Dimensionality Reduction and Motion Coordination in Learning Trajectories with Dynamic Movement Primitives

Adrià Colomé and Carme Torras

Abstract— **Dynamic Movement Primitives (DMP) are nowadays widely used as movement parametrization for learning trajectories, because of their linearity in the parameters, rescaling robustness and continuity. However, when learning a movement with a robot using DMP, many parameters may need to be tuned, requiring a prohibitive number of experiments/simulations to converge to a solution with a locally or globally optimal reward.**

We propose here strategies to palliate this dimensionality problem: the first is to explore only along the most significant directions in the parameter space, and the second is to add a reduced second set of Gaussians that would optimize the trajectory after fixing the Gaussians approximating the demonstrated movement. Both strategies result in less Gaussian computations and better performance on learning algorithms.

To further speed up the learning and allow for a better biased exploration, we also propose to coordinate the motion of different joints, by computing a coordination matrix initialized with the demonstrated movement and then automatically updating it by eliminating the degrees of freedom least affecting task performance.

Our three proposals have been experimentally tested and the obtained results show that similar (or even better) performance can be obtained at a significantly lower computational cost by reducing the dimensionality of the exploration space.

I. INTRODUCTION

Learning robotic skills is a tough problem which can be adressed in several ways. The most common approach is Learning from Demonstration (LfD), in which the robot is shown an initial way of solving a task, and then tries to reproduce, improve and/or adapt it to variable conditions. The learning of tasks is usually performed in the kinematic domain by learning trajectories [1], [2], although it can also be done in the force domain [3], [4]. The problem of learning to execute complex motor tasks can be tackled with different approaches as follows.

A training data set is often used in order to fit a relation between an input (experiment conditions) and an output (a good behaviour of the robot). This fitting, which often uses regression models such as Gaussian Mixture Models (GMM) [5] or Locally Weighted Projection Regression (LWPR) [6], is then adapted to the external conditions in order to modify the robot's behaviour [5] (see Fig. 1).

This work is partially funded by EU Project IntellAct (FP7-269959) and by the Spanish Ministry of Science and Innovation under project PAU+ DPI2011-27510 and by the CSIC Project CINNOVA (201150E088). A. Colomé is also supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant (AP2010-1989)

The authors are with Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Llorens Artigas 4-6, 08028 Barcelona, Spain. E-mails: [acolome,torras]@iri.upc.edu

Despite its high adaptability, reproducing the demonstrated behaviour and adapting it to new situations does not always solve a task optimally, thus Reinforcement Learning (RL) is also being used, where the robot learns from a demonstration, and improves the solution by trial-and-error (see Fig. 2). RL is capable of finding better solutions than the one demonstrated to the robot.

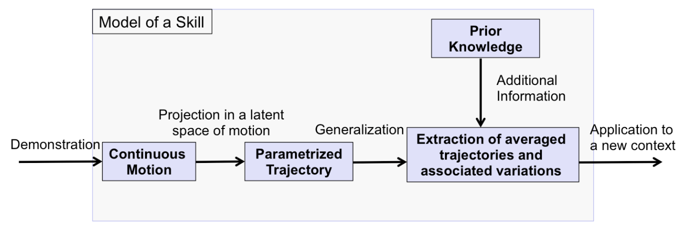

Fig. 1. Typical LfD scheme. A trajectory (or a set of trajectories) is shown by some means to the robot, and then the robot obtains a robust generalization of those trajectories, which is then useful to adapt to similar situations. Inspired by Fig. 59.9 of [7].

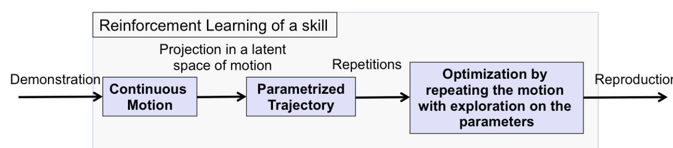

Fig. 2. Learning with Reinforcement Learning. Once the trajectory is parametrized, exploration on these parameters is done with several repetitions and a reward function, to improve over the initial demonstration.

In order to learn a task, a robot must have a framework characterizing its motion. Movement Primitives (MP) are parametrized trajectories for a robot that can be represented in many different ways, such as splines or neural networks. A desired trajectory is represented by fitting certain parameters, which can then be used to improve or change this desired trajectory, while a proper control (a computed torque control [8], for example) tracks this reference signal. Once a task is learned with MP from a single or several demonstrations, it can be generalized to similar contexts [9], [10].

