Skill learning and action recognition by arc-length dynamic movement primitives

Timotej Gašpar[a,b,*], Bojan Nemec[a], Jun Morimoto[a], Aleš Ude[a,b]

[a]*Humanoid and Cognitive Robotics Lab,*
Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute
Jamova cesta 39, 1000 Ljubljana, Slovenia
[b]*Department of Brain Robot Interface, ATR Computational Neuroscience Labs*
2-2-2 Hikaridai, Seika-cho, Kyoto 619-0288, Japan
e-mail: {timotej.gaspar, bojan.nemec, ales.ude}@ijs.si, xmorimo@atr.jp

Abstract

Effective robot programming by demonstration requires the availability of multiple demonstrations to learn about all relevant aspects of the demonstrated skill or task. Typically, a human teacher must demonstrate several variants of the desired task to generate a sufficient amount of data to reliably learn it. Here a problem often arises that there is a large variability in the speed of execution across human demonstrations. This can cause problems when multiple demonstrations are compared to extract the relevant information for learning. In this paper we propose an extension of dynamic movement primitives called *arc-length dynamic movement primitives*, where spatial and temporal components of motion are well separated. We show theoretically and experimentally that the proposed representation can be effectively applied for robot skill learning and action recognition even when there are large variations in the speed of demonstrated movements.

Keywords: programming by demonstration, skill learning, action recognition,

*Corresponding author
Email address: timotej.gaspar@ijs.si (Timotej Gašpar)

1. Introduction

A powerful movement representation is essential for a successful implementation of robot learning and action recognition algorithms. In recent years nonlinear dynamic systems have become widely used in robotics due to the many benefits they offer: ability to represent both point-to-point and periodic movements, easy computation of open parameters for accurate trajectory representation, ability to incorporate coupling terms for interaction with the environment, robustness against perturbations, ease of modulation of control parameters, etc. [1]. Dynamic movement primitives (DMPs) developed by Ijspeert et al. [2, 1] were the first variant of nonlinear dynamic systems, i. e. systems of nonlinear differential equations, proposed for robot trajectory representation and control. A number of methodologies based on DMPs have been proposed in the literature since then, e. g. methods for learning from multiple demonstrations [3, 4, 5], reinforcement learning [6, 7], synchronization of dual-arm behaviors [8, 9], Cartesian space movement generation [10, 11], human-robot interaction [12], adaptation to new start and goal constraints [13], etc.

While it is possible to compute the appropriate DMPs on the fly when the training data comprises multiple demonstrations [14], there exist also other types of dynamic systems that can represent multiple variants of the desired behaviour within a single equation system: dynamic systems with Gaussian mixture models [15, 16, 17] and probabilistic movement primitives [18, 19]. These representations require more data for learning and are computationally more expensive than one-shot learning of DMPs, but they can represent variants of a given movement skill

within the same dynamical system. In this paper we focus on how to deal with speed variability within the DMP framework, but our approach is applicable also to other movement representation schemes based on dynamic systems.

The issue of how to time-align multiple human movements has been addressed in the recent work on interaction primitives [20, 19]. While initially standard dynamic time warping [21] was applied to solve the time alignment problem [20], a smooth and continuous warping function was developed in [19] to enhance the performance of the approach. In another work, Ewerton et al. [22] applied expectation-maximization algorithm to time-align multiple phase parameters.

The problem of time-scaling of robot movements has been addressed also in our research [23, 24]. The main idea of our work was to introduce additional parameters to nonlinearly scale the DMP equations, which results in speeding up or slowing down the movement represented by a DMP. We showed how reinforcement learning and iterative learning control can be used to compute the optimal scaling parameters for tasks that involve complex liquid dynamics, e.g. carrying glass filled with liquid, pouring, etc. In this paper we further advance the scaling of dynamic movement primitives and develop a new representation that effectively separates the temporal and spatial aspects of motion. This is achieved by formulating the DMP equations as derivatives of arc-length (natural parameter) instead of time. The speed of movement, which carries information about timing, is encoded separately. The resulting representation is called arc-length dynamic movement primitive (AL-DMP). Unlike in [20, 22, 19], where the issue of variability in the speed of execution of human demonstrations was addressed by explicitly estimating the optimal time alignment, this is not necessary in our approach because spatial and temporal aspects of motion are well separated in AL-

DMPs. Thus learning of the spatial course of movement can be made independent of timing issues.

In the following we first explain how time and space can be separated in dynamic movement primitives, which leads to the concept of arc-length dynamic movement primitives. Next we show how arc-length dynamic movement primitives can be used for trajectory representation, robot control, action recognition, and skill learning from multiple demonstrations. The paper concludes with a discussion of issues arising in the applications of AL-DMPs.

2. Arc-length dynamic movement primitives

A point-to-point motion $\boldsymbol{y}(t) \in \mathbb{R}^d$ of a robot with d degrees of freedom can be specified by a second-order system of nonlinear differential equations [1]

$$\tau \dot{\boldsymbol{z}} = \alpha_z(\beta_z(\boldsymbol{g}-\boldsymbol{y})-\boldsymbol{z})+\mathbf{F}(x), \tag{1}$$

$$\tau \dot{\boldsymbol{y}} = \boldsymbol{z}, \tag{2}$$

where $\boldsymbol{z} \in \mathbb{R}^d$ is equal to the scaled velocity of motion, $\boldsymbol{g} \in \mathbb{R}^d$ is the final value of $\boldsymbol{y}$ on the trajectory, $\mathbf{F}$ is a nonlinear forcing term, and $x \in \mathbb{R}$ is the phase variable defined as

$$\tau \dot{x} = -\alpha_x x. \tag{3}$$

Phase x has been introduced to avoid explicit time dependency. It is fully defined by setting $\alpha_x > 0$ and x(0) = 1. Eq. system (1) – (3) constitutes a *dynamic movement primitive* (DMP). By properly selecting constants $\tau, \alpha_z, \beta_z \in \mathbb{R}$, e. g. $\tau > 0$ and $\alpha_z = 4\beta_z$, the linear part of equation system (1) – (2) becomes critically damped and $\boldsymbol{y}$, $\boldsymbol{z}$ monotonically converge to a unique attractor point at $\boldsymbol{y} = \boldsymbol{g}$,

$\boldsymbol{z} = 0$ [1]. The forcing term $\mathbf{F}(x)$ is usually defined by a linear combination of radial basis functions

$$\mathbf{F}(x) = \mathrm{diag}(\boldsymbol{g} - \boldsymbol{y}^0)\frac{\sum_{i=1}^{N} \boldsymbol{w}_i \Psi_i(x)}{\sum_{i=1}^{N} \Psi_i(x)} x,\ \Psi_i(x) = \exp\left(-h_i\,(x - c_i)^2\right), \qquad (4)$$

where c_i are the centers of Gaussians distributed along the phase of the movement, h_i their widths, and $\mathrm{diag}(\boldsymbol{g} - \boldsymbol{y}^0) \in \mathbb{R}^{d \times d}$ denotes a diagonal matrix with components of vector $\boldsymbol{g} - \boldsymbol{y}^0$ on the diagonal. The role of $\mathbf{F}$ is to adapt the dynamics of (1) – (2) to the desired trajectory, thus enabling the system to reproduce any smooth movement from the initial position $\boldsymbol{y}^0 \in \mathbb{R}^d$ to the final configuration $\boldsymbol{g}$. This can be accomplished by computing the free parameters $\boldsymbol{w}_i \in \mathbb{R}^d$ so that the desired behavior is achieved.

The DMP representation introduced above has been shown to possess many advantageous properties for robot trajectory generation and control [1] because DMPs can be learnt, modulated, and are robust against perturbations. While DMPs can be used also to compare movements between each other, e. g. in action classification tasks, this can be quite problematic because Eqs. (1) – (3) contain time derivatives. Since time derivatives are dependent both on speed and shape of movement, they cannot separate spatial and temporal aspects of motion. Thus even if the motion changes only in speed, it is not possible to determine that the spatial course of movement remained the same if DMPs are used to represent the movements. In the following we therefore propose a new representation that enables separation of spatial and temporal components of motion.

