Dimensionality Reduction of Movement Primitives in Parameter Space

Dimensionalitätsreduktion von Bewegungsprimitiven im Parameterraum
Bachelor thesis in the field of study “Computational Engineering” by Jonas Elias Stadtmüller
Date of submission: 18.02.2020

1. Review: Prof. Jan Peters, Ph.D.
2. Review: Samuele Tosatto
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

field of study:
Computational Engineering

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Jonas Elias Stadtmüller, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 18.02.2020

J. Stadtmüller

Abstract

Although there have been promising advancements in recent years, Reinforcement Learning is not yet directly applicable to robotics, since the dimensionality of robot movements is high, so the computation time of the RL algorithms becomes prohibitive. Having a learning system on real robotics will enable industrial robotics to be easily reprogrammable to new tasks, and at the same time avoid the need for training in simulated environments. Engineering expenses to design the simulation can be saved and the reality gap can be overcome. We propose a new framework that works in parameter space to resolve the dimensionality of the representation. Our approach takes into consideration the similarity between movements instead of the redundancy in the kinematic system. Additionally, the proposed method is independent of the number of chosen features. We empirically show that the framework we introduce works efficiently for both complex human movement as well as in a simpler robotic scenario.

Zusammenfassung

Obwohl es in den letzten Jahren vielversprechende Fortschritte gegeben hat, ist Reinforcement Learning noch nicht direkt auf die Robotik anwendbar, da die Dimensionalität der Roboterbewegungen hoch ist, so dass die Berechnungszeit der RL-Algorithmen untragbar wird. Mit einem lernenden System für die reale Robotik kann die industrielle Robotik leicht auf neue Aufgaben umprogrammiert werden, gleichzeitig wird die Notwendigkeit des Trainings in simulierten Umgebungen vermieden. Die Kosten für die Entwicklung der Simulation können eingespart und die Lücke zur Realisierung überwunden werden. Wir schlagen eine neue Herangehensweise vor, die im Parameterraum arbeitet, um die übermäßige Dimensionalität der Darstellung zu vermeiden. Unser Ansatz berücksichtigt die Ähnlichkeit zwischen den Bewegungen und nicht die Redundanz im kinematischen System. Weiterhin ist die vorgeschlagene Methode unabhängig von der Anzahl der gewählten Features. Wir zeigen empirisch, dass unsere Herangehensweise sowohl bei komplexen menschlichen Bewegungen als auch in einem einfacheren Roboter-Szenario effizient funktioniert.

Contents

List of Figures

1 Introduction

In recent years Reinforcement Learning (RL) has accomplished impressive results [27] in areas like video games [17] and simulated environments [14]. Reinforcement Learning is an approach to solve tasks that require decision making [28]. The framework consists of an agent interacting with an environment in the form of actions, receiving direct feedback through a reward signal.
RL in robotics has high potential, however, the direct application of RL algorithms to robotics is challenging because it usually requires the processing of high dimensional data and scarce demonstration availability. A feasible way to apply RL algorithms to robotics is by considering abstract actions rather than letting the agent directly control the robot's torques. This could lead to dangerous behavior, which could potentially damage the robot due to bad policies. Therefore, to handle this interaction a framework for separating the RL agent and the robot is needed.
For many industrial applications, such as object manipulation or assembly of items, it is enough to consider the state representation as the position of the object and the robot's possible movement as a whole action, because the environment is generally static. It does not change during action execution. The bottleneck of this approach is the action space. The representation of movement is usually high dimensional, caused by a large number of time steps and degrees of freedom for complex robot movements. Therefore, a method is needed for converting trajectories into compact actions and conversely for reconstructing movements out of compact action representations. Figure 1.1 illustrates this setup.

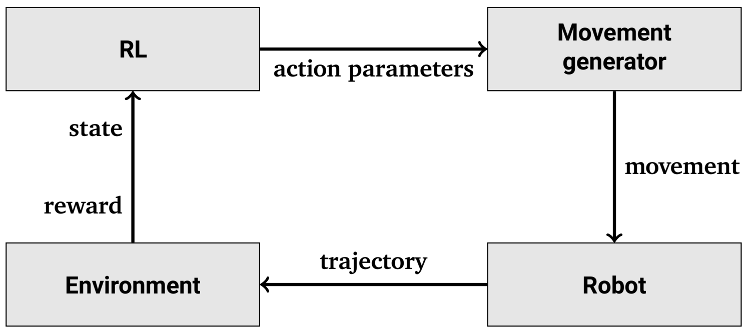

Figure 1.1: Learning framework showing the connection of robot and environment using a method to convert compact action parameters to executable robot trajectories.

It is common to represent methods of Movement Primitives to parameterize the robot's trajectories as a weight vector. A trajectory is a path that a robot will follow while executing the movement. For some applications, the dimensionality of the parameterized action-space is too extensive. Therefore, it is desirable to employ methods of dimensionality reduction in order to decrease the dimensionality of the data while simultaneously keeping most of the data's information. Many methods addressing this problem aimed to reduce the trajectories' size directly by exploiting coupling between the robot's joints.
We propose a different approach to reduce a trajectory's dimensionality by applying methods of dimensionality reduction (DR) in the parameter-space instead of in the joint-space. This approach, which we call Principal Movement Primitives, or PriMos for short may be advantageous for various settings, where a concise action space is of great importance. We demonstrate that our method enables the representation of complex trajectories using only small parameter vectors. Furthermore, we show that the proposed method allows for sampling random, smooth trajectories using only a compact covariance matrix.
As a preliminary, we provide the most essential methods and techniques we will use throughout this thesis and are therefore necessary prerequisites.

1.1 Problem Formulation

Describing the RL context more formally, we describe the state of the environment at each time-step by $S_t \in \mathbb{S}, t \in \mathbb{N}^+$, where $\mathbb{S}$ is the state-space. In every state S_t, the agent selects an action $A_t \in \mathbb{A}$, receiving an immediate reward $R_{t+1} \in \mathbb{R}$ [28]. It is important to note that both $\mathbb{S}$ and $\mathbb{A}$ can embody discrete or continuous sets of states and actions. Similar to [28], we define $p(S_{t+1}|S_t, A_t)$ to be the transition probability.
The Markov property asserts that the transition probability only depends on the last state S_t and last action A_t and is therefore independent of every state and action before those. The RL setting is called a Markov Decision Process (MDP). The action selection is made according to a policy, the probability distribution of action A_t, given the current state is S_t. We denote this probability distribution with $\pi(A_t|S_t)$. In the discounted infinite-horizon case, the object of the RL agent is to maximize the expected return

$$J_\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right], \tag{1.1}$$

where $\gamma \in [0, 1)$ is the discount factor determining the emphasis on earlier obtained rewards.
A wide range of algorithms has been proposed in the literature to maximize the objective in 1.1, for example, policy gradient approaches [29, 3, 7] or value-function based methods [32]. However, to enhance sample efficiency, the RL-agent ideally relies only on low-dimensional representations of states and actions. In every time-step, the agent selects an action α in its low-dimensional action-space according to the current policy $\pi(\alpha|S_t)$. We assume α to be a parameter vector that is then passed to some black box that handles the movement reconstruction. According to some learned model, it returns a trajectory, that may directly be passed to the robot for it to interact with the environment. We use parameterization to represent the trajectories because it enables to account for movements of varying length as it normalizes the time axis to the space of $[0, 1]$. For parameterization of the robot's trajectories we use Movement Primitives [11]. In recent years new extensions and variations have been made to this approach [19, 26]. In order to learn Movement Primitives, a set of weights has to be found, so the feature representation of the movement does approximate the original trajectory best. To find these weights it is common to use ridge regression.
As stated earlier it is necessary to reduce the dimensionality of the trajectories or movements. There are numerous techniques and frameworks that address this problem [25, 30]. We chose to use Principal Component Analysis [10] to achieve the reduction since

it is a well known and popular method. Recent work focused on reducing the data size before parameterizing the data. We will discuss this in Section 1.5. Considering the overall background of robotics, it is common to collect trajectories in joint space. A trajectory in joint-space describes the angular displacement of every joint of the robot for every time-step. It is a common approach to perform the DR directly on every trajectory in the dataset.

