# Learning Compact Parameterized Skills with a Single Regression

Freek Stulp[1,2], Gennaro Raiola[1,3], Antoine Hoarau[1,2], Serena Ivaldi[4], Olivier Sigaud[4]

***Abstract*— One of the long-term challenges of programming by demonstration is achieving generality, i.e. automatically adapting the reproduced behavior to novel situations. A common approach for achieving generality is to learn parameterizable skills from multiple demonstrations for different situations. In this paper, we generalize recent approaches on learning parameterizable skills based on dynamical movement primitives (DMPs), such that task parameters are also passed as inputs to the function approximator of the DMP. This leads to a more general, flexible, and compact representation of parameterizable skills, as demonstrated by our empirical evaluation on the iCub and Meka humanoid robots.**

## I. Introduction

One of the main appeals of programming by demonstration is that it provides an intuitive method – for experts and lay people alike – to teach a robot new skills. This alleviates the need to program such skills by hand, which is tedious and error-prone, and requires expert knowledge. One of the long-term challenges of programming by demonstration is achieving *generality*, i.e. automatically adapting the reproduced behavior to novel situations [1]. Achieving generality hinges on providing demonstrations for different situations. Methods for trajectory-level skill encoding, the focus of this paper, are able to leverage multiple demonstrated trajectories to determine relevant movement constraints [2], generalize across start states [3], [4], or for adaptation to varying task parameters [5]–[9].

Fig. 1 illustrates the latter approach with one of the tasks considered in our evaluation. The left image depicts a human demonstrating a trajectory for grasping a box. Here, the task parameter vector $\mathbf{q}$ represents the 3D pose of the box, which is detected with a Kinect. Given $K$=30 such demonstrations ($\{\tau_k, \mathbf{q}_k\}_{k=1}^{K}$), a *parameterizable skill* is learned ($\pi(\mathbf{x}, \mathbf{q}) \mapsto \mathbf{a}$, with states $\mathbf{x}$ and actions $\mathbf{a}$), which is able to generalize and generate trajectories for novel box poses.

In recent years, dynamical movement primitives (DMPs) [10] have become a common underlying skill representation for learning parameterized skills [5]–[9]. In most current approaches to parameterized skills, movement generation is a two-step process: 1) determine the skill parameters from the task parameters $\theta = m(\mathbf{q})$, 2) execute the DMP with parameters $\theta$.

Our key innovation is to merge these two steps into one. We do so by generalizing the DMP formulation, such that task parameters are passed directly and on-line to the DMP's function approximator. Training a DMP from multiple demonstrations is done with one single regression, rather than current approaches which require two separate regressions [5]–[8]. Some of the important features of this reformulation are that: 1) **Generality** – it generalizes several previous approaches [4]–[7]. 2) **Flexibility** – different function approximator implementations may be used interchangeably. 3) **Compactness** – parameterizable skills can be represented more compactly.

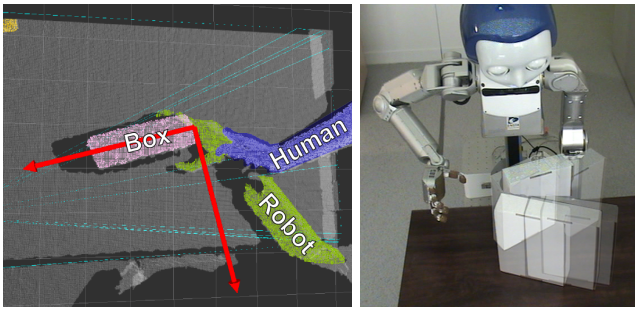



Fig. 1. Left: using kinesthetic teaching to demonstrate a trajectory $\tau_k$ for grasping the box. The Meka humanoid robot detects the task parameters $\mathbf{q}_k$ – the 3D box pose $\langle x, y, \text{yaw} \rangle$ – with a Kinect. Right: overlay of 8 of $K$=30 box poses for which a trajectory was demonstrated. This yields the training set $\{\tau_k, \mathbf{q}_k\}_{k=1}^{K}$.

The rest of this paper is structured as follows. In the next section, we present the DMP formulation and related work. Our DMP reformulation with augmented function approximators is presented in Section III. Section IV contains an evaluation of our approach, and we conclude with Section V.

## II. Foundations and Related Work

In this section we present DMPs [10], on which our method is based, and related work on parameterizable skills.

