Robotica: (2021) volume 39, pp. 1299–1315.
doi:10.1017/S0263574720001186

Motion Adaptation Based on Learning the Manifold of Task and Dynamic Movement Primitive Parameters

Yosef Cohen†, Or Bar-Shira† and Sigal Berman†,‡*

†*Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel*
‡*Zlotowski Center for Neuroscience, Ben-Gurion University of the Negev, Beer-Sheva, Israel*

(Accepted November 21, 2020. First published online: December 18, 2020)

SUMMARY
Dynamic movement primitives (DMP) are motion building blocks suitable for real-world tasks. We suggest a methodology for learning the manifold of task and DMP parameters, which facilitates runtime adaptation to changes in task requirements while ensuring predictable and robust performance. For efficient learning, the parameter space is analyzed using principal component analysis and locally linear embedding. Two manifold learning methods: kernel estimation and deep neural networks, are investigated for a ball throwing task in simulation and in a physical environment. Low runtime estimation errors are obtained for both learning methods, with an advantage to kernel estimation when data sets are small.

KEYWORDS: Dynamic movement primitives, Kernel estimation, Deep Neural networks, Motion planning, Learning.

1. Introduction

Using a finite set of motor primitives has been suggested as the method employed by vertebrates for producing a myriad of dexterous motor behaviors.[1] In light of this, several forms of motor primitives have been suggested for robot motion planning and control.[2,3] When motion is based on motor primitives, learning to perform a task is related to learning the suitable motion primitive parameters. Similarly, performance adaptation and improvement during runtime are related to adaptation of the motion primitive parameter values. The current work suggests learning a continuous mapping of task requirements to motion primitive parameter values, which can be used during runtime for motion adaptation to changes in task requirements. Since the mapping is formed *a priori*, it ensures predictable and robust task performance.

Learning motion primitive parameters for a required task is typically performed based on learning from demonstration or reinforcement learning (RL), and most commonly, by combining both methods.[4–6] Several methods have been suggested for handling changes in task requirements without additional learning epochs, for example, selection of a suitable motion primitive from a set of pre-trained primitives,[7] concatenation of several motion primitives,[8–10] and linear extrapolation of the motion primitive's parameters.[10] The relationship between task parameters and motion primitive parameters is typically nonlinear, forming a manifold in the parameter space. Therefore, linear methods are bounded to a small region about the original parameter values in which local linearity can be assumed. Differently, nonlinear methods can lead to an efficient representation of the parameter manifold and to a larger continuum of task requirements that can be successfully met.

* Corresponding author. E-mail: sigalbe@bgu.ac.il

Using meta parameters as a way to generalize learning is a common machine learning methodology,[11] and to this end, meta parameters have also been defined for motion primitives.[12,13] We present a method facilitating adaptation of motion primitive meta parameters with respect to task parameters based on an *a priori* learned mapping of the parameter manifold. Before learning the mapping, we analyze the relationship between the parameters for ensuring learning efficiency. We compare two common manifold mapping learning methods: deep neural networks (DNNs) and kernel estimation. The methods are examined for a soft ball throwing task. Low runtime estimation errors were obtained for both estimation methods, with an advantage to kernel estimation when training data sets are small. The rest of this paper is organized as follows: related work is presented in Section 2; the methodology is presented in Section 3; an in-depth case study of implementing the methodology for a robotic soft ball throwing task in simulation and in a physical setup is presented in Section 4; results are presented in Section 5; and discussed in Section 6.

2. Related Work

Dynamic movement primitives (DMPs)[14–17] are a commonly used form of motion primitives. They were derived based on combining notions from optimal control theory and dynamic systems. A DMP is a nonlinear second-order dynamic system. Where the nonlinearity facilitates representing complex motion polices. Their parametrization has been inspired by physical models, for example, a damped spring system perturbed by a nonlinear acceleration and summation of acceleration fields.[17] In several approaches aimed at DMP generalization, the DMP parameters have been divided into three categories:[12,13] shape parameters, meta parameters, and external parameters. Shape parameters define the spatio-temporal shape of the movement. The shape of a movement is maintained under affine transformations in time and/or in space, for example, change of size, time scale, rotation, and translation. The meta parameters are the values that determine the DMPs frame of operation, for example, the start and goal positions, movement duration, and internal gains. The external parameters are not specific to the DMP formulation, and they determine external, high-level issues that affect DMP operation, for example, when to start the motion with respect to additional task constraints. The task parameters determine the required outcome of the task, for example, for the soft ball throwing task used in the current work, task parameters determine the required landing position on the floor. DMP parameters (shape, meta, and external) are determined for given task parameters. For example, for the DMP vectorial equations defined as in.[17]

$$\begin{aligned} \dot{v} &= K(x_g - x) - Dv - K(x_g - x_0)s + Kf(s) \\ \dot{x} &= v \end{aligned} \tag{1}$$

where $x(t)$ is the position and $v(t)$ is the velocity of the required trajectory; the meta parameters are the initial position x_0, the goal position x_g, the time scaling factor τ, the stiffness K, and the damping D.

The nonlinearity is introduced to the DMP control equations using $f(s)$, which is a nonlinear function defined by a weighted sum of Gaussian kernels, Ψ_{I},

$$\begin{aligned} f(s) &= \frac{\sum_i \psi_i(s) w_i}{\sum_i \psi_i(s)} s \\ \psi_i(s) &= e^{-b^i (s - c_i)^2} \\ \tau \dot{s} &= -\varepsilon s \end{aligned} \tag{2}$$

where s is the phase variable and ε is the time convergence constant. The Gaussian kernels are defined by the Gaussian distribution parameters b and c, such that the kernels span the path space. Finally, the weights of the kernels, w_i, are the shape parameters as they determine the spatio-temporal shape of the trajectory. The external parameters are not part of the DMP formulation.

Several RL methods have been applied for finding optimal DMP parameters.[18] Research has concentrated mainly on shape and meta parameters. The use of external parameters is less common, and their determination and optimization are typically related to general task sequencing and thus less addressed within literature on DMP parameters. Most researchers develop methods for learning shape parameters for a given set of meta and task parameters (e.g.,[5,6,13]). Some researchers have developed methods for learning meta parameters for given shape parameters.[12,19–21] Others have integrated learning meta and shape parameters, either hierarchically[9] or in parallel.[21–23] While RL

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

indeed facilitates acquiring new skills, its use for adaptation during runtime has several limitations. The continuous parameter space along with the high dimensionality of typical robotic motor-learning problems lead to long learning epochs. This may be unacceptable during runtime for many tasks, especially when human–robot collaboration is required.[6,24,25]

A different body of research on data-driven models of motion, seeks to generalize available data, forming concise representations that capture salient motion characteristics. For example, wrist configurations required for achieving high-quality grasps for a given object were determined and represented using graspability maps and grasp density functions.[26,27] A similar concept can be used for constructing models of DMP meta and task parameters. Rueckert et al.[28] suggest using hierarchical Bayesian models for estimating both meta and shape parameters for probabilistic movement primitives. Ugur and Girgin[20,21] suggest learning parametric hidden Markov models from multiple demonstrations for encoding relations between shape parameters and environment properties. Probabilistic movement primitives facilitate encoding optimal behaviors in stochastic systems, yet when deterministic system behavior (i.e., completely predictable operation) is required, they are less suitable. Additionally, multiple demonstrations are typically required for learning primitive distribution parameters taxing the initial learning process.[29]

Two manifold learning methods are suggested for learning the mapping of the task and meta parameter manifold based on training data, kernel estimation, and DNNs. Both methods are suitable for learning efficient manifold representations,[30–32] yet they have different strengths and they are suitable for different estimation conditions. DNNs can efficiently represent complex manifold structures. Moreover, DNNs scale in a straight forward manner to high manifold dimensions, and they can handle correlations between output variables. However, DNNs are notoriously known for requiring large training data sets for attaining an accurate representation. Kernel estimation is a classical machine learning method, highly suitable for capturing the nonlinear data structures such as manifolds. Due to the ability to locally adapt the kernel weights, a smooth and highly accurate representation can be achieved with only a moderate training data set size. Moreover, task-related knowledge regarding the relationship between the parameters can be readily incorporated in the learning process, making it more efficient. However, kernel estimation does not scale well to high manifold dimensions and is not well suited for handling correlations between output variables. According to these insights, in the current work, the two learning methods are examined with respect to the size of the available training data set and the complexity of the task and meta parameter manifold.