Among all MP, the most used ones are Dynamic Movement Primitives (DMP) [11], [12], which characterize a movement by means of a second order dynamical system, using a position error, a velocity term and an excitation function for obtaining the acceleration profile generating the movement:

$$\begin{aligned} \dot{\mathbf{z}}/\tau &= \alpha_z \left(\beta_z \left(\mathbf{y_g} - \mathbf{y}\right) - \mathbf{z}\right) + \mathbf{f}(x) \\ \mathbf{f}(x) &= \mathbf{\Theta} \cdot \mathbf{g}(x), \end{aligned} \tag{1}$$

where $\mathbf{y}$ is the joint position vector, $\mathbf{y_g}$ the goal/ending joint position, τ a time constant, x is a transformation of time verifying $\dot{x} = -\alpha_x x/\tau$ and $\mathbf{z} = \dot{\mathbf{y}}/\tau$ a rescaled velocity vector. In addition, $\mathbf{\Theta}$ is the parameter matrix, where each row represents the parameters used for each joint to learn an initial move, applied to a set of basis functions $\mathbf{g}(x)$ defined as:

$$g_i(x) = \frac{\phi_i\left(x\right)}{\sum_j \phi_j\left(x\right)} x, i = 1..N_f, \tag{2}$$

where $\phi_i\left(x\right) = exp\left(-0.5(x-c_i)^2/d_i\right)$, and c_i, d_i represent the fixed center and width of the *i*th Gaussian used of a total of N_f per Degree of Freedom (DoF).

With this motion representation, the robot can be taught a demonstration movement, to obtain the weights and Gaussians of the motion by using least squares techniques on:

$$f_{de}^{(j)}(x) = \dot{z}_{de}^{(j)}/\tau + \alpha_z\left(\beta_z\left(y_{de}^{(j)} - y_g^{(j)}\right) + z_{de}^{(j)}\right) = \boldsymbol{\theta}_{(j)}^T \mathbf{g}(x), \tag{3}$$

where (j) indicates *j*th joint, the subscript *de* represents the demonstration movement taught to the robot and $\boldsymbol{\theta}_{(j)}^T$ the j-th row of $\mathbf{\Theta}$. The DMP representation of trajectories has good scaling properties wrt. trajectory time and initial/ending positions, has an intuitive behaviour, does not have an explicit time dependence and is linear in the parameters, among other advantages [11]. For these reasons, DMP are being widely used with policy optimization RL, where the general goal is to optimize the policy parameters $\mathbf{\Theta}$ so that an expected reward $\mathbf{J}(\mathbf{\Theta})$ is maximal. After each rollout, the reward/cost function is evaluated and used to search for a set of parameters that improve the performance over the initial movement.

Some of the common approaches used in this framework are Policy Gradients (PG) [13], [14], Policy Improvement with Path Integrals (PI2) [15], [16], [17] or Relative Entropy Policy Search (REPS) [18], [19]. However, these ideas have resulted in algorithms that require several roll-outs to find a proper update for the motion parameter. In addition, to have a good fitting of the initial movement, many parameters are required, while we want to have few in order to reduce the dimensionality of the optimization problem. When applying learning algorithms with DMP, several aspects must be taken into account [20]:

- **Parameter dimensionality**. Despite DMP having less parameters than other motion representations, such as neural networks, complex robots still require many parameters for a proper trajectory representation. The number of parameters needed strongly depends on the trajectory length or speed. In a 7-DoF robot following a long 20-second trajectory, the use of more than 20 Gaussian kernels per joint might be necessary, thus having at least 140 parameters in total. Exploring all these parameters can result in the algorithm interpreting that a weight of a kernel is good because its neighbours are. This large parameter dimensionality then requires a lot of exploration rollouts in order to obtain good optimization values. Although there exist procedures for segmenting and joining short MP [21], the use of a single MP for each task provides a simpler framework.
- **Model availability**. RL can be performed by simulation or with a real robot. The first case is more practical when a good simulator of the robot and its environment is available. However, in the case of manipulation of non-rigid objects or when a model is not available, reducing the number of repetitions is critical. In addition, certain exploration values might result in dangerous acceleration or motion on the real robot, such as strong oscillations and acceleration changes.
- **Cost function**. RL needs a cost/reward function indicating how good the performance of the robot was after a reproduction of the task. In previous work, the cost function considered is an indicator of whether the task is being accomplished or not, adding a term to reduce the DMP weights' values and an acceleration cost. These terms are linearly weighted in order to have a similar order of magnitude. However, large accelerations coming from the DMP characterization can generate large costs, so when working in a simulated environment, it is common to start with a minimum jerk trajectory [16], [22]. In addition, certain tasks may not depend on the whole number of DoFs of the robot, meaning that the RL algorithm used might be exploring motions that are irrelevant to the task, as we will see later.

When fitting a demonstrated movement with MP, there is a tradeoff between a good fitting of the initial movement, a low dimensionality of the parameter space obtained and a good exploration strategy for learning. In [23], LWPR is used for efficiently fitting trajectories, but with no guarantees of obtaining a good exploration framework for using RL. Due to these reasons, in Section II we will present alternatives to reduce the parameter dimensionality of the DMP characterization, while in Section III we will be focusing on the robot's DoF.

II. Parameter Dimensionality Reduction

In this section, we propose to reduce the dimensionality of the exploration parameters by using a multilayer excitation function. In [22], it is shown that for the PI2 algorithm, performing constant exploration rollouts has a similar performance to that of variable timestep exploration. This exploration can be done in the acceleration domain, with each acceleration value associated to a proper Gaussian, or directly in the parameter space. However, the exploration magnitude for the parameters is often manually tuned. In [22], after a number of exploration rollouts, the algorithm exploration parameter, represented as the covariance matrix, is adapted according to the rewards and weights given by the PI2 algorithm, namely Covariance Matrix Adaptation (CMA). CMA's exploration matrix represents an ellipse that converges to an exploration magnitude for each parameter vector. However, using this covariance matrix with fewer rollouts per update than the number of parameters restricts the exploration to the already explored parameter subspace, unless a filter is used in order not to reduce the potential task improvement limiting parameter space. Also, it is recommended to add a minimal value to the diagonal elements of the covariance matrix after each update to avoid premature convergence of this matrix to zero in certain Gaussians.