1.2 Ridge Regression

Let the feature matrix $\boldsymbol{\Phi}$ be a representation of $\boldsymbol{y_i}, i \in 1, \dots, n$ that can be obtained by multiplying it with a weight vector $\boldsymbol{w}$. the vector $\boldsymbol{y}$ could, for example, be a robot's trajectory. A common evaluation of this models accuracy is the mean squared error

$$\frac{1}{n}\Big(\boldsymbol{\Phi w} - \boldsymbol{y}\Big)^\intercal \Big(\boldsymbol{\Phi w} - \boldsymbol{y}\Big).$$

Since the error should be as small as possible, the weights have to be chosen to minimize the error. This is known as the standard Linear Regression problem

$$\boldsymbol{w}^* = \text{arg min}_{\boldsymbol{w}} \frac{1}{n}\Big(\boldsymbol{\Phi w} - \boldsymbol{y}\Big)^\intercal \Big(\boldsymbol{\Phi w} - \boldsymbol{y}\Big). \tag{1.2}$$

Incautious usage of the problems analytical solution

$$\boldsymbol{w}^* = \Big(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi}\Big)^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{y}$$

can lead to big values for the elements of $\boldsymbol{w}$ with an alternating sign and thus interpolate the data points. This is known as over-fitting and leads to a model that lacks generality as it interpolates the training data.
To avoid this shortcoming ridge regression [9] was proposed as a method to penalize parameter values of great magnitude.
Formally 1.2 is extended by

$$\boldsymbol{w}^*_{\text{ridge}} = \text{arg min}_{\boldsymbol{w}} \frac{1}{n}\Big(\boldsymbol{\Phi w} - \boldsymbol{y}\Big)^\intercal \Big(\boldsymbol{\Phi w} - \boldsymbol{y}\Big) + \frac{\lambda}{d}\boldsymbol{w}^\intercal \boldsymbol{w}, \tag{1.3}$$

where we call $\frac{\lambda}{d}\boldsymbol{w}^\intercal \boldsymbol{w}$ the penalization term with its corresponding ridge parameter λ. The coefficient d denotes the number of dimensions for every trajectory $\boldsymbol{y}$ and $\mathbb{I}$ is the identity matrix of appropriate dimensions. This new modified problem yields the solution

$$\boldsymbol{w}^*_{\text{ridge}} = \Big(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi} + \frac{n}{d}\lambda \mathbb{I}\Big)^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{y}. \tag{1.4}$$

We will use this solution throughout this work. The solution's derivation can be found in the appendix.

1.3 Probabilistic Movement Primitives

The core idea of the Movement Primitive approach [11] is to represent a complex trajectory with a set of weighted basis functions. Paraschos et al. proposed a framework to parameterize trajectories by a number of weights, each corresponding to a Gaussian basis function [19]. By doing so they were able to introduce several advantageous characteristics like temporal modulation, perturbation of via-points and combination of distinct Movement Primitives. These bring practical properties for various tasks in robotics [20] and have been frequently used for robotics tasks [13, 16].
In order for the ProMP framework to be applicable to datasets with varying numbers of time steps, the definition of feature matrices is needed. For this let T be the number of time steps for every dimension and trajectory sampled and f the number of basis functions to use. We define the feature matrix according to the context of Probabilistic Movement Primitives, where movements are expressed as a summation of Gaussian basis functions

$$\varphi_i(z) = \frac{b_i(z)}{\sum_{j=1}^{n} b_j(z)},\ i \in \{1, \dots, f\}, \tag{1.5}$$

$$b_i(z) = \exp\Big(- \frac{(z - c_i)^2}{2h} \Big),\ i \in \{1, \dots, \mathrm{f}\}. \tag{1.6}$$

The Gaussian basis functions are centered at c_i and have a fixed variance of h, also called the bandwidth. In order to compactly represent our derivations in closed form fashion, the basis functions are stored in a matrix, called the feature matrix. For one dimension it is given by

$$\boldsymbol{\varphi} = \begin{bmatrix} \varphi_1(t_1) & \dots & \varphi_{\mathrm{f}}(t_1) \\ \vdots & \ddots & \vdots \\ \varphi_1(t_{\mathrm{T}}) & \dots & \varphi_{\mathrm{f}}(t_{\mathrm{T}}) \end{bmatrix}, \tag{1.7}$$

It should be mentioned that the time dimension was normalized to the interval $[0, 1]$ as shown in Equation 1.5 and Equation 1.6. We refer to this as the phase space with z being the corresponding phase variable. This holds for the feature matrices as well, despite being indexed by t_i. This is to emphasize the time dependency. Furthermore, we address the placing of the basis functions' centers c_i similar to [19] so that they are placed equidistantly in $[-2h, (1+2h)]$ to smooth the resulting combination of basis functions. Finally, we set the bandwidth $h = \frac{1}{\mathrm{f}\sqrt{2\mathrm{f}}}$.

For the straightforward application of regression techniques, we extended the definition of our feature matrix 1.7 for the multidimensional context. For this, we constructed a block diagonal matrix as follows:

$$\boldsymbol{\Phi} = \begin{bmatrix} \varphi & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \varphi \end{bmatrix}.$$

This allows for a compact matrix formulation in the multidimensional case. For the sake of simplicity, we refer to this extended matrix as the feature matrix. Nevertheless, it is still possible to perform the calculations for every dimension separately using 1.7. Especially for trajectories with a high number of degrees of freedom or time-steps, this is the more efficient method with respect to both space and time complexity. This only holds for the matrix computation not being optimized for computation on the GPU.
With the formulation of feature matrices at hand, it is possible to formalize the method of calculating every trajectories corresponding weight. In the ProMP approach a set of weights $\boldsymbol{w}^*$ is found for every trajectory $\boldsymbol{y} \in \mathbb{R}^{T \times d}$ in the dataset that solves

$$\boldsymbol{w}^* = \text{arg min}_{\boldsymbol{w}} \frac{1}{T}\Big(\boldsymbol{\Phi w} - \boldsymbol{y}\Big)^{\mathsf{T}}\Big(\boldsymbol{\Phi w} - \boldsymbol{y}\Big) + \frac{\lambda}{d}\boldsymbol{w}^{\mathsf{T}}\boldsymbol{w},$$

where λ is again the ridge parameter in the setting of ridge regression and d is the number of dimensions or degrees of freedom in the context of robotics. The solution of this equation is given by

$$\boldsymbol{w}^* = \Big(\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\Phi} + \frac{T}{d}\lambda\mathbb{I}\Big)^{-1}\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{y}.$$

Having obtained this vector representation, it can be stored as a compact representation of the trajectory or processed in another way. Furthermore, the movement can be reconstructed easily using

$$\hat{\boldsymbol{y}} = \boldsymbol{\Phi w}.$$

1.4 Principal Component Analysis

Principal Component Analysis [23, 10, 12] intends to find a series of linearly independent eigenvectors that form an orthogonal base on the data. It can be used to reduce the data's

dimensionality by selecting the c eigenvectors that contain the most variance.
Formally the initial step is to perform an eigendecomposition on the dataset's covariance matrix.
The covariance is estimated by

$$\boldsymbol{\Sigma} = \frac{1}{n-1}(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^{\intercal},$$

where $\mu = \sum \boldsymbol{X_i}$ is the empirical mean over the respective axis. Thereafter we perform eigendecomposition. This is equivalent to finding $\boldsymbol{E}$ and $\boldsymbol{\Lambda}$ so that

$$\boldsymbol{\Sigma} = \boldsymbol{E}\boldsymbol{\Lambda}\boldsymbol{E}^{-1}.$$

The square matrix $\boldsymbol{E}$ holds the eigenvectors of $\boldsymbol{\Sigma}$ in its columns. The diagonal matrix $\boldsymbol{\Lambda}$ contains the corresponding eigenvalues as its diagonal elements.
Having obtained the matrices $\boldsymbol{\Lambda}$ and $\boldsymbol{E}$, the eigenvectors have to be sorted by the absolute magnitude of their respective eigenvalue in descending order. In the following, it is assumed the sorted matrix $\hat{\boldsymbol{E}}$ contains the eigenvectors row-wise.
By definition, the first eigenvectors capture most of the initial data's information. Therefore it is sensible to only use a fixed number c of them to reduce the dimensionality. Let $\boldsymbol{C}$ denote the submatrix with only c components of $\hat{\boldsymbol{E}}$. It is possible to project our data onto a lower-dimensional space using

$$\hat{\boldsymbol{X}} = \boldsymbol{C}\boldsymbol{X}.$$

Similarly, the reconstructed data can be calculated by

$$\boldsymbol{X_r} = \boldsymbol{C}^{\intercal}\hat{\boldsymbol{X}}.$$

By using the subscript notation, we emphasize the fact that the reconstruction will only approximate the original data.