For ensuring efficient learning, the complexity of the task and meta parameter manifold can be examined using problem dimension estimation methods. Two commonly used methods are principal component analysis (PCA)[33,34] and locally linear embedding (LLE).[35] PCA is a linear method that determines a coordinate system, in which each axis is a linear transformation of the original data features and the data's principal components, i.e., the axes onto which the retained variance under projection is maximal.[33,34] PCA additionally enables representation of dominant patterns in the data by plotting the primary principal components. LLE is a nonlinear method which computes low-dimensional, locally linear, neighborhood-preserving embedding of high-dimensional inputs.[35] The method is based on a simple geometric intuition: if data are sampled from a smooth manifold, then neighboring points remain similarly co-located in the low-dimensional space. In LLE, each point in the data set is linearly embedded into a locally linear patch of the manifold. The low-dimensional data are constructed in a way that ensures that the locally linear relations of the original data are preserved.[36] The downside of using LLE is that unlike PCA analysis, the relationship between the output and input parameters is not readily available. This is typical for nonlinear manifold learning methods. Both PCA and LLE facilitate visualization of the parameter response surface in the lower dimensional space. PCA is more suitable for data with linear or near linear dependencies and LLE for data with significant nonlinear dependencies.[37]

In the classical DMP formulation, shape parameters are established *a priori* and retained for motion generation during runtime. In the tight DMP (TDMP) formulation,[38] the coordinates of a typical movement trajectory are retained for motion generation. During runtime, these coordinates are linearly transformed based on trajectory landmarks (e.g., start and goal positions) and suitable shape parameters are calculated using linear regression from the transformed trajectory along with determined meta parameters. This formulation ensures a high fit of the trajectory formed by TDMP to the shape of the typical trajectory.

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

In the presented methodology, TDMP parameter adaptation during runtime is based on a hierarchical process; first meta parameters are determined based on the task and meta parameters manifold, and then shape parameters are found based on these meta parameters and the transformed coordinates of a typical trajectory. The methodology is examined for a soft ball throwing task, in which the ball is thrown to a designated landing position.

3. Methodology

3.1. Overview

The suggested methodology for robust, runtime, DMP parameter adaptation based on task requirements has an *a priori* learning stage and two sequential runtime stages. In the *a priori* stage, the dimensionality of the parameter space is qualitatively evaluated using PCA and LLE. This analysis is used for guiding the parameter mapping processes, that is, determining the regression equations for the Gaussian kernel estimation or the hyper-parameters governing the network training processes. The task and meta parameter mapping is learned using kernel estimation or DNNs based on a training data set of task and meta parameters (Fig. 1). During runtime, the meta parameters are determined based on the task and meta parameter mapping and then shape parameters are determined for the meta parameters and a typical trajectory using the TDMP method. The typical trajectory is a predetermined (recorded, computed, or learned) movement trajectory. In the general case, the task requirement space can be divided into several instances of such typical trajectories. For example, typical trajectories for table tennis can be defined for each of the four basic strokes (forehand drive, backhand drive forehand push, and backhand push).

3.2. Learning a representation of the task and meta parameter manifold

For a given robotic system, environment, and task, there are n task parameters, $\delta_1, \ldots, \delta_n$, m meta parameters, $\gamma_1, \ldots, \gamma_m$, and p shape parameters, $w_1, \ldots, w_p$. The task and meta parameters are related through a manifold $\Phi(\delta_1, \ldots, \delta_n, \gamma_1, \ldots, \gamma_m)$ embedded in the parameter space.

3.2.1. Kernel estimation. A set of m functions, $g_j(\delta_1, \ldots, \delta_n, \gamma_1, \ldots, \gamma_{i\neq j}, \ldots, \gamma_m)$, $j = 1 \ldots m$, relating each meta parameter γ_j to the n task parameters δ_i, and the rest of the $m-1$ meta parameters γ_i, $I \neq j$ form a map of dimension $n+m-1$ of the task and meta parameter manifold.[39] These functions can be estimated using kernel estimation by a weighted sum of $n+m-1$-dimensional Gaussians kernels,

$$\begin{aligned}\gamma_j &= \tilde{g}_j(\delta_1, \ldots, \delta_n, \gamma_1, \ldots, \gamma_{i\neq j}, \ldots, \gamma_m) \\ &= \frac{\Theta_j \cdot \Gamma_j(\delta_1, \ldots, \delta_n, \gamma_1, \ldots, \gamma_{i\neq j}, \ldots, \gamma_m)}{\sum_{i_{n+m-1}}^{k_j} \cdots \cdots \sum_{i_3=1}^{k_j} \sum_{i_2=1}^{k_j} \sum_{i_1=1}^{k_j} \Gamma_j(\delta_1, \ldots, \delta_n, \gamma_1, \ldots, \gamma_{i\neq j}, \ldots, \gamma_m)}\end{aligned} \tag{3}$$

where k_j is the number of kernels in each axis (for simplicity, we assume the same number of kernels in each axis), and $k_j \cdot (n+m-1)$ is the number of kernels representing each meta parameter, Θ_j is a $k_j \cdot (n+m-1)$-dimensional tensor of weights, and Γ_j is a $k_j \cdot (n+m-1)$-dimensional tensor of kernel functions,

$$\begin{aligned}\psi_{i_l}^j(\delta_1, \ldots, \delta_n, \gamma_1, \ldots, \gamma_{i\neq j}, \ldots, \gamma_m) &= \exp\left(\frac{-(\delta_1 - c_{i_l}^j(\delta_1))}{2(\sigma_{i_l}^j(\delta_1))^2}\right) \exp\left(\frac{-(\delta_2 - c_{i_l}^j(\delta_2))}{2(\sigma_{i_l}^j(\delta_2))^2}\right) \cdots \\ \exp\left(\frac{-(\delta_n - c_{i_l}^j(\delta_n))}{2(\sigma_{i_l}^j(\delta_n))^2}\right) &\exp\left(\frac{-(\gamma_1 - c_{i_l}^j(\gamma_1))}{2(\sigma_{i_l}^j(\gamma_1))^2}\right) \exp\left(\frac{-(\gamma_2 - c_{i_l}^j(\gamma_2))}{2(\sigma_{i_l}^j(\gamma_2))^2}\right) \cdots \\ \exp\left(\frac{-(\gamma_{i\neq j} - c_{i_l}^j(\gamma_{i\neq j}))}{2(\sigma_{i_l}^j(\gamma_{i\neq j}))^2}\right) &\ldots \exp\left(\frac{-(\gamma_m - c_{i_l}^j(\gamma_m))}{2(\sigma_{i_l}^j(\gamma_m))^2}\right), \quad j = 1, \ldots, n+m-1, \; l = 1, \ldots, k_j\end{aligned} \tag{4}$$

where $c_{i_l}^j$ are the means and $\sigma_{i_l}^j$ are the standard deviations.