In order to reduce the dimensionality while performing a meaningful exploration, we propose both to explore in the relevant directions of the parameter space according to the Gaussian kernels, and to create a second layer of Gaussians, as will be described in the following two subsections, respectively.

A. Exploring in Relevant Directions

When having a complex trajectory that requires several Gaussians to be properly characterized, a first option is to explore in the direction of those parameter vectors that have most influence on the robot's motion, by computing once the matrix:

$$\mathbf{W} = \begin{bmatrix} g_1(x_0) & \dots & g_{N_f}(x_0) \\ \dots & & \dots \\ g_1(x_{N_t})) & \dots & g_{N_f}(x_{N_t}) \end{bmatrix}, \tag{4}$$

M being the number of Gaussians for each joint of the robot and N_t the number of timesteps in the trajectory. This matrix $\mathbf{W}$ has dimension $N_t \times N_f$, with $N_t >> N_f$. If we compute its Singular Value Decomposition, we get its eigenvectors $\mathbf{v}_1, ..., \mathbf{v}_{N_f}$ in the parameter space (dimension N_f), with gains associated to the singular values obtained $\sigma_1 > ... > \sigma_{N_f}$. Then, instead of exploring each DMP parameter θ_i, we can take the $M < N_f$ most relevant directions $\mathbf{v}_1, ..., \mathbf{v}_M$ and, for each exploration repetition of the movement, use an updated parameter computed as $\boldsymbol{\theta}_e^{(j)} = \boldsymbol{\theta}^{(j)} + \epsilon_1 \mathbf{v}_1 + ... + \epsilon_M \mathbf{v}_M$, with ϵ_s following a normal distribution with a predefined exploration variance. Despite not being reward-oriented, this approach reduces the number of parameters of each degree of freedom by performing an exploration meaningful for motion, and provides a good initial guess for the exploration covariance matrix when using CMA, already removing the exploration along some parameter directions.

B. Dual-layer excitation functions

Another strategy to tackle high-dimensionality in the exploration is to create a dual-layer excitation function, in the sense that in Eq. (1), one can take

$$\mathbf{f}(x) = \boldsymbol{\Theta}_\mathbf{0}\mathbf{g}_\mathbf{0}(x) + \boldsymbol{\Theta}_e\mathbf{g}_\mathbf{e}(x),$$

with $\boldsymbol{\Theta}_\mathbf{0}$ and $\mathbf{g}_\mathbf{0}(x)$ the weights and Gaussians of the excitation function learned with the demonstrated move, and $\boldsymbol{\Theta}_\mathbf{e}$ and $\mathbf{g}_\mathbf{e}(x)$ a reduced set of Gaussians that does not need to approximate the learned movement but just to optimize the trajectory. Thus this reduced set is less constrained and can have wider kernels in order to avoid high-frequency oscillations coming from random exploration. $\boldsymbol{\Theta}_\mathbf{e}$ can be initialized to zero and be updated as in any learning algorithm in literature, such as policy gradients [13] or path integral approaches [15]. In addition, the new kernels $\mathbf{g}_\mathbf{e}(x)$ do not have to be the same as $\mathbf{g}_\mathbf{0}(x)$, so we propose to take:

$$(g_e)_i(x) = \frac{\phi_i(x)}{\sum_j \phi_j(x)} x(1-x)^2, \tag{5}$$

ϕ_j being the Gaussian kernels, and the term $(1-x)^2$ reducing the acceleration at the beginning of motion.

C. Effect of parameter reduction on exploration trajectories

In order to compare our two proposals with the standard way of exploring parameters, we learned a motion of a 7-DoF WAM robot. We obtain values for the weights of the robot's second joint's DMP, with different exploration methods for a 20 seconds trajectory with 30 Gaussian kernels. The *standard* exploration (all Gaussians explored) adds a normal random value to each weight of the 30 Gaussians, while the *relevant* directions strategy (as in Section II-A) only takes the 10 most important parameter vectors, and the *dual-layer* (Section II-B) uses a second layer of only 5 Gaussian kernels following equation (5).

In Fig. 3 we can see samples of exploration with different frameworks. The standard way of exploring can be dangerous in a real robot, due to large oscillations, while the other approaches are better behaved to operate out of simulation. This fact is more relevant when a simulation framework is not available, making the exploration magnitude of parameters critical, as a large value would result in large oscillations, while a small one may need a lot of updates to reach convergence.

