Overcoming Some Drawbacks of Dynamic Movement Primitives

Michele Ginesi[a,*], Nicola Sansonetto[a], Paolo Fiorini[a]

[a]*Department of Computer Science, University of Verona, Strada Le Grazie 15, Verona, Italy*

Abstract

Dynamic Movement Primitives (DMPs) is a framework for learning a point-to-point trajectory from a demonstration. Despite being widely used, DMPs still present some shortcomings that may limit their usage in real robotic applications. Firstly, at the state of the art, mainly Gaussian basis functions have been used to perform function approximation. Secondly, the adaptation of the trajectory generated by the DMP heavily depends on the choice of hyperparameters and the new desired goal position. Lastly, DMPs are a framework for 'one-shot learning', meaning that they are constrained to learn from a unique demonstration. In this work, we present and motivate a new set of basis functions to be used in the learning process, showing their ability to accurately approximate functions while having both analytical and numerical advantages w.r.t. Gaussian basis functions. Then, we show how to use the invariance of DMPs w.r.t. affine transformations to make the generalization of the trajectory robust against both the choice of hyperparameters and new goal position, performing both synthetic tests and experiments with real robots to show this increased robustness. Finally, we propose an algorithm to extract a common behavior from multiple observations, validating it both on a synthetic dataset and on a dataset obtained by performing a task on a real robot.

Keywords: Learning from Demonstrations, Motion and Path Planning, Kinematics, Dynamic Movement Primitives.

1. Introduction

The recent improvements in robot dexterity have given rise to increasing attention to Learning from Demonstration (LfD) approaches to make robots faithfully mimic human motions.
Dynamic Movement Primitive (DMP) [1, 2, 3, 4] is one of the most used frameworks for trajectory learning from a single demonstration. They are based on a system of second-order Ordinary Differential Equations (ODEs), in which a *forcing term* can be "learned" to encode the desired trajectory. This approach has already been proven effective in teaching a robot how to perform some human task as, for instance, (table) tennis swing [1, 5], to play drums [6], to write [7, 8], and to perform surgical-related tasks [9, 10]. The framework of DMPs has been shown to be flexible and robust enough to allow learning sensory experience [11, 12, 13], handling obstacle avoidance [14, 15, 16, 17], describing bimanual tasks [18], learning orientations [19, 20], and working in scenarios with human-robot interaction [21].

Despite their wide use, DMPs approach has still some shortcomings that need to be fixed to obtain a more robust framework.

In this work, we discuss three aspects of DMPs and propose modifications that guarantee a more robust implementation of the framework. In more detail, we discuss different classes of basis functions to be used instead of the Gaussian radial basis functions. Then, we show how to exploit the invariance of DMPs with respect to particular transformations in order to make the generalization of the learned trajectory more robust. Finally, we present an algorithm to learn a unique DMP from multiple observations without the need to rely on probabilistic approaches or additional parameters.

We remark that probabilistic approaches for trajectory learning can deal with generalization and learning from multiple demonstrations. These approaches include, for instance, *Probabilistic Movement Primitives* [22], *Kernelized Movement Primitives* [23], and *Gaussian Mixture Models* [24]. However, these methods necessarily require multiple trajectories to extract a common behavior, and, differently from DMPs, cannot be used as one-shot learning frameworks. Moreover, generalization is heavily limited by the quality of the dataset. Indeed, if the dataset is not descriptive enough of the task and all the different scenarios, the learned behavior may fail to generalize in certain situations. Finally, these methods have not only a probabilistic learning phase but also a

*Corresponding author

stochastic execution. This aspect makes them not suitable in some critical scenarios, such as Robotic Minimally Invasive Surgery. On the other hand, DMP is a completely deterministic approach. Thus, the improvements we present in this work allow to extend the DMP framework adding some of the strong points of probabilistic frameworks while maintaining the strengths of a deterministic approach.

The work is organized as follows. In Section 2 we review the two classical formulations of DMPs, emphasizing their shortcomings. In Section 3 we review multiple classes of basis functions and introduce a new one, which has both the desirable properties of being smooth and compactly supported. We then test and compare various numerical aspects of all the sets of basis functions presented, showing that our proposed one gives rise to a numerically more stable and faster learning phase. In Section 4 we show how to generalize the DMPs to any new starting and goal positions by exploiting the invariance property of DMPs under affine transformations. In Section 5 we present and test a novel algorithm to learn a unique DMP from a set of observations. Lastly, in Section 6 we present the conclusions.

Our implementation of DMPs, written in Python 3.8, is available at the link `https://github.com/mginesi/dmp_pp`. Together with the implementation of DMPs and our extensions, the repository contains the scripts to run all the tests presented in Sections 3.3, 4.2, and 5.2, together with the tests that are mentioned but not shown.

2. Dynamic Movement Primitives: an Overview

DMPs are used to model both periodic and discrete movements [1, 2, 3]. However, in this work, we will focus on the latter.
They consist of a system of second-order ODEs (one for each dimension of the ambient space) of mass-spring-damper type with a forcing term. DMPs aim to model the forcing term in such a way to be able to generalize the trajectory to new start and goal positions while maintaining the shape of the learned trajectory.

The one-dimensional formulation of DMPs is [1, 2, 3]:

$$\begin{cases} \tau \dot{v} = K(g - x) - Dv + (g - x_0) f(s) & \text{(1a)} \\ \tau \dot{x} = v & \text{(1b)} \end{cases}$$

where $x, v \in \mathbb{R}$ are, respectively, position and velocity of a prescribed point of the system. $x_0 \in \mathbb{R}$ is the initial position, and $g \in \mathbb{R}$ is the *goal*. Constants $K, D \in \mathbb{R}^+$ are, respectively, the spring and damping terms, chosen in such a way that the associated homogeneous system is critically damped: $D = 2\sqrt{K}$. Parameter $\tau \in \mathbb{R}^+$ is a temporal scaling factor, and f is a real-valued, nonlinear *forcing* (also called *perturbation*) *term*. $s \in (0, 1]$ is a re-parametrization of time $t \in [0, T]$, governed by the so-called *canonical system*:

$$\tau \dot{s} = -\alpha s. \tag{2}$$

$\alpha \in \mathbb{R}^+$ determines the exponential decay of canonical system (2). The initial value is set to $s(0) = 1$.
The forcing term f in (1a) is written in terms of basis functions as

$$f(s) = \frac{\sum_{i=0}^{N} \omega_i \psi_i(s)}{\sum_{i=0}^{N} \psi_i(s)} s, \tag{3}$$

where

$$\psi_i(s) = \exp\left(-h_i (s - c_i)^2\right) \tag{4}$$

are *Gaussian Basis Functions* (GBFs) with centers c_i and widths h_i. Centers c_i are defined as:

$$c_i = \exp\left(-\alpha \, i \, \frac{T}{N}\right), \quad i = 0, 1, \dots, N, \tag{5}$$

so that they are equispaced in time interval $[0, T]$. Widths h_i [25] are defined as

$$\begin{aligned} h_i &= \frac{\tilde{h}}{(c_{i+1} - c_i)^2}, \quad i = 0, 1, \dots, N-1, \\ h_N &= h_{N-1}, \end{aligned} \tag{6}$$

where $\tilde{h} > 0$ controls the overlapping between the basis functions. Usually, $\tilde{h}$ is set to one, $\tilde{h} = 1$ [19, 26].
The *learning process* focuses on the computation of the weights $\omega_i \in \mathbb{R}$ that best approximate the desired forcing term, obtained by solving (1a) for f. Given a desired trajectory $x(t), t \in [0, T]$ with velocity $v(t)$, we set $\tau = 1$, $x_0 = x(0)$, and $g = x(T)$. The forcing term is then computed as

$$f(s(t)) = \frac{1}{g - x_0}\Big(\dot{v}(t) - K(g - x(t)) + Dv\Big), \tag{7}$$

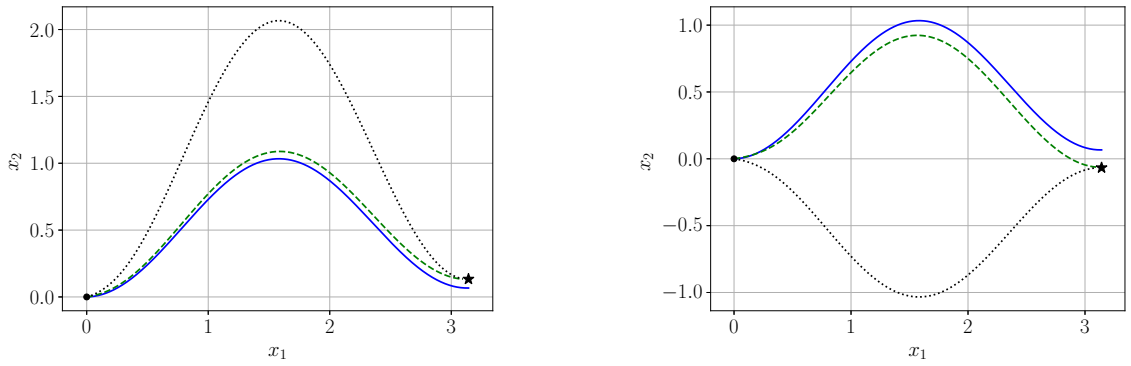
where $s(t) = \exp(-\alpha t)$. Then, the vector $\boldsymbol{\omega} = [\omega_0, \omega_1, \dots, \omega_N]^\mathsf{T}$ can be computed by solving the linear system

$$\mathbf{A}\boldsymbol{\omega} = \mathbf{b}, \tag{8}$$

where $\mathbf{A} = [a_{hk}]$ is a $(N+1)\times(N+1)$ matrix and $\mathbf{b} = [b_h]$ is a vector in $\mathbb{R}^{N+1}$ with components, respectively,

$$a_{hk} = \int_{s(0)}^{s(T)} \frac{\psi_h(s)\psi_k(s)}{\left(\sum_{i=0}^{N} \psi_i(s)\right)^2} s^2 \, ds, \tag{9a}$$

$$b_h = \int_{s(0)}^{s(T)} \frac{\psi_h(s)}{\sum_{i=0}^{N} \psi_i(s)} f(s) \, s \, ds. \tag{9b}$$



(a) Slight change in goal position.

(b) Change of sign in goal position.

Figure 1: Comparison between DMPs' formulation (10) and (11). In both plots, the blue solid line shows the learned trajectory. The dot and star mark, respectively, the new initial position $\mathbf{x}_0$ (which, for simplicity, in this example is kept the same as the learned one) and new goal position $\mathbf{g}$. The black dotted line shows the generalization to the new goal using DMP formulation (10), while the dashed green line shows the generalization using formulation (11). The grid is plotted to emphasize that in Figure 1b the vertical component of the new goal position is of opposite sign than the learned one.

When dealing with d-dimensional trajectories, we make d decoupled copies of system (1), obtaining the following vector formulation:

$$\begin{cases} \tau\dot{\mathbf{v}} = \mathbf{K}(\mathbf{g}-\mathbf{x}) - \mathbf{D}\mathbf{v} + (\mathbf{g}-\mathbf{x}_0)\odot\mathbf{f}(s) & \text{(10a)} \\ \tau\dot{\mathbf{x}} = \mathbf{v} & \text{(10b)} \end{cases}$$

where $\mathbf{x}, \mathbf{v}, \mathbf{g}, \mathbf{x}_0, \mathbf{f}(s) \in \mathbb{R}^d$ and $\mathbf{K}, \mathbf{D} \in \mathbb{R}^{d\times d}$ are diagonal matrices, so to maintain each component decoupled from the others. The operator $\odot$ denotes the component-wise multiplication: given $\mathbf{v} = [v_i], \mathbf{w} = [w_i] \in \mathbb{R}^d$, we define $\mathbf{v}\odot\mathbf{w} = [v_i w_i]_i$.
Thanks to the decoupling of the system, the forcing term can be learned component by component.

Three main drawbacks characterize DMP formulation (10) (see [15]). Firstly, if in any component the goal position coincides with the starting position, $g = x_0$, the perturbation term f clearly does not contribute to the evolution of the dynamical system. Secondly, if $g - x_0$ is "small" the scaling of the perturbation term f by the quantity $g - x_0$ may produce unexpected (and undesired) behaviors. Finally, if the scaling factor $g - x_0$ changes sign from the learned trajectory to the new one, the trajectory will result mirrored.
In Figure 1 we present an example similar to that in Figure 2 of [15] to show the aforementioned drawbacks. In both tests, the vertical component of the difference $\mathbf{g} - \mathbf{x}_0$ is quite small: $(\mathbf{g} - \mathbf{x}_0)|_2 = \frac{1}{15} \approx 0.0667$. Figure 1a shows that even a slight change in goal position results in a complete different trajectory when using formulation (1). Figure 1b, instead, shows the 'mirroring' problem. In this case, the vertical component of $\mathbf{g} - \mathbf{x}_0$ changes sign from the learned to the new execution. Thus, the trajectory results mirrored w.r.t. the horizontal axis $x_2 = 0$.