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

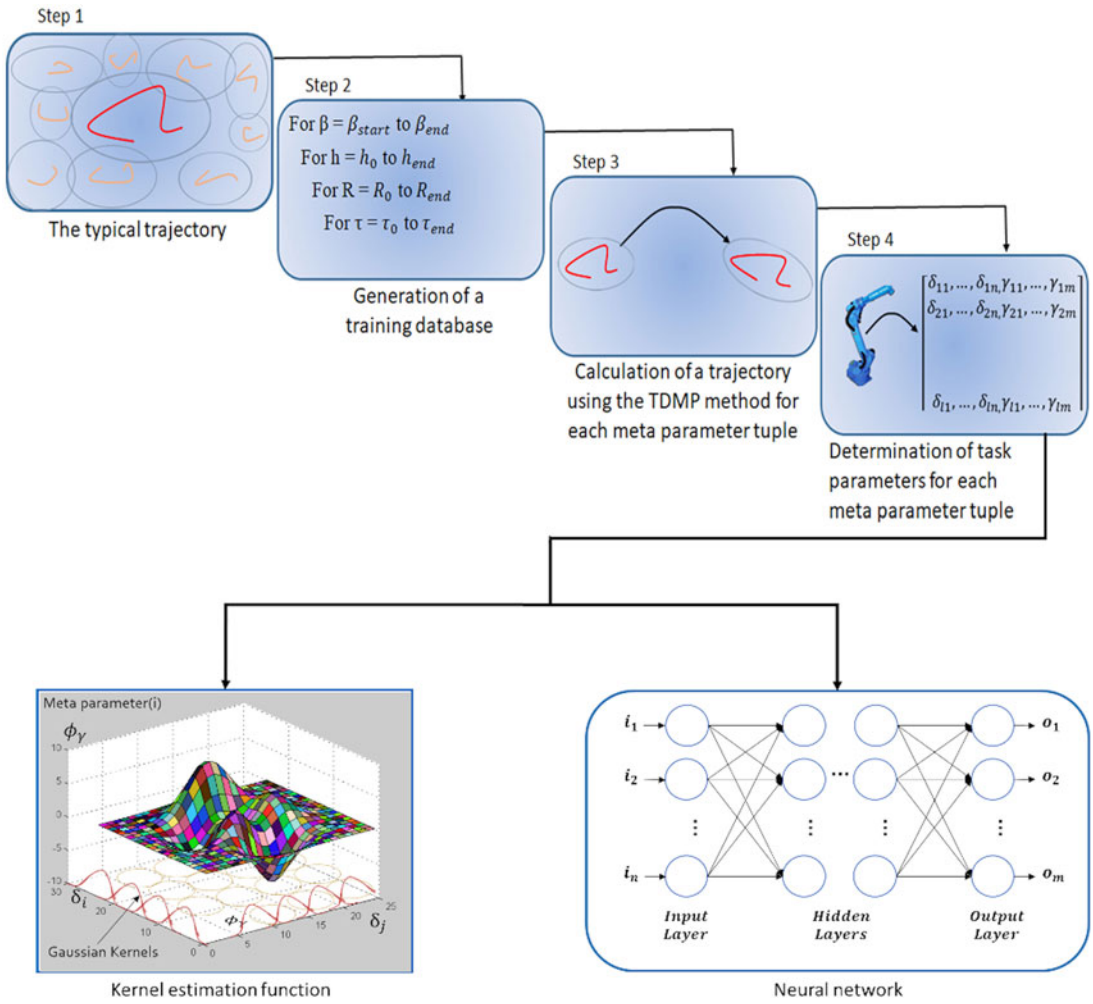

Fig. 1. Generation of a data set for the task and meta parameter mapping. Step 1: Select a typical trajectory, Step 2: Determine a set of meta parameters. Step 3: For all the meta parameters in the set, calculate a trajectory using TDMP. Step 4: For each trajectory, extract the task parameters. The database of task and meta parameters is used for the kernel estimation or DNN training.

For Gaussian kernels, the kernel estimation parameters include the number of kernels, the kernel means, and their standard deviations. The number of kernels balances problem complexity with attainable estimation resolution. The kernel means must assure good coverage of the ranges of data values. The standard deviation values affect attainable estimation accuracy. Selecting large standard deviations may result in a smooth function, but with a low fit to the data, and selecting small standard deviations may result in overfitting the data. There are several methods for choosing the optimal standard deviations for a given data set.[40,41] Among them, the most commonly applied method is based on finding the standard deviations that yield the smallest average square error for the validation data set.[41] We chose this method due to its relative simplicity and high attained performance in many practical cases.

Knowledge regarding the relationship between the parameters can be readily incorporated in the estimation process, by pre-determining a subset of the task and meta parameters that will be included in the estimation of each meta parameter. Reducing the dimensionality of the problem in such a way reduces the estimation time and the size of the required training data set. Additionally, it increases the robustness of the solution to noisy measurements.

3.2.2. Deep neural network. The hierarchical, layered structure of DNNs facilitates capturing complex data structures, where the outputs are continuous piecewise linear functions of the inputs.[32] Accordingly, a neural network can take as input the vector of task parameters and supply as output

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

the vector of suitable meta parameters. The networks chosen in the current work were feed-forward networks with a rectified linear unit activation function, and mean absolute error loss function. The network structure was optimized with an Adam optimizer.[42–44] We performed a grid search[45] to determine the number of hidden layers and the number of neurons in each layer and to set the learning rate.

3.3. Task and meta parameter manifold structure

Examining the structure of the task and meta parameter manifold is important for evaluating problem complexity and for understanding the interactions between the parameters. Such knowledge facilitates establishing suitable regression equations and regression estimation parameters (e.g., the number of kernels). It is similarly important for correctly establishing the search space for the hyperparameters of the DNN. We examined the structure of the task and meta parameter manifold using two methods, PCA and LLE. The kernel estimation equations were determined based on the weights of the PCA components. Task and meta parameters whose weights were high in the same principal component were used in the same regression equation. Problem complexity was assessed by examining the variance explained by the PCA components and the layout of the top components of both PCA and LLE. For PCA, when accumulation of 95% of the variance requires more components, the problem space is more complex. When PCA or LLE parameter dispersion is less uniform, problem complexity is higher. Higher problem complexity requires more kernels or deeper networks with more neurons in each hidden layer. Finally, the PCA weights influence the effort to minimize the error of each meta parameter when learning the structure of the DNN. A meta parameter that has larger weights in the top principal components is marked as more important, and the error threshold guiding the training for this parameter is lower.

4. Case Study: A Robotic Soft Ball Throwing Task

A robotic soft ball throwing task was chosen for demonstrating the methodology. The target of the task is to throw a soft ball to a designated landing point. The task was chosen since a basic motion trajectory can be readily defined for the throwing motion. However, the physical task dynamics are complex since due to drag, they are influenced by the ball's surface. These dynamics have nonlinear components which make the estimation of the task and meta parameter manifold non-trivial. The task was examined in simulation and in a physical environment. Simulation facilitates collection of large amounts of data. However, the physical implementation is important for attaining physical grounding. The throwing task parameters are the required landing point on the floor defined by the radius r and the horizontal angle α (Fig. 2a). The meta parameters are the endpoint of the robotic throwing motion (where the robot stops at the end of the task), defined by the horizontal angle β, height h, and radius R, the time scaling factor τ, the stiffness K, and the damping D. For a typical throwing trajectory, the endpoint of the robotic throwing motion directly influences the ball release point and velocity. Therefore, the endpoint of the robotic throwing motion has a major impact on the ball's landing position. In the current work, we investigated adaptation of the endpoint meta parameter, while the rest of the meta parameters were defined as constants with values appropriate for the required task parameter range.

4.1. The physical environment

A 6 degrees of freedom (DOF) HP6 manipulator (Yaskawa, Motoman, Japan) was used in the physical setup (Fig. 2b). Its manipulator was fit with a ball holder. The orientation of the ball holder was set such that the ball will be released when the robot is moving at its maximal velocity, before it starts decelerating. The room setup was suitable for testing landing positions in a distance of 0.8–1.6 m from the manipulator and ±27 degree about it. According to these dimensions, a green plastic mat was printed marking distance and angles about the robot's base. An overhead video camera was mounted for determining the landing position on the mat. A soft red ball made out of clay gel polymer was used for throwing (radius 26 mm, weight 90 g). The moderate adhesiveness caused the ball to remain at the landing point after the initial impact, which simplified the identification of the landing point and added some nonlinearity to the formed trajectory. Communication between the station computer and the robot controller was established using RS-232 serial communication and the Motocom32^{TM} software. A typical trajectory suitable for throwing the ball was manually programmed. Before each run, the ball was loaded into the ball holder and the robot was positioned at

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

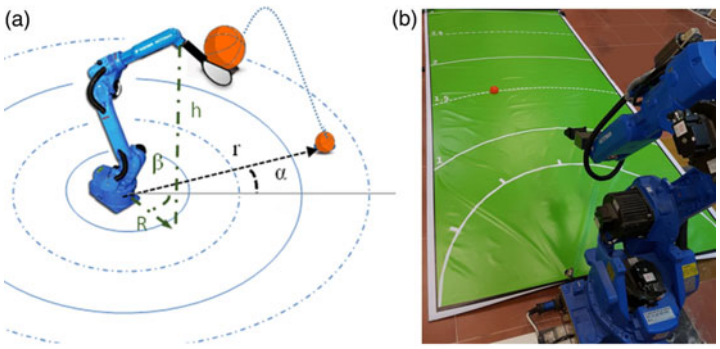

Fig. 2. (a) The task and meta parameters. The task parameter – landing point on the floor is defined by the radius r and the horizontal angle α. The meta parameter – motion endpoint is defined by the horizontal angle β, height h, and radius R (and motion duration τ). (b) The physical implementation with a HP6 Motoman robot, ball holder, and the mat for determining ball landing position.