### A. Dynamical Movement Primitives

DMPs combine a feedback controller with an open loop forcing term consisting of a function approximator $h$ [10]:

$$\tau \ddot{x}_t = \underbrace{\alpha(\beta(x^g - x_t) - \dot{x}_t)}_{\text{feedback controller}} + \underbrace{s_t(x^g - x_0)h(s_t)}_{\text{open loop controller}} \quad (1)$$

$$\tau \dot{s}_t = -\alpha_s s_t \qquad \text{canonical system} \quad (2)$$

When integrated over time, DMPs generate trajectories $[x_t \ \dot{x}_t \ \ddot{x}_t]$, which, for instance, are used as a desired joint angle or desired end-effector coordinate. The main advantage of DMPs is that the open loop controller is able to represent arbitrary time-dependent movements, whereas the feedback controller makes the resulting motion robust to perturbations

[1]Robotics and Computer Vision, ENSTA-ParisTech, Paris, France
[2]FLOWERS Team, INRIA Bordeaux Sud-Ouest, Talence, France
[3]Pal Robotics S.L., Barcelona, Spain
[4]ISIR, Université Pierre Marie Curie CNRS UMR 7222, Paris, France
This work was supported by the French ANR program MACSi (ANR 2010 BLAN 0216 01), the INRIA/ADT project Carroman, and the European Commission within the CoDyCo project (FP7-ICT-2011-9, No. 600716).

and ensures convergence towards the goal $x^g$. This convergence also depends on the multiplication of the output of $h$ with the movement phase $s_t$, which is 1 at the beginning of the movement and decays exponentially towards 0. The phase $s$ is thus an alternative 1D representation of time.

The function approximator $h$ takes the movement phase $s$ as an input, and is commonly implemented as a weighted combination of $B$ Gaussian basis functions with centers $c_{b=1:B}$ and widths $\sigma_{b=1:B}$. The vector of weights $\mathbf{w}_{b=1:B}$ is thus also of length $B$. Since $\mathbf{w}$ determines the shape of the movement between $x_0$ and $x^g$, we refer to them as the 'shape parameters'. Since $h_{\mathbf{w}}$ is linear in the parameters $\mathbf{w}$, it can easily be learned through linear regression.

$$h_{\mathbf{w}}(s) = \frac{\sum_{b=1}^{B} \mathbf{\Psi}_b(s)[\mathbf{w}]_b}{\sum_{b=1}^{B} \mathbf{\Psi}_b(s)} \qquad \text{Function approx.} \quad (3)$$

$$\mathbf{\Psi}_b(s) = \exp\left(\frac{-(s-c_b)^2}{2\sigma_b^2}\right) \qquad \text{Gaussian kernel} \quad (4)$$

Equation (1) describes the evolution of a 1-dimensional variable $x$. Multi-dimensional DMPs, that represent for instance the joint angles of a 7-DOF arm, or its 3-DOF end-effector position, are acquired by coupling several systems as in (1) with one canonical system (2).

Our key innovation is to also pass task parameters to the function approximator of the DMP, which enables the DMP to generalize to novel task parameterizations, for instance previously unseen box poses. Whilst leaving the derivation to Section III, we provide a preview of the resulting DMP formulation in (5).

$$\tau\ddot{x}_t = \alpha(\beta(x^g - x_t) - \dot{x}_t) + s_t(x^g - x_0)h(s_t, \mathbf{q}_t) \quad (5)$$

### B. Parameterized Skills

For a classification of related work on parameterizable skills, it is important to distinguish between three types of skill parameters when DMPs are used as the underlying skill representation: 1) *DMP shape parameters* are the weights $\mathbf{w}$ associated with the basis function, cf. (3). 2) *DMP meta-parameters* are all DMP parameters *except* for the weights, i.e. $\mathbf{x}^g$, $\tau$, $\alpha$, etc. This term was introduced by Kober et al. [9]. 3) *External parameters* are parameters of the skill not specific to the DMP formulation. For instance, in a ball-throwing task, da Silva et al. [7] specify $\lambda$ as the time when a robot lets go of the ball, and include this in the skill parameter vector. The generic term for combinations of these parameters is 'skill parameter vector', and is denoted $\theta$.

As DMPs are not usually able to generalize to varying task parameters $\mathbf{q}$, recent approaches use multiple demonstrations to learn *parameterizable skills* to achieve such generalization [5]–[8]. Learning parameterizable skills is done by first acquiring $K$ DMPs – one for each of the task parameterizations $\mathbf{q}_{k=1:K}$ – with supervised learning [5], [6], or optimization [7]. This yields a set of skill parameters associated with the corresponding task parameters $\{\theta_k, \mathbf{q}_k\}_{k=1:K}$.

Then, a regression from task parameters to skill parameters is learned, given the data in $\{\theta_k, \mathbf{q}_k\}_{k=1:K}$. For the shape parameters $\mathbf{w}$, this corresponds to $B$ functions $m_b(\mathbf{q})$ that map task parameters $\mathbf{q}$ to the $b^{th}$ weight $[\mathbf{w}]_b$:

$$[\mathbf{w}]_b = m_b(\mathbf{q}) \quad (6)$$

Ude et al. [5] perform this second regression with a combination of Locally Weighted Regression (LWR) for the shape parameters $\mathbf{w}$, and Gaussian Process Regression (GPR) for the other skill parameters. Forte et al. [6] build on [5] and use GPR for $\mathbf{w}$ also. Da Silva et al. [7] use a combination of ISOMAP and Support Vector Machines. An interesting aspect of [7] is that different regressors are learned for different parts of the parameter space, thus avoiding the problem of averaging across multiple skill parameters that achieve the same task. For instance, reaching around an obstacle will succeed by going around it to the left *or* right, but the average of these two motions will not succeed. The method of da Silva et al. [7] may be readily combined with [5], [6], [8], and also our work, to avoid such averaging.