1.5 Related Work

Compactly representing movement primitives is a prevalent topic of several publications. The literature is not limited to the context of robotics. In order to efficiently predictively model human motion, [4] proposed the representation of Dynamic Movement Primitives (DMPs) [26] by fitting under complete Deep Autoencoders [2] on movement data. They evaluated their algorithm on high-dimensional humanoid movements gathered by motion capturing. They achieve better generalization of the DMPs because of the nonlinear architecture of the Autoencoders they make use of but lack a method to sample random movements.
To improve the computational efficiency of ProMPs, [5] introduced a method of probabilistic dimensionality reduction based on an Expectation-Maximization like algorithm [18]. The proposed framework enabled them to train the Relative Entropy Policy Search algorithm [24] more efficiently on a robot system. They did, however, demonstrate a way of generating trajectories.
Both of the preceding publications perform the reduction in the joint space, compared to the reduction in parameter-space that our method employs.
Similar to PriMos, [21] used PCA in order to generate basis functions of movement primitives. These basis functions are then turned into trajectories by optimizing a cost function. They used this framework to efficiently generate approximations of humanoid demonstrations. They compared these generated movements to those generated by B-Splines [6] and demonstrated that their method generated more natural-looking movements using less computational time. However, they only evaluated their approach on the generation of a single movement, with only 4 degrees of freedom.
The above publications focused on movements like walking that do not require fine motor skills. Tasks that do require these skills are for example humanoid hand movements, which were studied by [1], who proposed a method to generalize grasping movements for multi-fingered humanoid hands. They performed PCA on the joint-space data to extract a low-dimensional grasp-space. Subsequently, they used DMPs to parameterize the grasp-space hand movements. Additionally, they provided a contact-warping algorithm that enabled them to perform hand movements with different objects. These objects only had to be comparable to those shown to the robot in the human demonstrations. Their method proved to be robust to changes in the environment. Reducing the dimensionality of the data on the joint-space, however, will generally result in a higher number of parameters used, to fully describe the trajectories, compared to our proposed framework.
For Robots to be incorporated into certain real-life settings, they need to be able to learn new movements, when needed. In order to achieve this, [15] combined Gaussian Processes

(GPs) and DMPs to create a strategy for active incremental learning that a robot can use to decide when to request a human demonstration in order to adapt the robot's model. They provided an approach to train DMPs on trajectories encoded by GPs, which enables the DMPs to generalize to different types of movements. With increasing dimensionality of the demonstrations, the GPs' computational demand may become too large, as [15] do not employ dimensionality reduction.

2 Principal Movement Primitives

Instead of reducing the dimensionality of the data directly in the joint space and subsequently computing the ProMP representation on the reduced data set, we perform the reduction step after the ProMP parameterization. Consequently, we achieve a dimensionality reduction on the movement or parameter space. Furthermore, we extend the approach by additionally minimizing the squared error

$$\boldsymbol{a}^* = \text{arg min}_{\boldsymbol{\alpha}} \frac{1}{T}\Big(\boldsymbol{\Phi}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{M}^\intercal\boldsymbol{\alpha} - \boldsymbol{y}\Big)^\intercal \Big(\boldsymbol{\Phi}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{M}^\intercal\boldsymbol{\alpha} - \boldsymbol{y}\Big) + \frac{\lambda}{c}\boldsymbol{\alpha}^\intercal\boldsymbol{\alpha}. \tag{2.1}$$

This enables us to find a parameter vector $\boldsymbol{\alpha}$ describing the original trajectory. It weighs the dimensionality-reduced version of our data's feature representation so that it reconstructs our original data better.
The following image shows the seminal steps in both approaches and highlights the differences by sketching the basic sequence of actions.

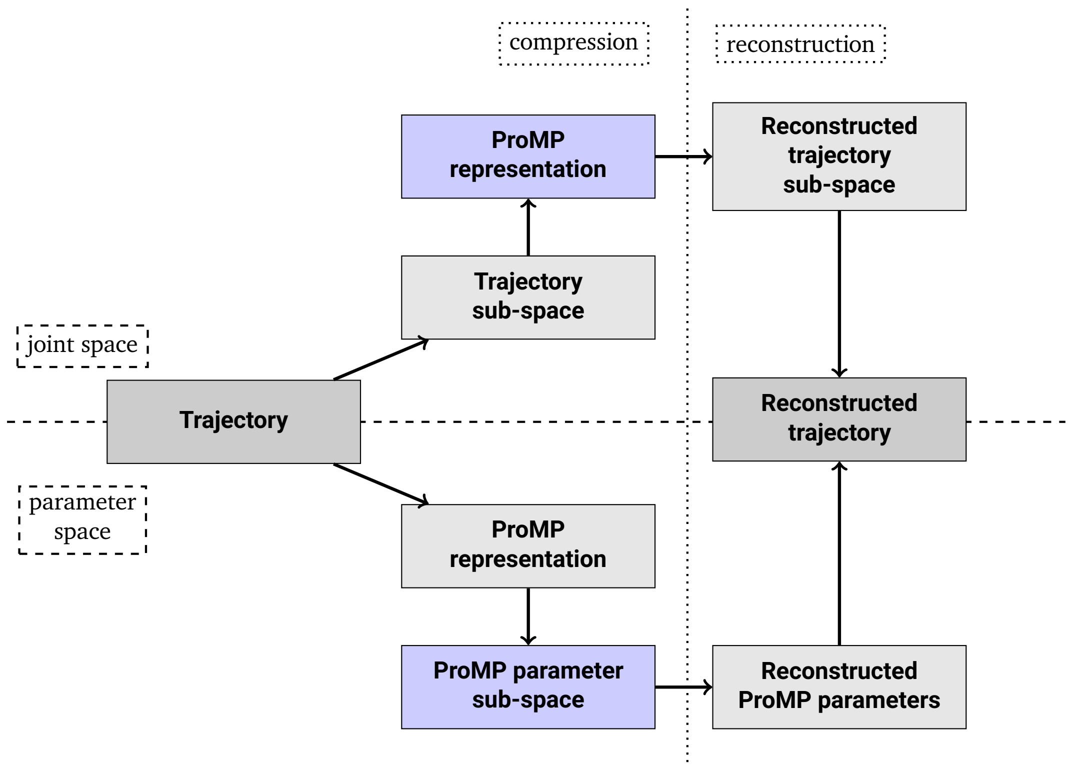

Figure 2.1: Comparison of both compression and reconstruction steps for parameter-space and joint-space methods. The blue node resembles the representation of lowest dimensionality.

Both approaches will have to start with the same trajectory, but the following step differs. Whereas the joint-space method projects the trajectory onto a subspace using the process outlined in Section 2.1, the movement-space approach transforms the trajectory to the ProMP encoded movement space. The latter method then reduces the resulting movement parameters, to achieve the highest rate of compression, the former transforms the subspace trajectory to the movement space.

2.1 Joint Space PCA

Commonly PCA is performed directly on the gathered data. In the context of robotics, these are usually trajectories in joint space of the form $\boldsymbol{Y_i} \in \mathbb{R}^{T \times d}$, with T samples per joint d_j. It makes sense to assume coupling between certain joints, so the intention of joint-space reduction is to use this relation in order to represent the trajectory in a subspace of a lower dimension. Therefore the reduction is performed on the dimensions to retrieve the compressed joint space $\hat{\boldsymbol{Y}} \in \mathbb{R}^{T \times c}$, where c is the selected number of components or eigenvectors to use for the PCA. The PCA algorithm can be applied to the whole dataset $\mathbb{D}$ at once, by concatenating all trajectories like

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{Y}_1 \\ \vdots \\ \boldsymbol{Y}_\mathrm{n} \end{bmatrix}.$$

With the resulting reduced data set the application of the ProMPs framework is obvious and the result for every trajectory $\boldsymbol{Y_i} \in \mathbb{D}$ is the parameter vector $\boldsymbol{w} \in \mathbb{R}^{c \times f}$, where f is the number of basis functions to represent the movement with.

2.2 Movement Space PCA

Instead of looking for similarities in the joints of the trajectory, as in Section 2.1, it is possible to examine the data for resemblance in its movements. Since ProMPs transform a trajectory into the movement space, PCA can thus be employed to ProMPs to exploit the coupling of movements. The method of applying PCA on ProMPs is straightforward. For every trajectory $\boldsymbol{Y_i} \in \mathbb{D}, i \in \{1, \dots, \mathrm{n}\}$ a ProMP is fit on $\boldsymbol{Y_i}$, resulting in a parameter vector $\boldsymbol{w_i}$. These can be combined in a matrix

$$\boldsymbol{\Omega} = \begin{bmatrix} \boldsymbol{w}_1^\intercal \\ \vdots \\ \boldsymbol{w}_\mathrm{n}^\intercal \end{bmatrix}.$$

The steps in Section 1.4 are performed on this matrix. After being fit, the dimensionality reduction can be achieved as $\boldsymbol{\alpha_i} = \boldsymbol{E}\boldsymbol{w_i}$,$\boldsymbol{\alpha_i} \in \mathbb{R}^{c \times 1}$, where $\boldsymbol{E}$ is the eigenvector matrix, satisfying the conventions detailed in Section 1.4.