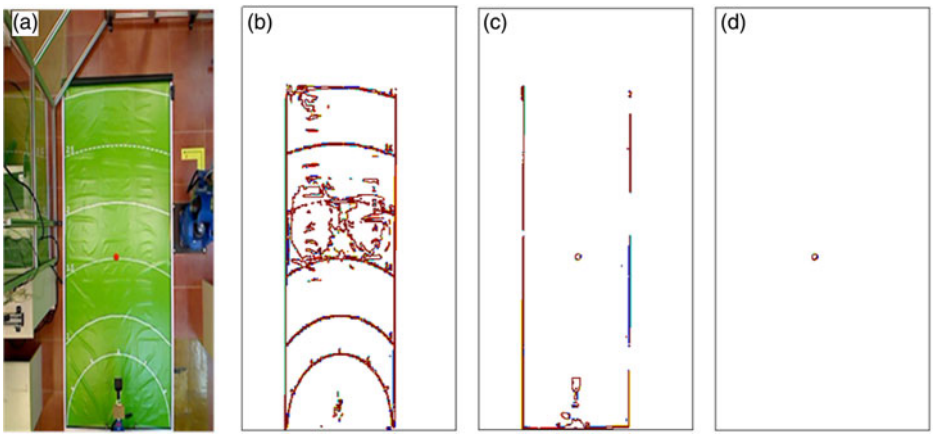

Fig. 3. (a) An overhead camera was mounted above the arena, and an image was taken with a short delay after the ball was thrown. (b) The image mask after band-pass filtering of the green image plan (low green values: Low = 2 and High = 100). (c) The image mask after band-pass filtering the red image plane (high red values: Low = 150 and High = 300). (d) Multiplication of images b & c followed by elliptical dilation for smoothing.

an initial configuration in which the end effector was low and close to the robot's base. The required motion trajectory was computed by the station computer based on the meta parameters and the transformed typical trajectory. The trajectory was sent to the robot controller, and the robot executed the motion. The robot's configuration at the end of the motion was sent back to the control computer for verification, and the landing position of the ball was automatically extracted from the image taken by the overhead camera using color-based identification (Fig. 3) and either added to the formed training database or used for outcome evaluation.

4.2. Simulations

The simulation was developed using Matlab (Mathworks Inc., U.S) on a computer with an Intel® core™ i7, and 8 GB RAM. The HP6 manipulator (Yaskawa, Motoman, Japan) used in the physical setup and was modeled in simulation with the Matlab Robotic Toolbox.[46] The robot's initial configuration and the typical trajectory programmed in the physical environment were transformed to the coordinates of the simulated environment. The landing point in each run was determined based on the modeled dynamic equations of the ball's trajectory after it was released by the robot. Two simulation models with different ball trajectory dynamic equations were developed: a ballistic model and an attenuated model. For both models, the meta parameter values used for establishing the training database were $\beta = [-45°{:}45°]$, $h = [40\text{ mm}{:}140\text{ mm}]$, $R = [500\text{ mm}{:}700\text{ mm}]$. The ballistic model disregarded drag forces was based on the standard ballistic motion equations (Eq. 5). The meta

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

parameter values and these equations led to ball release velocities in the range of 0.2–0.5 m/s and to landing distances in the range of 0.5–1.8 m with landing angles in the range of ±72 degree.

$$\begin{aligned} r_x(t) &= r_z(0) + v_x(0)t \\ r_y(t) &= r_y(0) + v_y(0)t \\ r_z(t) &= r_z(0) + v_z(0)t + \frac{1}{2}gt^2 \end{aligned} \tag{5}$$

where $v_0 = [v_x(0)\ v_y(0)\ v_z(0)]$ is the velocity of the robot when the ball is released from the holder (the maximal velocity of the robotic motion profile).

The attenuated model was adapted in order to better account for some of the characteristics of the motion of the physical robot. The velocity equation of the attenuated model (Eq. 6) reflects the fact that the velocity of the physical robot at the ball release point was higher than the velocity prescribed by the DMP profile due to differences between the modeled and actual acceleration used by the robot. In addition, it was harder for the robot to reach high velocities when throwing the ball toward the side using the defined typical motion profile (reflected by the added cosine function of β) and it was harder for the robot to develop high velocities for very low or very high throwing distances (reflected by the added exponential function of R) (Eq. 6). These led ball release velocities in the range of 0.3–1.2 m/s and to landing distances in the range of 0.5–8 m with landing angles in the range of ±79 degree.

$$v_{0\_a} = v_0\left(1 + e^{\frac{-|R_{max}-R|}{c_w}}\right)(1+\cos(\beta)),\ -90 < \beta < 90 \tag{6}$$

where R_{max} is the goal point radius for which the maximal velocity was obtained over all trials and c_w is a scaling constant. In the current work, c_w was empirically set to 20.

4.3. Data sets

Data sets of task and meta parameter tuples were created based on a set of meta parameters using the simulation models (ballistic and attenuated) and with the physical apparatus. Given the meta parameters and transformed typical trajectory, shape parameters were calculated using TDMP. Then, the resulting task parameters were calculated by using the DMP controller of the robotic motion, in simulation (ballistic or attenuated) or within the physical environment.

Different training data sets were constructed for different analyses. For all data sets, motion duration was set as a constant value of $\tau = 5$ since preliminary analysis with the physical robot showed that this is an appropriate duration value for a successful throw. Similarly, the stiffness K was set to 150, and the damping D was set to 24.9. The data sets were constructed by varying the values of the motion endpoint defined by the horizontal angle β, height h, and radius R. For kernel estimation, training data sets of different sizes were constructed as full factorial array of meta parameter values. Data set size was determined by the number of equally spaced parameter values used. For examining data complexity, for training the DNNs, and for testing, data sets of different sizes were constructed by randomly sampling motion endpoint values from a uniform distribution over the range of each parameter.

Both kernel estimation and DNNs can be trained using iteratively converging algorithms which minimize a cost function, for example, gradient decent. Kernel estimation can also be achieved by linear regression with a least squares cost. In this case, the solution can be found analytically and, therefore, ordered data can be used. When using iterative estimation algorithms, it is critical that incoming data are not ordered, as ordered data may lead to convergence to a local minimum. Sampling data points from a uniform distribution is a suitable method for constructing unordered data sets. Yet, for small data sets, a stochastically sampled data set may not cover the input range well. Therefore, for small data sets, if possible, it is better to use ordered sets.

We hypothesize that kernel estimation will have advantages over DNNs when using small data sets. To verify this issue, we opt to use the data set that was most suitable for each method (full factorial for kernel estimation and uniform distribution for DNNs). For each method, with q meta parameter values, where $q = 2-1, 0$ therefore 8, 27, 64, 125, 216, 343, 512, 729, and 1000 parameter tuples training data sets, and 125 parameter tuples test data set were constructed for each simulation model (ballistic, attenuated). Data complexity was analyzed using the randomly sampled 1000

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

parameter tuples training set for each simulation model. In the physical environment, a training data set with 45 parameter tuples and a test data set with 12 parameter tuples were constructed.

4.4. Procedure

4.4.1. Examining data complexity. The relationship between the parameters was examined with PCA and LLE. After the PCA was performed, the weights of each parameter in the top principal components were examined. The number of required dimensions was set based on the number of principal components required for explaining 95% of the variance. The number of required linear dimensions is an upper bound for the number of required nonlinear dimensions. Therefore, the number of dimensions established based on PCA was used for computing the LLE. The kernel estimation equations and the neural network structure were adjusted according to the outcome of the PCA and LLE analysis, as discussed in Section 2.3

4.4.2. Defining the kernel estimation parameters. For determining the number of required kernels, we examined the average estimation error as a function of the number of kernels for both simulation models (ballistic, attenuated). The number of kernels examined was 8, 27, 64, 125, and 343. For each kernel density, the kernel means were uniformly distributed in the parameter space. The standard deviations were found as in[41] by minimizing the square error. The number of kernels was examined using a medium size training data set with 125 records constructed for each of the simulation models as full factorial array of meta parameter values.

4.4.3. Defining network structure. To ensure the best possible trained network for each data set, a DNN was devised for each data set for each of the simulation models (ballistic, attenuated). For each training data set, 20% of the tuples were randomly defined as a validation set and kept aside for model selection. For all networks, the number of neurons in the input layer was equal to the number of task parameters (r, α) and the number of neurons in the output layer was equal to the number of meta parameters (R, β, h). The batch size and the number of epochs were empirically set to 32 and 1000, respectively. A grid search was performed to adjust the number of hidden layers, the number of neurons in each layer, and the learning rate for the optimizer. The values for the number of layers and neuron were chosen based on the results of the PCA and LLE analysis. More neurons and layers are required for more complex problems, since poor performance is achieved when the network is too shallow for the problem. On the other hand when a network is too deep, redundancy can be created which ultimately decreased the performance of the network. The learning rates tested were [0.01, 0.001, 0.0005]. An early stop technique was used, according to which the training was stopped when the model performance stops improving with the validation set over 10 sequential epochs.[47] The final model for each data set was devised with the combination of hyper-parameter values that minimized the mean absolute error (MAE) between the estimated and true values, for the validation set,

$$\mathrm{MAE}_y = \frac{1}{n} \sum_{t=1}^{n} \left| y_t - \hat{y}_t \right| \tag{7}$$

where y is one of the meta parameters (β, h, R), n is the number of samples, y_t is the truth value, and $\hat{y}_t$ is the estimated value.