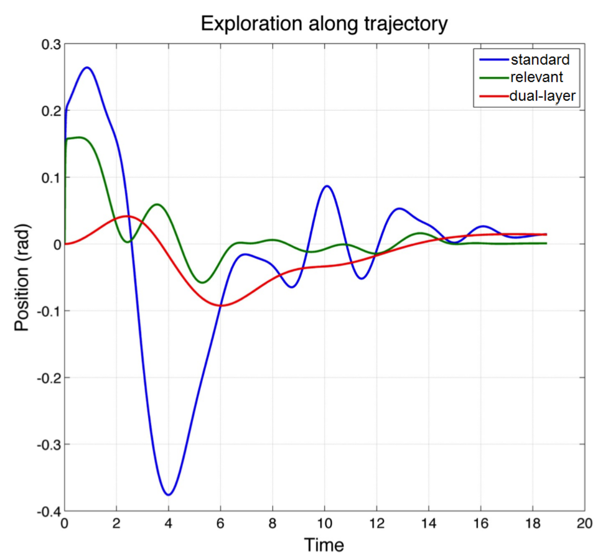

Fig. 3. Joint position variation wrt. initial movement. As we can see, the standard exploration may not only present an abrupt change at the start of the motion, but also strong oscillations along the whole trajectory.

III. DMP Coordination

In [24], a coordination framework for DMP was presented, where a robot's primitives were coupled with a coordination matrix, which was learned with RL algorithms. Kormushev et al. [25] extended this work, but with the sole objective of learning a square coordination matrix, rather than the motion parameters.

We now propose to use a coordination matrix in order to decrease the number of actuated DoF which will not necessarily be squared, and may be used to reduce the number of parameters further.

A. Coordination Matrix

Let $\mathbf{y}$ be the joint vector of a robot. We can rewrite Eq. (1) as:

$$\ddot{\mathbf{y}}/\tau^2 = \alpha_z \left(\beta_z \left(\mathbf{G} - \mathbf{y}\right) - \dot{\mathbf{y}}/\tau\right) + \mathbf{f}_{co}(x), \tag{6}$$

where $\mathbf{f}_{co}(x)$ is:

$$\mathbf{f}_{co}(x) = K_{co} \cdot \mathbf{\Theta} \cdot \mathbf{g}(x), \tag{7}$$

K_{co} being a $(d \times r)$ matrix, with d the number of robot DoF and $r \leq d$ a reduced dimensionality. $\mathbf{\Theta}$ is a $(r \times N_f)$ matrix with the motion parameters, and $\mathbf{g}(x)$ a N_f-dimensional vector with the Gaussian values. Note that, with this representation, we have r movement primitives that encode the d-dimensional acceleration command vector $\mathbf{f}_{co}(x)$. In order to learn this coordination matrix, we need an initial guess, but also an algorithm to update it and eliminate unnecessary degrees of freedom from the motion, according to the reward/cost obtained.

B. Learning Coordination Matrices with PCA

An initial guess for the coordination matrix K_{co} is to perform a Principal Component Analysis (PCA) over the demonstrated values of $\mathbf{f}(x)$ (see Eq(1)). Taking the matrix $\overline{\mathbf{F}}$ of size $(d \times N_t)$ as:

$$\overline{\mathbf{F}} = \begin{bmatrix} f_{de}^{(1)}\left(x_0\right) - \overline{f}_{de}^{(1)} & ... & f_{de}^{(1)}\left(x_N\right) - \overline{f}_{de}^{(1)} \\ ... & & ... \\ f_{de}^{(d)}\left(x_0\right) - \overline{f}_{de}^{(d)} & ... & f_{de}^{(d)}\left(x_N\right) - \overline{f}_{de}^{(d)} \end{bmatrix}, \tag{8}$$

$\overline{f}_{de}$ being the average over each joint component of the DMP excitation function (it is recommended to subtract this average before performing PCA). Then we can perform PCA, obtaining $\overline{\mathbf{F}} = U_{pca} \cdot \Sigma_{pca} \cdot V_{pca}^T$.

Now having set $r < d$ as a fixed value, we can take the r most relevant eigenvectors, which will be the first r columns of $U_{pca} = [\mathbf{u}_1, ..., \mathbf{u}_r, ..., \mathbf{u}_d]$ and use

$$K_{co} = [\mathbf{u}_1, ..., \mathbf{u}_r] \tag{9}$$

as coordination matrix in Eq. (7), having a reduced set of DoF of dimension r, which activate the robot joints (dimension d), minimizing the error in the reprojection $e = \|\overline{\mathbf{F}} - K_{co} \cdot \overline{\Sigma} \cdot V_{pca}^T\|_{Frob}^2$, with $\overline{\Sigma}$ the part of σ_{pca} corresponding to the first r singular values. However, this dimensionality reduction will not take any reward/cost function into consideration, so an alternative is to start with a full-rank coordination matrix and progressively reduce its dimension, according to the costs or rewards of the rollouts.

C. Adapting the Coordination Matrix through coaching

When reducing the dimensionality of the parameter space, we may be cutting off the exploration along DoF that do not have much influence on the initial motion. As we mentioned, this is an important drawback of the CMA algorithm if we do not filter the covariance updates, and could also occur in certain situations when using a non-square coordination matrix. To avoid this situation, we can modify the coordination matrix by repeating the initial motion with a compliant controller, and manually force the robot to move certain joints while reproducing it. Then we can recompute the coordination matrix, as in the previous section, by using the new data or adding the new and initial data. In this way, the robot can be easily told what to explore. The idea behind this modification is similar to many daily situations in which a teacher tells a learner a motion, and manually helps the learner in certain situations to show him how to improve his performance (e.g., teaching how to play golf).will see how this modification improves the performance of the coordination matrix.