For this purpose we borrowed an idea from differential geometry of curves [25], where the parametrization with arc-length, i e. the spatial length of a curve, has turned out to be useful. The arc-length of a time-parametrized trajectory $\mathbf{y}(t)$

is defined as [25]

$$s(t) = \int_0^t \|\dot{\mathbf{y}}(u)\| du. \tag{5}$$

Thus the spatial length L of the complete trajectory $y(t)$ on time interval $[0,T]$ can be calculated as follows

$$L = \int_0^T \|\dot{\mathbf{y}}(t)\| dt, \tag{6}$$

where T is the duration. Related to arc-length is the speed of movement, which is given as the time derivative of s

$$\dot{s}(t) = \|\dot{\mathbf{y}}(t)\|. \tag{7}$$

Trajectories with speed greater than zero, i. e. $\dot{s} > 0$, are called regular curves and can be parametrized by arc-length [25]. If this condition is not fulfilled, then the trajectory needs to be segmented into constituent parts with nonzero speed, except possibly at the two end points. The basic idea of our approach is to parametrize the spatial course of movement with arc-length parameter s instead of time. As shown in [25], the arc-length parametrized curve $y(s)$ has unit speed, i. e.

$$\|y'(s)\| = \left\| \frac{dy}{ds} \right\| = 1. \tag{8}$$

Hence $\int_0^s \|y'(s)\| ds = \int_0^s ds = s$. Here and in the following we denote the derivative with respect to arc-length by $'$.

To obtain a speed-independent parametrization of the trajectory, we thus express Eqs. (1) – (3) with respect to arc-length instead of time. We obtain the following dynamical system:

$$L\boldsymbol{z}' = \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{y}) - \boldsymbol{z}) + \mathbf{F}(x), \tag{9}$$

$$L\boldsymbol{y}' = \boldsymbol{z}, \tag{10}$$

$$Lx' = -\alpha_x x, \tag{11}$$

where all derivatives are now taken with respect to arc-length instead of time. The time constant τ, which is used in standard DMPs to speed-up or slow-down the movement during its execution by the robot, was replaced by arc length $L > 0$, which can be calculated by integration (6). We denote this new representation as *arc-length dynamic movement primitive*, or with an acronym *AL-DMP*. As illustrated in Figure 1, an AL-DMP effectively separates the spatial and temporal component of motion. Given the initial condition $x(0) = 1$, Eq. (11) can be solved analytically

$$x(s) = \exp\left(-\alpha_x \frac{s}{L}\right). \tag{12}$$

Thus regardless of the length L of the trajectory, the phase is always defined on the

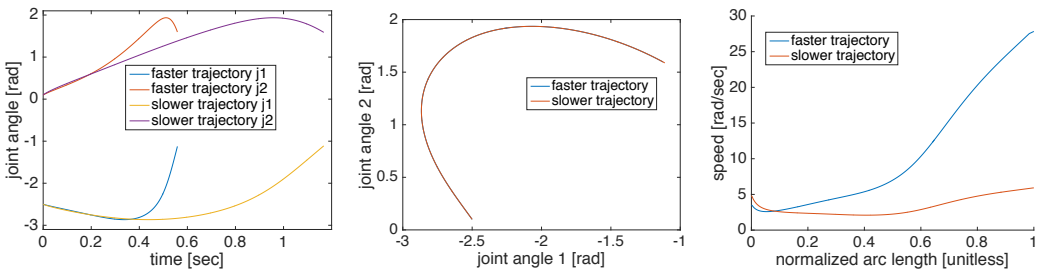

Figure 1: Trajectory representation with AL-DMPs. Left: trajectories of movement of a robot with two degrees of freedom, executed at two different speeds. The result are spatially identical but temporally different movements. The faster movement was generated by nonlinear scaling of the slower movement (its speed was multiplied by a factor of $2(t+1)e^{(t+1)^2-2}$, $t \in [0,T]$, $T = 0.5345$, T is the duration of the faster movement). The consequence of nonlinear scaling is that the spatial parameters $\mathbf{w}$ in the DMP forcing term change. Middle: the reproduction of the spatial course of movement by AL-DMP. Since arc-length derivatives are independent of speed, in AL-DMP the weights $\mathbf{w}$ do not change and consequently the spatial courses of the two movements are identical. Right: Speed $\dot{s}$ of both movements as a function of normalized arc-length, i. e. s/L. As shown in the graph, the speed change is nonlinear and cannot be reproduced by linear scaling of the standard DMP, i. e. changing the parameter τ.

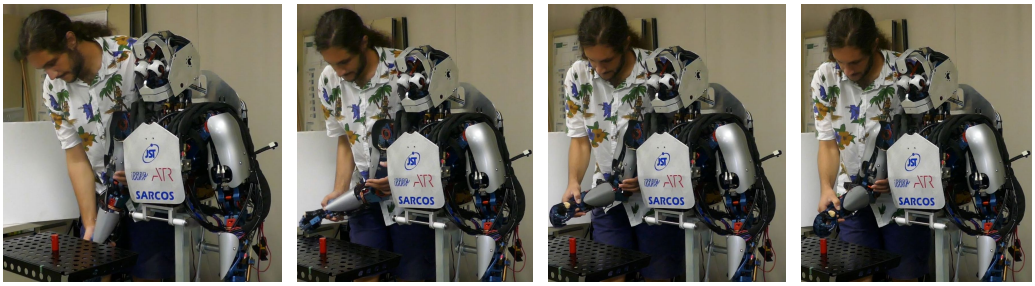

Figure 2: Teaching of humanoid reaching trajectories by kinesthetic guiding from the initial configuration (left) to the final reaching position (right).

same interval: $1 \geq x \geq \exp(-\alpha_x),\ \forall s \in [0, L]$. This is important when comparing different trajectories.

3. Estimation of arc-length dynamic movement primitives

An important feature of dynamic movement primitives is that they can be estimated from a single demonstration of the desired movement. An example demonstration of a reaching movement is shown in Fig. 2. The training data is usually specified as follows

$$\mathscr{G} = \{\boldsymbol{y}_k, t_k\}_{k=1}^{K}, \boldsymbol{y}_k \in \mathbb{R}^d, \tag{13}$$

where d is the number of the robot's degrees of freedom, $t_1 = 0$, $t_K = T$.

To estimate the free parameters of an AL-DMP, i. e. weights $\boldsymbol{w}_i$ associated with radial basis functions in (4), we first rewrite the equation system (9) – (10) as a single second-order system

$$\mathbf{F}(x) = L^2 \boldsymbol{y}'' - \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{y}) - L\boldsymbol{y}'). \tag{14}$$

For every measurement time $t_k,\ k = 1, \ldots, K$, we obtain the following equation linear in $\boldsymbol{w}_i$

$$\sum_{i=1}^{N} \frac{\Psi_i(x_k)}{\sum_{j=1}^{N} \Psi_j(x_k)} \boldsymbol{w}_i = \frac{1}{x_k} \mathrm{diag}(\boldsymbol{g} - \boldsymbol{y}^0)^{-1} \left(L^2 \boldsymbol{y}_k'' - \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{y}_k) - L\boldsymbol{y}_k') \right), \tag{15}$$

The weights $\boldsymbol{w}_i$ can be computed by solving these linear equations in a least squares sense. But for this purpose we first need to compute the derivatives $\boldsymbol{y}'_k = \boldsymbol{y}'(s_k)$, $\boldsymbol{y}''_k = \boldsymbol{y}''(s_k)$.