Executing such a parameterized skill is a 2-step procedure: 1) compute the skill parameters $\theta$ from the task parameters $\mathbf{q}$ with (6) 2) execute the DMP with parameters $\theta$. For this reason, we refer to them as "2-step" skills.

It is important to realize that learning $[\mathbf{w}]_b = m_b(\mathbf{q})$ in the second step will work *only* if the number (and locations) of basis functions $B$ resulting from the first step is consistent for each of the $K$ task instances. For example, if the DMP for the first demonstration $k$=1 has $B_k$=10 weights, and the second one $k$=2 has $B_k$=12 weights, we cannot learn the mapping $[\mathbf{w}]_b = m_b(\mathbf{q})$ because there is an inconsistency in what the index $b$ refers to. For this reason, the number of basis functions (and also their centers and widths) resulting from the first step must be fixed. This *precludes* the use of for instance Locally Weighted Projection Regression (LWPR) [12] for the first step, because LWPR determines these parameters itself from the data. Rather, in the first step of the 2-step approach one is *obliged* to use representations and function approximators where these parameters are fixed (e.g. LWR, used to implement the first step in [5], [6]). As we will see in Section III, our reformulation does not enforce such constraints, and other function approximator implementations (e.g. LWPR and GPR) may readily be used.

### C. Further Related Work

Our approach is a generalization of [5], [6] discussed above, but also generalizes other works. Parlaktuna et al. [4] consider the special case where task parameters $\mathbf{q}$ are considered to be the time-varying state of the DMP $\mathbf{q} = \langle \mathbf{x}\ \dot{\mathbf{x}} \rangle$. Matsubara et al. [8] also pass task parameters $\mathbf{q}$ (which they call 'style parameter') to function approximator $h$, and consider the special case where $h_{\mathbf{w}}(s, \mathbf{q}) = \sum_{b=1}^{B} \mathbf{\Psi}_b(s)[\mathbf{w}]_b[\mathbf{q}]_b / \sum_{b=1}^{B} \mathbf{\Psi}_b(s)$. Our approach generalizes this by allowing arbitrary implementations for $h(s, \mathbf{q})$. An interesting aspect of [8] is that task parameters are not provided, but rather extracted from the data itself through principal components analysis. We trivially generalize the standard DMP formulation in [10], where $\mathbf{q} = \varnothing$.

Kober et al. [9] present an altogether different approach for learning to map task parameters to DMP meta-parameters based on cost-regularized kernel regression. They consider the shape parameters $\mathbf{w}$ to be fixed, and learn only the meta-parameters. Kupcsik et al. [11] apply policy search to parameterized skills, where the first and second step are called the upper-level and lower-level policy respectively.

DMPs represent a *trajectory-based* approach to motion generation. The parameterizable skills in this paper extend this by enabling the DMP to be adapted to different task parameters. This is different from *state-based* approaches to motion generation [3], [4], which extend trajectory-based methods by adapting the motion to a time-varying state input.

## III. Parameterizable Skills with Augmented Function Approximators

Before introducing our approach, let us briefly return to the 2-step methods [5]–[7] presented in Section II-B, which compute the weight parameters in a separate step with $[\mathbf{w}]_b = m_b(\mathbf{q})$. In particular, let us consider [5], where the mapping $m$ is computed as a sum of basis functions (7), in the same way as the function approximator $h$ of the DMP ((8), repeated from (3)). For the mapping $m$, each function $m_b$ uses its own parameter vector $\mathbf{v}_b$ to compute the DMP basis function weight $[\mathbf{w}]_b$.

$$[\mathbf{w}]_b = m_b(\mathbf{q}) = \frac{\sum_{d=1}^{B_q} \mathbf{\Phi}_d(\mathbf{q})[\mathbf{v}^b]_d}{\sum_{d=1}^{B_q} \mathbf{\Phi}_d(q)} \qquad \text{Step 1} \quad (7)$$

$$h_{\mathbf{w}}(s) = \frac{\sum_{b=1}^{B_s} \mathbf{\Psi}_b(s)[\mathbf{w}]_b}{\sum_{b=1}^{B_s} \mathbf{\Psi}_b(s)} \qquad \text{Step 2 (uses } \mathbf{w}) \quad (8)$$

The 2-step procedure of first computing weights $\mathbf{w}$ from $\mathbf{q}$, and then executing the DMP with $\mathbf{w}$ is visualized in Fig. 2.