To overcome these disadvantages, the following formulation was proposed in [14, 27, 15]:

$$\begin{cases} \tau\dot{\mathbf{v}} = \mathbf{K}(\mathbf{g}-\mathbf{x}) - \mathbf{D}\mathbf{v} - \mathbf{K}(\mathbf{g}-\mathbf{x}_0)s + \mathbf{K}\mathbf{f}(s) & \text{(11a)} \\ \tau\dot{\mathbf{x}} = \mathbf{v} & \text{(11b)} \end{cases}$$

where the evolution of s is still described by the canonical system (2), and the forcing term is still written in terms of basis functions as in (3).

By removing the scaling term $\mathbf{g} - \mathbf{x}_0$ from the forcing term $\mathbf{f}$, this new formulation solves the aforementioned problems of (1), as it can be seen in Figure 1. However, while solving the drawbacks characterizing formulation (10), DMPs' formulation (11) still presents an important drawback: the generalization to different spatial scales is not always feasible since the forcing term does not have any dependence on the relative position between starting and ending points. We will discuss the details of this aspect in Section 4 when presenting our proposed modifications to solve this drawback.

3. New Set of Basis Functions

A change of the set of basis functions is motivated by the fact that a desirable property of basis functions is to be compactly supported [8] since compactly supported basis functions provide an easier update of the learned

trajectory. Indeed, if only a portion of the trajectory has to be modified, only a subset of the weights needs to be re-computed (we will discuss the details of this aspect in Section 3.2). Moreover, while with GBFs the forcing term never vanishes, compactly supported basis functions allow the forcing term to be zero outside the support, thus providing a faster convergence of the system to the goal position.

In [8] the authors proposed to use *Truncated Gaussian Basis Functions* (TGBFs):

$$\tilde{\psi}_i(s) = \begin{cases} \exp\left(-\frac{h_i}{2}(s-c_i)^2\right) & \text{if } s - c_i \leq \theta_i \\ 0 & \text{otherwise} \end{cases}, \tag{12}$$

where c_i and h_i are defined as in (5) and (6) respectively, and $\theta_i = K/\sqrt{h_i}$. Unfortunately, this approach has two main drawbacks. Firstly, a discontinuity is introduced at the truncation points θ_i, which may produce unexpected behaviors. Secondly, in order to work properly, this approach requires the introduction of *biases terms* in (3), which now reads

$$f(s) = \frac{\sum_{i=0}^{N}(\omega_i s + \beta_i)\,\tilde{\psi}_i(s)}{\sum_{i=0}^{N}\tilde{\psi}_i(s)}, \tag{13}$$

thus doubling the number of parameters that need to be learned (a weight ω_i and a bias β_i for each basis function), and increasing the overall computational cost.
Finally, we remark that these basis functions are not compactly supported since their support is an interval $(-\infty, L]$ with $L \in \mathbb{R}$. Moreover, since the truncation point is at the "right" in s (i.e. $\tilde{\psi}_i(s) \equiv 0$ when s is above a certain quantity), and s is a decreasing function in t, we remark that these basis functions never vanish as $t \to \infty$.

In this part, we propose a new set of basis functions that improves the TGBFs (12). Indeed, our proposed set contains compactly supported basis functions, which then vanish in both directions. Moreover, since no truncation (and, thus, discontinuities) are introduced, there is no need to double the number of parameters, and our formulation will require the learning of only the weights ω_i in (3).

We will also show that our proposed set of basis functions results in a minimization problem that is both numerically more stable and faster to solve.

3.1. Mollifier-like and Wendland Basis Functions

In this work, we propose a new set of basis functions that are both smooth and compactly supported.
Consider the *mollifier* $\varphi : \mathbb{R} \to \mathbb{R}$ defined as

$$\varphi(x) = \begin{cases} \frac{1}{I_n}\exp\left(-\frac{1}{1-|x|^2}\right) & \text{if } |x| < 1 \\ 0 & \text{otherwise} \end{cases}, \tag{14}$$

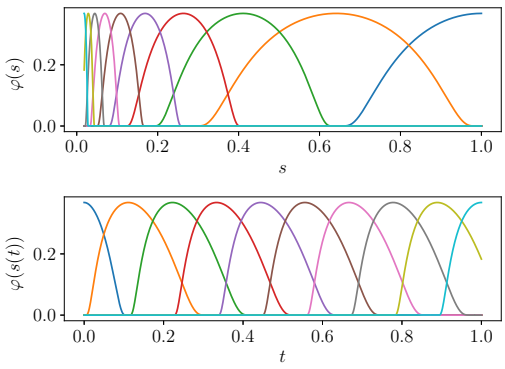


Figure 2: Plot of mollifier-like basis functions as defined in (15). In this example $N = 9$, $\alpha = 4$ and $T = 1$. The first plot shows the basis functions as a function of s, while the second plot shows them as functions of time t.

where I_n is set so that $\int_{\mathbb{R}} \varphi(x)\,dx = 1$. Function φ is smooth and compactly supported (its support is the interval $[-1, 1]$).
It is then natural to define the set of *mollifier-like basis functions* $\{\varphi_i(s)\}_{i=0,1,\dots,N}$ given by

$$\varphi_i(s) = \begin{cases} \exp\left(-\frac{1}{1-r^2}\right) & \text{if } r < 1 \\ 0 & \text{otherwise} \end{cases}, \tag{15}$$

where r is a function of s defined as

$$r = |a_i(s - c_i)|, \tag{16}$$

where, given the centers c_i as in (5), we define the widths a_i as

$$\begin{aligned} a_i &= \frac{\tilde{h}}{|c_i - c_{i-1}|}, \quad i = 1, 2, \dots, N, \\ a_0 &= a_1. \end{aligned} \tag{17}$$

As in (6), scalar $\tilde{h} > 0$ determines the overlapping between basis functions. In all our tests, we fix $\tilde{h}$ to value one: $\tilde{h} = 1$. We remark that the usual normalization term I_n in (14) is omitted in (15) since it does not enter our approach.
In Figure 2 we plot an example of mollifier-like basis functions both as function of s and of t. We remark that the basis functions are equispaced in t. Moreover, since s is a strictly decreasing function of t, their order changes from s to t (the first basis function in s is the last in t and vice versa), as it was for the GBFs used so far in the literature.

Other well known regular (see Table 1 for details on the regularity) and compactly supported basis functions are *Wendland functions* [28, 29]. In Section 3.3 we test

Table 1: Summary of the properties (regularity and support compactness) of the different sets of basis functions presented in Section 3. Bold font is used to mark the more desirable properties (smoothness and compact support).

Function		Regularity	Compactly Supported
Gaussian	$\psi_i(s)(\cdot)$	$\boldsymbol{C^\omega(\mathbb{R})}$	No
Truncated Gaussian	$\tilde{\psi}_i(s)(\cdot)$	Discontinuous	No
Mollifier-like	$\varphi_i(s)(\cdot)$	$\boldsymbol{C^\infty(\mathbb{R})}$	**Yes**
Wendland	$\phi^{(2)}(\cdot)$	$C^0(\mathbb{R})$	**Yes**
	$\phi^{(3)}(\cdot)$	$C^0(\mathbb{R})$	**Yes**
	$\phi^{(4)}(\cdot)$	$C^2(\mathbb{R})$	**Yes**
	$\phi^{(5)}(\cdot)$	$C^2(\mathbb{R})$	**Yes**
	$\phi^{(6)}(\cdot)$	$C^4(\mathbb{R})$	**Yes**
	$\phi^{(7)}(\cdot)$	$C^4(\mathbb{R})$	**Yes**
	$\phi^{(8)}(\cdot)$	$C^6(\mathbb{R})$	**Yes**

the following Wendland basis function (where the operator $(\cdot)_+$ denotes the *positive part function*: $(x)_+ = \max\{0, x\}$):

$$\phi_i^{(2)}(r) = (1-r)_+^2, \tag{18a}$$
$$\phi_i^{(3)}(r) = (1-r)_+^3, \tag{18b}$$
$$\phi_i^{(4)}(r) = (1-r)_+^4\,(4r+1), \tag{18c}$$
$$\phi_i^{(5)}(r) = (1-r)_+^5\,(5r+1), \tag{18d}$$
$$\phi_i^{(6)}(r) = (1-r)_+^6\,(35r^2+18r+3), \tag{18e}$$
$$\phi_i^{(7)}(r) = (1-r)_+^7\,(16r^2+7r+1), \tag{18f}$$
$$\phi_i^{(8)}(r) = (1-r)_+^8\,(32r^3+25r^2+8r+1). \tag{18g}$$

where c_i are the centers as in (5) and a_i are the widths as in (17), and r is defined as in (16).

In Table 1 we summarize the properties of the basis functions we presented in this section. As can be seen, mollifier-like basis functions are the only ones being both smooth and compactly supported.

3.2. Trajectory Update

In [8] was pointed out that if only a portion of the trajectory has to be re-learned, only a subset of weights has to be re-computed. To be more precise, assume that a desired trajectory $\gamma(t)$ has to be updated in the time interval $[t_0, t_1] \subset [0, T]$. Then, not all the weights ω_i have to be re-computed, but only those whose basis functions φ_i satisfy (where the support has to be intended on the values of t and not s)

$$\operatorname{supp}(\varphi_i) \cap [t_0, t_1] \neq \emptyset.$$

From this, it is easy to notice that for classical GBFs this means that all weights have to be re-computed, since their support is the whole real line $\mathbb{R}$: $\operatorname{supp}(\psi_i) = \mathbb{R}$. For compactly supported basis functions, this intersection will be usually smaller than the whole set of indexes $\{0, 1, \ldots, N\}$, and it will depend only on the length $t_1 - t_0$ of the interval.

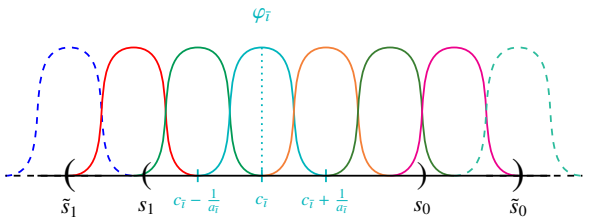


Figure 3: Depiction of compactly supported basis functions whose support intersects an interval (s_1, s_0). The basis functions drawn using solid lines are those whose weights have to be updated. As one can observe, they satisfy condition (19). Interval $(\tilde{s}_1, \tilde{s}_0)$ shows the domain in which the integrals in (9) have to be computed.

The support, in s, of each basis function φ_i is the interval $(c_i - 1/a_i, c_i + 1/a_i)$. Therefore, the set of weights $\{\omega_i\}_{i\in I}$ that has to be updated are those for which

$$c_i - \frac{1}{a_i} \leq s_0 \quad \text{and} \quad c_i + \frac{1}{a_i} \geq s_1, \tag{19}$$

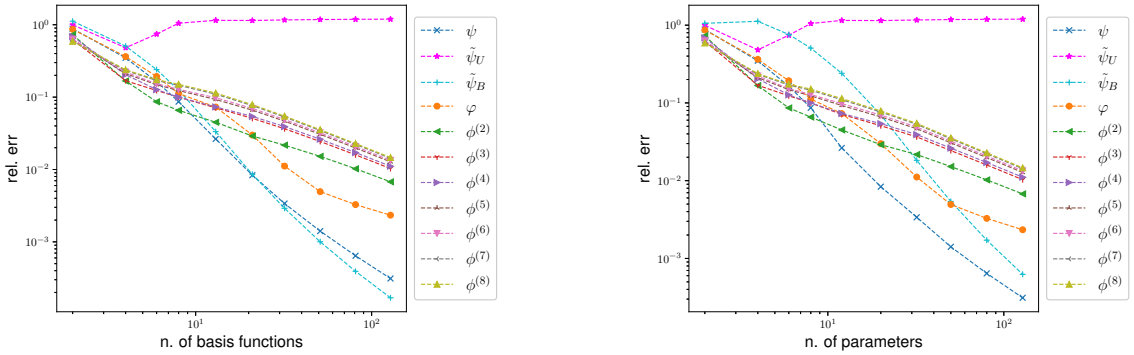
where $s_0 = \exp(-\alpha\, t_0)$ and $s_1 = \exp(-\alpha\, t_1)$.