4.4.4. Testing the influence of data set size on the estimation error. The size of the required training data set is an important consideration for choosing a learning method since requirements for large training data sets may be very costly. The analysis of the influence of the training data set size of the estimation error was performed with both learning methods, kernel estimation and DNNs, for both simulation models, the ballistic model and the attenuated model.

4.4.5. Examining the methodology in the physical environment. In the physical environment, three alternative data sets for learning the task to meta parameter mapping were examined: a training data set constructed in the physical environment (with 45 parameter tuples) and two training data sets (with 125 parameter tuples) based on the simulated data (the ballistic model and the attenuated model). Since collecting data in simulation is considerably faster than that in the physical

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

implementation, both simulation training data sets are considerably larger than the physical data set. Yet both simulation models make simplification assumptions with respect to the physical reality. All mappings were tested using the physical test set.

4.5. Analysis

We examined the errors of the meta parameter mapping and the resulting task parameter errors. The meta parameter mapping error is directly computed comparing the output of the learned estimation function to the ground truth value of the required meta parameters. Computing the error of the resulting task parameters with respect to the ground truth values requires an additional execution step, that is, computing the task parameters which result from the throwing motion conducted based on the estimated meta parameters (using simulation or executing with hardware). The errors in task parameters characterize the overall performance, yet the meta parameter estimation errors and their relationship with the task parameter errors are important for evaluation of the learning outcome and for understanding possible tradeoffs.

Distance errors to the ground truth values, R (endpoint radius) and r (ball landing point radius), were calculated as the mean scaled absolute error (MSAE),

$$\mathrm{MSAE}_y = \frac{1}{n} \sum_{t=1}^{n} \frac{|y_t - \hat{y}_t|}{y_t} \tag{8}$$

where y is one of the parameters (R, r), n is the number of samples, y_t is the truth value, and $\hat{y}_t$ is the estimated value.

Angle errors to the estimated ground truth values β (endpoint horizontal angle) and α (ball landing point horizontal angle) were calculated as the MAE (Eq. 7).

4.6. Statistical analysis

For analyzing the effects of data set size on the estimation error in simulation, we applied a mixed ANOVA with method (Kernel estimation, DNN) and model (Ballistic, Attenuated) as between-subjects parameters and data set size as the within-subject parameter. For analyzing the effects of the training data set source on the estimation error in the physical environment, we applied a repeated-measures ANOVA analysis with mapping data source (Physical, Ballistic, Attenuated) as within-subject parameters.

5. Results

5.1. Examining data complexity

In both the ballistic model and the attenuated model, the top 3 PCA components and the top 4 PCA components explain more than 95% and 99% of the variance, respectively (Fig. 4). Based on this, we determine that the complexity of both models is moderate. The dispersion of the first 3 PCA components was similar and nearly uniform in both models (Fig. 5), but the dispersion of the first 3 LLE components differed, showing different concentrations along different axes, suggesting stronger internal constraints among the parameters of the attenuated model (Fig. 6). This suggests that the attenuated model will require higher parameter accuracy for attaining a low error than the ballistic model.

Examining the PCA components shows a clear separation between the parameters (Fig. 5). The angles (α, β) have very high weights in the 1st and 5th components, the distances (r, R) in the 2nd and 4th components, and the height (h) in the 3rd component. This implies separation between distances and angles and a weaker relationship between the values of height and those of other parameters. There was still some influence of the height on the 2nd and 4th components (distances) and some influence of the angles on the 3rd component (height). These phenomena are easily related to the equations of motion describing the model. For the same release distance, for different release heights, a different final ball landing distance will be attained. The release height is also related to the attained final angle for the same release angle.

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

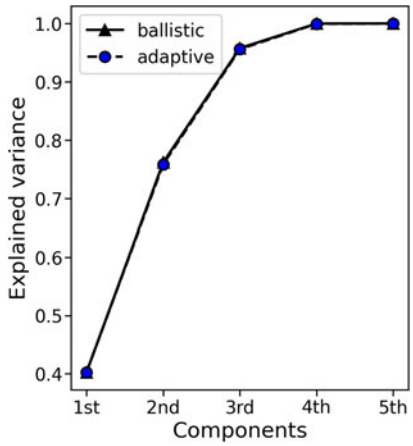

Fig. 4. Cumulative variance of the principal components. (a) Ballistic model. (b) Attenuated model.

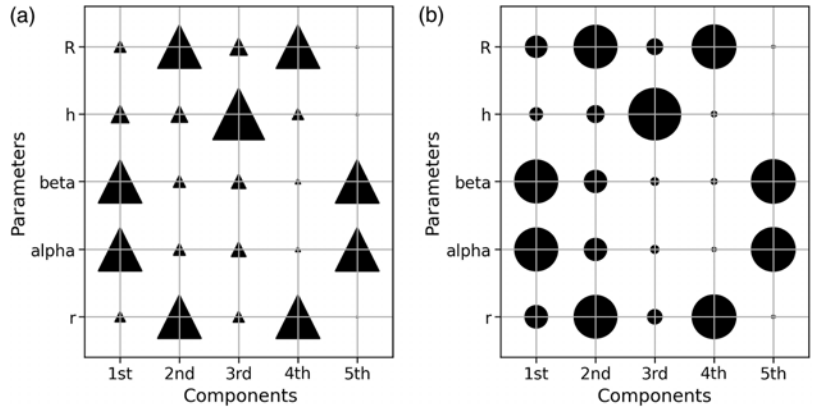

Fig. 5. Weights of the parameters in the principal components. (a) Ballistic model. (b) Attenuated model. The shapes symbolize the type of the model.

5.2. Kernel estimation parameters

Based on the PCA results, two separate mapping functions were defined one for task distance (R) and one for task angle (β) for describing the $\Phi(R, \beta, h, r, \alpha)$ manifold,

$$\gamma_{\mathrm{R}}(\mathrm{r}, \alpha, \mathrm{h}) = \frac{\Theta_{\mathrm{R}} \cdot \Gamma_R(\mathrm{r}, \alpha, \mathrm{h})}{\sum_{i_3=1}^{k_R} \sum_{i_2=1}^{k_R} \sum_{i_1=1}^{k_R} \Gamma_R(\mathrm{r}, \alpha, \mathrm{h})}$$
$$\gamma_{\beta}(\mathrm{r}, \alpha, \mathrm{h}) = \frac{\Theta_{\beta} \cdot \Gamma_\beta(\mathrm{r}, \alpha, \mathrm{h})}{\sum_{i_3=1}^{k_\beta} \sum_{i_2=1}^{k_\beta} \sum_{i_1=1}^{k_\beta} \Gamma_\beta(\mathrm{r}, \alpha, \mathrm{h})} \quad (9)$$

where $\gamma_R(r, \alpha, h)$ and $\gamma_\beta(r, \alpha, h)$ are smooth functions over the three-dimensional space of r, α, and h, Θ_R and Θ_β are the weight tensors and Γ_R and Γ_β are the kernel tensors.

For the ballistic model, kernel density had little effect on the error of both angles (β, endpoint, and α ball landing point) (Fig. 7b), while the radial errors (R, endpoint, and r, ball landing point) decreased similarly, as the number of kernels increased until reaching a plateau after 3 kernel per dimension (over all 27 kernels) (Fig. 7a). For the attenuated model, kernel density affected both the horizontal angle error (α, β) (Fig. 7d) and the radial distance (R, r) (Fig. 7c), where the error decreased as the number of kernels used increased. The error did not further decrease when increasing the number of kernels used beyond 4 kernel per dimension (over all 64 kernels).