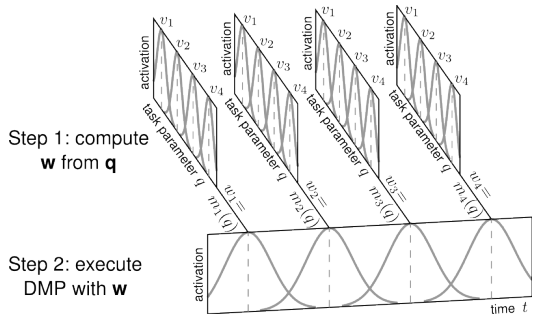



Fig. 2. Visualization of the two-step procedure for parameterized skills.

If the set of parameter vectors $\{\mathbf{v}\}_{b=1}^{B}$ are of equal length, as in [5], we may write it as a matrix $\mathbf{W}$ with $B$ rows:

$$[\mathbf{w}]_b = m_b(\mathbf{q}) = \frac{\sum_{d=1}^{B_q} \mathbf{\Phi}_d(\mathbf{q})[\mathbf{W}]_{b,d}}{\sum_{d=1}^{B_q} \mathbf{\Phi}_d(q)} \quad (9)$$

The first step in our new formulation for parameterizable skills with augmented function approximators is to plug (9) into (8):

$$h_{\mathbf{W}}(s,q) = \frac{\sum_{b=1}^{B_s}\sum_{d=1}^{B_q} \mathbf{\Psi}_b(s)\mathbf{\Phi}_d(q)[\mathbf{W}]_{b,d}}{\sum_{b=1}^{B_s}\sum_{d=1}^{B_q} \mathbf{\Psi}_b(s)\mathbf{\Phi}_d(q)}. \quad (10)$$

The second step is to expand the basis functions into a higher-dimensional space. To do so, we use Gaussian kernels for $\mathbf{\Phi}_d(q)$, i.e. $\mathbf{\Phi}_d(q) = \mathbf{\Psi}_d(q)$. Ude et al. [5] use tricube kernels, but note that this selection "is rarely critical for the performance of LWR". We then exploit the fact that multiplying two 1D Gaussians yields a 2D Gaussian.

$$\begin{aligned}\mathbf{\Psi}_b(s)\mathbf{\Psi}_d(q) &= \exp\left(\frac{-(s-c_b)^2}{2\sigma_b^2}\right)\exp\left(\frac{-(q-c_d)^2}{2\sigma_d^2}\right) \quad (11)\\ &= \exp\left(-\frac{1}{2}\begin{bmatrix} s-c_b \\ q-c_d\end{bmatrix}^{\intercal}\begin{bmatrix}\sigma_b^2 & 0\\ 0 & \sigma_d^2\end{bmatrix}^{-1}\begin{bmatrix} s-c_b \\ q-c_d\end{bmatrix}\right)\\ &= \exp\left(-\frac{1}{2}\left(\begin{bmatrix}s\\q\end{bmatrix}-\mathbf{c}_{b,d}\right)^{\intercal}\mathbf{\Sigma}_{b,d}\left(\begin{bmatrix}s\\q\end{bmatrix}-\mathbf{c}_{b,d}\right)\right)\\ &= \mathbf{\Psi}_{b,d}(s,q) \quad (12)\end{aligned}$$

Thus, the multiplication of the two univariate Gaussians kernels $\mathbf{\Psi}_b(s)\mathbf{\Psi}_d(q)$ has been simplified into one multivariate Gaussian with expanded kernels $\mathbf{\Psi}_{b,d}(s,q)$. Plugging the expanded basis functions (12) into the function approximator (10) yields:

$$h_{\mathbf{W}}(s,\mathbf{q}) = \frac{\sum_{b=1}^{B_s}\sum_{d=1}^{B_q} \mathbf{\Psi}_{b,d}(s,\mathbf{q})[\mathbf{W}]_{b,d}}{\sum_{b=1}^{B_s}\sum_{d=1}^{B_q} \mathbf{\Psi}_{b,d}(s,\mathbf{q})}. \quad (13)$$

These expanded basis functions are visualized in Fig. 3, where the $x$-axis represents time, and the $y$-axis a 1D task parameter $q$. The weighted activation of one basis function is acquired by *directly* computing the activation $\mathbf{\Psi}_{b,d}(s,\mathbf{q})$ in the augmented space, and multiplying it with $[\mathbf{W}]_{b,d}$. In previous approaches depicted in Fig. 2, this is done by computing the activation $\mathbf{\Psi}_b(s)$ in phase space only, and multiplying it with a weight $[\mathbf{w}]_b$ that depends on $\mathbf{q}$ with $[\mathbf{w}]_b = m_b(\mathbf{q})$.