This is accomplished by sampling the trajectory along arc-length s. Assuming that the time step is constant, i. e. $\Delta t = t_{k+1} - t_k$, $\forall k$, we can estimate the arc-length s_k at every measurement time t_k

$$s_k = \int_0^{t_k} \|\dot{\mathbf{y}}(t)\| \mathrm{d}t \approx \mathrm{Trapzd}(k), \tag{16}$$

where $\mathrm{Trapzd}(k)$ denotes the trapezoidal rule for numerical quadrature [26]

$$\mathrm{Trapzd}(k) = \begin{cases} \Delta t \left(\frac{1}{2} \|\dot{\boldsymbol{y}}_1\| + \sum_{n=2}^{k-1} \|\dot{\boldsymbol{y}}_n\| + \frac{1}{2} \|\dot{\boldsymbol{y}}_k\| \right), \ k \geq 2 \\ 0, \ k = 1 \end{cases} \tag{17}$$

The assumption of constant time steps can be relaxed by applying a different numerical quadrature formula. See [26] for the alternatives. The length of the movement is given by the integral of speed along the trajectory. It can thus be approximated by

$$L = \int_0^T \|\dot{\mathbf{y}}(t)\| \mathrm{d}t \approx \mathrm{Trapzd}(K). \tag{18}$$

Eq. (17) requires the availability of time derivatives $\|\dot{\boldsymbol{y}}_n\|$. They can be estimated using a standard numerical differentiation formula, e. g.

$$\dot{s}_n = \|\dot{\boldsymbol{y}}_n\| = \frac{\|\boldsymbol{y}_{n+1} - \boldsymbol{y}_n\|}{\Delta t}. \tag{19}$$

Given s_k, $k = 1, \ldots, K$, and input data $\mathscr{G}$, we can finally calculate $\boldsymbol{y}'_k$ and $\boldsymbol{y}''_k$ by numerical differentiation

$$\boldsymbol{y}'_k = \frac{\boldsymbol{y}_{k+i_k} - \boldsymbol{y}_k}{s_{k+i_k} - s_k}, \ \boldsymbol{y}''_k = \frac{\boldsymbol{y}'_{k+i_k} - \boldsymbol{y}'_k}{s_{k+i_k} - s_k}. \tag{20}$$

In practice we need to make certain that the arc-length step $\Delta s_k = s_{k+i_k} - s_k$ is large enough to ensure numerically stable calculation of derivatives (20) because unlike the time step, the arc-length step is not uniform in the sampled trajectory. This can be done by choosing i_k to be the smallest index $|i_k|$ (positive or negative) so that $|s_{k+i_k} - s_k| \geq \delta$, $1 \leq k + i_k \leq K$, where $0 < \delta < L$ is a constant specifying the desired arc-length step. Otherwise the calculation of numerical derivatives can become unstable at locations with low speed. This can happen for example with point-to-point movements, which have zero speed at the beginning and the end of motion. Note, however, that except at the end points we assume that $\dot{s} > 0$, otherwise the trajectory needs to be segmented into constituent parts that from regular curves.

Now finally all the data to compute the parameters of equation system (15) has been made available. For every degree of freedom l, $1 \leq l \leq d$, we solve the following linear system of equations in a least squares sense

$$\boldsymbol{A}\boldsymbol{b}_l = \boldsymbol{f}_l,\ l = 1, \ldots, d, \tag{21}$$

with

$$\boldsymbol{b}_l = \begin{bmatrix} w_{1,l} \\ \vdots \\ w_{N,l} \end{bmatrix}, \quad \boldsymbol{f}_l = \begin{bmatrix} f_{1,l} \\ \vdots \\ f_{K,l} \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} \dfrac{\psi_1(x_1)}{\sum_{j=1}^N \psi_j(x_1)} & \cdots & \dfrac{\psi_N(x_1)}{\sum_{j=1}^N \psi_j(x_1)} \\ \vdots & \vdots & \vdots \\ \dfrac{\psi_1(x_K)}{\sum_{j=1}^N \psi_j(x_K)} & \cdots & \dfrac{\psi_N(x_K)}{\sum_{j=1}^N \psi_j(x_K)} \end{bmatrix}, \tag{22}$$

and

$$f_{k,l} = \frac{1}{(g_l - y_l^0)x_k} \left(L^2 y''_{k,l} - \alpha_z(\beta_z(g_l - y_{k,l}) - L y'_{k,l}) \right), \tag{23}$$

$$x_k = \exp\left(-\alpha_x \frac{s_k}{L} \right). \tag{24}$$

Here $y_l, g_l, y_l^0 \in \mathbb{R}$ denote the components of $\boldsymbol{y}, \boldsymbol{g}, \boldsymbol{y}^0 \in \mathbb{R}^d$ and $\boldsymbol{w}_i = \left[w_i^1, \ldots, w_i^d\right]^T$. For a given number N of basis functions Ψ_i in forcing term (4), we define their parameters using the following formulae:

$$c_i = \exp\left(-\alpha_x \frac{i-1}{N-1}\right),\ i = 1, \ldots, N, \tag{25}$$

$$h_i = \begin{cases} \dfrac{2}{(c_{i+1} - c_i)^2},\ i = 1, \ldots, N-1, \\ h_i = h_{i-1},\ i = N. \end{cases} \tag{26}$$

An AL-DMP does not contain information about the speed of motion, which is also needed to reproduce the demonstrated trajectory. To provide the information about speed, or equivalently, the time derivative of arc-length, we approximate $\dot{s}$ with a linear combination of radial basis functions defined as a function of phase x

$$\dot{s}(x) = 1 + \frac{\sum_{i=1}^{M} v_i \Psi_i(x)}{\sum_{i=1}^{M} \Psi_i(x)} x. \tag{27}$$

The number of radial basis functions M for the estimation of speed is not necessarily the same as the number of basis function N in the forcing term (4). The centers c_i and widths h_i in basis functions Ψ_i are computed as in (25), (26), with M replacing N. The time derivatives of s at times t_k are estimated by numerical differentiation (19). For each measurement time t_k, we obtain the following equations linear in v_i

$$\frac{\dot{s}_k - 1}{x_k} = \frac{\sum_{i=1}^{M} v_i \Psi_i(x_k)}{\sum_{i=1}^{M} \Psi_i(x_k)},\ k = 1 \ldots, K. \tag{28}$$

Just like (15), (28) is a standard overdetermined system of linear equations that can be solved in a least squares sense. Note that beyond the demonstrated trajectory, $\dot{s}$ as defined by Eq. (27) converges to 1 as the phase x tends to zero. This property ensures the convergence of AL-DMP to its desired final configuration $\boldsymbol{g}$.

4. Trajectory reproduction and robot control with AL-DMPs

While AL-DMPs have many favourable properties for learning and recognition, they cannot be used directly for control because a robot is controlled at constant time steps, not constant arc-length steps. To construct AL-DMP integration at constant time steps, we first need to derive some relationships between time and arc-length derivatives

$$\dot{\boldsymbol{y}} = \frac{d}{dt}\boldsymbol{y}(s(t)) = \boldsymbol{y}'\dot{s}, \tag{29}$$

$$\ddot{\boldsymbol{y}} = \frac{d^2}{dt^2}\boldsymbol{y}(s(t)) = \boldsymbol{y}''\dot{s}^2 + \boldsymbol{y}'\ddot{s}. \tag{30}$$

$$\dot{x} = \frac{d}{dt}x(s(t)) = x'\dot{s}. \tag{31}$$

We can now express arc-length derivatives in terms of time derivatives

$$\boldsymbol{y}' = \frac{1}{\dot{s}}\dot{\boldsymbol{y}}, \tag{32}$$

$$\boldsymbol{y}'' = \frac{1}{\dot{s}^3}\left(\ddot{\boldsymbol{y}}\dot{s} - \dot{\boldsymbol{y}}\ddot{s}\right). \tag{33}$$

From (32) – (33) and (9) - (10) we obtain

$$\boldsymbol{z} = L\boldsymbol{y}' = \frac{L}{\dot{s}}\dot{\boldsymbol{y}} \tag{34}$$

$$\dot{\boldsymbol{z}} = L\frac{\ddot{\boldsymbol{y}}\dot{s} - \dot{\boldsymbol{y}}\ddot{s}}{\dot{s}^2}, \tag{35}$$

$$\boldsymbol{z}' = L\boldsymbol{y}'' = \frac{1}{\dot{s}}L\frac{\ddot{\boldsymbol{y}}\dot{s} - \dot{\boldsymbol{y}}\ddot{s}}{\dot{s}^2} = \frac{1}{\dot{s}}\dot{\boldsymbol{z}}. \tag{36}$$

Using the above equations it is possible to integrate AL-DMP with a constant time step as required by standard robot controllers. The details of the integration process are explained in Algorithm 1. Note that if in this algorithm we set the speed to $\dot{s} = 1$, which can easily be done by setting $v_i = 0$ in (27), then $\dot{\boldsymbol{y}}_i = \boldsymbol{y}'_i$, $\dot{\boldsymbol{z}}_i = \boldsymbol{z}'_i$, $\dot{x}_i = x'_i$ and $\Delta t = \Delta s$.