5.3. Error as a function of data set size

The PCA and LLE analysis indicated that the complexity of the problem was moderate; thus, moderate values for the lower and upper boarders of the grid search for the DNN hyper-parameters were

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

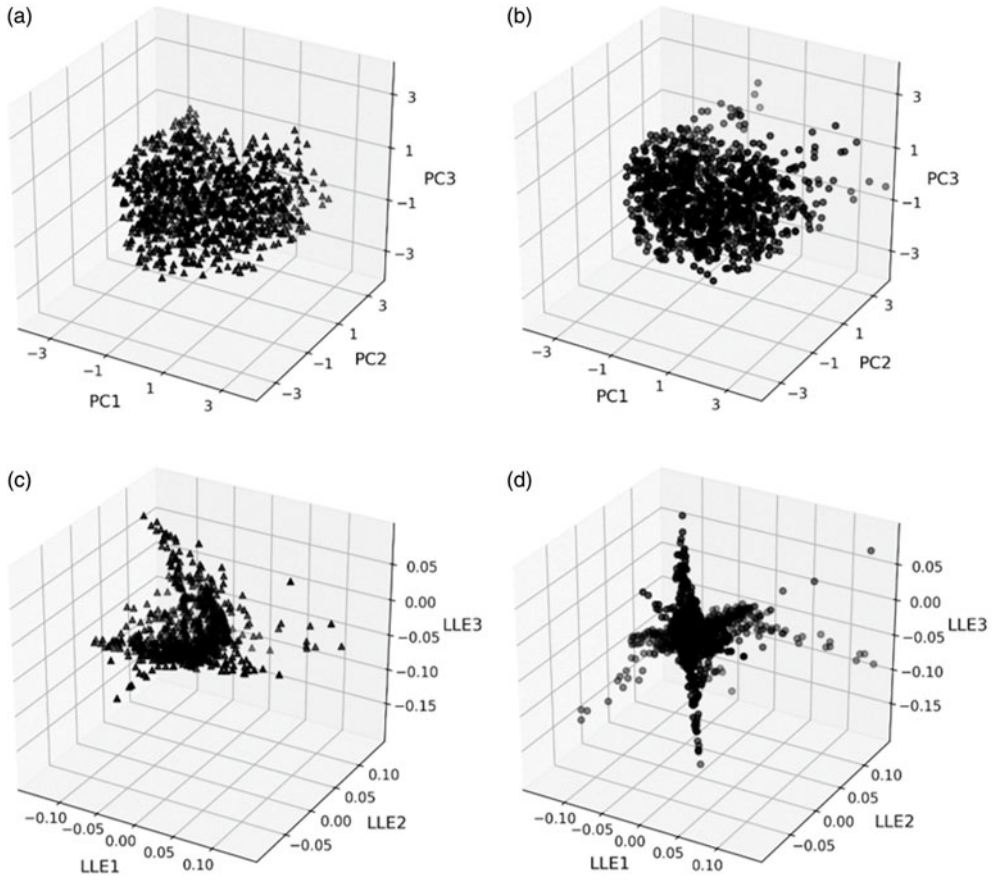

Fig. 6. Top: 3-dimensional visualization of the top 3 principal components. (a) Ballistic model. (b) Attenuated model. Bottom: 3-dimensional visualization of the top 3 LLE components. (c) Ballistic model. (d) Attenuated model. The shapes symbolize the type of the model.

tested: the number of hidden layers tested was 3, 5, 7, 8 and the number of neurons in each layer tests was 10, 20, 32, 64. The search was conducted separately for each database size. Similarly, based on the PCA and LLE analysis, 27 kernels were used for the ballistic model and 64 kernels for the attenuated model for the kernel estimation.

For both the ballistic and attenuated models and for both estimation methods (kernel estimation and DNNs), the meta parameter estimation error decreased as the size of the training data set increased (Fig. 8). For kernel estimation, the reduction in estimation error was very small when data set size was increased beyond 30 tuples for the ballistic model and 64 tuples for the attenuated model. For DNNs, the reduction in estimation error was very small when increasing data set size beyond 500 tuples for both the ballistic model and the attenuated model. Beyond this value (500 tuples), the error converged in both methods to values of 3% for the landing distance (r) and 1° for the landing angle (α) in the ballistic model and to values of 12% for the distance and 2° for the angle in the attenuated model.

Both the distance and angle errors were smaller when data set size was increased ($r : F_{1,4340} = 147\ p < 0.0001,\ \alpha : F_{1,4340} = 121\ p < 0.0001$). Both were smaller for the ballistic model than for the attenuated model ($r : F_{1,4340} = 578\ p < 0.0001,\ \alpha : F_{1,4340} = 257 p < 0.0001$). The distance error was similar for both estimation methods, yet the angle error was larger for the DNN ($F_{1,4340} = 533\ p < 0.001$). For both errors, there was a significant three-way interaction between model, method, and data set size ($r : F_{8,4340} = 2.9\ p < 0.01,\ \alpha : F_{8,4340} = 10.7\ p < 0.0001$).

5.4. Physical environment

The physical training data set size was determined based on the results of the simulation analysis and accounting for the fact that the area in the physical implementation is smaller than in the simulation

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

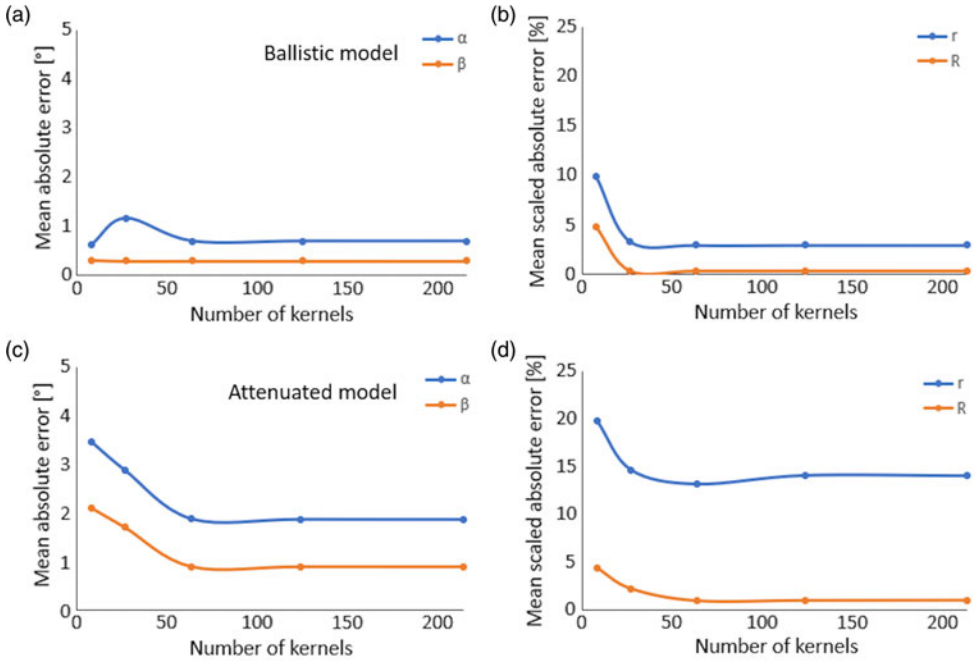

Fig. 7. Average estimation error of the task and meta parameters as a function of the number of kernels, presented parameters: r: ball landing radius; R: endpoint radius; α: ball landing horizontal angle; β: endpoint horizontal angle. Ballistic model: A and B, Attenuated model: C and D.

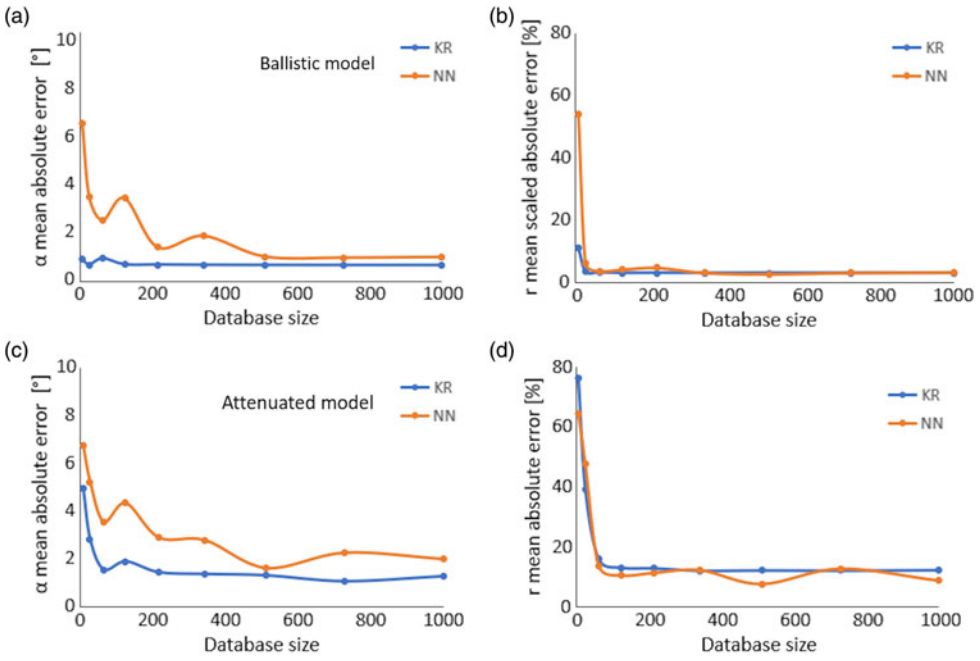

Fig. 8. Average estimation error as a function of data set size, r: ball landing radius; R: endpoint radius; α: ball landing horizontal angle; β: endpoint horizontal angle. Kernel estimation (KR, blue) and DNN (NN, orange). Ballistic model: A and B, Attenuated model: C and D.