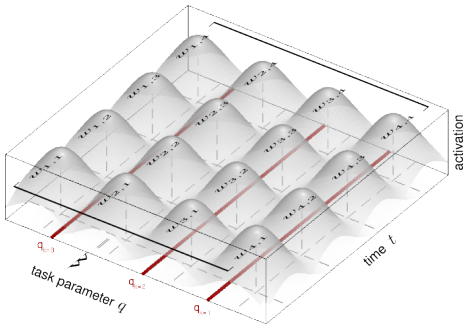



Fig. 3. Expanding the basis functions, such that they are defined in a space with dimensionality $\dim(\mathbf{q})+1$. Here, $\mathbf{q}$ is 1-dimensional for ease of visualization.

Before performing the final step of the derivation, let us briefly describe some features that this intermediate representation of the function approximator has. 1) It combines all the functions $m_{b=1:B}$ into one function $h$, and demonstrates that the function approximator may be computed directly in one step, rather than computing $[\mathbf{w}]_b = m_b(\mathbf{q})$ as an intermediate representation. 2) The function approximator no longer depends only on $s$, but also on $\mathbf{q}$. 3) Our derivation has used a 1D task parameter $q$ for brevity. The multivariate Gaussian kernels may be readily expanded for multi-dimensional task parameter vectors $\mathbf{q}$. In this case, the

kernels are defined in a $\dim(\mathbf{q})$+1 dimensional space. 4) The parameters are all stored together in the matrix $\mathbf{W}$, rather than distributed in the separate vectors $\{\mathbf{v}\}_{b=1}^{B}$.

The main advantages of our approach arise from applying the third and final step of the derivation. In [5]–[7], the number of basis functions $B_b$ and their centers must be fixed across task parameters in order to learn $[\mathbf{w}]_b = m_b(\mathbf{q})$. This is because the phase space and task parameter space are orthogonal, and considered in different steps. This constraint is inherited by the expanded basis functions; the indices $b$ and $d$ in $\mathbf{\Psi}_{b,d}(s, \mathbf{q})$ imply that the basis functions are organized on a grid, as Fig. 3 also makes obvious. However, we may overcome this limitation by organizing the $B_s \times B_q$ grid of basis functions as a 1D list of length $B_e = B_s \cdot B_q$ with index $e = (b \times d)$, as in (14). This reorganization is not possible with [5]–[7], as phase space and task space are considered in different steps, cf. Fig. 2.

$$h_{\mathbf{u}}(s, \mathbf{q}) = \frac{\sum_{e=1}^{B_e} \mathbf{\Psi}_e(s, \mathbf{q})[\mathbf{u}]_e}{\sum_{e=1}^{B_e} \mathbf{\Psi}_e(s, \mathbf{q})}. \tag{14}$$

The main consequence of this reorganization of basis function weights in a vector $\mathbf{u}$ instead of a matrix $\mathbf{W}$ is that the centers of the kernels must not necessarily be organized on a grid anymore, and can be placed anywhere in the multi-dimensional space, as illustrated in Fig. 4. Thus, task parameters for which high-frequency movement are required may be assigned more basis functions in phase space than task parameters that require very smooth movement. Algorithms such as LWPR [12] are able to determine such varying requirements automatically, and place and tune the basis function accordingly.

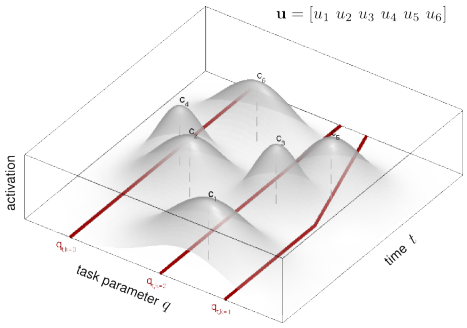



Fig. 4. Expanded basis functions with arbitrary centers, not necessarily on a grid as in Fig. 3

The automatic placement of basis functions at arbitrary locations has several advantages: 5) Alleviates the user from specifying the number of basis functions and their parameters by hand. 6) Leads to more compact representations than simply generating a grid of basis functions, because kernels are only placed where they are needed, as we demonstrate in the evaluation. 7) In fact, the function approximator $h$ need not be based on basis functions at all. It may just as well be represented with for instance a Gaussian Process, as we demonstrate in Section IV. One of the main advantages of our approach is that different implementations for $h$ may be used interchangeably.

A disadvantage is that expanding the basis functions to higher-dimensional spaces leads to the curse of dimensionality. Note however that $\mathbf{q}$ is not the *state*; it is the high dimensionality of the state that makes it a curse for reinforcement learning. Rather, $\mathbf{q}$ has the dimensionality of the task parameterization, which is determined by the user. In previous work $\dim(\mathbf{q})$ is 1 or 2 [5]–[8].

## IV. Experimental Evaluation

The aim of this evaluation is to compare the 1-Step approach with augmented function approximators over the 2-Step approach. In particular, on three different tasks, we 1) evaluate the compactness of learned parameterized skills; 2) compare different function approximators implementations and parameterizations; 3) evaluate our approach for robot control on two humanoid robot platforms.