D. Automatic Coordination Matrix fitting

In order to tune the coordination matrix once initialized as described in Section III-B, we assume we have performed N_k reproductions of motion, namely rollouts, obtaining an excitation function $\mathbf{f}_{(i,j)}^k$, for each rollout $k = 1..N_k$, timestep $i = 1..N_t$, and DoF j. Now having a cost function, we can also associate a cost-to-go C_i^k to each rollout and timestep, and associate a probability P_i^k to each rollout and timestep as it is done by the PI2 algorithm as in[16] (see Appendix). We can then obtain a new matrix $\mathbf{F}_{co}$ defined as:

$$\mathbf{F}_{co}^{new} = \begin{bmatrix} \sum_{k=1}^{N_k} \mathbf{f}_{(1,1)}^k P_1^k & ... & \sum_{k=1}^{N_k} \mathbf{f}_{(N_t,1)}^k P_{N_t}^k \\ ... & & ... \\ \sum_{k=1}^{N_k} \mathbf{f}_{(1,d)}^k P_1^k & ... & \sum_{k=1}^{N_k} \mathbf{f}_{(N_t,d)}^k P_{N_t}^k \end{bmatrix}, \tag{10}$$

which is a $(d \times N_t)$ matrix containing information of the excitation functions, weighted by their relative importance according to the rollout result and obtain a coordination matrix K_{co} as in the previous section. However, the parameter matrix $\mathbf{\Theta}$ has to be redefined in order to represent the same excitation function, by setting

$$K_{co}^{new}\mathbf{\Theta}^{new} \simeq K_{co}\mathbf{\Theta} \Rightarrow \mathbf{\Theta}^{new} \simeq (K_{co}^{new})^{\dagger} K_{co}\mathbf{\Theta}, \tag{11}$$

$\dagger$ indicating the Moore-Penrose pseudoinverse.

E. Eliminating irrelevant degrees of freedom

In RL, the task the robot tries to learn does not always necessary depend on all the degrees of freedom of the robot we are using to learn it. For example, if we want to track a cartesian position with a 7-DoF robot, it is likely that some degrees of freedom, which mainly alter the end-effector's orientation, may not affect the outcome of the task. However, these DoF are still considered all through the learning of the task, causing unnecessary motions which may slow down the

Algorithm 1 Coordination Matrix Update (CMU)

Input:
Rollout and timestep probabilities P_i^k, $i = 1..N_t$, $k = 1..N_k$.
Excitation function $\mathbf{f}_{(i,j)}^k$, $j = 1..d$.
Previous update (or initial) excitation function $\mathbf{F}_{co}$.
Current K_{co}.
Matrix dimension r.
DoF discarding threshold η.
Current DMP parameter matrix $\mathbf{\Theta}$.

Initialize $\mathbf{F}_{co}^{new} = \mathbf{0}_{N_t \times d}$
Compute $\mathbf{F}_{co}^{new}$ as in Eq. (10)
Filter excitation matrix: $\mathbf{F}_{co}^{new} = \alpha \mathbf{F}_{co}^{new} + (1-\alpha)\mathbf{F}_{co}$
Subtract average as in Eq. (8)
Perform PCA and obtain $U_{pca} = [\mathbf{u}_1, ..., \mathbf{u}_r, ..., \mathbf{u}_d]$ (see Eq.(9)), $S_{pca} = [\sigma_1, ..., \sigma_r, ..., \sigma_d]$
if $\sigma_1/\sigma_r > \eta$ **then**
    r=r-1
end if
$K_{co}^{new} = [\mathbf{u}_1, ..., \mathbf{u}_r]$
Reassign weights: $\mathbf{\Theta}^{new} = (K_{co}^{new})^{\dagger} K_{co} \mathbf{\Theta}$

learning process or generate a final solution in which a part of the motion was not necessary. In some cases, we might not even know what the true space is, it probably being a lower-dimensional subspace of our considered task space.

For this reason, the authors think that the main use of a coordination matrix should be to remove those unnecessary degrees of freedom, and the coordination matrix, as built in Section III-D, can easily provide such result. Given a threshold η for the ratio of the maximum and minimum singular values of $\mathbf{F}_{co}^{new}$, we can discard the last column of the coordination matrix if those singular values verify $\sigma_1/\sigma_r < \eta$.

In Algorithm 1, we show the process of updating and reducing the coordination matrix.