2.3 Scaling The Projection matrix

To account for each eigenvectors contribution to the error we require the data's variance they contribute. This is proportional to the square root of their respective eigenvalue. This represents the eigenvector's standard deviation

$$\hat{\boldsymbol{\lambda}} = \sqrt{\boldsymbol{\lambda}}.$$

Since both $\hat{\boldsymbol{\lambda}}$ and $\boldsymbol{\lambda}$ are column vectors we define the square root element-wise. Furthermore, we determine the mean across all n parameter vectors. Having calculated the $\boldsymbol{\Omega}$ matrix we can simply take the mean over the first axis, which we will refer to by $\boldsymbol{\mu}_{\boldsymbol{\Omega}}$. As with the joint-space PCA, we attain the initial projection matrix $\boldsymbol{E}$ and can immediately formulate the modified projection matrix

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{E}_1 \hat{\lambda}_1 \\ \vdots \\ \boldsymbol{E}_c \hat{\lambda} \end{bmatrix}. \tag{2.2}$$

We note that this scaling is not necessary for achieving an accurate result but leads to parameter values of smaller magnitude and may thus come with advantageous numerical stability.

2.4 Derivation of solution

We can calculate our reconstructed parameterized trajectory as

$$\hat{\boldsymbol{y}} = \boldsymbol{\Phi}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{M}^{\intercal}\boldsymbol{\alpha}. \tag{2.3}$$

Since we want to achieve an accurate approximation, it is reasonable to minimize the mean squared error. Thus we define the loss as

$$\mathcal{L} = \frac{1}{T}\Big(\boldsymbol{\Phi}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{M}^{\intercal}\boldsymbol{\alpha} - \boldsymbol{y}\Big)^{\intercal}\Big(\boldsymbol{\Phi}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{M}^{\intercal}\boldsymbol{\alpha} - \boldsymbol{y}\Big).$$

Since it is desirable to avoid overfitting we use the ridge regression approach to penalize complex models. Adding the penalization term gives

$$\mathcal{L} = \frac{1}{T}\Big(\boldsymbol{\Phi}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{M}^{\intercal}\boldsymbol{\alpha} - \boldsymbol{y}\Big)^{\intercal}\Big(\boldsymbol{\Phi}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{M}^{\intercal}\boldsymbol{\alpha} - \boldsymbol{y}\Big) + \frac{\lambda}{c}\boldsymbol{\alpha}^{\intercal}\boldsymbol{\alpha}.$$

In order to derive the solution, it is easiest to first reformulate the problem using the substitutions

$$\hat{\boldsymbol{\Phi}} = \boldsymbol{\Phi}\boldsymbol{M}^\intercal$$
$$\hat{\boldsymbol{y}} = \boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}.$$

Subsequently, we can express the loss as

$$\mathcal{L} = \frac{1}{T}\left(\hat{\boldsymbol{\Phi}}\boldsymbol{\alpha} - \hat{\boldsymbol{y}}\right)^\intercal\left(\hat{\boldsymbol{\Phi}}\boldsymbol{\alpha} - \hat{\boldsymbol{y}}\right) + \frac{\lambda}{c}\boldsymbol{\alpha}^\intercal\boldsymbol{\alpha}.$$

With this modification, the problem is in the form of the ridge regression problem 1.3 and we can directly utilize the previously derived solution 1.4 to obtain

$$\boldsymbol{\alpha}^* = \left(\hat{\boldsymbol{\Phi}}^\intercal\hat{\boldsymbol{\Phi}} + \frac{T}{c}\lambda\mathbb{I}\right)^{-1}\hat{\boldsymbol{\Phi}}^\intercal\hat{\boldsymbol{y}}.$$

Resubstitution yields the problems desired solution

$$\boldsymbol{\alpha}^* = \left(\boldsymbol{M}\boldsymbol{\Phi}^\intercal\boldsymbol{\Phi}\boldsymbol{M}^\intercal + \frac{T}{c}\lambda\mathbb{I}\right)^{-1}\boldsymbol{M}\boldsymbol{\Phi}^\intercal\left(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}\right). \tag{2.4}$$

To further outline the difference between the straightforward application of PCA on the movement space and PriMos, we constructed a simple visual example of the underlying projections of data onto different sub-spaces, which can be found in Figure 2.2.

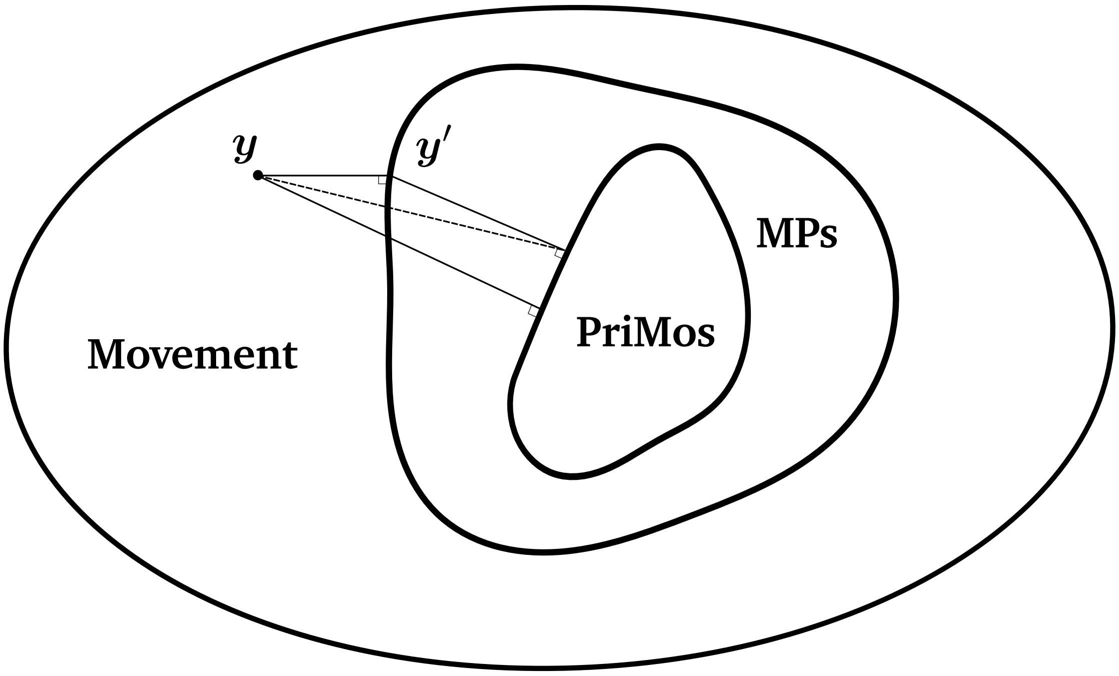

Figure 2.2: Projection of a trajectory y onto the different sub-spaces using parameter-space PCA and PriMos. The dotted line represents the linear combination of the initial projection on the movement-primitive subspace and the subsequent projection on the PriMo-subspace.

For any trajectory $\boldsymbol{y}$ in the set of all possible movements, the basic movement-space PCA first projects it onto the subspace of all movements representable by Movement Primitives. This is done using the shortest distance to this subset, which is equivalent to minimizing the mean squared error. From here on $\boldsymbol{y}'$ is transformed onto the dimensionality reduced subspace using PCA. The total distance to the PriMos-subset is, therefore, the linear combination of both previous projections.
PriMos instead seek to minimize the distance to the PriMos subspace directly, by differently weighing the reduced parameters, conditioned to minimize the distance to the original trajectory $\boldsymbol{y}$. Therefore the PriMos projection is always equally good or better than the linearly combined distances, since minimizing the mean squared error on the original trajectory $\boldsymbol{y}$ will result in the shortest distance from PriMos set to $\boldsymbol{y}$.

2.5 Principal Movement Primitives For Reinforcement Learning

In Section 2.2 we outlined the process of fitting the PCA model in movement space. We further extended this approach using the results of Section 2.4, to better fit the original trajectory. This yields a model that is able to compress the robot's trajectory into a compact vector representation, as well as reconstruct a smooth trajectory out of a given parameter vector, that is safe for the robot to execute.