Once the set of indexes I has been identified using (19), one must solve a linear problem as in (8), where $\mathbf{A}$ is a $|I| \times |I|$ matrix, $\mathbf{A} \in \mathbb{R}^{|I|\times|I|}$, and $\mathbf{b}$ is a vector with $|I|$ components, $\mathbf{b} \in \mathbb{R}^{|I|}$. The components of matrix $\mathbf{A}$ and vector $\mathbf{b}$ are given in (9). We remark that the integrals in (9) have to be evaluated on the interval

$$(\tilde{s}_1, \tilde{s}_0) = \bigcup_{i\in I} \operatorname{supp}(\varphi_i),$$

where the support has to be intended in s. By solving the linear problem, the weights ω_i, $i \in I$ are updated.

Figure 3 gives an intuition on how to identify the set $\{\varphi_i\}_{i\in I}$ of basis functions satisfying (19) and the interval $(\tilde{s}_1, \tilde{s}_0)$.



(a) Error as function of the number of basis functions.

(b) Error as function of the number of parameters.

Figure 4: Plot of the L^2 error done by approximating the hat function (20). The first plot shows the error w.r.t. the number of basis functions, while the second plot shows the error w.r.t. the number of parameters that has to be learned. TGBFs are tested both using the (classical) unbiased formulation (3) and the biased one (13). These two different approaches are denoted respectively by $\tilde{\psi}_U$ and $\tilde{\psi}_B$ in the legend.

Remark 1. In the case of TGBFs, the set of weights to update will depend not only on the length of the interval but also on its 'position'. For instance, if the last part of the trajectory has to be updated, i.e. $[t_0, t_1] = [t_0, T]$, all the weights must be updated since $T \in \text{supp}(\tilde{\psi}_i)$ for any $i = 0, 1, \ldots, N$.
Thus, in general, when a portion of the trajectory has to be modified, the number of weights to re-compute will be less when using compactly supported basis functions than when using TGBFs.

3.3. Results

We now investigate the goodness of the approximations obtained using the various examples of basis functions we introduced. In particular, we test both the classical (4) and the truncated (12) Gaussian basis functions, the mollifier-like basis functions (15), and the Wendland functions (18). We test three numerical aspects: in Section 3.3.1 the approximation error on particular target functions, in Section 3.3.2 the condition number of matrix $\mathbf{A}$ in (8), and in Section 3.3.3 we test the computtional time needed to solve the minimization problem (8). Additionally, in Section 3.3.4 we present a test on the trajectory update property presented in Section 3.2.

3.3.1. Numerical Accuracy.

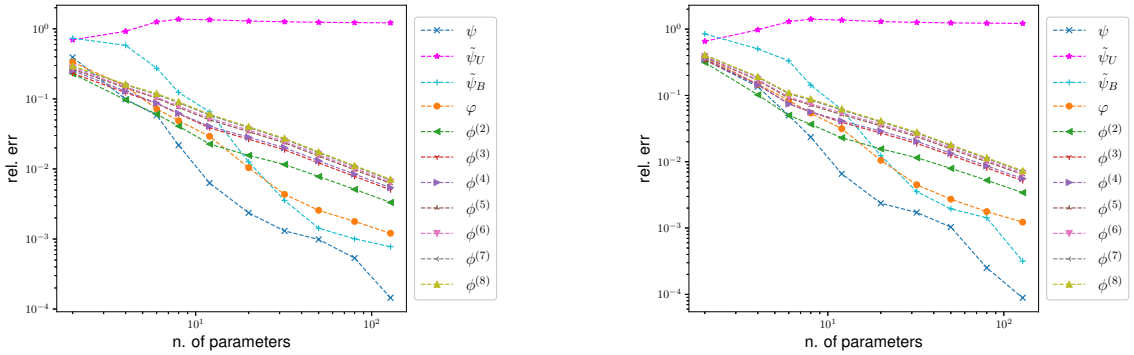
We test the accuracy of the approximation of the following function:

$$\eta(t) = t^2 \cos(\pi t), \quad t \in [0, 1]. \tag{20}$$

Figure 4 shows the approximation error, measured w.r.t. the L^2-norm, both w.r.t. the number of basis functions (Figure 4a), and w.r.t. the number of parameters that need to be learned (Figure 4b). We recall that for TGBFs the number of parameters is double the number of basis functions since every basis function requires one weight and one bias term. On the other hand, for all other classes of basis functions, the number of parameters coincide with the number of basis functions, since for each basis function only one parameter (the weight) has to be computed.

The plot shows that TGBFs (12) work properly only using the biased formulation (denoted by $\tilde{\psi}_B$ in the legend) for the forcing term (13), which means that given N basis functions we are solving a $2N$-length linear system when computing the weights and the biases. On the other hand, when using the unbiased formulation (3) (denoted by $\tilde{\psi}_U$ in the legend) the approximant does not converge to the desired forcing term.

We observe that when comparing the error w.r.t. the number of basis functions, the error obtained by using TGBFs is comparable to the error done using classical GBFs. On the other hand, when comparing the error w.r.t. the number of parameters that have to be computed, TGBFs approximate worse than mollifier basis functions when the number of parameters is below 60. Usually, in the applications, no more than fifty basis functions are used. For this particular value and target function, mollifier-like basis functions and truncated Gaussian have almost identical approximation errors. However, we remark that these results depend on the particular choice of the target function η in (20),



(a) `pushing_needle`.

(b) `positioning_needle`.

Figure 5: Plot of the L^2 error done approximating two real gestures. The error is given as a function of the number of parameters that need to be learned. TGBFs are tested both using the (classical) unbiased formulation (3) and the biased one (13). These two different approaches are denoted respectively by $\tilde{\psi}_U$ and $\tilde{\psi}_B$ in the legend.

and that different target functions may give different results. Tests performed with different target functions show similar results: classical Gaussian functions remain the best approximator, while truncated Gaussian and mollifier-like give similar approximation accuracy and usually work better than Wendland functions.

Additionally, we performed this test on trajectories obtained from real task executions. In particular, we relied on the JIGSAW dataset [30]. This dataset contains data on three elementary surgical tasks (suturing, knot-tying, and needle-passing). The demonstrations were collected from eight different subjects of different experience in robotic surgery. In Figure 5 we show the convergence error for two gestures extracted from the dataset. In particular, Figure 5a shows the results for the `pushing_needle` gesture, while Figure 5b shows the results for the `positioning_needle` gesture. Both gestures were extracted from the same user ('D') performing the same task (suturing). As it can be seen, the results are similar to those obtained by approximating function η in (20): GBFs (4) are the best approximators, TGBFs (12) and mollifier-like basis functions (15) have similar convergence result, and Wendland basis functions (18) are the less accurate basis functions.

In all tests, we tested classical Gaussians, Wendland, and mollifier-like basis functions also using the biased formulation (13), without noticing any difference in the goodness of the approximation.

3.3.2. *Condition Number.*

We now discuss the numerical accuracy of the minimization problem (8) used to compute the weights ω_i in (3). To do so, we investigate the condition number of matrix $\mathbf{A}$, $\text{cond}(\mathbf{A}) \stackrel{\text{def}}{=} \|\mathbf{A}\|\,\|\mathbf{A}^{-1}\|$, in (8). The importance of this test is given from the fact that when one solves numerically a linear system, the approximation error is directly proportional to the condition number. Indeed, when numerically solving a linear system $\mathbf{A}\boldsymbol{\omega} = \mathbf{b}$, whose exact solution is $\tilde{\boldsymbol{\omega}}$, and numerical solution is $\hat{\boldsymbol{\omega}}$, the following inequality holds:

$$\frac{\|\tilde{\boldsymbol{\omega}} - \hat{\boldsymbol{\omega}}\|}{\|\hat{\boldsymbol{\omega}}\|} \leq \text{cond}(\mathbf{A}) \frac{\|\boldsymbol{\varsigma}\|}{\|\mathbf{b}\|}, \tag{21}$$

where $\boldsymbol{\varsigma} \stackrel{\text{def}}{=} \mathbf{b} - \mathbf{A}\tilde{\boldsymbol{\omega}}$ is the *residual*. Inequality (21) says that the relative error done when numerically solving a linear system is amplified by the condition number of the matrix $\mathbf{A}$.

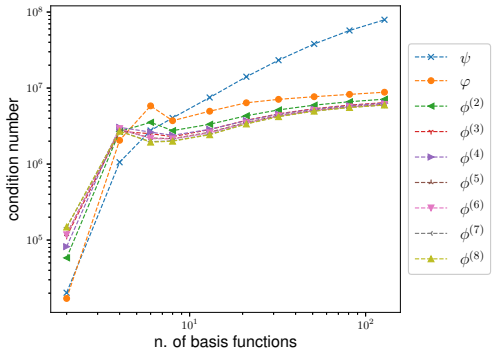


Figure 6: Condition number of the matrix $\mathbf{A}$ with elements a_{hk} defined in (32) w.r.t. the number of basis functions.

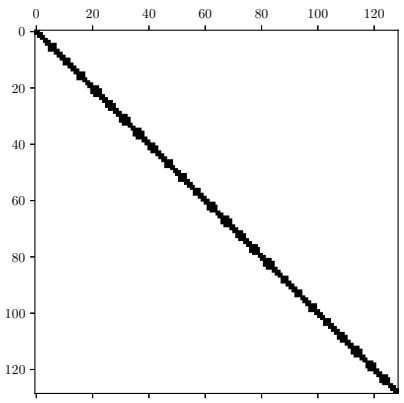


Figure 7: Sparsity pattern of matrix $\mathbf{A}$ with elements a_{hk} defined in (9a) in the case of $N = 2^7 = 128$ using mollifier-like basis functions.

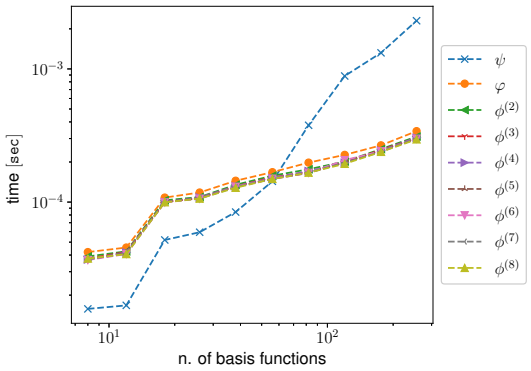


Figure 8: Average computational time when solving the minimization problem (8) as function of the number of basis functions N.

In this test, we do not consider TGBFs since, in this case, the components $\mathbf{A}$ and $\mathbf{b}$ of the linear problem (8) have different formulations, having to learn both the weights and the biases.

Figure 6 shows that the condition number cond($\mathbf{A}$) of matrix $\mathbf{A}$ increases faster for GBFs, while the compactly supported basis functions show a slower increase. Moreover, even with only twenty basis functions, the condition number obtained with GBFs is significantly bigger than any compactly supported basis function. This means that solving the minimization problem (8) using GBFs results in a more severe numerical cancellation error than mollifier-like or Wendland functions.

The lower condition number can be explained by the fact that mollifier-like and Wendland basis functions are compactly supported, and then the resulting matrix $\mathbf{A}$ will have many off-diagonal components equal to zero, as it can be seen in the sparsity pattern shown in Figure 7. On the other hand, when one uses GBFs, all the components of $\mathbf{A}$ are non-zero, since $\psi_i(s) > 0, \forall s \in \mathbb{R}$. Thus, $\mathbf{A}$ is a full matrix when using GBFs, while it is a sparse $n-$diagonal matrix when using compactly supported basis functions; and, in general, full matrices have a bigger condition number than sparse $n-$diagonal ones.

We remark that convergence analysis shown in Figures 4 and 5 is done on particular choices of target functions. The convergence error may differ depending on the forcing term that needs to be approximated. However, we stress that the condition number shown in Figure 6 does not depend on the target function but only on the choice of basis functions.

3.3.3. Computational Time

An additional advantage in using compactly supported basis functions lies in the improvements in computational time when solving the linear problem (8). Indeed, when the number of basis functions N increases, the entries in matrix $\mathbf{A}$ increases quadratically: $O(N^2)$. When GBFs are used, all entries in $\mathbf{A}$ are non-zero. On the other hand, when compactly supported basis functions are adopted, the number of non-zero entries in $\mathbf{A}$ increases linearly, $O(N\,\ell)$, where ℓ is the number of non-zero diagonals in $\mathbf{A}$, and is determined by the overlapping between basis functions, which depends on parameter $\tilde{h}$ in (17).
Consequently, as the number of basis functions N increases, the number of operations that are needed to solve the linear problem (8) increases cubically, $O(N^3)$, when the matrix is full; and only quadratically, $O(N^2\ell)$, when the matrix is sparse. Thus, one should expect the minimization problem (8) to be faster to solve when compactly supported basis functions are used.