due to room size limitations. The training data set size was set to 45 parameter tuples, created based on a full factorial design for three values of h (400 mm, 900 mm, 1400 mm) and R (500 mm, 600 mm, 700 mm) and five values for $\beta(-10°, -5°, 0°, 5°, 10°)$. Results were compared for the mapping learned with kernel estimation based on the physical training data set, and the ballistic and attenuated

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

Table I. Estimation errors when training with different training sets (Radius – Mean scaled absolute error ($MSAE_r$), Angle – Mean absolute error (MAE_α)).

Training dataset (size)	**$MSAE_r$ (SD) (%)**	**MAE_α (SD) (deg)**
Physical (45)	32 (0.07)	4.00 (2.72)
Ballistic model (125)	3 (0.00)	0.09 (.05)
Attenuated model (125)	2 (0.01)	2.41 (0.46)

simulation training data sets with 125 tuples (Table I). The number of kernels was 27, and the average error calculation for the three mappings was performed using the 12-point test set generated in a physical experiment.

The average errors for both task parameters using the mappings based on the simulation models were significantly lower than when using the mapping based on the physical data set (radius – $MSAE_r$: $p < 0.0001$ for both simulation models, angle – MAE_α: Physical-Ballistic $p < 0.0001$, Physical-Attenuated $p < 0.05$). There was no difference between training with each of the two simulation models for the radial error, but the angular error was larger for training with the attenuated model ($p < 0.01$). Results were similar when larger simulated data sets were used.

6. Discussion

The PCA and LLE analysis of the data from both simulation models of the soft ball throwing task indicated that the complexity of the problem was moderate, that is, that a moderate number of kernels and similarly a moderate number of layers, were required for estimating the task and meta parameter manifold. This was corroborated by the analysis of the number of required kernels and the analysis of network hyper-parameters. The additional nonlinear constrains in the attenuated model are in line with the larger estimation errors for this model using both methods (kernel estimation and DNNs). Not surprisingly, the estimation errors decreased rapidly as the size of the training data set increased for both kernel estimation and DNNs. However, kernel estimation attained good results even with small data sets. While for small data sets, kernel estimation outperformed the DNNs; for the larger data sets, their performance was similar.

In a physical environment, generating a large training data set is very time-consuming. Accordingly, the training data set generated in the physical environment was small. Therefore, the mapping learned based on this data set was not accurate, and the estimation errors using the mapping were large. Using mappings learned with training data sets generated by the simulation models produced considerably lower errors even though the simulation models were based on simplifying assumptions. The mapping based on the data generated using the ballistic model led to very low errors in both landing distance and angle. The mapping based on the attenuated model led to low errors in landing distance, but it did not capture the landing angle well, perhaps since the modeled reduction in the velocity as a function of β was too steep.

The low errors attained in both simulations and in the physical environment when training with simulated data, validate the use of the *a priori* learned mapping between task and DMP parameters for runtime motion adaptation. The motion adapted based on the mapping is fully predictable, and task performance is robust. For robotic applications that require high assurances of runtime behavior without forsaking runtime adaptability, such traits are important. Mapping the task to meta parameters facilitates a structured adaptation to anticipated changes in the environment (known unknowns). For examples, in the presented use case, changes to the required landing point. It is not equipped to handle un-anticipated changes (unknown unknowns). For example, in the presented use case, we did not prepare for changes in the elasticity of the ball. The DMP formulation itself does provide some flexibility for dealing with changes in the environment, yet without parameter adaptation, the ability to handle such changes is limited and the runtime operation may be hindered.

In many applications, the task enforces requirements on the path, for example, placing a ball in the cup,[48] pouring or carrying liquid,[10,22] or flipping a pancake.[49,50] In such cases, the number of task parameters is large; therefore, kernel estimation may not be suitable and using a DNN may be required. Seker et al.[3] suggest an alternative primitive representation based on the conditional neural processes DNN architecture. Augmenting such a representation with a learning and generation system facilitates encoding complex temporal multi-modal sensorimotor relations in connection with

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press

complex task constraints. Indeed, such a framework requires large learning data sets. However, for applications with complex task requirements along the motion path, for which the relationship with the DMP parameters is not straight forward, such a representation may be advantageous. In many other applications, requirements are enforced by the task only on the final configuration, for example, pick and place tasks,[20] hitting a baseball,[51] dart throwing,[12] tennis playing,[16,52] or drumming.[53,54] In such cases, the number of task parameters is small; therefore, forming the mapping of task to meta parameters may be advantageous, especially when using kernel estimation which does not require much training data.

7. Conclusions

The research presents a hierarchical method for adaptation of DMP parameters during runtime based on an *a priori* learned map of the task and DMP meta parameter manifold. Two methods for learning the mapping are compared, kernel estimation and DNNs. For facilitating faster learning convergence and correct emphasis on reducing errors of the more influential parameters, two methods are applied for analyzing the complexity of the task and meta parameter manifold: PCA and LLE. PCA assists in determining the regression equations and the number of required kernels and for determining values of the grid search for the hyper-parameters of the DNN. LLE assists in gaining a deeper understanding of model complexity.

Acknowledgment

Research supported by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev. The authors thank Mr. Noam Peles for his assistance in setting up the physical environment.

References

1. S. F. Giszter, E. Loeb, F. A. Mussa-Ivaldi and E. Bizzi, "Repeatable spatial maps of a few force and joint torque patterns elicited by microstimulation applied throughout the lumbar spinal cord of the spinal frog. Hum," *Mov. Sci.* **19**, 597–626 (2000).
2. A. Billard, S. Calinon, R. Dillmann and S. Schaal, "Robot Programming by Demonstration," **In**: *Handbook of Robotics* (B. Siciliano, O. Khatib, eds.) (Springer, Berlin, 2008) chapter 59, pp. 1371–1394.
3. M. Y. Seker, M. Imre, J. Piater and E. Ugur, "Conditional Neural Movement Primitives," *Robotics: Science and Systems (RSS) Conference*, Freiburg, Germany (2019). https://doi.org/10.15607/RSS.2019.XV.071.
4. B. D. Argall, S. Chernova, M. Veloso and B. Browning, "A survey of robot learning from demonstration," *Rob. Auton. Syst.* **57**, 469–483 (2009).
5. P. Kormushev, S. Calinon and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenge," *Robotics.* **2**(3), 122–148 (2013).
6. J. Kober, J. A. Bagnell and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Rob. Res.* **32**(11), 1238–1274 (2013).
7. P. Pastor, M. Kalakrishnan, F. Meier, F. Stulp, J. Buchli, E. Theodorou and S. Schaal, "From dynamic movement primitives to associative skill memories," *Rob. Auton. Syst.* **61**(4), 351–361 (2013).
8. B. C. da Silva, G. Baldassarre, G. Konisidaris and A. Barto, "Learning Parameterized Motor Skills on a Humanoid Robot," *IEEE International Conference on Robotics and Automation*, Hong Kong, China (2014) pp. 5239–5244.
9. K. Muelling, J. Kober, O. Kroemer and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *Int. J. Rob. Res.* **32**(3), 263–279 (2013).
10. B. Nemec and A. Ude, "Action sequencing using dynamic movement primitives", *Robotica*. **30**, 837–846 (2011).
11. A. McGovern and A. G. Barto, "Automatic Discovery of Subgoals in Reinforcement Learning Using Diverse Density," *Proceedings of the Eighteenth International Conference on Machine Learning*, Williamstown, MA, USA (2001).
12. J. Kober, E. Oztop and J. Peters, "Reinforcement Learning to Adjust Robot Movements to New Situations," *International Joint Conference on Artificial Intelligence*, Barcelona, Spain (2011) pp. 2650–2655.
13. F. Stulp, G. Raiola, A. Hoarau1, S. Ivaldi and O. Sigaud, "Learning Compact Parameterized Skills with a Single Regression", *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Atlanta, USA (2013) pp. 417–422.
14. A. J. Ijspeert, J. Nakanishi and S. Schaal, "Movement Imitation with Nonlinear Dynamical Systems in Humanoid Robots," *IEEE International Conference on Robotics and Automation*, vol. 2, Washington, DC, USA (2002) pp. 1398–1403.
15. A. J. Ijspeert, J. Nakanishi and S. Schaal, "Learning Attractor Landscapes for Learning Motor Primitives," **In**: *Advances in Neural Information Processing Systems* (S. Becker, S. Thrun and K. Obermayer, eds.), vol. 15 (Vancouver, Canada, 2003) pp. 1523–1530.