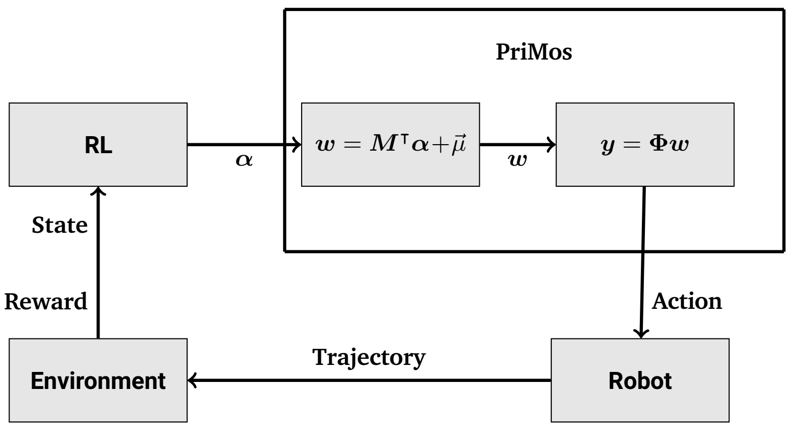
PriMos
RL
$\boldsymbol{\alpha}$
$\boldsymbol{w} = \boldsymbol{M}^\intercal \boldsymbol{\alpha} + \vec{\mu}$
$\boldsymbol{w}$
$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{w}$
State
Reward
Action
Trajectory
Environment
Robot

Figure 2.3: PriMo framework in the Reinforcement Learning context. $\boldsymbol{\alpha}$ is a parameter action the Reinforcement learning agent computes according to $\pi(\boldsymbol{\alpha}|s)$. The fit PCA object then reconstructs the movement parameters $\boldsymbol{w}$ using $\boldsymbol{M}^\intercal \boldsymbol{\alpha} + \boldsymbol{\mu}$, where $\boldsymbol{M}$ is the scaled projection matrix, introduced in Section 2.3. $\boldsymbol{w}$ can subsequently be converted into a trajectory $\boldsymbol{y}$ in joint space. This trajectory can be executed by the robot, to interact with the environment.

Referring to the introductory basic framework of Reinforcement Learning in Figure 1.1, we can utilize the PriMos as a bridge between the parameter actions and smooth trajectories, a robot may execute to alter the environment.
The RL agent will observe the environment's state S_t and compute an action $\boldsymbol{\alpha}$, according to the current policy $\pi(\boldsymbol{\alpha}|S_t)$. Using $\boldsymbol{\alpha}$ instead of the more common variable name a for some action, we emphasize the fact that $\boldsymbol{\alpha}$ is not an executable action, but instead the compressed parameter representation of a movement. The PriMo framework, that has been fit on a dataset according to the steps in Sections 2.2 and 2.4, is able to generate a

joint-space trajectory out of $\boldsymbol{\alpha}$. This can be done sequentially as outlined in Figure 2.3, or directly by applying equation 2.3. This gives the reconstructed movement$\boldsymbol{y}$, containing the corresponding angle for every joint and time step. Having a suitable interface to a robot system, this is sufficient for the movement to be executed by the robot.
If a suitable action is executed, the environment will be altered. In the case of a simple manipulation task, the placement of an object on a working table could have changed. The Reinforcement Learning Agent will then receive a reward for the executed action, depending on the reward function. The agent will consequently adapt the policy to $\pi^*(\boldsymbol{\alpha}|S_t)$. and repeat the process, until a terminal state S_T has been reached.

3 Experiments

In order to empirically evaluate our approach, we compared Principal Movement Primitives along different metrics. The reconstruction error is of interest since it indicates how accurate the reconstructed movements will be. Extending the number of basis functions f for the MPs to use will generally lower the reconstruction error, as will increasing the number of eigenvectors c used for the PCA. We want to calculate the reconstruction error of PriMos using different combinations of f and c. Using higher numbers for f and c comes at the cost of computational efficiency, so the analysis of the reconstruction error can help to find a combination that manages a good trade-off between accuracy and efficiency.
Furthermore, we want to compare PriMos to joint-space PCA. Since PriMos aim to describe movements using only a low number of parameters, we want to study how many total parameters are needed for PriMos to achieve a reconstruction error that is comparable to that of the joint-space approach. We want to repeat the experiment of parameter use on high-dimensional data, as well as low-dimensional data, to identify use cases of PriMos.

3.1 Data Generation

3.1.1 Low Dimensional Data

Our first source of data was the Darias robot, which consists of two Kuka lightweight robot arms, each with 7 degrees of freedom, used primarily for bi-manual manipulation tasks. We defined a task of pouring a designated amount of sugar pearls from a plastic cup, placed in the robot's hand, into a stationary bowl, at the center of a table in front of the robot. Originally we wanted to pour liquid, for safety reasons we decided to use pearls, as they allow for smooth pouring behavior as well. Figure 3.1 is a photograph of this setup, Figure 3.2 shows the robot performing a pouring movement using a reconstructed PriMos movement.

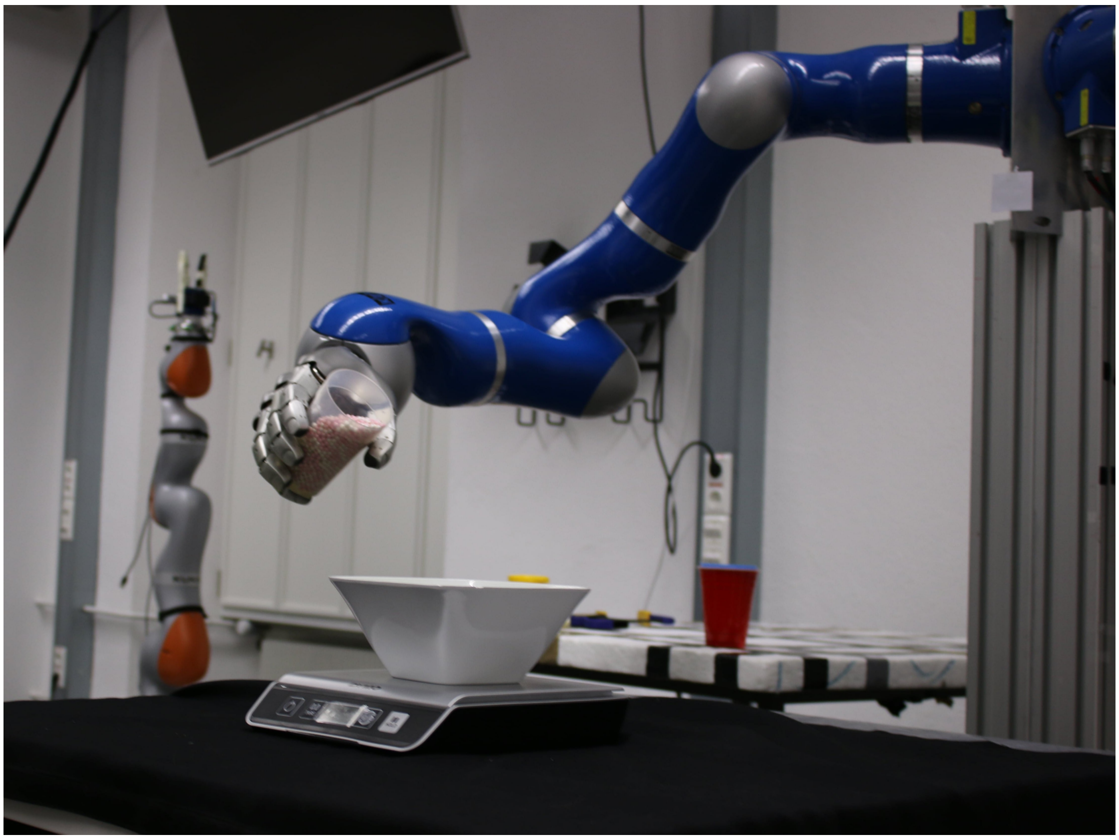

Figure 3.1: Experimental setup of the pouring task. This is the initial position of the robot. The bowl is placed on a digital scale that will measure the weight of sugar pearls poured into the bowl accurate to two grams.