For the 2-Step method, we use locally weighted regression (LWR) [5], [6], [10]. To emphasize the two steps, we denote it LWR$^{2S}$. For the 1-Step method, we use LWPR and GPR in the expanded space[1]. We denote these LWPR$^{1S}$ and GPR$^{1S}$. We would again like to emphasize that training the DMP parameters in the first step of the 2-Step method can only be done with a method using a fixed number of basis functions like LWR (the center of these basis functions being usually chosen to be on a grid).

### A. Task 1: Viapoint Task

The aim of the first task is to pass through a viapoint in 2D Cartesian space. This first low-dimensional simulated task is used to gather statistics over many experiments, and facilitate visual interpretation of the results, especially relevant to highlighting the compactness of the learned skills.

Training trajectories are minimum-jerk movements of duration $0.5s$, from the initial state $\mathbf{x}_0$ to the goal state $\mathbf{x}^g$ that pass through the viapoint $\mathbf{q}$. Although $\mathbf{q}$ is 2-dimensional, viapoints are constrained to lie on a 1D manifold, as depicted in Fig. 5 in the inset labeled "20 training trajectories".

Parameterizable skills were learned for an increasing number of training trajectories $k = 1, 2, \ldots 20$. The cost of reproduced movements is the distance to the viapoint at $t$=0.2$s$ of 20 separate test trajectories. Each training session was run 10 times with different test/train trajectories for randomly generated viapoints. LWR$^{2S}$ is trained with a varying total number of basis functions (32, 108, 256, 500, 864, 1372), and LWPR$^{1S}$ with varying regularization penalty terms (1.0, 0.1, 0.01, 0.001), where higher penalties lead to smoother functions, and therefore indirectly to models with less basis functions. No GPR$^{1S}$ parameter needed to be varied. The results of training these function approximators with 1, 4 and 20 training trajectories are depicted in Fig. 5.

From Fig. 5 we draw the following conclusions. 1) not surprisingly, more training trajectories (left to right graph) lead to lower costs. For LWPR$^{1S}$ the cost converges after 4 trajectories, and for LWR$^{2S}$ and GPR$^{1S}$ after 8 (not shown

[1]Unless stated otherwise LWPR parameters are set as follows: init-D=10; w-gen=0.3; w-prune=0.7; init-alpha=0.05; diag-only=0; meta=0. GPR uses the covariance function of the Matérn form with isotropic distance measure.

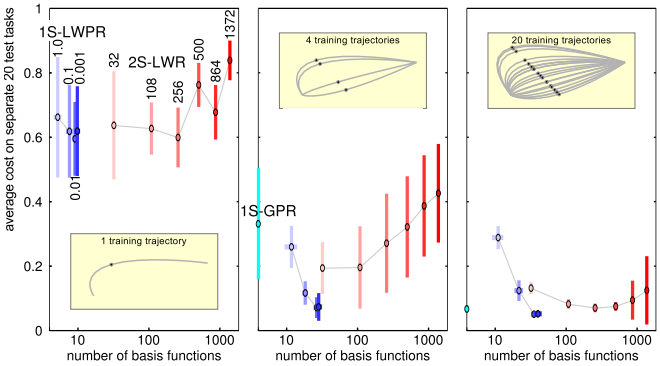
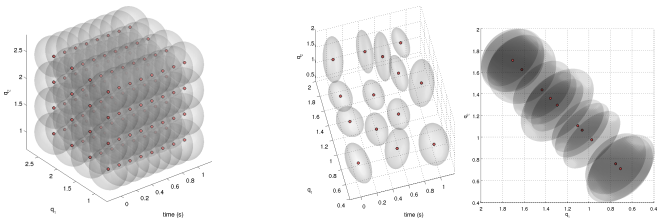



Fig. 5. Results of the viapoint tasks, after providing 1 (left), 4 (center) and 20 (right) training trajectories. Example trajectories are visualized in the insets at the top or bottom of the graphs. Each graph plots the average cost over 20 separate test trials ($y$-axis) against the number of basis functions in the skill ($x$-axis, logarithmic scale), for the function approximators LWR$^{2S}$ (red, with 32, 108, 256, 500, 864, 1372 basis functions), LWPR$^{1S}$ (blue, with penalty 1.0, 0.1, 0.01, 0.001), and GPR$^{1S}$ (cyan, where GPR$^{1S}$ lies on the $y$-axis, because the number of basis functions is not defined for this function approximator). Standard deviations over 10 separate training sessions are depicted as error bars in $x$ and $y$ direction. 20 different training and test trajectories are generated for each of the training sessions.

here). 2) LWPR$^{1S}$ (with penalty 0.01) always achieves a lower cost than the best LWR$^{2S}$ skill. For 20 trajectories, GPR$^{1S}$ performs similarly to LWPR$^{1S}$ in terms of cost. 3) LWPR$^{1S}$ is always able to achieve the same or lower cost with *less* basis functions than LWR$^{2S}$ (please note the logarithmic $x$-axis), and therefore has a more compact representation than LWR$^{2S}$.