To test the improvement in computational time, we perform the following test. For different values of the number of basis functions N, we compute the matrix $\mathbf{A}$ in (8). Then, for each value of N, we consider thirty different values for vector $\mathbf{b}$ and solve the linear problem (8) saving the time needed to solve it. We perform this test for the GBFs (4), mollifier-like basis functions (15), and Wendland basis functions (18).
Figure 8 shows the result of this test. In particular, the log-scale plot shows the average computational time for each value of N. As it can be seen, as the number of basis functions increases, the computational time needed to solve the linear problem (8) has a greater growth when GBFs are used. On the other hand, when compactly supported basis functions are adopted, the increment in computational time is reduced.

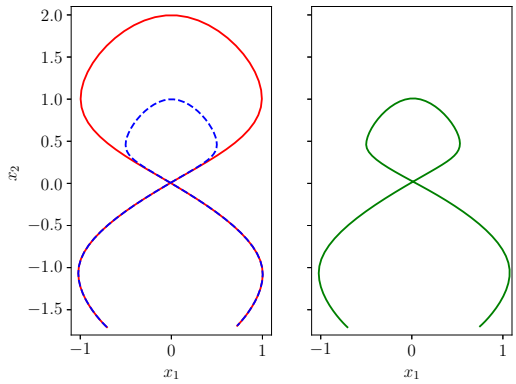


Figure 9: Result for the 'trajectory update' property of compactly supported basis functions. On the left, the two trajectories are shown, showing that they differ only in the central portion. On the right, the execution of the obtained DMP is shown. The DMP is initially learned from the 'big' trajectory, and then updated using the middle portion of the small one.

Tests were performed on a notebook with a quad-core Intel Core i7-7000 CPU with 16 GB of RAM.

3.3.4. Trajectory Update

In this Section, we present a synthetic test to show the "trajectory update" property presented in Section 3.2. To do so, we generate two trajectories, which can be seen on the left plot in Figure 9, using clamped splines (to have null initial and final velocities). The trajectories are identical on the 'tails' but they differ in the central portion (one is larger than the other). To show the update properties of compactly supported basis functions, we start by learning a DMP from the 'largest' trajectory (shown in red in Figure 9). DMPs parameters are $\mathbf{K} = K\,\mathrm{Id}_2$ and $\mathbf{D} = D\,\mathrm{Id}_2$ with $K = 150$ and $D = 2\sqrt{K} \approx 24.49$, $\alpha = 4$, and $N = 100$.
Next, we identificate, using (19), the set of basis functions that have to be updated when the middle portion of the trajectory has to be updated. This result in the set of indexes $I = \{24, 25, \dots, 63\} \subset \{0, 1, \dots, N\} = \{0, 1, \dots, 100\}$. Then, we re-compute the set of weights $\{\omega_i\}_{i \in I}$.
The plot on the right in Figure 9 shows the execution of the resulting, updated DMP. As one can observe, by updating only a subset of the weights the resulting trajectory mimics the new desired behavior.

3.4. General Considerations

The tests performed in Section 3.3 showed the advantages of using compactly supported basis functions.
Firstly, these sets of functions give a sparse matrix in minimization problem (8). As a consequence, the learning phase is both numerically faster and more stable than when using GBFs (4).
Secondly, compactly supported basis functions allow to 'update' a DMP when only a portion of the trajectory has to be modified, instead of learning a new one from scratch.
Finally, the usage of compactly supported basis functions ensures a faster convergence of the DMP system to the goal position since the resulting forcing term is null after a finite amount of time, $f(s(t)) = 0, \forall t > T^\star$ [8]. This is not true for (both classical and truncated) Gaussian basis functions (4), (12). Indeed, the support of the classical GBFs is the whole real line $\mathbb{R}$; while the support for TGBFs is an interval $(-\infty, L]$, $L \in \mathbb{R}$ in s, which is an interval $[L', +\infty)$, $L' \in \mathbb{R}$ in t.

Among the families of compactly supported basis functions we presented in this work, our proposed mollifier-like basis functions (15) are both the most regular and the best approximants.

For these reasons, we argue that mollifier-like basis functions may become the new standard when using DMPs.

Remark 2. The numerical advantages (stability and time efficiency) showed by compactly supported basis functions influence only the learning phase. Indeed, since the approximation errors are similar and, overall, small among the different classes of basis functions, the resulting DMP execution will have no meaningful difference.

As a final observation, we remark that other approaches based upon basis functions approximation (such as ProMPs [22]) could take advantage of the computational stability and efficiency of compactly supported basis functions.

4. Invariance Under Invertible Linear Transformations

Invariance under translations of DMPs formulations (1) and (11) is straightforward. Indeed, by changing the initial position from $\mathbf{x}_0$ to $\mathbf{x}'_0$, and by considering, instead of $\mathbf{g}$, the new goal position $\mathbf{g}' = \mathbf{g} + (\mathbf{x}'_0 - \mathbf{x}_0)$, the whole trajectory is translated by a quantity $\mathbf{x}'_0 - \mathbf{x}_0$.

DMPs are able to adapt to small changes of the relative position between goal and start, $\mathbf{g} - \mathbf{x}_0$; however, the robustness of DMPs w.r.t. sensible changes of vector $\mathbf{g} - \mathbf{x}_0$ heavily depends on the choice of the hyperparameters $\mathbf{K}, \mathbf{D}$ in (11) and α in (2) (see, for instance, the tests performed in Figures 10, 11). Generalization to arbitrary changes of quantity $\mathbf{g} - \mathbf{x}_0$ can be achieved by exploiting the invariance property of equations (11)

w.r.t. (invertible) affine transformations, see [15]. At the best of the authors' knowledge, this well-known property has never been exploited in order to make DMPs globally robust w.r.t. arbitrary changes of $\mathbf{g} - \mathbf{x}_0$.
The invariance of (11) can be easily proven (see [15]). Consider the invertible transformation matrix $\mathbf{S} \in \mathbb{R}^{d \times d}$. By substituting

$$\begin{aligned} &\mathbf{x}' = \mathbf{S}\mathbf{x}, \quad \mathbf{v}' = \mathbf{S}\mathbf{v}, \qquad \mathbf{x}'_0 = \mathbf{S}\mathbf{x}_0, \\ &\mathbf{g}' = \mathbf{S}\mathbf{g}, \quad \mathbf{K}' = \mathbf{S}\mathbf{K}\mathbf{S}^{-1}, \quad \mathbf{D}' = \mathbf{S}\mathbf{D}\mathbf{S}^{-1}, \end{aligned} \tag{22}$$

in (11) we obtain the transformation law of the forcing term

$$\mathbf{f}' = \mathbf{S}\mathbf{f}. \tag{23}$$

Thus, if we want to generate a new trajectory $\mathbf{S}\mathbf{x}$, it is sufficient to apply the transformations in (22) and (23) to (11). The resulting DMP system, thus, reads

$$\begin{cases} \tau \dot{\mathbf{v}} = \mathbf{K}'(\mathbf{g} - \mathbf{x}) - \mathbf{D}'\mathbf{v} - \mathbf{K}'(\mathbf{g} - \mathbf{x}_0)s + \\ \qquad\qquad\qquad\qquad\qquad \mathbf{K}'\mathbf{f}(s) & \text{(24a)} \\ \tau \dot{\mathbf{x}} = \mathbf{v} & \text{(24b)} \end{cases}$$

We remark that if the elastic and damping terms are the same for each degree-of-freedom, then $\mathbf{K}$ and $\mathbf{D}$ are multiple of the identity matrix and thus $\mathbf{K} = \mathbf{K}'$ and $\mathbf{D} = \mathbf{D}'$.

4.1. Invariance Under Roto-Dilatation

A case of particular interest is when $\mathbf{S}$ in (22), (23) represents a *roto-dilatation*.
Consider a trajectory which is learned together with the relative position between the goal and the starting point, i.e. the vector $\mathbf{g} - \mathbf{x}_0$; and a new trajectory, with starting and ending points $\mathbf{x}'_0$ and $\mathbf{g}'$ respectively has to be reproduced.
We first compute the rotation matrix $\mathbf{R}^{\widehat{\mathbf{g}'-\mathbf{x}'_0}}_{\widehat{\mathbf{g}-\mathbf{x}_0}}$, which maps the unit vector $\widehat{\mathbf{g} - \mathbf{x}_0}$ to $\widehat{\mathbf{g}' - \mathbf{x}'_0}$, where operator $\widehat{\cdot}$ denotes the normalization: $\widehat{\mathbf{v}} \stackrel{\text{def}}{=} \mathbf{v}/\|\mathbf{v}\|$. Rotation matrix $\mathbf{R}^{\widehat{\mathbf{g}'-\mathbf{x}'_0}}_{\widehat{\mathbf{g}-\mathbf{x}_0}}$ can be computed using the algorithm presented in [31].
After performing the rotation, we perform a dilatation of $\|\mathbf{g}' - \mathbf{x}'_0\|/\|\mathbf{g} - \mathbf{x}_0\|$ obtaining the transformation matrix

$$\mathbf{S} = \frac{\left\|\mathbf{g}' - \mathbf{x}'_0\right\|}{\|\mathbf{g} - \mathbf{x}_0\|} \mathbf{R}^{\widehat{\mathbf{g}'-\mathbf{x}'_0}}_{\widehat{\mathbf{g}-\mathbf{x}_0}}. \tag{25}$$

Therefore, when a learned DMP has to be used to generate a new trajectory, characterized by the parameters $\mathbf{x}'_0$ and $\mathbf{g}'$, we compute the matrix $\mathbf{S}$ as in (25), and then the quantities $\mathbf{K}'$, $\mathbf{D}'$, and $\mathbf{f}'$ as in (22) and (23).
This approach can be used even if the goal position changes with time during the execution of the trajectory simply by updating the matrix $\mathbf{S}$ at each time. This, in particular, means that $\mathbf{S}$ depends on time. In Section 4.2 we will test this approach both for static and moving goal positions.

We remark that this method cannot be performed when starting and ending points coincide: $\mathbf{x}_0 = \mathbf{g}$, since the matrix $\mathbf{S}$ in (25) would result in a null matrix, $\mathbf{S} = \mathbf{0}$. However, in such cases, one should use *rhythmic* (or *periodic*) DMPs instead of discrete ones, which are beyond the scope of this paper.

Remark 3. The choice to use a roto-dilatation as the transformation matrix $\mathbf{S}$ is not a unique possibility. Indeed, any invertible matrix can be used. For example, if we write $\mathbf{S}$ as the diagonal matrix $\mathbf{S} = \mathrm{diag}(s_1, s_2, \ldots, s_d)$ in which each component of the diagonal is defined as (assuming that the learned goal and starting positions differ in each component)

$$s_i = \frac{\mathbf{g}'_i - \mathbf{x}'_{0\,i}}{\mathbf{g}_i - \mathbf{x}_{0\,i}}, \quad i = 1, 2, \ldots, d,$$

we would obtain a formulation similar to (10), in which each component is scaled by $g - x_0$.

Remark 4. Choosing roto-dilatation has the advantage that the evaluation of the inverse transformation does not require numerically inverting the matrix. Indeed, since the matrix $\mathbf{S}$ of the transformation can be written as a non-zero scalar value multiplied by an orthogonal matrix

$$\mathbf{S} = a\,\mathbf{R}, \quad a \in \mathbb{R} \setminus \{0\},\ \mathbf{R} \in \mathbb{R}^{d \times d},\ \mathbf{R}^{-1} = \mathbf{R}^{\mathsf{T}},$$

where $\mathbf{R}$ is the rotation matrix, the inverse is simply

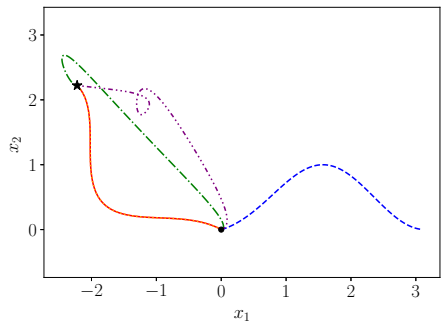
$$\mathbf{S}^{-1} = \frac{1}{a}\,\mathbf{R}^{-1} = \frac{1}{a}\,\mathbf{R}^{\mathsf{T}}.$$

Thanks to this property, the computation of the inverse matrix $\mathbf{S}^{-1}$ in (22) can be performed efficiently (since transposing a matrix is faster than inverting one).

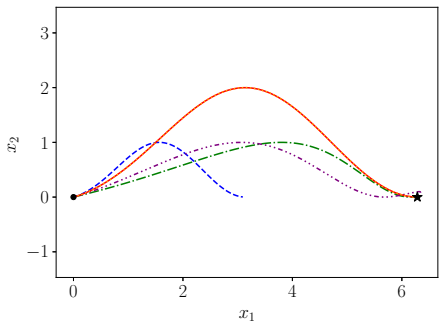
4.2. Results

In this Section, we present several synthetic tests and robotic experiments to show how DMP formulation (24) results in better behaviors when compared to formulation (11).