Often it is not necessary to continue integrating AL-DMP beyond the training interval, i. e. for $x < \exp(-\alpha_x)$. If no movement modulations have been applied and no perturbations have occurred, then the trajectory will be completely reproduced by integration on the interval $1 \geq x \geq \exp(-\alpha_x)$. A point-to-point movement should reach the desired final configuration $\boldsymbol{y} = \boldsymbol{g}, \dot{\boldsymbol{y}} = 0$ at $x = \exp(-\alpha_x)$, provided the phase follows Eq. (11), or equivalently, (12).

An important point to make is that regardless of what approximation we take

---

Algorithm 1: Euler integration of AL-DMP with a constant time step

---

Input: Current AL-DMP state $\boldsymbol{y}_i, \boldsymbol{z}_i, x_i$

AL-DMP constants $L, \boldsymbol{g}, \alpha_x, \alpha_z, \beta_z$, robot servo rate Δt

parameters $N,\ c_i,\ h_i,\ \boldsymbol{w}_i$ of forcing term (4)

parameters M, c_i, h_i, v_i of speed estimate (27)

Output: Next AL-DMP state $\boldsymbol{y}_{i+1}, \boldsymbol{z}_{i+1}, x_{i+1}$, robot velocity $\dot{\boldsymbol{y}}_i$

begin

1 Compute arc-length derivatives $\boldsymbol{y}'_i, \boldsymbol{z}'_i, x'_i$ at current state $\boldsymbol{y}_i, \boldsymbol{z}_i, x_i$ using equations (9) - (11)

2 Compute speed $\dot{s}_i$ at current phase x_i using approximation (27)

3 Compute time derivatives $\dot{\boldsymbol{y}}_i, \dot{\boldsymbol{z}}_i, \dot{x}_i$ by respectively applying equations (29), (36) and (31)

4 Euler integration:

$$\boldsymbol{y}_{i+1} = \boldsymbol{y}_i + \dot{\boldsymbol{y}}_i \Delta t$$

$$\boldsymbol{z}_{i+1} = \boldsymbol{z}_i + \dot{\boldsymbol{z}}_i \Delta t$$

$$x_{i+1} = x_i + \dot{x}_i \Delta t$$

end

---

for $\dot{s}$, the spatial course of trajectory generated by AL-DMP will not change. This becomes clear if we rewrite Eq. (9) and (10) by respectively replacing $\boldsymbol{y}'$ and $\boldsymbol{z}'$ with (32) and (36). This results in AL-DMP expressed with temporal derivatives

$$L\dot{\boldsymbol{z}} = \dot{s}\left(\alpha_z(\beta_z(\boldsymbol{g}-\boldsymbol{y})-\boldsymbol{z})+\mathbf{F}(x)\right), \tag{37}$$

$$L\dot{\boldsymbol{y}} = \dot{s}\boldsymbol{z}. \tag{38}$$

Phase equation (11) can also be rewritten in a time-dependent form

$$L\dot{x} = -\dot{s}\alpha_x x. \tag{39}$$

Since all equations are multiplied by the same scaling factor $\dot{s}$, the spatial course of movement does not change regardless of the definition of $\dot{s}$. This form of writing is related to speed-scaled dynamic movement primitives introduced in [23, 24]. However, if we want to reproduce the demonstrated trajectory both spatially and temporally, then $\dot{s}$ must be equal to the demonstrated speed of motion.

5. Statistical skill learning with arc-length dynamic movement primitives

Statistical learning methods can be applied for generalization of trajectories stemming from multiple demonstrations of variants of the same skill. For example, if the robot is to reach from the same initial configuration to different positions distributed on the surface of a table, the training data could consist of reaching trajectories in robot joint space, supplemented with information about 2-D final reaching positions in the Cartesian space. We obtain the following training data

$$\mathscr{G} = \{\boldsymbol{y}_{n,k}, t_{n,k};\ \boldsymbol{q}_n\}_{k=1,n=1}^{K_n,NumEx}, \tag{40}$$

where $\boldsymbol{y}_{n,k}$ are the measurements captured at time $t_{n,k}$, $\boldsymbol{q}_n \in \mathbb{R}^m$ are the query points describing the aim of the task, K_n is the number of measurements on the

n-th example trajectory, $NumEx$ is the number of example trajectories, and m is the dimension of the query point. In the reaching example described above, the measurements $\boldsymbol{y}_{n,k} \in \mathbb{R}^d$ would be the robot joint space configurations and $\boldsymbol{q}_n \in \mathbb{R}^2$ the desired reaching positions on the table. Here d is the number of the robot's degrees of freedom.

We are interested in finding the optimal movement to accomplish the task for any query point $\boldsymbol{q} \in \mathbb{R}^m$. For the above example this means that we want to compute a reaching trajectory for any reaching position on the table. We represent such a trajectory with an AL-DMP. Thus we need to estimate the following function $\mathbf{G}$ which, given the training data $\mathscr{G}$, maps the desired query point $\boldsymbol{q}$ into the parameters that define an AL-DMP:

$$\mathbf{G}_{\mathscr{G}} : \boldsymbol{q} \mapsto \left[L, \boldsymbol{g}^T, \boldsymbol{w}^T, \boldsymbol{v}^T\right]^T, \tag{41}$$

where $\boldsymbol{w} = [\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_N^T]^T$ and $\boldsymbol{v} = [v_1, \ldots, v_M]^T$.

Locally weighted regression (LWR) [27] and related approaches, e. g. [28], were created as nonparametric method that can approximate a wide range of functions and are widely used in the context of programming by demonstration. For example, in [3] LWR was applied to synthesize new trajectories based on a library of demonstrated example trajectories. Applying this approach to AL-DMPs, the weights $\boldsymbol{w}$ of the forcing term can be computed by solving the following optimization problem for every dimension of $\boldsymbol{y}$

$$\min_{\boldsymbol{b}_l} \sum_{n=1}^{NumEx} \|\boldsymbol{A}\boldsymbol{b}_l - \boldsymbol{f}_{n,l}\|^2 \mathrm{K}(\mathrm{d}(\boldsymbol{q}, \boldsymbol{q}_n)),\ l = 1, \ldots, d, \tag{42}$$

where $\boldsymbol{b}_l$ contains the parameters of the forcing term for l-th dimension. $\boldsymbol{A}$ and $\boldsymbol{f}_{n,l}$ are defined as in (22), with $\boldsymbol{f}_{n,l}$ changing across the training trajectories. K

is the weighting kernel that should put more emphasis on the data associated with queries $\boldsymbol{q}_n$ closer to the current query $\boldsymbol{q}$. d is the metrics measuring the distance between the queries. Euclidean distance was used in the examples in this paper, but other choices are possible. We selected the tricube kernel K [27] for weighting

$$\mathrm{K}(d) = \begin{cases} (1-|d/h|^3)^3 & \text{if } |d/h| < 1 \\ 0 & \text{otherwise} \end{cases}, \tag{43}$$

where h is the scaling factor that influences the range on which $\mathrm{K}(d) > 0$. The tricube kernel has finite support and continuous first and second derivatives. Thus, the first two derivatives of the generalization function $\mathbf{G}_{\mathscr{G}}$ are also continuous. The computational complexity of the optimization problem (42) is reduced through this choice of K because K vanishes for query points $\boldsymbol{q}_n$ that are far from $\boldsymbol{q}$ and therefore do not influence $\mathbf{G}_{\mathscr{G}}$. This makes the system matrix associated with objective function (42) banded.