The reason for this last point is visualized in Fig. 6. The LWR$^{2S}$ that achieves lowest cost uses 64=8×4×4 basis functions (BFs) for one transformation system (there are two transformation systems, hence the total number of 128 in Fig. 5a), because the BFs *must* be organized on a grid. LWPR$^{1S}$ on the other hand places BFs only on the manifold in which task parameters actually arise ($\mathbf{q}_1 = \mathbf{q}_2$) allowing for a much more compact representation with only 14 BFs. Furthermore, these BFs may have arbitrary orientations, also enabling more compact representations.



a) LWR b) LWPR c) LWPR (side view)

Fig. 6. Learned models for one of the two transformation systems, after 20 training trials. a) LWR$^{2S}$ with 128 basis functions. b/c) LWPR$^{1S}$ with penalty 0.1, leading to 14 basis functions. These models lead to approximately the same cost.

**Conclusion.** In the expanded basis function space of the 1-step approach, the positions and orientations of the basis functions may be freely chosen. This may lead to more compact models when algorithms such as LWPR are used.

### *B. Task 2: Object Transport*

In this experiment we reproduce the results described by Matsubara et al. [8, Section 4.1.2]. The iCub humanoid robot transports an object from one place to another, over an obstacle of varying height, cf. Fig. 7. The obstacle is a children's toy, consisting of a pile of 10 stackable hemispherical items; each item has an increasing diameter (8-12cm) and variable height (4-8cm); by stacking the items the obstacle height ranges from 8 to 25cm, cf. Fig. 7.

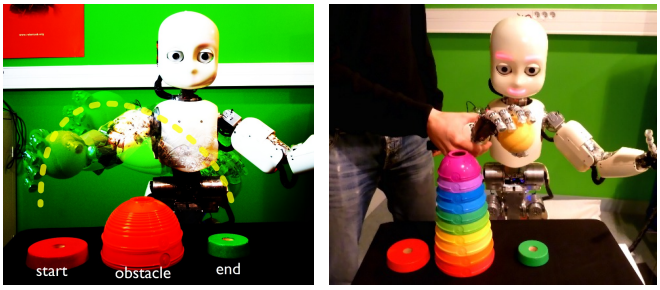



Fig. 7. Transport task: iCub brings its hand from a starting position (indicated by the red ring on the table) to a final position (green ring) avoiding an obstacle of variable height.

To acquire training trajectories, a human teacher manually guides the end-effector of the robot to demonstrate 5 trajectories for each of the 11 obstacle sizes. A zero-torque controller is used to gently move the robot upper-body, exploiting the force/torque sensors of the robot limbs [14]. The video attachment shows the acquisition of several of these 55 demonstrated trajectories. Due to the variation in the movements, we take the average of the 5 trajectories for one obstacle height. More sophisticated methods may be used to model this variation [2], but this is beyond the scope of this paper. A 3D parameterizable DMP is then trained with 6 of the 11 averaged 3D trajectories of the end-effector. The remaining 5 trajectories are used to test generalization.

In the execution phase, the parameterizable DMP generates a 3D Cartesian trajectory for the robot's end-effector. An inverse kinematics controller maps the Cartesian velocities to the joint velocities of the torso (3 DOF), and arms (7 DOF each). An optimization routine takes into account the redundancy of arm and torso, as well as the physical constraints of the platform. The hand posture is fixed during the movement.

The video attachment also shows the resulting generated motions for obstacle heights observed during training (6 examples), and also for those not previously observed. LWPR$^{1S}$ uses only 41% of the number of kernels that LWR$^{2S}$ does (311 vs. 750). Fig. 8 summarizes the aim of learning parameterized skills: being able to adapt motions to different task parameterizations with a single, compact representation.

**Conclusion.** In this experiment, our 1-step approach is able to learn more compact models than the 2-step approach.

### *C. Task 3: Object Grasping*

In this experiment, visualized in Fig. 1, the Meka humanoid robot learns a parameterizable skill for grasping

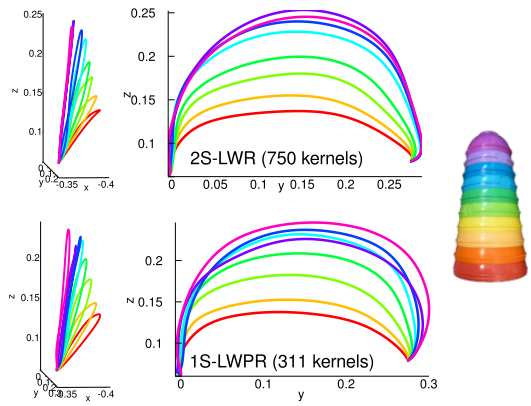



Fig. 8. Parameterized skill for object transport. Depending on the height of the obstacle, the robot adapts its motion. Here, the task parameters are set to 8 equidistant values, none of which have been observed during training. The obstacle serves as a color bar.

a box from trajectories demonstrated through kinesthetic teaching. 30 such trajectories are demonstrated (see video supplement for sample demonstrations). The robot automatically extracts the 3D box pose $\mathbf{q} = \langle x, y, \text{yaw} \rangle$ with a Kinect.