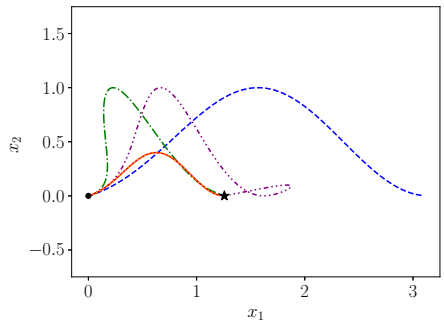
In the following, we refer to *classical* DMPs when talking about the implementation without exploiting the invariance property. Similarly, we refer to *extended* DMPs when talking about the formulation (24) we introduced in Section 4.1, in which the invariance property is exploited by using, as linear invertible transformation, the roto-dilatation defined in (25).



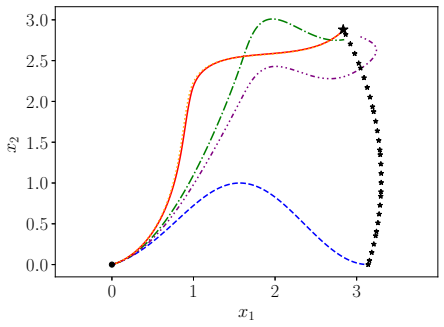
(a) Rotation



(b) Dilatation



(c) Shrinking



(d) Moving goal

Figure 10: Different behaviors obtained when changing the goal position. In Figures 10a, 10b, and 10c the goal is different from the learned one and still. In Figure 10d the goal starts in the learned position and moves continuously. The desired curve is given by (26). In all four plots, the desired curve is plotted using the blue dashed line. The execution of extended DMP is shown by the red full line for $\alpha = 4$, and by the orange dotted line for $\alpha = 2$ (in all these experiments, they overlap). The execution of classical DMPs are marked with the dash-dotted green line when $\alpha = 4$, and with the purple one dash three dots line when α is set to 2. The dot and star mark, respectively, the desired starting and goal positions $\mathbf{x}_0$ and $\mathbf{g}$. The stars trial in Figure 10d shows the movement of the goal.

4.2.1. Synthetic Tests.

We investigate the robustness gain ensured by the invariance property under rotation and dilatation of the reference system by considering two examples in which a trajectory is learned and then executed changing the relative position between goal and starting point $\mathbf{g} - \mathbf{x}_0$. In these tests, we do not change $\mathbf{x}_0$ since DMPs are natively translational invariant. We compare the different behaviors obtained with and without the scaling term $\mathbf{S}$ defined in (25).[1] For all the tests presented in this Section, we use mollifier-like basis functions (15). However, we remark that tests done with other classes of basis functions give similar results.

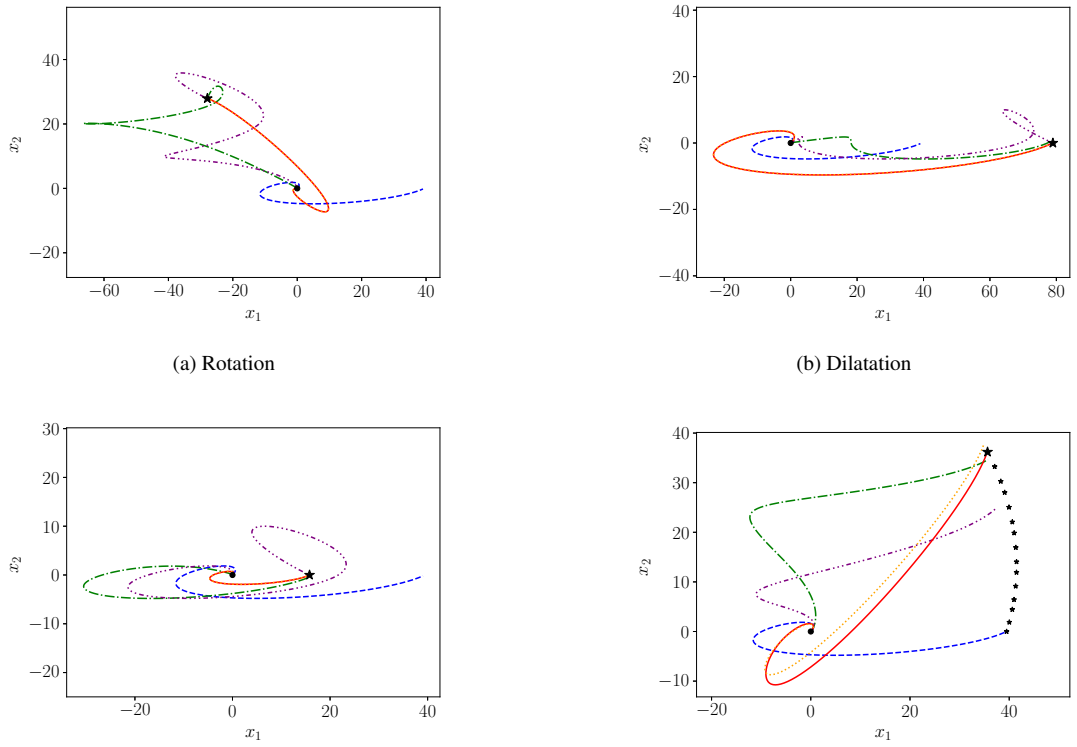
In the first simulation, we generate the desired curve in the plane:

$$\mathbf{x}(t) = \left(t, \sin^2(t)\right), \quad t \in [0, \pi]. \tag{26}$$

Then we perform the learning step to compute the weights $\boldsymbol{\omega}_i$, and we test the generalization properties of DMPs by changing in different manners the goal position. In particular, in Figures 10a, 10b, and 10c the goal is set to a different configuration and is still. Instead, in Figure 10d, the goal starts in the learned position and moves towards a final target configuration while the DMP is being executed. All tests are executed using both classical and extended DMPs.

We performed these tests with elastic term $K = 150$, and damping term $D = 2\sqrt{150} \approx 24.49$ for both components x_1 and x_2, that is $\mathbf{K} = K\mathbf{I}_2$ and $\mathbf{D} = D\mathbf{I}_2$, being $\mathbf{I}_2$ the 2×2 identity matrix. We test the generalization by using two different values of α in the canonical system (2), $\alpha = 4$ and $\alpha = 2$. Both these values for α

[1] We will not compare the results with the original DMPs (10) since the drawbacks of such formulation have already been discussed in the literature [14, 15, 27].



(a) Rotation

(b) Dilatation

(c) Shrinking

(d) Moving goal

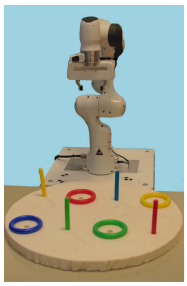
Figure 11: Different behaviors obtained when changing the goal position. In Figures 11a, 11b, and 11c the goal is different from the learned one and still. In Figure 11d the goal starts in the learned position and moves continuously. The desired curve is given by (27). In all four plots, the desired curve is plotted using the blue dashed line. The execution of extended DMP is shown by the red full line for $K = 150$, and by the orange dotted line for $K = 15$ (in Figures 11a–11c, they overlap). The execution of classical DMPs are marked with the dash-dotted green line when $K = 150$, and with the purple one dash three dots line when K is set to 15. The dot and star mark, respectively, the desired starting and goal positions $\mathbf{x}_0$ and $\mathbf{g}$. The stars trial in Figure 11d shows the movement of the goal.

result in a canonical system that vanishes at the final time within a tolerance smaller than 1%. Indeed, since the final time is $t_1 = \pi$ in (26), for $\alpha = 2$ we have $\exp(-\alpha T) = \exp(-2\pi) \approx 1.9\mathsf{e}-03$; while for $\alpha = 4$ we have $\exp(-\alpha T) = \exp(-4\pi) \approx 3.5\mathsf{e}-06$. Figure 10 shows the results of our tests.
In the second simulation, the desired curve is:

$$\mathbf{x}(t) = \left(t^2 \cos(t), t \sin(t)\right), \quad t \in [0, 2\pi]. \tag{27}$$

Also in this case we test the generalization properties of both classical DMPs and extended DMPs when the relative position $\mathbf{g} - \mathbf{x}_0$ is changed. In Figures 11a, 11b, and 11c the goal is set to a different configuration before we start executing the DMP. Instead, in Figure 11d, the goal starts in the learned position and moves towards a final target configuration while the DMP is being executed. The main difference is that in this second test we set $\alpha = 4$, and we use two different values for $\mathbf{K}$ (and, thus, for $\mathbf{D}$). Indeed, we set $\mathbf{K} = K\mathbf{I}$ and set K to assume value 150 and 15. In both cases, the damping term is set to $\mathbf{D} = \sqrt{K}\,\mathbf{I}_2$.

Both these tests show how extended DMPs are more robust than the classical DMP formulation since the trajectories generated by taking advantage of the invariance property have a shape that closely resembles the learned trajectory, while classical DMPs may generate trajectories that do not resemble the learned behavior. Moreover, we notice that the goodness of the generalization of classical DMPs heavily depends on the choice of the parameters. For instance, in the cases of dilatation and moving goal for (26), they generalize well when $\alpha = 4$, but fails when $\alpha = 2$ (see Figure 10b and 10d). Similarly, we observe that the generalization of classical DMPs heavily depends also on the choice of parameter $\mathbf{K}$ (and, consequently, $\mathbf{D}$). For instance, Figure 11d shows that the trajectories are different, even if



(a) Panda setup.

(b) daVinci setup.

Figure 12: Setups of the Peg & Ring task.

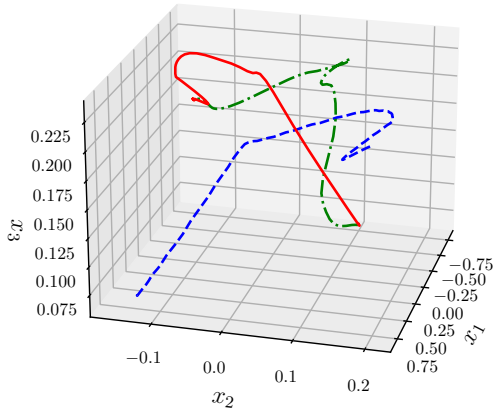


Figure 13: Generalization of extended DMPs (24). The blue dashed line shows the desired behavior, learned from the `carry` movement for the red ring. The solid red line shows the executed extended DMP (24) for the starting and ending points for the `carry` movement for the blue ring. The green dash-dotted line shows the behavior obtained using classical DMPs (11) adapted to the critical points of the blue ring.

the change in goal position is the same. On the other hand, we see that the generalization of the trajectory is robust against the particular choice of hyperparameters when using extended DMPs. Indeed, in almost all our tests, the generalizations obtained by extended DMPs completely overlap when changing the goal position. The only case in which there is an actual difference, even if hardly noticeable, is for the test in Figure 11d, i.e. when the goal is moving and we compare two DMPs with different values of $\mathbf{K}$ (and, thus, $\mathbf{D}$). This is due to two factors: the scaling matrix $\mathbf{S}$ depends explicitly on time (since $\mathbf{g}$ is moving), and different values for $\mathbf{K}$ influence the un-perturbed evolution of dynamical system (11). On the other hand, we observe that, in the case presented in 10d, there is no difference between the two trajectories, even if $\mathbf{S}$ depends on time because the un-perturbed evolution of the system is not influenced by changes in α.

4.2.2. Tests on Real Robots.

On a real setup, we test our new formulation by performing a task on a 7 Degrees-of-Freedom (DoF) industrial manipulator, a Panda robot from Franka Emika shown in Figure 12a. We will perform two tests. In the first test, the learned behavior will be used to execute the task on the same robot, and we will show how extended DMPs (24) result in a better adaptation than classical DMPs (11). The second test, instead, will show how extended DMPs (24) can be used to *transfer* the learned behavior to a different setup, in our case, the daVinci surgical robot from Intuitive.

The *Peg & Ring* task consists of grabbing, one at a time, four colored rings and moving them to the same-colored peg. The task consists of two gestures, namely `move`, in which the end-effector has to reach the ring, and `carry`, in which the ring is moved toward the peg. Automation of this task on surgical setups is a crucial aspect in Autonomous Robotic Surgery since it presents several challenges of real surgery (e.g. grasping) [9, 10]. During the learning phase, we learn a 3−dimensional DMP for both gestures, on the Panda, via kinesthetic teaching. For both tests, DMPs parameters are $\mathbf{K} = K\,\mathbf{Id}_3$ and $\mathbf{D} = D\,\mathbf{Id}_3$, with $K = 150$ and $D = 2\sqrt{K} \approx 24.49$, and $\alpha = 4$. As it can be seen from Figure 12a, the four `carry` movements are qualitatively different for each color: the red and blue rings must be moved 'in diagonal' along the base to be put in the corresponding pegs, while the green and yellow rings must be moved 'horizontally'.