To estimate the length L and final configuration $\boldsymbol{g}$ on the trajectory, we extract the data directly from the training set

$$L_n = \int_{t_{n,1}}^{t_{n,K_n}} \|\dot{\boldsymbol{y}}_n(t)\| dt \approx \mathrm{Trapzd}(K_n),\ \boldsymbol{g}_n = \boldsymbol{y}_{n,K_n},\ n = 1,\ldots,NumEx. \tag{44}$$

Given a new query point $\boldsymbol{q}$ and using LWR, the length and final configuration can be generalized as follows

$$L = \sum_{n=1}^{NumEx} \frac{\mathrm{K}(\boldsymbol{q},\boldsymbol{q}_n)L_n}{\sum_{j=1}^{NumEx}\mathrm{K}(\boldsymbol{q},\boldsymbol{q}_j)},\ \boldsymbol{g} = \sum_{n=1}^{NumEx} \frac{\mathrm{K}(\boldsymbol{q},\boldsymbol{q}_n)\boldsymbol{g}_n}{\sum_{j=1}^{NumEx}\mathrm{K}(\boldsymbol{q},\boldsymbol{q}_j)}. \tag{45}$$

The weights of the generalized speed $\boldsymbol{v}$ are estimated by solving a least squares problem similar to (42), which is used to estimate $\boldsymbol{w}$. We omit the details here.

5.1. Evaluation of skill learning with AL-DMPs in simulation

First we evaluated the performance of statistical skill learning on simulated data. For this purpose we synthetically generated a set of 100 reaching planar

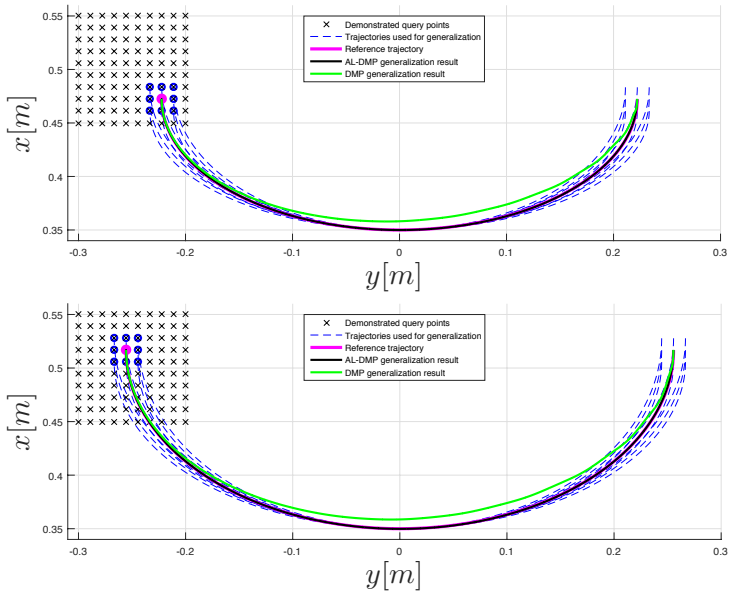

Figure 3: Two example generalizations in 2-D plane using AL-DMPs and standard DMPs. The original trajectories that were used for generalization are shown in blue and dashed. They all pass through a unique via point at $y = 0$. The trajectories generalized using AL-DMPs (black) also pass through this via point. The trajectories generalized using DMPs (green) do not approximate the original trajectories well and do not pass through the via point.

trajectories that all passed through a via point located at half way, i. e. at $s = L/2$. Using inverse kinematics of a humanoid robot (see Section 5.2), these planar trajectories were converted into the upper arm shoulder and elbow movement of the robot (four degrees of freedom). The training trajectories were then temporally scaled with a nonlinear scaling factor, which caused the training trajectories to reach the via point at different times, whereas their spatial course of movement was unaffected. The 2-D position of the final point on the trajectory was used as a query for LWR. The trajectories were generalized in the robot's joint space, but Fig. 3 shows their course in 2-D plane after the direct kinematics was ap-

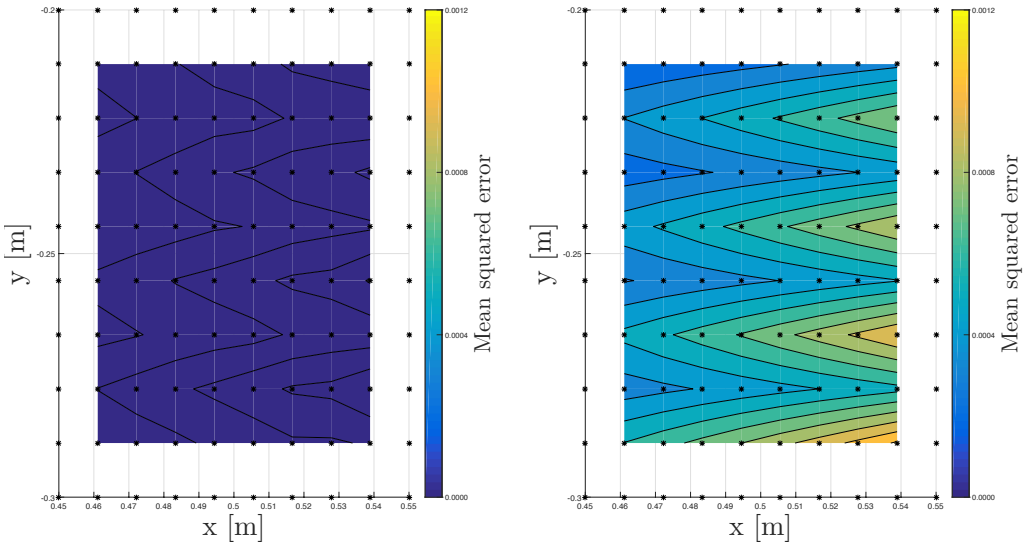

Figure 4: Contour plots showing the accuracy of generalization with AL-DMPs (left) and DMPs (right) in simulation. Generalization with AL-DMPs is clearly better.

plied. Only the trajectories that were generalized using AL-DMPs pass through the via point, whereas the trajectories generalized using standard DMPs do not approximate the original trajectories well enough to pass through the via point.

We used mean squared error criterion to evaluate the difference between the generalized trajectory represented by AL-DMP $\boldsymbol{y}_{AL}$ and the true trajectory $\boldsymbol{y}$

$$\text{MSE} = \frac{1}{K}\sum_{k=1}^{K} \|\boldsymbol{y}_{AL}(s_k) - \boldsymbol{y}(s_k)\|^2, \tag{46}$$

where K is the number of integration steps needed to integrate the AL-DMP to reach the desired final position. MSE was estimated both for the generalized AL-DMPs and generalized standard DMPs. Fig. 4 shows that generalization results with AL-DMPs are significantly better than when standard DMPs are used. This simulation experiment thus confirmed our intuition that generalization using AL-DPMs should produce better results.

5.2. Evaluation of skill learning with AL-DMPs in real experiments

A skill learning experiment with real data was performed on a humanoid robot CB-i [29]. Altogether we collected 99 reaching movements from the same initial arm configuration, where the arm was extended below the table, to 99 final reaching positions on the table. The human demonstrator intentionally performed some demonstrations slower than the others. Snapshots from one of the training movements are shown in Fig. 2. Four different final reaching positions are shown in Fig. 5. 2-D position of the red peg on the table was used as query for generalization.