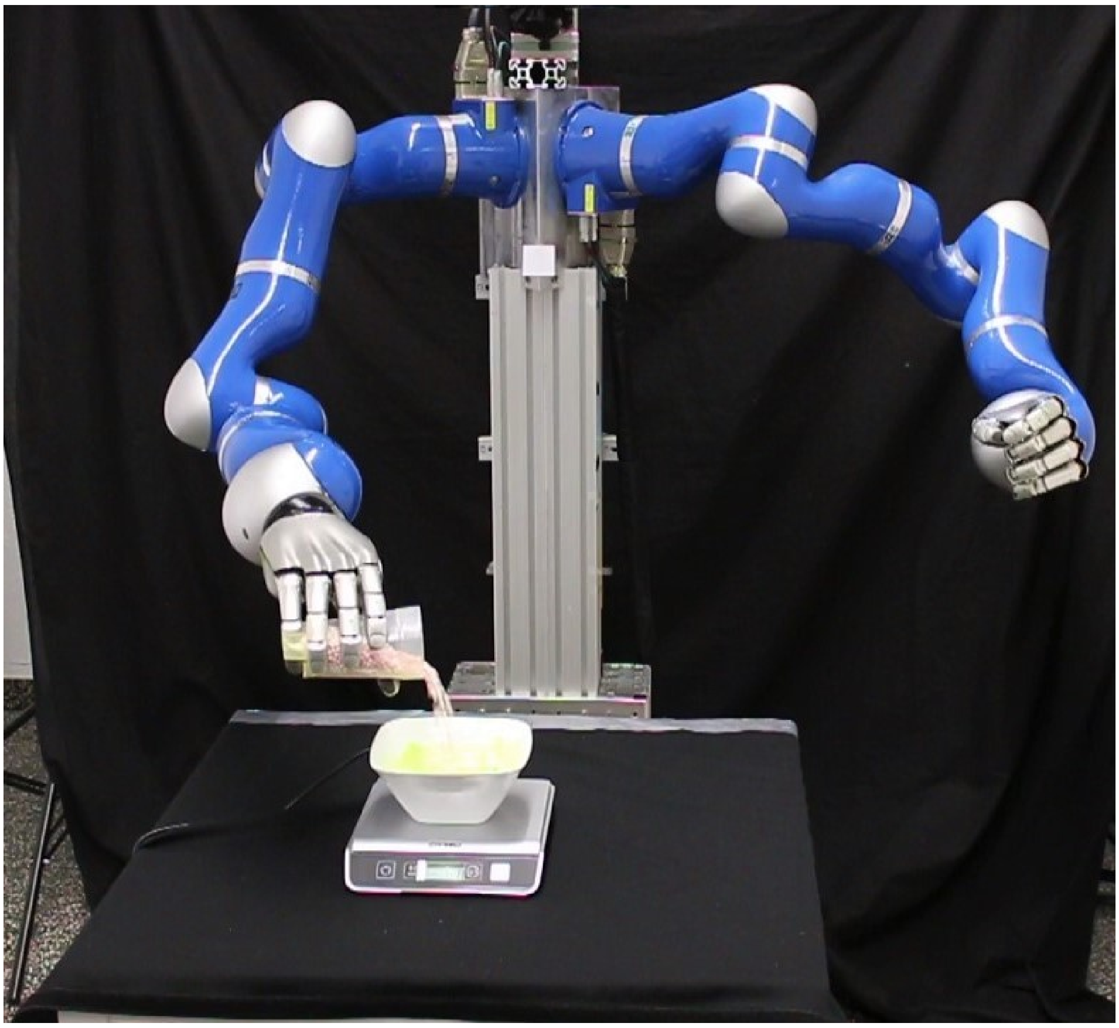

Figure 3.2: Darias robot performing a pouring movement. The movement executed was parameterized and compressed using the PriMos approach.

After each movement we weighed the bowl, using a digital scale. We reiterated this process 20 times, to obtain the `Darias` dataset. Since we only used the right robot arm and did not move the robot's fingers, the collected trajectories only have 7 joints. Each trajectory is therefore of dimensionality $\boldsymbol{\tau} \in \mathbb{R}^{T \times 7}$, where the number of time steps T depends on each trajectory's duration and the sampling frequency the recordings were made with. To evaluate our results more accurately, we fixated the cup onto the robot's hand, to ensure the placement and tilt of the cup would stay the same for retries. An image of this fixation can be found in Figure 3.3.

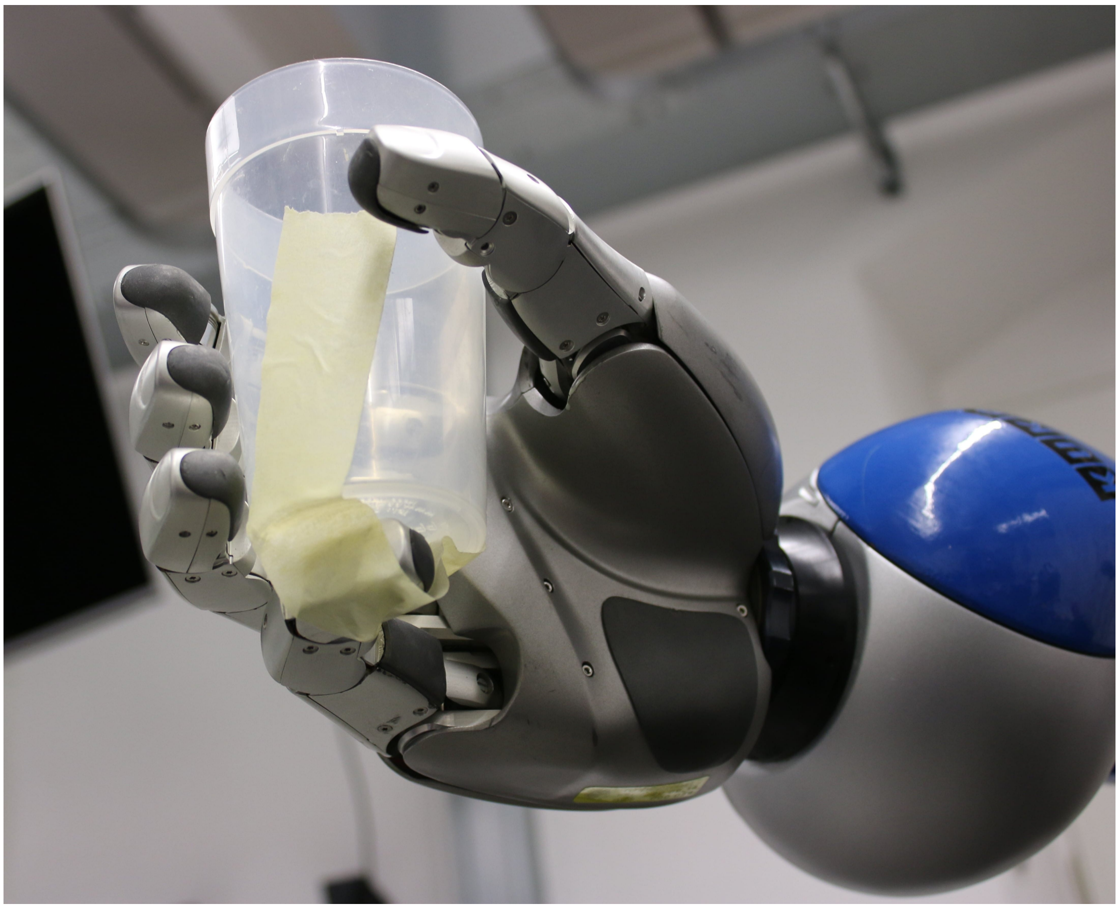

Figure 3.3: Fixation of plastic cup on a saved finger configuration.

3.1.2 High Dimensional Data

Since we wanted to test the algorithms in diverse settings, we sought to incorporate high-dimensional data. For this, we used various subjects and movements from the CMU Graphics Lab Motion Capture Database(http://mocap.cs.cmu.edu/). The data this

database comprises was generated by motion capturing of human subjects performing diverse tasks like walking, running, jumping or dancing. For an exemplary motion capture pipeline, see [22]. This is not the exact setup of the CMU Graphics Lab Motion Capture Database. The placement of the motion capture markers may not be the same. The joint values of the `Mocap` data are given in Euler angles. For better comparability with the `Darias` data, we converted them to radians. Similar to [4] we used the data of subjects 35, 49, 74, 120 and 143. We will reference these by `CMU #subject`. The data describes the movement of the whole body and is therefore high dimensional. Similar to the Darias robot, we treated the joint data for every axis of rotation separately. In these datasets, every trajectory has a total of 62 degrees of freedom and is therefore of dimensionality $\boldsymbol{\tau} \in \mathbb{R}^{T \times 62}$.
To evaluate the performance of PriMos compared to a standard movement-space compression, we compute the mean squared error on the `Darias` dataset.