To prove the adaptability of extended DMPs, we learn the desired behavior from the execution of the `carry` gesture for the red ring and use it to execute the gesture for all the rings in the scene. Figure 13 shows the result of this test. In particular, one can observe that the shape of the executed trajectory (for the blue ring) maintains a trajectory of similar shape to the learned behavior (from the execution for the red ring). For completeness, we plot the behavior obtained with classical DMPs (11). From this, it is clear that the behavior of extended DMPs success in maintaining the same shape as the learned behavior, while classical DMPs fail in doing so.

As the second test, we show how extended DMPs allow to 'transfer' the desired behavior between different robotic setups. To do so, we use the DMPs obtained from the execution performed on the Panda industrial manipulator to execute the Peg & Ring task on

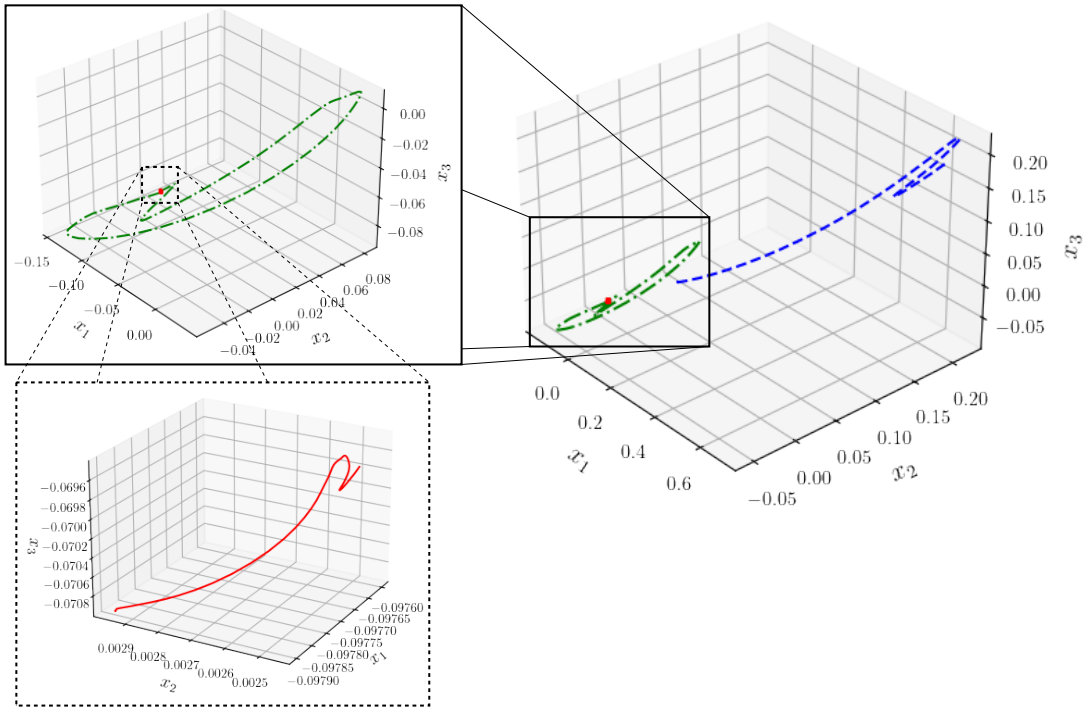


Figure 14: Generalization of DMPs between the learned trajectory on the Panda and the executed trajectory on the daVinci surgical robot. The blue dashed line shows the desired (and learned) trajectory, obtained on the Panda industrial manipulator via kinesthetic teaching. The red solid line shows the executed extended DMPs (24) on the daVinci surgical robot. The green dash-dotted line shows the adaptation of classical DMPs (11) to the same starting and goal positions as the red line.

the daVinci surgical robot (which setup can be seen in Figure 12b). Figure 14 shows the resulting trajectories for this test for an instance of the `move` gesture. Additionally, we plot the trajectory obtained by adapting classical DMPs (11) to the new starting and goal positions. This experiment shows that extended DMPs (24) are able to generalize to a very different length-scale, while maintaining a shape similar to the learned behavior, effectively permitting to transfer movement from the industrial manipulator to the surgical robot. Moreover, it is possible to observe that the trajectory obtained using classical DMPs fails in adapting to the new length-scale, generating a behavior that cannot be executed with the surgical robot.

5. Learning from Multiple Trajectories

DMPs have been used to learn trajectories from a single demonstration, because a set of demonstrated trajectories, in general, does not have the same starting and ending points, making the learning phase highly non-trivial.

So-called *Stylistic DMPs* [5, 32] were developed to learn from multiple demonstrations by introducing an additional variable, called *style parameter*, and by making the execution dependent on this new variable. This approach is useful when the "style" is needed to describe a trajectory (for instance, the trajectory may depend on the height of an obstacle). However, when the style is not an issue, this approach introduces additional and undesired variables that increase the complexity of the problem.

5.1. Linear Regression Over Aligned DMPs

We propose a technique that allows to extract a unique set of weights $\mathbf{w}_i \in \mathbb{R}^d$, $i = 0, 1, \ldots, N$, from a set of multiple observations as a single linear regression problem.

We consider a set of M *demonstrated trajectories* $\left\{\mathbf{x}^{(j)}(t), t \in \left[t_0^{(j)}, t_1^{(j)}\right]\right\}_{j=1}^{M}$. Using the technique introduced in Section 4.1 we transform each trajectory in such a way that all the starting and ending points are the same. We choose $\mathbf{x}_0 = \mathbf{0}$ and $\mathbf{g} = \mathbf{1}$, but we emphasize that any choice satisfying $\mathbf{x}_0 \neq \mathbf{g}$ would give the same

results. We compute the roto-dilatation matrices, for $j = 1, 2, \dots, M$,

$$\mathbf{S}^{(j)} = \frac{\|\mathbf{1} - \mathbf{0}\|}{\left\|\mathbf{x}^{(j)}\left(t_1^{(j)}\right) - \mathbf{x}^{(j)}\left(t_0^{(j)}\right)\right\|} \widehat{\mathbf{R}^{\mathbf{1}-\mathbf{0}}_{\mathbf{x}^{(j)}\left(t_1^{(j)}\right)-\mathbf{x}^{(j)}\left(t_0^{(j)}\right)}}, \tag{28}$$

and use these matrices to create the new set of transformed trajectories

$$\left\{\check{\mathbf{x}}^{(j)}(t) = \mathbf{S}^{(j)}\mathbf{x}^{(j)}(t), t \in \left[t_0^{(j)}, t_1^{(j)}\right]\right\}_{j=1,2,\dots,M}.$$

Next, we perform a *time scaling*, so that each curve has $[0, T]$ as time domain, for a given $T > 0$. To do so, we define

$$\tilde{\mathbf{x}}^{(j)}(t) = \check{\mathbf{x}}^{(j)}\left(\frac{t_1^{(j)} - t_0^{(j)}}{T}\, t + t_0^{(j)}\right). \tag{29}$$

It's easy to see that

$$\tilde{\mathbf{x}}^{(j)}(0) = \check{\mathbf{x}}^{(j)}\left(t_0^{(j)}\right),$$
$$\tilde{\mathbf{x}}^{(j)}(T) = \check{\mathbf{x}}^{(j)}\left(t_1^{(j)}\right).$$

From these two step, we obtain a new set of 'transformed' curves $\{\tilde{\mathbf{x}}^{(j)}(t), t \in [0, T]\}_{j=1}^{M}$, each with time domain $[0, T]$, and $\mathbf{0}$ and $\mathbf{1}$ as starting and ending points respectively.

Equation (11a) allows to compute the set of forcing terms $\{\mathbf{f}^{(j)}(s)\}_{j=1,2,\dots,M}$, and then we are able to compute the set of weights $\{\mathbf{w}_i\}_{i=0,1,\dots,N}$ that minimizes the sum of the squared errors (w.r.t. the L_2-norm) between the function generated using (3) and the forcing terms $\mathbf{f}^{(j)}(s)$, $j = 1, 2, \dots, M$.

For this purpose, we decompose the problem in each independent component. For each component $p = 1, 2, \dots, d$, we look for the *weight vector* $\boldsymbol{\omega}^\star = [\omega_0^\star, \omega_1^\star, \dots, \omega_N^\star]^\mathsf{T} \in \mathbb{R}^{N+1}$ satisfying

$$\boldsymbol{\omega}^\star = \arg\min_{\boldsymbol{\omega}\in\mathbb{R}^{N+1}} \underbrace{\sum_{j=1}^{M} \left\| \frac{\sum_{i=0}^{N} \omega_i \varphi_i(s)}{\sum_{i=0}^{N} \varphi_i(s)}\, s - f_p^{(j)}(s) \right\|_2^2}_{F(\boldsymbol{\omega})},$$

where $f_p^{(j)}(s)$ denotes the p-th component of $\mathbf{f}^{(j)}(s)$ and is computed as

$$f_p^{(j)}(s(t)) = \frac{\dot{v}_p^{(j)} + D_p v_p}{K_p} - (g_p^{(j)} - x_p(t)) + (g_p - x_{0,p})s(t), \tag{30}$$

with $s(t) = \exp(-\alpha t)$.

The existence and uniqueness of a minimum for F is guaranteed by its continuity and strict convexity. Moreover, since F is smooth, $\boldsymbol{\omega}^\star$ is the vector that nullifies its gradient: $\nabla_\omega F(\cdot)|_{\omega^\star} = \mathbf{0}$. Let us denote with $\mathbf{G}$ the gradient of F: $\mathbf{G}(\cdot) \doteq \nabla_\omega F(\cdot) : \mathbb{R}^{N+1} \to \mathbb{R}^{N+1}$. Its h-th component, denoted by G_h is (recalling that $\|\zeta(s)\|_2^2 = \int_{s_0}^{s_1} (\zeta(s))^2\, ds$)

$$\begin{aligned} G_h(\boldsymbol{\omega}) &= \frac{\partial F}{\partial \omega_h}(\boldsymbol{\omega}) \\ &= \sum_{j=1}^{M} \int_{s_0}^{s_1} 2\left(\frac{\sum_{i=0}^{N} \omega_i \varphi_i(s)}{\sum_{i=0}^{N} \varphi_i(s)}\, s - f_p^{(j)}(s)\right) \\ &\qquad \left(\frac{\varphi_h(s)}{\sum_{i=0}^{N} \varphi_i(s)}\, s\right) ds. \end{aligned}$$

Function G_h is linear in $\boldsymbol{\omega}$, thus the minimization problem can be written as a linear system:

$$\mathbf{A}\,\boldsymbol{\omega} = \mathbf{b}. \tag{31}$$

The component in row h and column k of $\mathbf{A}$, a_{hk}, is, for $h, k \in \{0, 1, \dots, N\}$,

$$\begin{aligned} a_{hk} &= \frac{\partial G_h}{\partial \omega_k}(\omega) \\ &= \sum_{j=1}^{M} \int_{s_0}^{s_1} 2\frac{\varphi_h(s)\varphi_k(s)}{\left(\sum_{i=0}^{N} \varphi_i(s)\right)^2}\, s^2\, ds \\ &= 2M \int_{s_0}^{s_1} \frac{\varphi_h(s)\varphi_k(s)}{\left(\sum_{i=0}^{N} \varphi_i(s)\right)^2}\, s^2\, ds, \end{aligned} \tag{32}$$

while the h-th component of the vector $\mathbf{b}$, b_h, is, for $h = 0, 1, \dots, N$,

$$b_h = \sum_{j=1}^{M} \int_{s_0}^{s_1} 2\,\frac{\varphi_h(s)}{\sum_{i=0}^{N} \varphi_i(s)}\, f_p^{(j)}(s)\, s\, ds. \tag{33}$$

Using any quadrature formula (e.g. Simpson's rule), the integrals in equations (32) and (33) can be computed for each $h, k = 0, 1, \dots, N$ giving matrix $\mathbf{A}$ and vector $\mathbf{b}$ of (31). Thus, we can solve the linear problem and obtain $\boldsymbol{\omega}^\star$. We recall that this procedure has to be repeated for each component $p = 1, 2, \dots, d$.

The algorithm to perform regression is summarized in Algorithm 1.

Remark 5. Although we presented the algorithm using mollifier-like basis functions $\{\varphi_i\}_{i=0,1,\dots,N}$, this algorithm works for any choice of the set of basis functions.