For analysis we performed an experiment in which at each internal query point $\boldsymbol{q}_n$, the training trajectory $\{\boldsymbol{y}_k^n\}_{k=1}^{t_{K_n}}$ was removed from the training set (40). 63 internal trajectories were used for testing, i. e. trajectories that are associated with query points inside the training space (see Fig. 7). The generalization function (41) was then computed for $\boldsymbol{q} = \boldsymbol{q}_n$ as input, with $\{\boldsymbol{y}_k^n\}_{k=1}^{K_n}$ removed from $\mathscr{G}$. Generalization was performed both with standard DMPs and AL-DMPs. The generalized DMPs and AL-DMPs were then compared to the trajectories that were left out from the training set using mean squared error criterion (46). All trajectories were represented as a function of arc-length.

Figure 5: Final reaching positions on four of the 99 training trajectories.

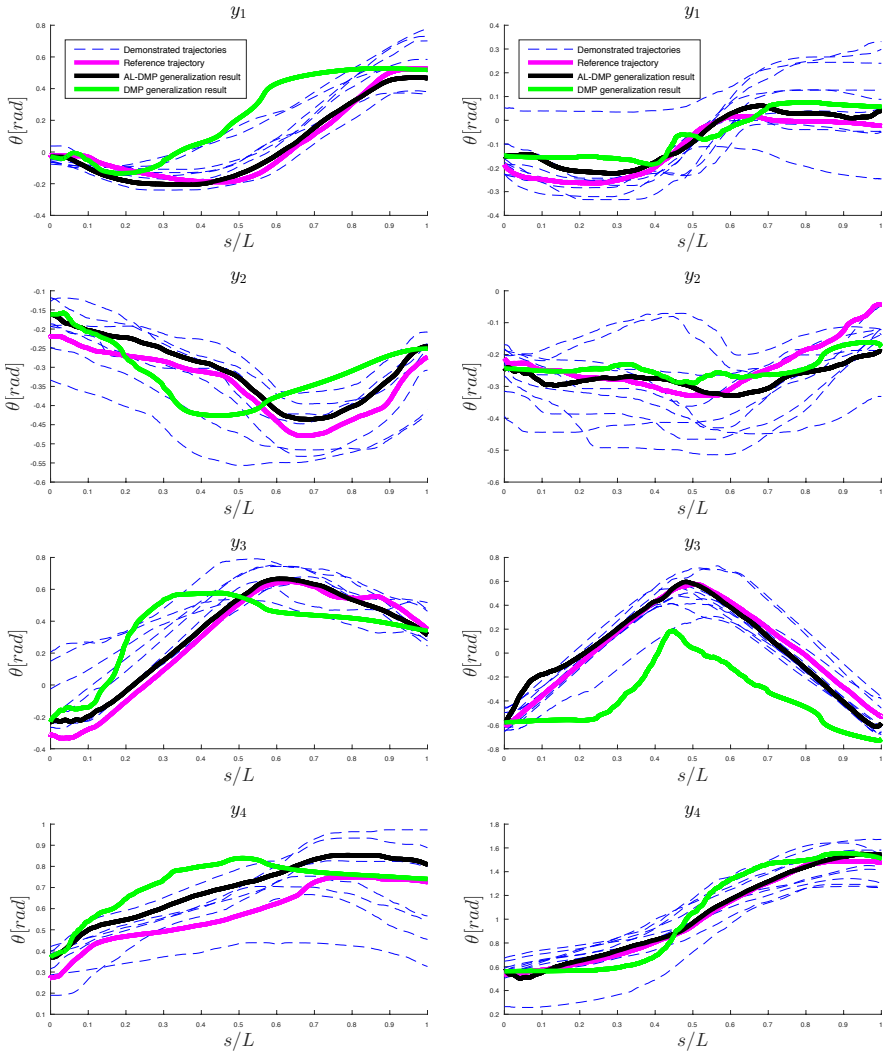

Figure 6: Accuracy of generalization for AL-DMPs (black) and DMPs (green) compared to the demonstrated movements (magenta) for two example reaching movements (left and right column)

Fig. 6 shows the result of generalization using AL-DMPs and standard DMPs. It is clear from these figures that the generalized AL-DMPs approximate the trajectories that were left out much better than the generalized DMPs. The reason for this difference in performance are the variations in speed profiles of the training trajectories. Fig. 7 shows the MSE results as contour plots for all internal trajecto-

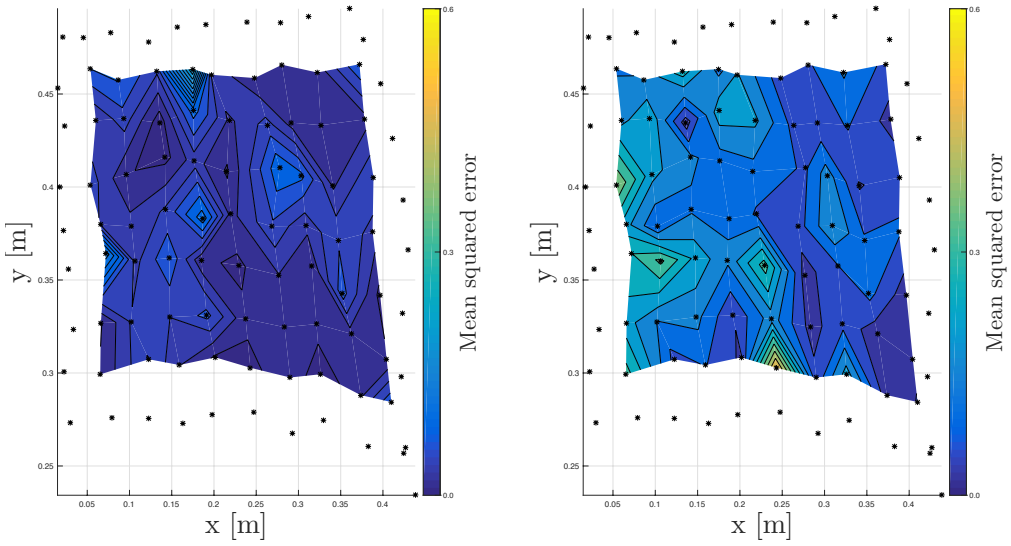

Figure 7: Contour plots showing accuracy of generalization for AL-DMPs (left) and DMPs (right)

ries. These results again demonstrate that the generalized AL-DMPs approximate the trajectories that were left out better than the generalized DMPs.

6. Action recognition with arc-length dynamic movement primitives

Action recognition from motion trajectories is a vast field and many different approaches have been proposed in the literature [30]. In this section we show that if motion trajectories are used for recognition, AL-DMPs outperform more standard robot trajectory representations such as B-splines [31] or DMPs [1]. It should be mentioned here that raw motion trajectories are not necessarily the best input data for every classification problem. Other measures can be extracted form motion capture data [32] to characterize the captured actions. The best representation for a specific classification problem cannot be selected without considering the actual task.

For standard DMPs, Ijspeert et al. [2] showed that the weights of the forcing

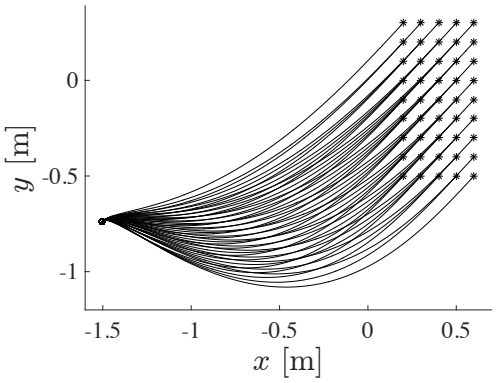

Figure 8: Cartesian paths of the reaching movements that define 5 classes in the data set used for testing classification with simulated data. Reaching distance in the *x* direction defines one class. For each class 161 example trajectories were generated (only 9 of them are shown), with uniformly distributed reaching distances in *y* direction.