16. A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Comput.* **25**, 328–373 (2013).
17. H. Hoffmann, P. Pastor, D. H. Park and S. Schaal, "Biologically-inspired Dynamical Systems for Movement Generation: Automatic Real-time Goal Adaptation and Obstacle Avoidance," *IEEE International Conference on Robotics and Automation*, vol. 2, Kobe, Japan (2009), pp. 1–7.
18. P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou and S. Schaal, "Skill Learning and Task Outcome Prediction for Manipulation," *IEEE International Conference on Robotics and Automation*, Shanghai, China (2011) pp. 9–13.
19. M. Tamošiunaite, T. Asfour and F. Worgotter, "Learning to reach by reinforcement learning using receptive field based function approximation with continuous actions," *Biol. Cybern.* **100**, 249–260 (2009).
20. E. Ugur and H. Girgin, "Compliant parametric dynamic movement primitives," *Robotica*. 1–18 (2019). First view https://doi.org/10.1017/S026357471900078X.
21. H. Girgin and E. Ugur, "Associative Skill Memory Models," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Madrid, Spain (2018) pp. 6043–6048.
22. M. Tamošiunaite, B. Nemec, A. Ude and F. Worgotter, "Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives," *Rob. Auton. Syst.* **59**, 910–922 (2011).
23. F. Stulp and S. Schaal, "Hierarchical Reinforcement Learning with Movement Primitives," *IEEE-RAS International Conference on Humanoid Robots*, Bled, Slovenia (2011) pp. 231–238.
24. M. Kalakrishnan, P. Pastor, L. Righetti and S. Schaal, "Learning Objective Functions for Manipulation," *IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany (2013) pp. 1323–1328.
25. M. Deisenroth, P. Englert, J. Peters and D. Fox, "Multi-task policy search for robotics," *arXiv preprint*, arXiv:1307.0813 (2013).
26. D. Eizicovits and S. Berman, "Efficient sensory-grounded grasp pose quality mapping for gripper design and online grasp planning," *Rob. Auton. Syst.* **62**, 1208–1219 (2014).
27. C. de Granville, D. Wang, J. Southerland, R. Platt, J. Andrew and H. Fagg, "Grasping Affordances: Learning to Connect Vision to Hand Action," **In**: *The Path to Autonomous Robots* (G. Sukhatme ed.) (Springer-Verlag Germany, 2009) pp. 1–22.
28. E. Rueckert, J. Mundo, A. Paraschos, J. Peters and G. Neumann, "Extracting Low-Dimensional Control Variables for Movement Primitives," *IEEE International Conference on Robotics and Automation*, Seattle, USA (2015) pp. 1511–1518.
29. A. Paraschos, C. Daniel, J. Peters and G. Neumann, "Probabilistic Movement Primitives," *Advances in Neural Information Processing Systems*, vol. 26, Lake Tahoe, USA (2013) pp. 2616–2624.
30. T. He, R. Kong, A. J. Holmes, M. R. Sabuncu, S. B. Eickhoff, D. Bzdok, J. Feng and B. T. T. Yeo, "Is Deep Learning Better than Kernel Estimation for Functional Connectivity Prediction of Fluid Intelligence?," *International Workshop on Pattern Recognition in Neuroimaging*, Singapore (2018) pp. 1–4.
31. N. C. Mutono, G. Anthony Waititu and W. A. Kiberia, "Feed forward neural network versus kernel estimation a case of body mass index and body dimensions," *Am. J. Theor. Appl. Stat.* **5**(4), 180–185 (2016).
32. R. Basri and D. W. Jacobs, "Efficient Representation of Low-Dimensional Manifolds Using Deep Networks," *International Conference on Learning Representations*, Toulon, France (2017).
33. I. Jolliffe, *Principal Component Analysis* (Springer-Verlag, New York, 1986).
34. S. Wold, K. Esbensen and P. Geladi, "Principal component analysis," *Chemom. Intell. Lab. Syst.* **2**(1–3), 37–52 (1987).
35. S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*. **290**(5500), 2323–2326 (2000).
36. J. Wang, *Geometric Structure of High-Dimensional Data and Dimensionality Reduction* (Springer-Verlag, Berlin, 2012) Chapter 10.
37. K. Liu, A. Weissenfeld and J. Ostermann, "Parameterization of mouth images by LLE and PCA for image-based facial animation," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Toulouse, France (2006) pp. 461–464.
38. Y. Cohen and S. Berman, "Tight Dynamic Movement Primitives for Complex Trajectory Generation," *IEEE International Conference on Systems, Man, and Cybernetics*, Manchester, UK (2013) pp. 2402–2407.
39. B., Schultz, *Geometrical Method of Mathematical Physics* (Cambridge: Cambridge University Press, 1999).
40. J. Racine and Q. Li, "Nonparametric estimation of regression functions with both categorical and continuous data," *J. Economet.* **119**(1), 99–130 (2004).
41. H. Shinamazaki and S. Shinomoto, "Kernel bandwidth optimization in spike rate estimation," *J. Comput. Neurosci.* **29**(1–2), 171–182 (2010).
42. V. Sze, Y. H. Chen, T. J. Yang and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, **105**(12), 2295–2329 (2017).
43. S. Ruder, "An overview of gradient descent optimization algorithms", *arXiv preprint* arXiv:1609.04747 (2016).
44. X. Glorot, A. Bordes and Y. Bengio, "Deep Sparse Rectifier Neural Networks," *International Conference on Artificial Intelligence and Statistics*, Ft. Lauderdale, USA (2011) 315–323.
45. J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.* **13**, 281–305 (2012).

46. P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB* (Springer-Verlag, Germany, 2011).
47. L. Prechelt, "Early Stopping-but When? **In**: *Neural Networks: Tricks of the Trade* (G. Montavon, G. B. Orr and K. Müller, eds.) (Springer-Verlag, Germany 1998) pp. 55–69.
48. J. Kober and J. Peters, "Policy Search for Motor Primitives in Robotics," *Advances in Neural Information Processing Systems*, vol. 22, Vancouver, Canada (2009) pp. 171–203.
49. P. Kormushev, S. Calinon and D. G. Caldwell, "Approaches for Learning Human-like Motor Skills which Require Variable Stiffness During Execution", *IEEE International Conference on Humanoid Robots*, Santa Monica, USA (2010).
50. P. Kormushev, S. Calinon and D. G. Caldwell, "Robot Motor Skill Coordination with EM-based Reinforcement Learning," *IEEE/RSJ International Conference Intelligent Robots and Systems, on Intelligent Robots and Systems*, *Taipei, Taiwan* (2010) pp. 3232–3237.
51. J. Peters and S. Schaal, "Policy Gradient Methods for Robotics", *IEEE/RSJ International Conference Intelligent Robots and Systems, Beijing, China* (2006) pp. 2219–2225.
52. S. Schaal, P. Mohajerian and A. Ijspeert, "Dynamics Systems vs. Optimal Control — A Unifying View," **In**: *Progress in Brain Research* (P. Cisek, T. Drew and J. F. Kalaska eds.), vol. 165 (2007) pp. 425–445.
53. S. Schaal, S. Kotosaka and D. Sternad, "Nonlinear Dynamical Systems as Movement Primitives," *IEEE International Conference Humanoid Robotics*, *Santa Monica, USA* (2000).
54. D. Pongas, A. Billard and S. Schaal, "Rapid Synchronization and Accurate Phase-Locking of Rhythmic Motor Primitives," *IEEE/RSJ International Conference Intelligent Robots and Systems*, Edmonton, Canada (2005) pp. 2911–2916.

https://doi.org/10.1017/S0263574720001186 Published online by Cambridge University Press