Remark 6. This approach does not encode information about 'styles', differently from [32]. Thus, the generalization of the DMP depends only on the starting and goal positions $\mathbf{x}_0$ and $\mathbf{g}$.

Algorithm 1 Regression Over Multiple Demonstrations

Input: Set of desired trajectories $\{\mathbf{x}^{(j)}(t)\}_{j=1,2,\dots,M}$, set of basis functions $\{\varphi_i(s)\}_{i=0,1,\dots,N}$, DMPs hyperparameter $\mathbf{K}, \mathbf{D}$ and α, final time $T > 0$;

Output: Set of weights $\{\boldsymbol{\omega}^{(p)}\}_p$ for each direction $p = 1, 2, \dots, d$.

▷ Align the trajectories (both temporally and spatially)

for $j = 1, 2 \dots, M$ **do**
  Compute $\mathbf{S}^{(j)}$ using (28)
  Compute $\check{\mathbf{x}}^{(j)}(t) = \mathbf{S}^{(j)}\mathbf{x}(t)$
  Compute $\tilde{\mathbf{x}}^{(j)}(t)$ using (29)
end for
Evaluate the canonical system extrema $s_0 = 1, s_1 = e^{-\alpha T}$

▷ Compute the matrix $\mathbf{A} = [a_{hk}]_{h,k=0,1,\dots,N}$

for $h = 0, 1, \dots, N$ **do**
  $a_{hh} = 2M \int_{s_0}^{s_1} \frac{\varphi_h(s)^2}{\left(\sum_{i=0}^{N} \varphi_i(s)\right)^2} s^2 \, ds$
  for $k = h + 1, h + 2, \dots, N$ **do**
    $a_{hk} = 2M \int_{s_0}^{s_1} \frac{\varphi_h(s)\varphi_k(s)}{\left(\sum_{i=0}^{N} \varphi_i(s)\right)^2} s^2 \, ds$
    $a_{kh} = a_{hk}$
  end for
end for

▷ For loop along the directions

for $p = 1, 2, \dots, d$ **do**
  For each trajectory $\tilde{\mathbf{x}}^{(j)}(t)$ compute the forcing term $f^{(j)}(s)$ for direction p using (30).

▷ Compute the vector $\mathbf{b} = [b_h]_{h=0,1,\dots,N}$

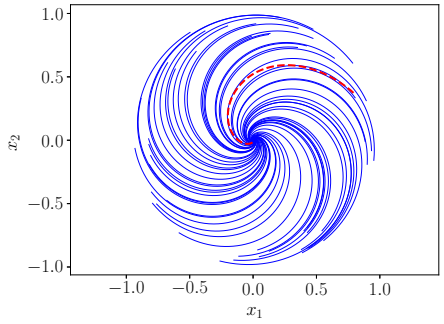
  for $h = 0, 1, \dots, N$ **do**
    $b_h = \sum_{j=1}^{M} \int_{s_0}^{s_1} 2 \frac{\varphi_h(s)}{\sum_{i=0}^{N} \varphi_i(s)} f^{(j)}(s) \, s \, ds$
  end for
  Solve the minimization problem: $\mathbf{A}\boldsymbol{\omega}^{(p)} = \mathbf{b}$
end for

Remark 7. The presented method performs classical linear regression over a set of observed trajectories, and it gives a non-probabilistic result, differently from other approaches, such as Probabilistic Movement Primitives [22] and Kernelized Movement Primitives [23], which are purely probabilistic approaches.

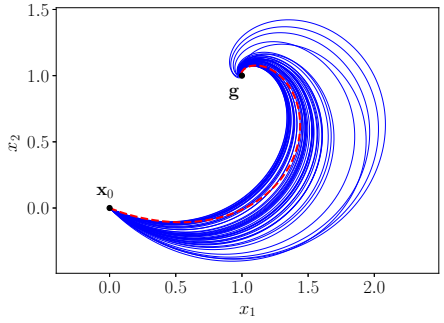
5.2. Results

In this Section, we test the ability to learn from multiple demonstrations by performing both synthetic tests and experiments with real robots.

As first synthetic test, we create a set of trajectories $\{\mathbf{x}^{(j)}(t) = (x_1^{(j)}(t), x_2^{(j)}(t))\}_j$ by integrating a known dynamical system. The second synthetic test is performed using a well-know dataset containing trajectories extracted from surgical tasks.



(a) Synthetic test before rescaling.



(b) Synthetic test after rescaling.

Figure 15: Tests for our proposed approach to perform DMPs learning from multiple trajectories. The trajectories are obtained by integrating (34) with different initial conditions. In both plots the solid lines represent the demonstrated trajectories, while the dashed one is an execution of the learned DMP. The tests are plotted both before (Figure 15a) and after (Figure 15b) the spatial scaling step. In Figure 15b, the dot represents the initial position $\mathbf{x}_0 = (0, 0)$, while the star marks the goal $\mathbf{g} = (1, 1)$.

On the real setup, we use real executions to automate the *Peg & Ring* task on poth the Panda industrial manipulator and the daVinci surgical robot. In the experiment on the daVinci, we make the task more challenging by integrating extended DMPs (24) with obstacle avoidance methods.

5.2.1. Synthetic Test.

For the test with synthetic data, we generate the trajectories computing the solution of the dynamical system

$$\begin{cases} \dot{x}_1 = x_1^3 + x_2^2 \, x_1 - x_1 - x_2 \\ \dot{x}_2 = x_2^3 + x_1^2 \, x_2 + x_1 - x_2 \end{cases}, \tag{34}$$

which is known to have a *alpha-limit* on the circumference of radius 1 and an attractive equilibrium at the origin. The set of (fifty) trajectories is generated by

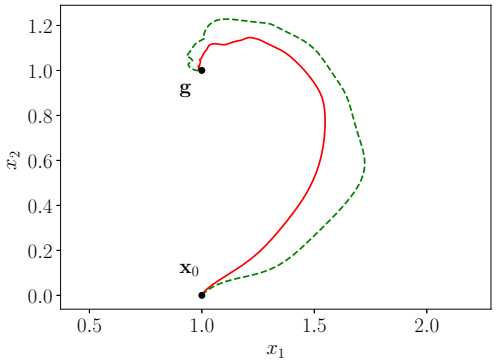


Figure 16: Comparison of two DMPs obtained by a noisy data-set. The solid red line shows the trajectory obtained using the proposed method for regression over multiple observations. The dashed green line shows the execution of a DMP learned from one sample of the dataset.

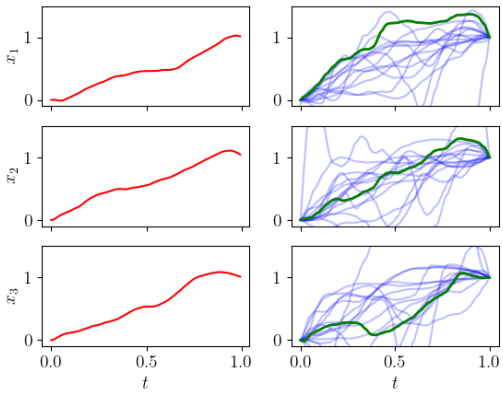


Figure 17: Comparison between two DMPs obtained by the JIGSAW dataset. The solid red line (on the left) shows the execution of the DMP obtained via regression. The dataset is shown on the right in blue. The green line marks a DMP obtained by a single element of the dataset.

choosing a random angle $\theta_0 \in [0, 2\pi)$ and a random radius $\rho_0 \in (0.8, 1)$, and then setting as initial position $x_1(0) = \rho_0 \cos(\theta_0)$, $x_2(0) = \rho_0 \sin(\theta_0)$. Then the dynamical system is integrated using the *classic fourth-order Runge-Kutta method* up to a random final time $T \in (5, 10)$. Choosing a final time greater than 5 allows the system to reach the origin with an error smaller than 2%: $\|\mathbf{x} - \mathbf{0}\| < 0.02$. Moreover, having the final time change within executions mimics the time inconsistencies that may arise when learning trajectories from real executions. After generating the set of observations, we use Algorithm 1 to generate a DMP. The DMP's hyperparameters are $\mathbf{K} = K\,\mathbf{I}_2$, $\mathbf{D} = D\,\mathbf{I}_2$, with $K = 150\,D = 2\sqrt{K} \approx 24.49$, and $\alpha = 4$. Finally, we randomly select an initial point $\mathbf{x}_0$ with $\|\mathbf{x}_0\| \in (0.8, 1)$ and execute the obtained DMP by setting as goal the attractive equilibrium of the system: $\mathbf{g} = (0, 0)$.
Figure 15 shows the results of these tests. In particular Figure 15a shows the set of trajectories obtained by integrating the ODE (34), together with an execution of the obtained DMP. Figure 15b shows the same trajectories when spatially rescaled to have $\mathbf{0}$ and $\mathbf{1}$ as starting and ending points respectively.

To highlight the usefulness of learning from a set of observations, we added a Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ of null mean and variance $\sigma^2 = 5 \cdot 10^{-5}$ to the set of trajectories obtained by solving the dynamical system (34). We then use Algorithm 1 to extract a DMP (the hyperparameters are the same as the previous test). Moreover, we learned a second DMP, with the same hyperparameters, using as desired trajectory a single trajectory of the dataset.
In Figure 16 it can be seen that learning from a single sample of the dataset results in a DMP with undesired oscillation. On the other hand, performing regression heavily reduces these oscillations.
To emphasize this aspect, we perform a similar comparison on the JIGSAW [30] dataset. In particular, we extract all the occurrences of the gesture 'G1' (*reaching needle with the right hand*) and use them to extract a unique DMP via Algorithm 1. DMPs' parameters are $K = 150, D = 2\sqrt{K} \approx 24.49$, and $\alpha = 4$. Since this gesture involves only the right arm, we extract only the Cartesian components for it and we ignore the left end-effector. As it can be seen in Figure 17, in particular from the second component x_2, oscillations are reduced when a unique behavior is extracted from multiple demonstrations.

5.2.2. *Tests on Real Robots.*

In this Section, we present two experiments on real robotic setups to shows the ability of our proposed method for DMPs regression to extract a unique behavior from multiple observations.

Test on the Panda Industrial Manipulator. For the experiments on a real setup, we executed the Peg & Ring task with a Panda industrial manipulator. The main phases of the task are presented in Figure 18. The task can be decomposed into two gestures, namely `move` and `carry`. The first gesture is executed to reach the grasping point of the ring with the robot end-effector. The second gesture, instead, is used to carry the ring toward the same colored peg. The task uses four colored rings (and, thus, four same-colored pegs) and we execute the

(a) Initial setup.

(b) Grasped ring.

(c) Carrying the ring.

(d) Released ring.

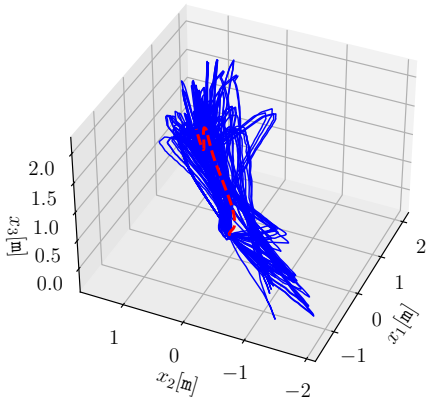
Figure 18: The Peg&Ring task with the Panda robot.

task a total of fifty times on the Panda robot, changing the grasping and releasing positions (i.e., the starting and final positions of each gesture). Since we executed the task fifty times, we obtained a total of 200 samples for both the `move` and `carry` gestures.

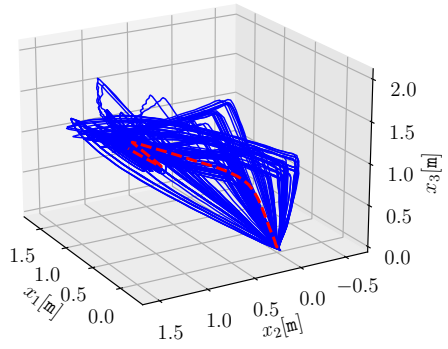
Figure 19a and 19b show the trajectories of, respectively, `move` and `carry` obtained with the industrial manipulator together with an execution of the obtained DMP with parameters $K = 150, D = 2\sqrt{K} \approx 24.49$, and $\alpha = 4$. These are plotted after the rescaling so that $\mathbf{x}_0 = \mathbf{0}$, and $\mathbf{g} = \mathbf{1}$ for both gestures.

We then used these DMPs to automate the Peg & Ring task. We implemented a Finite State Machine which takes as input the order of colors and then automatically executes the task by alternating the `move` and `carry` gestures. The learned DMPs are used to model the Cartesian evolution of the robot's end-effector of the robot. Figure 18 shows the step of the task's execution for the green ring. Figure 20 shows the movement of the end-effector.