3.2 Comparison With Unmodified Parameter-Space PCA

As the Principal Movement Primitives framework incorporates and extends the movement space PCA method, we decided to start with the evaluation of PriMos by comparing these two movement-space approaches. In order to achieve this, we considered the `Darias` dataset. We evaluated the reconstruction error for both methods. As is possible to observe in Figure 3.5, both methods have a very high error for less than four basis functions. This is to be expected, as ProMP representation of the trajectory is not complex enough to reasonably parameterize it. In addition, the PriMo approach seems to produce more accurate reconstructions for a small number of components ranging between 2 and 11.

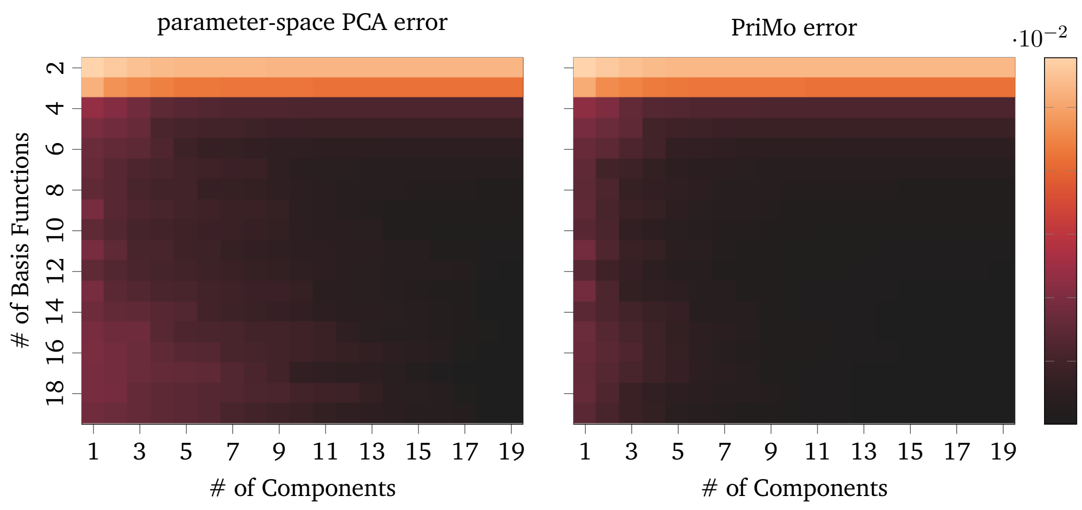

Figure 3.4: Reconstruction error heat-map on the `Darias` dataset for both models. The error was computed for different combinations of components used in the PCA and numbers of basis functions to use for parameterizing the trajectory. The reconstruction error is calculated using the mean squared difference of demonstrated trajectory y and the corresponding reconstruction $\hat{y}$. We averaged this error for all trajectories.

Since the relative performance is of great importance and it is not that easy to see notable differences in 3.5, it is sensible to illustrate the difference of both error plots. The results are shown in Figure 3.6.

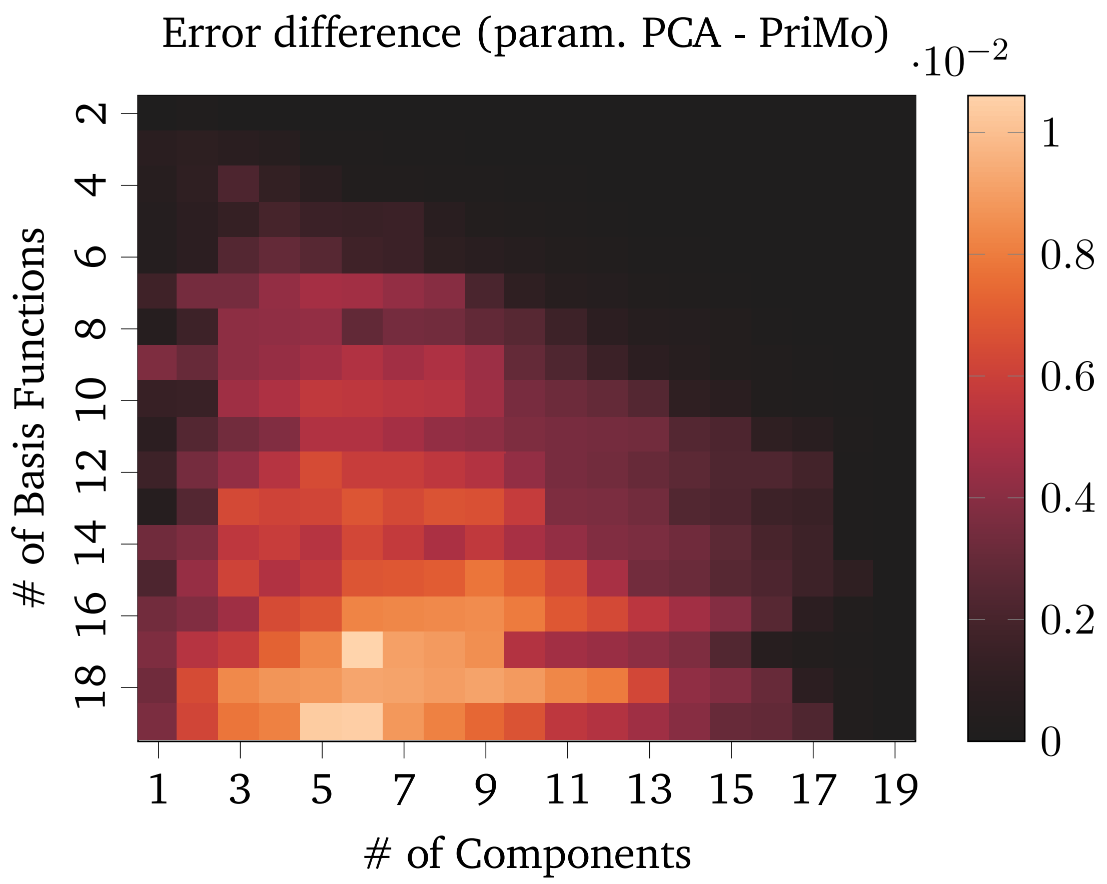

Figure 3.5: Depiction of the difference between the two heat-maps in Figure 3.5. It is the mean squared error, averaged for all trajectories in the dataset. Higher values indicate that the PriMos outperformed the unmodified parameter-space PCA by a greater amount.

Notably, there are no values where the movement-space PCA outperforms the PriMo approach. This is according to expectation, as the calculation of α is constrained on the reconstruction error on the original trajectories. The evaluation furthermore suggests that Principal Movement Primitives are especially advantageous with a low number of components used. In conclusion, the PriMo framework appears to be more accurate than the movement space PCA and all further comparisons can be done using the PriMo approach.

3.3 Comparison With Joint-Space PCA

As Principal Movement Primitives pose an alternative order of performing parameterization and dimensionality reduction compared to the joint space approach, key differences are of great interest. At its core, the Principal Movement Primitives approach intends to find a considerably small representation of movements. Therefore comparing the total number of parameters to describe the aforementioned trajectory. Reducing the data's size using the joint space PCA results in the use of $c \times f$ parameters in total, whereas PriMos compress it into a parameter vector of length c. We evaluated the minimum number of parameters needed so that the $NRMSE$ stays below a certain threshold. The Normalized Root Mean Squared Error, or $NRMSE$ for short is obtained by

$$NRMSE = \frac{RMSE}{\sigma_{\mathbb{D}}},$$

where $RMSE$ is the Root Mean Squared Error of the model's reconstruction and $\sigma_{\mathbb{D}}$ is the standard deviation of the dataset. This normalization allows for a better comparison between errors of different datasets. Similar to Section 3.2 we averaged the $NRMSE$ for all trajectories in the dataset as the basis of the parameter-use analysis. Figure 3.7 shows the minimum total number of parameters necessary so that the error does not exceed a certain $NRMSE$ for the `Darias` data.