A 6D DMP (3D end-effector position and yaw/pitch/roll orientation) is then trained with these trajectories to learn the shape parameters $\mathbf{w}$. Based on a visual inspection of the reproduced trajectories, appropriate parameters for the algorithms are determined. An LWR$^{2S}$ with a total number of 5250=6×875 basis functions was trained, and for LWPR$^{1S}$ the penalty parameter is 0.1, which leads to a far more compact skill with only 559 basis functions. The same GPR$^{1S}$ parameters as for the viapoint task are used.

Since the end-point of the movement represents that grasp posture, adapting the DMP goal $\mathbf{x}^g$ is essential for successful task completion. In fact, we have chosen this task because it requires both adaptation of the goal (the final grasp) *and* the trajectory towards this goal (the more the object is turned away, the more you need to reach around it). Because the goal must be adapted, we also learn a mapping $x^g = m(\mathbf{q})$ for each transformation system, as suggested by [5]. For this simple mapping and LWPR, LWR and GP perform essentially equal, and their parameterization is not critical. The task parameter dependent duration of the movement could also be learned in this way, as in [5], [7], but we have kept it constant in this work.

During testing, we execute the parameterized skill for 25 task parameterizations, i.e. different poses of the box on the table. An execution is deemed successful if the robot is able to lift the object after grasping it; also see the video supplement. The success rates for LWR$^{2S}$ and GPR$^{1S}$ are 0.72 and 0.92. respectively. The two failure cases for GPR occur when the box is close to the robot, as shown in the video. LWR$^{2S}$ does not perform well because it is difficult to find a number of basis functions that generalize well, whilst also capturing the local variations required to generate the right trajectory shape. GPR$^{1S}$ achieves better performance because it generalizes well across the entire 3D space. However, GPR$^{1S}$ trajectories must be generated offline due to its high computational complexity.

**Conclusion.** With the augmented function approximator, different function approximator implementations may be used interchangeably. In this experiment, exchanging LWR with GPR substantially increases the success rate of the parameterizable skill (0.72→0.92). Our main point here is not as much to compare such implementations, which has been extensively done elsewhere [13], but rather to show that we may use and compare different implementations in the first place, which is not possible in the first step of 2-step approaches.

## V. Conclusion

Learning parameterizable skills from multiple demonstrations enables robots to generalize their motions to novel task parameterizations. We propose a novel DMP formulation for parameterized skills, based on additionally passing task parameters to the DMP function approximator. This generalizes previous approaches, in particular those which train and execute parameterized skills with two separate regressions. Learning the function approximator with one regression in the full space of phase *and* tasks parameters allows for more compact models, and the flexible use of different function approximator implementations such as LWPR and GPR, as we demonstrate on the Meka and iCub humanoids robots.

## References

[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, *Springer Handbook of Robotics*. Springer, 2008, ch. 59. Robot programming by demonstration.

[2] S. Calinon, F. Guenter, and A. Billard, "On learning, representing and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 37, no. 2, pp. 286–298, 2007.

[3] S. M. Khansari-Zadeh and A. Billard, "Learning stable non-linear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, 2011.

[4] M. Parlaktuna, D. Tunaoglu, E. Sahin, and E. Ugur, "Closed-loop primitives: A method to generate and recognize reaching actions from demonstration," in *ICRA*, 2012, pp. 2015–2020.

[5] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[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] B. da Silva, G. Konidaris, and A. G. Barto, "Learning parameterized skills," in *ICML*, 2012, pp. 1679–1686.

[8] T. Matsubara, S. Hyon, and J. Morimoto, "Learning parametric dynamic movement primitives from multiple demonstrations," *Neural Networks*, vol. 24, no. 5, pp. 493–500, 2011.

[9] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *R:SS*, 2010.

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

[12] S. Vijayakumar and S. Schaal, "Locally Weighted Projection Regression," in *ICML*, 2000.

[13] A. Droniou, S. Ivaldi, V. Padois, O. Sigaud, *et al.*, "Autonomous online learning of velocity kinematics on the icub: a comparative study," in *IROS*, 2012.

[14] S. Ivaldi *et al.*, "Computing robot internal/external wrenches by means of inertial, tactile and F/T sensors: theory and implementation on the iCub", in *IEEE-RAS Int'l Conference on Humanoid Robots*, 2011.