Test on the daVinci Surgical Robot. As a second experiment on real robots, we perform the Peg & Ring task also on the daVinci surgical manipulator (which setup is shown in Figure 21). In this case, the task is more complicated since it is executed with two end-effectors, and an additional movement, called `pass` has to be learned. This gesture is used to transfer the ring from one end-effector to the other. The workspace is split into two parts (see Figure 21), left and right and each



(a) `move`



(b) `carry`

Figure 19: Experiments on the real robot. For both gestures (`move` and `carry`), the trajectories of the dataset are plotted using solid lines, while the execution of the obtained DMP is plotted using a dashed line. All the trajectories are plotted after the rescaling, so that $\mathbf{x}_0 = \mathbf{0}$, and $\mathbf{g} = \mathbf{1}$ for both gestures.

end-effector must move only in one of the portions. For this reason, if the ring initial position and the target-peg placement are in the same half, the end-effector picking the ring will be the same placing it, and the `pass` gesture will be unnecessary. On the other hand, when the ring is in one half of the workspace and the same-colored peg is in the other, the ring will be transferred between end-effectors.

In our test, we set the initial placement of the rings in such a way that the arm picking up the ring always differs from the hand placing it. Thus, the ring has always to be passed from one end-effector to the other. On this setup, the available workspace is proportionally smaller w.r.t. the robot dimension. Moreover, the pegs and rings are proportionally bigger. For this reason, we integrate

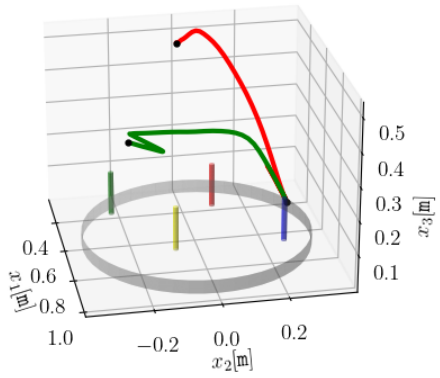


Figure 20: Plot of the `move` (in red) and `carry` (in green) gesture in an automatic execution of the Peg & Ring task on the Panda robot.

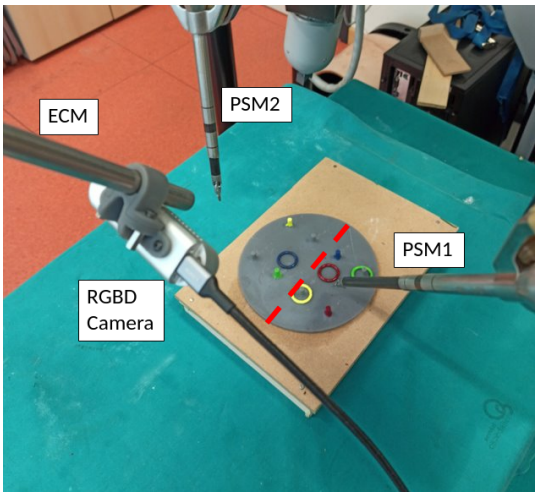


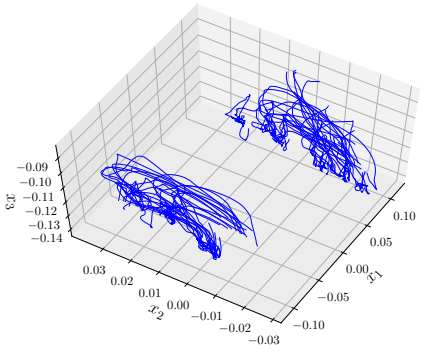
Figure 21: daVinci setup for the bimanual Peg & Ring task. The red line splits the workspace in left and right portion.

the extended DMP system (24) with the obstacle avoidance method presented in [16] in order to avoid collisions with the pegs. Thus, (24a) reads

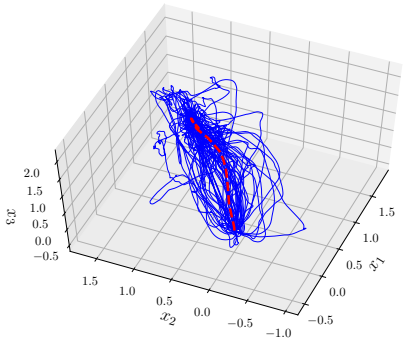
$$\tau\dot{\mathbf{v}} = \mathbf{K}'(\mathbf{g} - \mathbf{x}) - \mathbf{D}'\mathbf{v} - \mathbf{K}'(\mathbf{g} - \mathbf{x}_0)s + \mathbf{K}'\mathbf{f}(s) + \mathbf{p}(\mathbf{x}),$$

where, intuitively, $\mathbf{p}(\mathbf{x})$ is a 'repulsive term' that pushes the trajectory away from the obstacle.

Since the `move` and `carry` gestures require the movement of a single end-effector at a time, we model both with a 3−dimensional DMP. On the other hand, the `pass` gesture requires that the two end-effectors move together towards the center of the workspace. Thus, this third gesture is modeled using a single 6−dimensional DMP. In this way, the two end-effectors share the same canonical system so that their movement is synchronized. Figure 22 shows the dataset and obtained DMP for the `move` gesture.



(a) Original dataset.



(b) Roto-dilatated dataset (in blu) and obtained DMP (in dashed red).

Figure 22: Trajectory samples for the `move` gesture of the Peg & Ring task on the daVinci surgical robot.

The obtained DMPs allowed to successfully automate the task.

6. Conclusions

In this work, we have highlighted some of the weak aspects of the DMP framework, proposing three major modifications to overcome them.

At first, we proposed a new formulation for the set of basis functions. We proved the proposed mollifier-like basis functions to be good approximators for the forcing term, while providing a faster and more stable learning phase. Moreover, being compactly supported, this new set of basis functions allows to update a DMP when only a portion of the trajectory has to be modified. Secondly, we proposed a strategy to exploit the invariance under affine transformations of the DMP formulation (11). In particular, we showed how to generalize the learned trajectory to any length scale and any rotation of the reference frame maintaining the qualitative shape of the

learned trajectory, specifying that the same approach can be extended to any invertible transformation. Finally, we solved one of the major issues in DMPs, which is the inability of the original framework to learn a common behavior from multiple observations.

As future work, we aim to extend these improvements to the unit quaternion formulation of DMPs [20], in order to make more robust also the orientation formulation of DMPs.
Moreover, we aim to use the proposed formulation to improve the 'Surgical Task Automation Frameworks' presented in [9] and [10], in which DMPs are used to generate surgical gestures.

Funding

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, ARS (Autonomous Robotic Surgery) project, grant agreement No. 742671.

References

[1] A. J. Ijspeert, J. Nakanishi, S. Schaal, Movement imitation with nonlinear dynamical systems in humanoid robots, in: Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on, Vol. 2, IEEE, 2002, pp. 1398–1403.

[2] A. J. Ijspeert, J. Nakanishi, S. Schaal, Learning attractor landscapes for learning motor primitives, in: Advances in neural information processing systems, 2003, pp. 1547–1554.

[3] S. Schaal, Dynamic movement primitives-a framework for motor control in humans and humanoid robotics, in: Adaptive motion of animals and machines, Springer, 2006, pp. 261–280.

[4] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, S. Schaal, Dynamical movement primitives: learning attractor models for motor behaviors, Neural computation 25 (2) (2013) 328–373.

[5] T. Matsubara, S.-H. Hyon, J. Morimoto, Learning stylistic dynamic movement primitives from multiple demonstrations, in: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, 2010, pp. 1277–1283.

[6] A. Ude, A. Gams, T. Asfour, J. Morimoto, Task-specific generalization of discrete and periodic dynamic movement primitives, IEEE Transactions on Robotics 26 (5) (2010) 800–815.

[7] T. Kulvicius, K. Ning, M. Tamosiunaite, F. Worgötter, Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting, IEEE Transactions on Robotics 28 (1) (2012) 145–157.

[8] R. Wang, Y. Wu, W. L. Chan, K. P. Tee, Dynamic movement primitives plus: For enhanced reproduction quality and efficient trajectory modification using truncated kernels and local biases, in: Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on, IEEE, 2016, pp. 3765–3771.

[9] M. Ginesi, D. Meli, H. C. Nakawala, A. Roberti, P. Fiorini, A knowledge-based framework for task automation in surgery, in: 2019 19th International Conference on Advanced Robotics (ICAR), 2019, pp. 37–42.

[10] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, P. Fiorini, Autonomous task planning and situation awareness in robotic surgery*, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020, pp. 3144–3150.

[11] P. Pastor, L. Righetti, M. Kalakrishnan, S. Schaal, Online movement adaptation based on previous sensor experiences, in: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 365–371.

[12] P. Pastor, M. Kalakrishnan, L. Righetti, S. Schaal, Towards associative skill memories, in: Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on, IEEE, 2012, pp. 309–315.

[13] D. Kappler, P. Pastor, M. Kalakrishnan, M. Wüthrich, S. Schaal, Data-driven online decision making for autonomous manipulation., in: Robotics: Science and Systems, 2015.

[14] D.-H. Park, H. Hoffmann, P. Pastor, S. Schaal, Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields, in: Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on, IEEE, 2008, pp. 91–98.

[15] H. Hoffmann, P. Pastor, D.-H. Park, S. Schaal, Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance, in: Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, IEEE, 2009, pp. 2587–2592.

[16] M. Ginesi, D. Meli, A. Calanca, D. Dall'Alba, N. Sansonetto, P. Fiorini, Dynamic movement primitives: Volumetric obstacle avoidance, in: 2019 19th International Conference on Advanced Robotics (ICAR), 2019, pp. 234–239.

[17] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, P. Fiorini, Dynamic movement primitives: Volumetric obstacle avoidance using dynamic potential functions, Journal of Intelligent & Robotic Systems 101 (4) (2021) 79. doi:10.1007/s10846-021-01344-y.
URL https://doi.org/10.1007/s10846-021-01344-y

[18] A. Gams, B. Nemec, A. J. Ijspeert, A. Ude, Coupling movement primitives: Interaction with the environment and bimanual tasks, IEEE Trans. Robotics 30 (4) (2014) 816–830.

[19] A. Ude, B. Nemec, T. Petrić, J. Morimoto, Orientation in cartesian space dynamic movement primitives, in: Robotics and Automation (ICRA), 2014 IEEE International Conference on, IEEE, 2014, pp. 2997–3004.

[20] M. Saveriano, F. Franzel, D. Lee, Merging position and orientation motion primitives, in: International Conference on Robotics and Automation (ICRA), 2019, 2019.

[21] H. B. Amor, G. Neumann, S. Kamthe, O. Kroemer, J. Peters, Interaction primitives for human-robot cooperation tasks, in: Robotics and Automation (ICRA), 2014 IEEE International Conference on, IEEE, 2014, pp. 2831–2837.

[22] A. Paraschos, C. Daniel, J. R. Peters, G. Neumann, Probabilistic movement primitives, in: Advances in neural information processing systems, 2013, pp. 2616–2624.

[23] Y. Huang, L. Rozo, J. Silvério, D. G. Caldwell, Kernelized movement primitives, The International Journal of Robotics Research 38 (7) (2019) 833–852.

[24] S. Calinon, A tutorial on task-parameterized movement learning and retrieval, Intelligent service robotics 9 (1) (2016) 1–29.

[25] A. Sidiropoulos, Z. Doulgeri, A reversible dynamic movement primitive formulation, arXiv preprint arXiv:2010.07708 (2020).

[26] F. J. Abu-Dakka, V. Kyrki, Geometry-aware dynamic movement primitives, arXiv preprint arXiv:2003.06061 (2020).

[27] P. Pastor, H. Hoffmann, T. Asfour, S. Schaal, Learning and generalization of motor skills by learning from demonstration, in: Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, IEEE, 2009, pp. 763–768.

[28] H. Wendland, Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree, Advances in computational Mathematics 4 (1) (1995) 389–396.

[29] R. Schaback, A practical guide to radial basis functions, Electronic Resource 11 (2007).

[30] Y. Gao, S. S. Vedula, C. E. Reiley, N. Ahmidi, B. Varadarajan, H. C. Lin, L. Tao, L. Zappella, B. Béjar, D. D. Yuh, et al., Jhu-isi gesture and skill assessment working set (JIGSAWS): A surgical activity dataset for human motion modeling, in: MICCAI Workshop: M2CAI, Vol. 3, 2014, p. 3.

[31] O. I. Zhelezov, N-dimensional rotation matrix generation algorithm, American Journal of Computational and Applied Mathematics 7 (2) (2017) 51–57.

[32] T. Matsubara, S.-H. Hyon, J. Morimoto, Learning parametric dynamic movement primitives from multiple demonstrations, Neural networks 24 (5) (2011) 493–500.