient descend, it is vital that the *K-means* clustering initialization is performed in a deterministic manner, as in [18]. Furthermore, when no convergence is reached after a certain number of iterations, $\boldsymbol{\nu}$ is set to zero for that policy update and the optimization is performed only with attractor Gaussians.

REFERENCES

[1] L.P. Kaelbling, M.L. Littman; A.W. Moore. "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[2] J-B. Mouret, "Micro-Data Learning: The Other End of the Spectrum", *ERCIM News. Special theme: Machine Learning*, no. 107, pp. 18, 2016.

[3] S. Levine, P. Pastor, A. Krizhevsky, D. Quillen "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection", *International Symposium on Experimental Robotics (ISER)*, 2016.

[4] J. Peters, S. Schaal. "Policy gradient methods for robotics ." *Proc. IEEE/RSJ IROS*, pp. 2219-2225, 2006.

[5] J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients". *Journal of Neural Networks*, vol. 21, no. 4, pp. 682-697, 2008.

[6] M. Deisenroth, G. Neumann and J. Peters, "A Survey on Policy Search for Robotics". *Foundations and Trends in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2011.

[7] E. Theodorou, J. Buchli and S. Schaal, "Reinforcement Learning of Motor Skills in High Dimensions: A Path Integral Approach". *IEEE ICRA*, pp. 2397-2403, 2010.

[8] E. A. Theodorou, J. Buchli and S. Schaal. "A Generalized Path Integral Control Approach to Reinforcement Learning". *Journal of Machine Learning Research*, vol. 11, pp. 3137-3181, 2010.

[9] F. Stulp, E. A. Theodorou, S. Schaal. "Reinforcement learning with sequences of motion primitives for robust manipulation" *IEEE Transactions on Robotics*, vol. 28, no. 6, 2012.

[10] J. Peters, K. Mülling and Y. Altün, "Relative Entropy Policy Search". *24th National Conf. on Artificial Intelligence*, track 15, pp. 182-189, 2011.

[11] C. Daniel, G. Neumann and J. Peters, "Hierarchical Relative Entropy Policy Search". *Journal of Machine Learning Research*, track 22, pp. 273-281, 2012.

[12] S. Kullback and R.A. Leibler, "On Information and Sufficiency". *Annals of Mathematical Statistics 22*, vol 1, pp. 79–86, 1951.

[13] G. Neumann, "Variational Inference for Policy Search in Changing Situations", *International Conference on Machine Learning (ICML)*, pp. 817-824, 2011.

[14] V. Gómez, H.J. Kappen, J. Peters and G. Neumann, "Policy Search for Path Integral Control", *European Conf. on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, vol. 8724, no. 2014, pp. 482-497, 2014.

[15] S.P. Lloyd, "Least Squares Quantization in PCM." *IEEE Transactions on Information Theory*. vol. 28, pp. 129–137, 1982.

[16] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman and A.Y. Wu, "An efficient K-means clustering algorithm: Analysis and implementation". IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, pp. 881–892, 2009.

[17] S. Schaal, J. Peters, J. Nakanishi and A. Ijspeert, "Learning Movement Primitives," *International Symposium on Robotics Research (ISRR)*, pp. 561-572, Springer Tracts in Advanced Robotics, 2005.

[18] S.S. Khan, A. Ahmad, "Cluster center initialization algorithm for K-means clustering", *Pattern Recognition Letters*, vol. 5, pp. 1293–1302, 2004.

[19] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, S. Schaal. "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviours". *Neural Computation*, vol. 25, no. 2, pp. 328-373, 2013.