IV. Experimentation

In order to check how our proposals perform, we used a PI2 algorithm [15], [16], [17] with the different exploration strategies presented. The PI2 algorithm is derived from stochastic optimal control principles and performs well in many situations. It executes several updates consisting of a certain number of rollouts each and, after each epoch, the new parameters are computed with those of each rollout and the corresponding weights. One of the advantages of the PI2 algorithm is that it only requires two extra parameters: the cost comparison eliteness λ (see Appendix), used to weight the Gaussian parameters according to the rollout cost, and the exploration variance. This exploration variance had been set manually by trial-and-error until the CMA [22] approach was presented, in which the exploration covariance matrix was updated after each epoch (set of rollouts). This approach helps to modify the exploration magnitudes in cases where one does not have a good initial guess, but it does not significantly improve performance when such initial guess is available. CMA needs more rollouts per policy update, and it is recommended to be used with a base-level exploration to avoid premature convergence or that the algorithm favours too much a single parameter direction if the eliteness parameter is too large.

A. 10-DoF planar arm

In order to have comparable results, we firstly tested our proposals with the benchmark used in [16], in which a planar 10-DoF arm ($D = 10$) of $1m$ length, with $0.1m$ links moves from an initial position to a goal in a $0.5s$ trajectory at $50Hz$ sampling. The objective of this task is to pass through point $(0.5, 0.5)$ at time $t = 0.3s$ and end at the goal position. The cost function for this task was taken from [16] as:

$$J(\tau_{t_i}) = \delta(t-0.3) \cdot \left((x_t - 0.5)^2 + (y_t - 0.5)^2\right) + \delta(t-0.5) \cdot \left((x_t)^2 + (y_t - 0.6955)^2\right) + \frac{\sum_{s=1}^{d} (d+1-s)(\ddot{a}_t^2)}{\sum_{s=1}^{d} (d+1-s)} \cdot 10^{-6} \tag{12}$$

This cost function penalizes accelerations in the first joints, which move the whole robot. The PI2 algorithm parameters were set as: eliteness $\lambda = 10$, 100 epochs of 10 rollouts each, with a starting exploration of $\left[10^4, 10^5, 10^4\right]$ for the *standard*, *dual-layer* and *relevant* exploration strategies described in Section II, which were the parameters showing the best performance for each case. For both the *dual-layer* and *relevant* cases, we used a reduced set of 5 exploration parameters per DoF. In addition, we plot the results using a reduced dimensionality of 5, as explained in Section III-B. We also used a decay term of $\gamma_e = 0.995$ for the exploration magnitude.

We performed every experiment 10 times in order to reduce noise from *lucky/unlucky* guesses, and plotted the results in Fig. 4 in a logarithmic scale. We can see that the *relevant directions* framework has a similar behaviour to the *standard* exploration method, while having to perform less computations and storing less data and parameters, thus making a more efficient use of computation resources. Lastly, the *dual-layer* method for exploration converges faster than the standard exploration framework, while requiring less resources, despite converging to a slightly worse solution.

B. Robot arm following a circle

For a second ilustrative example, we used a 7-DoF WAM arm. We manually taught the real robot to follow a circular trajectory with its wrist in the cartesian space. From this initial motion, which was very inaccurate, the best circle fitting the initial trajectory was computed, and a cost function consisting in a point-to-point deviation from that circle, plus an acceleration-penalizing term, was taken. The DMP for the 7 DoF were learned with 30 Gaussians each, as the trajectory to be learned was a complex 20-second movement. Note that, in this task, the objective is to track a circular trajectory with the robot's wrist, thus since the 3 last DoF of the robot don't have any influence on the cost function other than the acceleration terms, one could eliminate those DoF and still obtain the same optimal solution. With this experiment, we

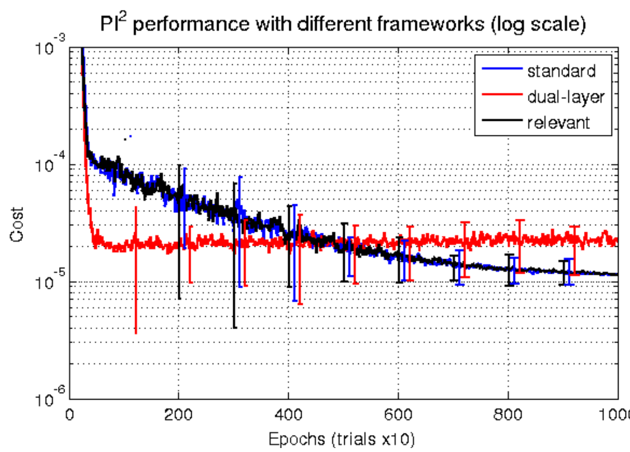

Fig. 4. Waypoint task results. Averaging over 10 repetitions. Note the non-symmetry of the standard deviation due to the log-scale.

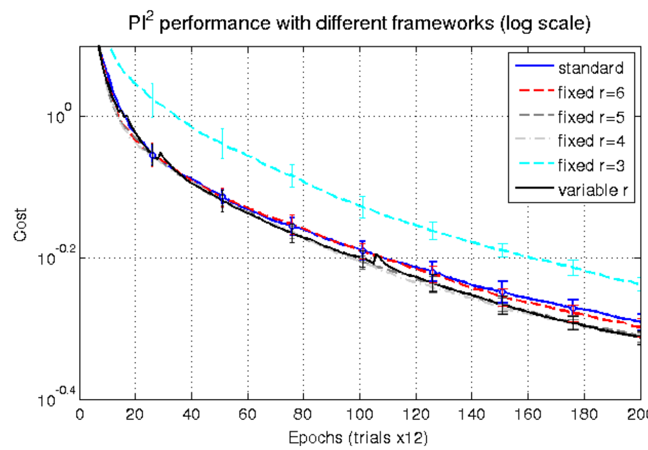

Fig. 5. Average cost (of 10 experiments) of the policy after each update in the circle-tracking experiment with 200 policy updates.

pretended to show if the proposed CMU algorithm is capable of efficiently detecting those unnecessary DoF.