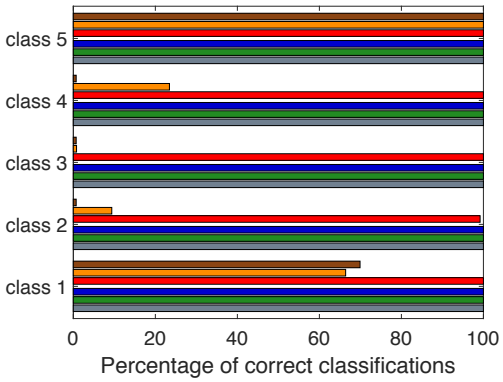

Figure 9: Classification performance for simulated reaching movements. The grey, green and blue bars show the classification performance of AL-DMPs, DMPs, and B-splines using the original data, while the red, orange and brown bars show the classification performance of AL-DMPs, DMPs, and B-splines using temporally scaled data.

term can be used for classification of trajectories with "similar velocity profiles". Their approach was based on correlations between feature vectors consisting of weights. They were able to show that correlation values are higher for trajectories belonging to the same class compared to trajectories from different classes.

As explained in the introduction, the speed of movement can vary significantly across human demonstrations, especially if they are performed by different subjects. In such cases the velocity profiles are not "similar" and DMP weights become a poor feature vector for classification. However since the weights of the forcing term in AL-DMP do not depend on speed, we expect that recognition performance can be significantly improved in such cases by applying the weights of AL-DMPs instead of standard DMPs as feature vectors.

6.1. Simulated data

For testing we created a simulated set of 805 reaching trajectories in Cartesian space for a simulated robot with two degrees of freedom. Some of the training trajectories are shown in Fig. 8. They were assigned to five different classes, where each class consisted of 161 reaching movements to a given distance in x direction, whereas the final destinations in y direction where different. We used inverse kinematics to convert the Cartesian space trajectories into joint space trajectories. Joint space trajectories were then approximated by AL-DMPs, standard DMPs, and B-splines. The computed weights $\boldsymbol{w}_i$ of the forcing terms and the parameters of B-splines were used as feature vectors for classification. Feature vectors had 40 dimensions because the number of basis functions N was set to 20 in all cases and the trajectories were two-dimensional.

Support vector machines (SVM, [33]) were applied for classification. We used multiclass (one-vs-one) SVMs with linear kernels in this experiment. Matlab Classification Learner App was used to train the classifiers. 20% of the whole data set, i. e. 161 trajectories, were used to train the SVM. In this noiseless experiment, all of the remaining 644 examples were classified by the SVM correctly, regardless whether AL-DMPs, standard DMPs, or B-splines were used to compute the feature vectors. These results are shown with green and blue bars in Fig. 9.

Next, we introduced non-linear temporal scaling into the 644 test examples. This was done by multiplying the speed by $2(t+1)e^{(t+1)^2-2}$, $t \in [0,T]$, where T was the duration of motion. This function was selected to make clear the benefits of AL-DMPs compared to DMPs and B-splines. Again DMPs, B-splines and AL-DMPs were used to compute the feature vectors for the temporally scaled example

trajectories and the classification performance was evaluated using SVMs trained with feature vectors stemming from original examples that were not temporally scaled. The results are shown in Figure 9. While the classification rate was perfect with all three types of feature vectors when using the original data for testing, it is clear from the bar graph that the classification rate with DMP based and B-splines based feature vectors was severely affected when using temporally scaled data. Significantly less than 50 % of the inputs were classified correctly. The cause for this behavior is obvious; the values of the weights of the DMP forcing term and B-spline parameters were significantly affected by temporal scaling. Therefore, the resulting feature vectors were not representative of the classes trained with the original data. On the other hand, the performance of classification with AL-DMP based feature vectors was virtually unaffected. The reason for this is that variations in the speed profiles of the trajectories do not change the weights of the forcing term in AL-DMP. Hence the speed-scaled trajectories were still classified correctly. This experiment confirms our expectation that AL-DMP based classification is much more robust compared to standard DMP features when the speed of motion varies nonlinearly.

6.2. Real data

In this final experiment we evaluated the classification performance of DMPs and AL-DMPs on a data set consisting of 5 motion classes, in total containing 225 hand trajectories performed by 4 different subjects. Of the 5 classes, 2 deal with manipulation (placing a book on a shelf, placing an object into a drawer) and 3 encompass sports actions (tennis swing, 2 versions of a ball throw). The trajectories were recorded using motion capture system Optotrak.

The training trajectories were encoded with standard DMPs and AL-DMPs.

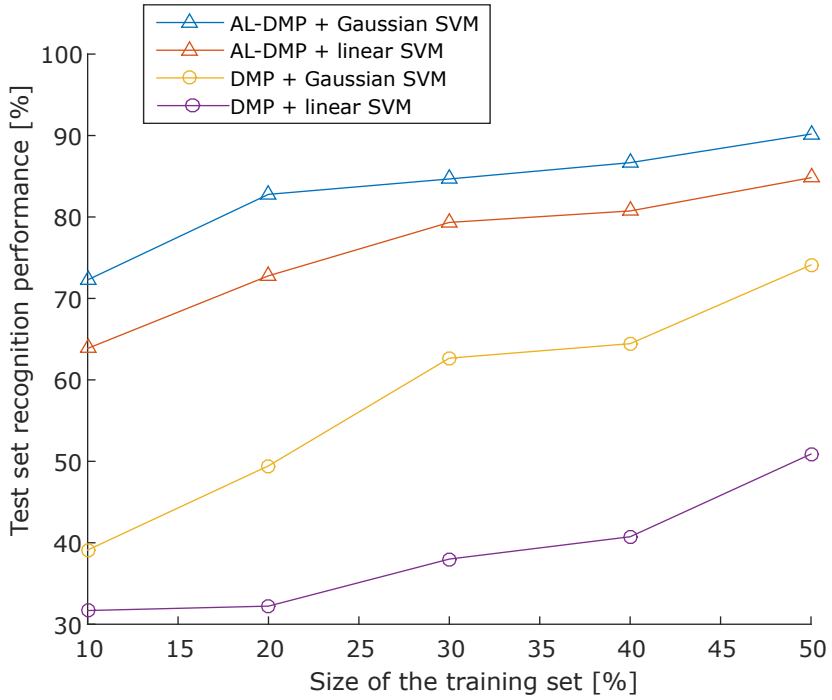

Figure 10: Classification performance as a function of the training set size. Blue and red graphs show the percentage of correctly classified examples using the Gaussian and linear SVM, respectively, with AL-DMP weights used as features. Yellow and magenta show classification performance when standard DMP weights were used as features.

The resulting weights $\boldsymbol{w}_i$ of the forcing terms were used as feature vectors for classification. Like in simulation experiments, the number of basis functions N was set to 20. The trajectories of 3-D wrist positions were used to build the training data set, thus the feature vector dimension was 60.

The classification performance was evaluated using SVMs with linear and Gaussian kernels. Compared to linear SVMs, Gaussian kernel SVMs offer better support for non-linear class boundaries. For this reason, Gaussian kernel SVMs are better suited to classify complex input data. The classification performance was tested using different training and test set ratios; we tested the cases where 10,

20, 30, 40 and 50 percent of the example trajectories in each class were randomly chosen to form the training set, while the corresponding test sets was composed of the remaining example trajectories.

Figure 10 shows that SVMs with AL-DMP based feature vectors significantly outperform the SVMs with standard DMP based feature vectors. This was true regardless of the applied kernel type. Our simulation results were thus confirmed in an experiment with real data. This graph also shows that the nonlinear Gaussian-kernel SVMs improve the results obtained with conventional DMP based feature vectors more than when AL-DMP based features are used. This is due to the fact that the weights of the forcing term in AL-DMP better reflect the changes in the shape of movement. Consequently, the demonstrated movements can be separated even with simpler linear boundaries.