The task was learned with the PI2 algorithm with the standard DMP framework, initialized with a coordination matrix of reduced dimension as explained in Section III-B, with a fixed value of $r = 3, 4, 5, 6$. We also applied the algorithm proposed in Section III-E, while using the CMA in all the simulations and the results can be seen in Fig. 5.

We can observe that, due to the trivial DoF elimination for this task, directly removing the last 3 DoF of the robot improves the algorithm's performance. However, removing a fourth DoF to leave only 3 hampers the RL outcome, yielding a bad result. In addition, we see that the automatic DoF elimination described Section III-E obtains a result which is as good as starting with the elimination of the 3 unnecessary DoF. However, it presents some discontinuities in the cost function, due to the loss of information which can occur when reducing the coordination matrix dimension, but nevertheless, after such losses of information, the algorithm quickly recovers from the loss of performance and still obtains good results. These discontinuities occur due to the difficulty of assigning costs or rewards to certain DoF, and a high-reward (or low-cost) rollout may include noise coming from those unnecessary DoF, thus columns from the K_{co} matrix which may still contain relevant information may be eliminated. That information is then lost, but can be quickly recovered in most cases, since those columns are the ones with the least energy on the PCA in Eq. (9).

V. Conclusions

Dynamical Movement Primitives are an adaptive alternative to precomputed trajectory representations. However, when using them with reinforcement learning algorithms, several issues arise:

- **The number of kernels**. In order to efficiently fit an initial motion, several kernels are required. Each kernel will later require a weighting parameter, thus there is a trade-off between the accuracy with which the trajectory is fitted and the number of parameters to optimize so that fewest reproductions are required to improve the performance.
- **Feasibility of reproduction**. When a simulator is not available, the real reproduction of a task with a robot may be dangerous due to large accelerations, which can appear when the exploration on close kernels has different signs. This fact can force us to significantly reduce the exploration on the parameters in order to maintain safety when a model of the environment or the task is not available.
- **Exploration range**. The exploration range used with DMP, in particular using the PI2 algorithm, is often hard to tune. CMA offers a way to guess an appropiate range, but CMA also has a tradeoff between losing unexplored directions in the parameter space and filter the covariance matrix, hampering its performance, as well as the need to add a minimum exploration to avoid premature convergence.

To address these issues, in this paper we have proposed two alternatives:

- Exploring only along the most relevant dimensions. These relevant dimensions are chosen according to the singular values of the normalized kernels. This strategy produces similar results as using all dimensions, while being more resource-efficient since it computes less Gaussians per timestep.
- Using a second layer of Gaussians. We also proposed to split the trajectory fitting problem during exploration. We introduce a second layer of Gaussians, which does not need to have the same kernels, and has a lower dimensionality than the first layer. The use of any kind of kernel can give us exploration motions with less acceleration, and the learning algorithm performs faster, as we have seen in the experimentation section.

In addition, following the intuition that motions are usually performed in a subspace of all possible motions, we proposed to use non-square coordination matrices for learning. These non-square matrices can be initially learned with a PCA decomposition of the task, and later modified by guiding the robot to find the relevant parameters to be learned. In addition, we proposed an algorithm to automatically update this coordination matrix and eliminate its unnecessary DoF. Despite occasionally experiencing discontinuities in the cost/reward vs updates curve, this last algorithm showed a better performance than the standard algorithm, while requiring less computation time. This proposed algorithm may become even more useful when working with highly articulated robots learning simple tasks.

These smaller coordination matrices can be combined with the other two dimensionality reduction strategies in order to attain even better results in less computation time. Our future work includes to devise new ways to find a relation between the cost function and each DoF, in order to minimize further the loss of information when reducing the dimensionality. Also prior to learning, we can use several demonstrations to find a better representation of the relevant directions in the parameter space [26]. Moreover, we will study when to reduce dimensionality (e.g. the parameter η) and perform a benchmark comparison with other frameworks tackling a similar problem, such as DMPSynergies [27] or Iterative Improvement [28].

APPENDIX

A. Converting the cost function to rollout probabilities

Given the instantaneous cost of a RL algorithm, such as Eq. (12), the cost-to-go at timestep i of an experiment k is defined [15] as the sum of the instantaneous cost of the reimaning trajectory:

$$S_i^k = \sum_{j=i}^{N_t} J(\tau_{t_j}).$$

These terms S_i^k can be normalized by:

$$\hat{S}_i^k = \frac{S_i^k - \min_k(S_i^k)}{\max_k(S_i^k) - \min_k(S_i^k)}$$

and converted to a probability associated to each rollout and timestep, which will be directly related to the relative reward of that rollout over all the experiments considered:

$$P_i^k = \exp\left(-\frac{1}{\lambda}\hat{S}_i^k\right) / \sum_{s=1}^{N_k} \exp\left(-\frac{1}{\lambda}\hat{S}_i^s\right),$$

where λ is called eliteness and represents how are these probabilities scaled.