7. Discussion

In this paper we developed the concept of arc-length dynamic movement primitives, where spatial and temporal aspects of motion are well separated. AL-DMPs achieve this by describing motion with 1) a system of nonlinear differential equations that are solely dependent on the spatial course of movement (through arc length derivatives) and 2) a feature vector that encodes the speed of movement, which is dependent only on temporal aspects of motion. We confirmed these properties in a number of skill learning and action recognition experiments, both in simulation and in real word. Just like standard DMPs, AL-DMPs represent kinematic aspects of motion and are not concerned with motion dynamics.

While we were able to effectively use AL-DMPs for skill learning and action recognition, their application for robot control can sometimes be problematic.

Unlike standard DMP equations (1) – (2), which have a unique attractor point at $\boldsymbol{y} = \boldsymbol{g},\ \dot{\boldsymbol{y}} = 0$, an AL-DMP does not converge to such a point within the training interval because the arc-length derivative $\boldsymbol{y}'$ is equal to the unit tangent of the curve $\boldsymbol{y}$ [25], i. e. $\|\boldsymbol{y}'\| = 1$ everywhere on the training interval. Thus $\boldsymbol{y}' \neq 0$ at the end of movement and an AL-DMP will deviate from the attractor point outside of the training interval if numerical integration of Eq. (9) – (10) continues. Consequently, the robot moves beyond the desired final configuration and slowly returns unless the speed is set to zero at the end of movement. But if the speed is set to zero, an AL-DMP cannot ensure convergence to the desired goal $\boldsymbol{g}$ if the goal is not reached at the end of movement. Thus AL-DMPs are not the best choice to control the robot, especially if the desired motion is modulated or in a feedback loop because in such cases the robot does not reach its final destination $\boldsymbol{g}$ at the end of the training interval, i. e. at $x = \exp(-\alpha_x)$. Thus for control it is better to convert an AL-DMP to a standard DMP representation, which does not suffer from this issue. The conversion is trivial as one can synthetically reproduce the trajectory generated by an AL-DMP using Algorithm 1 at a high sampling rate and sample the resulting robot configurations along the trajectory. Just like when computing DMPs from real motion data, DMPs can also be computed from the synthetically sampled data. The resulting DMP can then be used to control the robot.

Acknowledgement

This work has received funding from the EU's Horizon 2020 IA ReconCell (GA no. 680431); GOSTOP programme C3330-16-529000 co-financed by Slovenia and EU under ERDF; JSPS KAKENHI JP16H06565; New Energy and Indus-

trial Technology Development Organization (NEDO); SRPBS from AME; Im-PACT Program of Coun- cil for Science, Technology, and Innovation (Cabinet Office, Govern- ment of Japan); and the Commissioned Research of NICT.

References

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

[2] A. J. Ijspeert, J. Nakanishi, T. Shibata, S. Schaal, Nonlinear dynamical systems for imitation with humanoid robots, in: IEEE-RAS International Conference on Humanoid Robots (Humanoids), Tokyo, Japan, 2001, pp. 219–226.

[3] 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.

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

[5] A. Kramberger, A. Gams, B. Nemec, D. Chrysostomou, O. Madsen, A. Ude, Generalization of orientation trajectories and force-torque profiles for robotic assembly, Robotics and Autonomous Systems 98 (2017) 333–346.

[6] J. Kober, A. Wilhelm, E. Oztop, J. Peters, Reinforcement learning to adjust parametrized motor primitives to new situations, Autonomous Robots 33 (4) (2012) 361–379.

[7] F. Stulp, E. A. Theodorou, S. Schaal, Reinforcement Learning With Sequences of Motion Primitives for Robust Manipulation, IEEE Transactions on Robotics 28 (6) (2012) 1360–1370.

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

[9] T. Kulvicius, M. Biehl, M. J. Aein, M. Tamosiunaite, F. Wörgötter, Interaction learning for dynamic movement primitives used in cooperative robotic tasks, Robotics and Autonomous Systems 61 (12) (2013) 1450–1459.

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

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

[12] M. Prada, A. Remazeilles, A. Koene, S. Endo, Dynamic Movement Primitives for Human-Robot interaction: Comparison with human behavioral observation, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Tokyo, Japan, 2013, pp. 1168–1175.

[13] A. D. Dragan, K. Muelling, J. A. Bagnell, S. S. Srinivasa, Movement primitives via optimization, in: IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, 2015, pp. 2339–2346.

[14] D. Forte, A. Gams, J. Morimoto, A. Ude, On-line motion synthesis and adaptation using a trajectory database, Robotics and Autonomous Systems 60 (10) (2012) 1327–1339.

[15] S. M. Khansari-Zadeh, A. Billard, Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions, Robotics and Autonomous Systems 62 (6) (2014) 752–765.

[16] S. Calinon, P. Kormushev, D. G. Caldwell, Compliant skills acquisition and multi-optima policy search with EM-based reinforcement learning, Robotics and Autonomous Systems 61 (4) (2013) 369–379.

[17] L. Rozo, S. Calinon, D. G. Caldwell, P. Jiménez, C. Torras, Learning Physical Collaborative Robot Behaviors From Human Demonstrations, IEEE Transactions on Robotics 32 (3) (2016) 513–527.

[18] A. Paraschos, C. Daniel, J. Peters, G. Neumann, Probabilistic movement primitives, in: Advances in Neural Information Processing Systems 26 (NIPS), Lake Tahoe, Nevada, 2013.

[19] G. J. Maeda, G. Neumann, M. Ewerton, R. Lioutikov, O. Kroemer, J. Peters, Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks, Autonomous Robots 41 (3) (2017) 593–612.

[20] H. Ben Amor, G. Neumann, S. Kamthe, O. Kroemer, J. Peters, Interaction

primitives for human-robot cooperation tasks, in: IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 2014, pp. 2831–2837.

[21] H. Sakoe, S. Chiba, Dynamic programming algorithm optimization for spoken word recognition, IEEE Transactions on Acoustics, Speech and Signal Processing 26 (1) (1978) 43–49.

[22] M. Ewerton, G. Maeda, J. Peters, G. Neumann, Learning motor skills from partially observed movements executed at different speeds, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015, pp. 456–463.

[23] B. Nemec, A. Gams, A. Ude, Velocity adaptation for self-improvement of skills learned from user demonstrations, in: IEEE-RAS International Conference on Humanoid Robots (Humanoids), Atlanta, GA, 2013, pp. 423–428.

[24] R. Vuga, B. Nemec, A. Ude, Speed adaptation for self-improvement of skills learned from user demonstrations, Robotica 34 (2016) 2806–2822.

[25] A. Gray, Modern Differential Geometry of Curves and Surfaces with Mathematica, CRC Press, Boca Raton, FL, 1999.

[26] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes; The Art of Scientific Computing, Cambridge University Press, 2007.

[27] C. G. Atkeson, A. W. Moore, S. Schaal, Locally Weighted Learning, Artificial Intelligence Review 11 (1997) 11–73.

[28] F. Meier, P. Hennig, S. Schaal, Incremental local Gaussian regression, in: Advances in Neural Information Processing Systems 27 (NIPS), Montreal, Canada, 2014, pp. 972–980.

[29] G. Cheng, S.-H. Hyon, J. Morimoto, A. Ude, J. G. Hale, G. Colvin, W. Scroggin, S. C. Jacobsen, CB: a humanoid research platform for exploring neuroscience, Advanced Robotics 21 (10) (2007) 1097–1114.

[30] J. K. Aggarwal, L. Xia, Human activity recognition from 3D data: A review, Pattern Recognition Letters 48 (2014) 70–80.

[31] S. E. Thompson, R. V. Patel, Formulation of joint trajectories for industrial robots using B-splines, IEEE Transactions on Industrial Electronics IE-34 (2) (1987) 192–199.

[32] I. H. Suh, S. H. Lee, N. J. Cho, W. Y. Kwon, Measuring motion significance and motion complexity, Information Sciences 388-389 (2017) 84–98.

[33] C. J. C. Burges, A tutorial on support vector machines for pattern recognition, Data Mining and Knowledge Discovery 2 (2) (1998) 121–167.