REFERENCES

[1] A. J. Ijspeert, J. Nakanishi and S. Schaal. “Movement Imitation with Nonlinear Dyamical Systems in Humanoid Robots”. *IEEE ICRA*, pp 1398-1403 (2002).

[2] J. Kober, K. Mülling, O. Krömer, C. H. Lampert and B. Schölkopf. “Movement Templates for Learning of Hitting and Batting”. *IEEE ICRA*, pp 853 - 858 (2010).

[3] L. Rozo, P. Jiménez and C. Torras. “Robot Learning from Demonstration of Force-based Tasks with Multiple Solution Trajectories”. *15th IEEE Int. Conf. on Advanced Robotics*, pp 124-129 (2011).

[4] L. Rozo, P. Jiménez and C. Torras. “A robot learning from demonstration framework to perform force-based manipulation tasks”. *Intelligent Service Robotics*, vol. 6, no 1, pp 33-51 (2013).

[5] S. M. Khansari-Zadeh, A. Billard. “Learning Stable Nonlinear Dynamical Systems with Gaussian Mixture Models”. *IEEE Trans. on Robotics*, vol. 27, no 5, pp 943-957 (2011).

[6] S. Klanke, S. Vijayakumar and S. Schaal. “A library for Locally Weighted Projection Regression”. *Journal of Machine Learning Research*, vol. 9, pp. 623-626 (2008).

[7] A. Billard, S. Calinon, R. Dillmann and S. Schaal. “Robot Programming by Demonstration”. *Springer Handbook of Robotics*, part G, chapter 59.

[8] D. Nguyen-Tuong and J. Peters. “Learning Robot Dynamics for Computed Torque Control Using Local Gaussian Processes Regression”. *ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems*, pp 59-64 (2008).

[9] J. Kober, A. Wilhem, E. Oztop and J. Peters. “Reinforcement Learning to Adjust Parametrized Motor Primitives to New Situations”. *Autonomous Robots*, vol 33, no 4, pp 361-370 (2012).

[10] B.Nemec, D. forte, R. Vuga, M Tamosiunaite, F. Wörgötter and A. Ude. “Applying statistical generalization to determine search direction for reinforcement learning of movement primitives”. *IEEE-RAS Humanoids*, pp 65-70 (2012).

[11] 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).

[12] A. J. Ijspeert, J. Nakanishi and S. Schaal. “Movement imitation with nonlinear dyamical systems in humanoid robots”. *IEEE ICRA*, pp 1398-1403, 2002.

[13] J. Peters, S. Schaal. “Policy gradient methods for robotics ”. *IEEE/RSJ IROS*, pp 2219-2225, 2006.

[14] 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).

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

[16] 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).

[17] 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.

[18] J. Peters, K. Mülling and Y. Altün. “Relative Entropy Policy Search”. *Twenty-Fourth National Conf. on Artificial Intelligence*, track 15, pp 182-189 (2011).

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

[20] J. Kober, D. Banell, J. Peters. “Reinforcement Learning in Robotics: A survey”. *Int. Journal of Robotics Research*, vol 32, no 11, pp 1238-1274 (2013).

[21] T. Kulvicius, K. Ning, M. Tamosiunaite and F. Wörgötter. “Joining Movement Sequences: Modified Dynamic Movement Primitives for Robotics Applications Exemplified on Handwritting”. *IEEE Transactions on Robotics*, vol 28, no 1, pp 145-157 (2012).

[22] F. Stulp and O. Sigaud. “Path Integral Policy Improvement with Covariance Matrix Adaptation”. *Int. Conf. on Machine Learning*, pp 281-288 (2012).

[23] F. Stulp, E. Oztop, P. Pastor, M. Beetz and S. Schaal. “Compact Models of Motor Primitive Variations for Predictable Reaching and Obstacle Avoidance”. *IEEE-RAS Humanoids*, pp 589-595 (2009).

[24] D. Pardo. “Learning rest-to-rest Motor Coordination in Articulated Mobile Robots”. *Ph.D. Dissertation*, (2009).

[25] P. Kormushev, S. Calinon and G. Caldwell. “Robot Motor Skill Coordination with EM-based Reinforcement Learning”. *IEEE IROS*, pp 3232 - 3237 (2010).

[26] T. Matsubara, S-H Hyon and J. Morimoto. “Learning parametric dynamic movement primitives from multiple demonstrations”. *Neural Networks*, vol 24, no 5, pp 493-500 (2011).

[27] E. Rückert and A. dÁvellam. “Learned parametrized dynamic movement primitives with shared synergies for controlling robotic and muscoskeletal systems”. *Frontiers in Computational Neuroscience*, vol 7, pp 1-17 (2013).

[28] A. Jain, B. Wojcik T. Joachims, A. Saxena. “Learning Trajectory Preferences for Manipulators via Iterative Improvement”. *Advances in Neural Information Processing Systems*, pp 575-583 (2013).