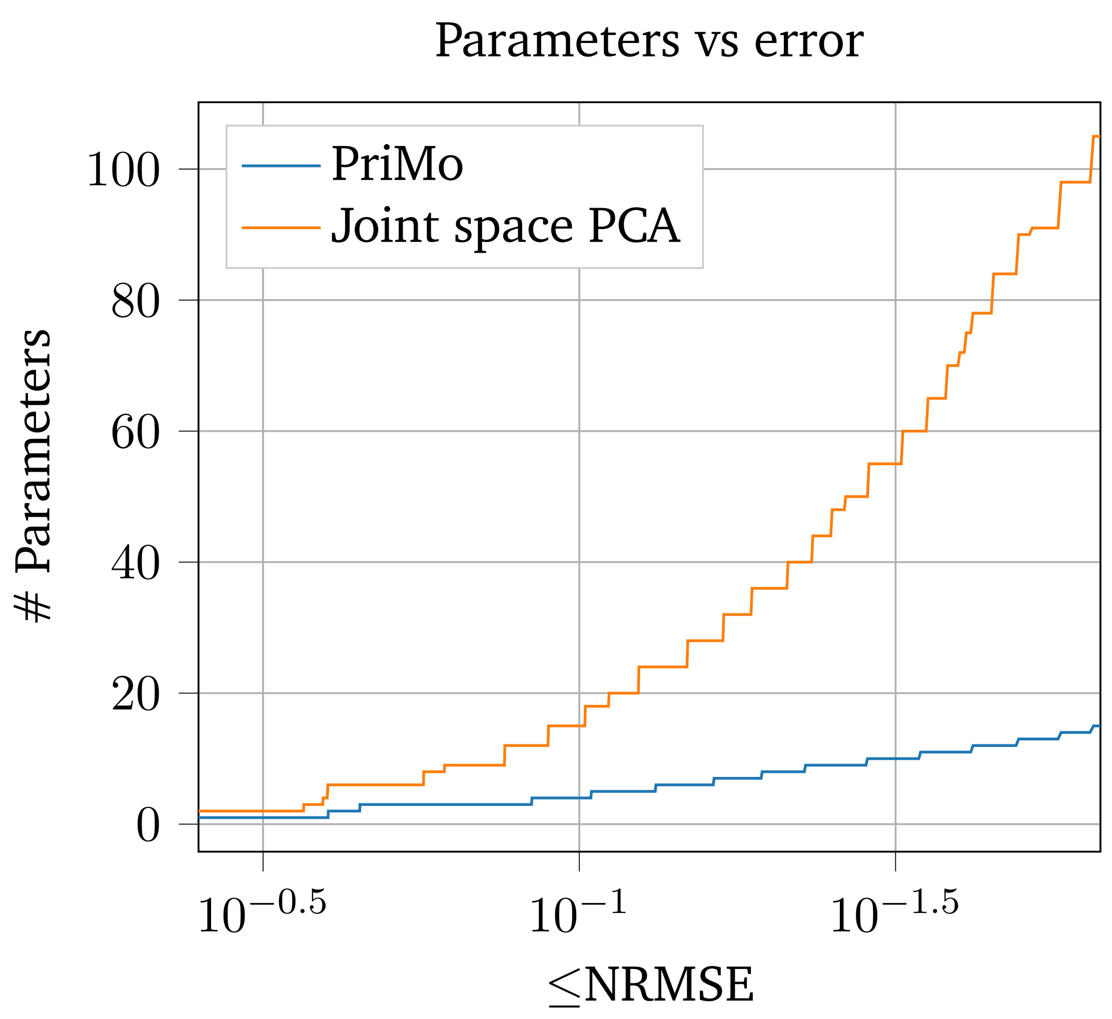

Figure 3.6: Evaluation of both movement space and joint space reduction methods on number of total parameters necessary for respective error to satisfy normalized error bound on the `Darias` dataset.

The empirical evaluation of Figure 3.7 suggests that PriMos use significantly less total parameters, compared to the joint space method.
However, the `Darias` set only consists of just 20 trajectories of the right robot arm performing similar movements. A great amount of similarity between the movements seems

very likely. As PriMos reduce the dimensionality on the movement's resemblance, we would expect them to be a suitable approach.
Testing the method on a substantially more high-dimensional set of movements is, therefore, necessary to infer the possible use cases for PriMos. For this purpose, we perform the calculation of the first number of parameters to fall below some normalized error bound ϵ again on the `CMU 143` dataset. We test this on the averaged $NRMSE$ for every trajectory in the `CMU 143` dataset. It contains 42 movements on 62 degrees of freedom, performing a variety of different actions, like climbing ladders, running and even dancing. These movements differ significantly from another, so the PriMos approach can not make use of movement resemblance. Similar to Figure 3.7, we normalized the empirical error by dividing by the dataset's standard deviation. The evaluation is depicted in Figure 3.8.

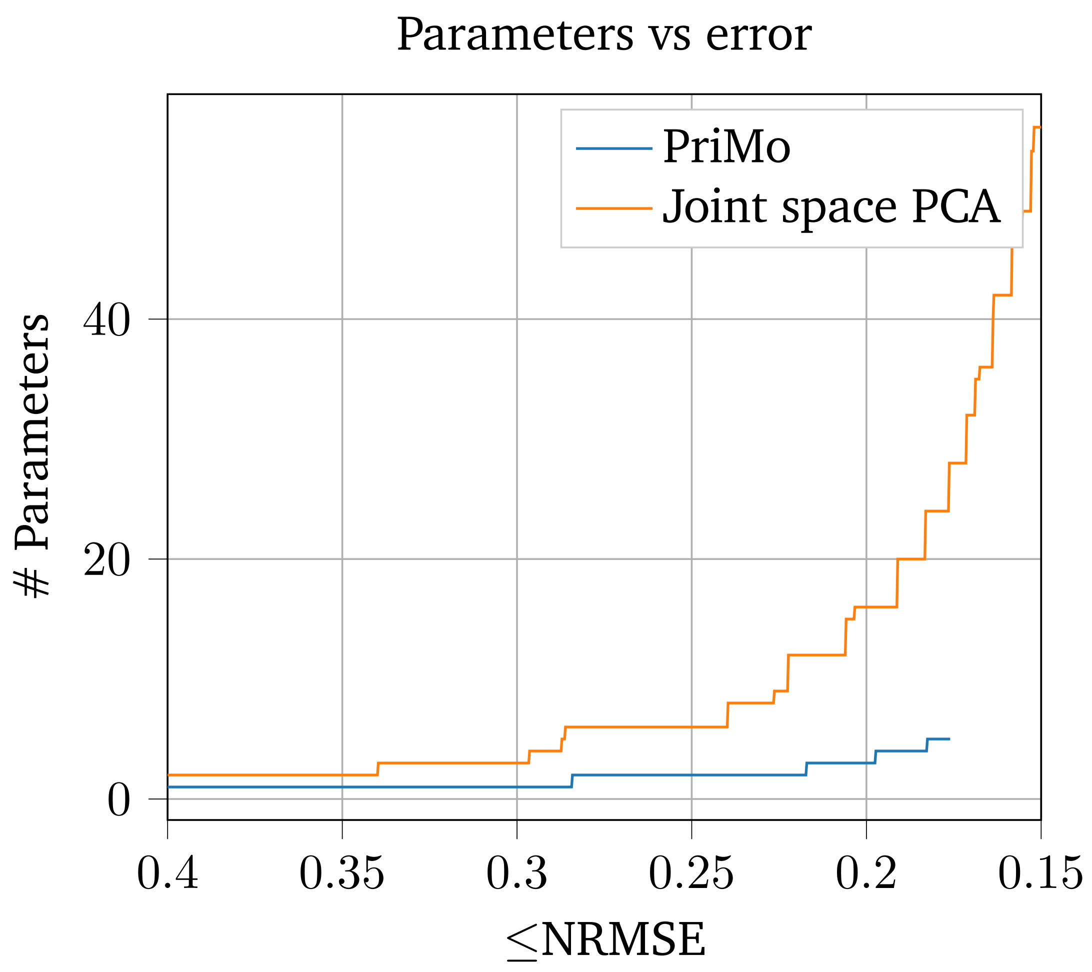

Figure 3.7: Evaluation of both methods on number of total parameters necessary for respective normalized error to satisfy error bound on the `CMU 143` dataset.

The most notable difference is the smaller error range on high-dimensional data. Furthermore, both methods perform worse on the `CMU 143` trajectories. At an $NRMSE$ threshold of 0.15, joint-space PCA requires almost 60 parameters. On the `Darias` data, the same method reached an error of 0.1 with less than 20 parameters. At an error of approximately 0.225 the PriMos plot stops. This means that the error will not be reached anymore, even if using all available components and basis functions. The joint-space PCA method, however, is still able to work within the error bound. The reason for the joint-space approach performing better on this high dimensional dataset is caused by a high coupling in the joint-space executing complex movements with a high number of degrees of freedom.

3.4 Comparison With ProMPs

Both, ProMPs and PriMos, describe a trajectory in parameter-space and are able to model the movement's variance. Therefore it is interesting to compare the respective variance. We are concerned with the movement's variance $\text{Var}(\boldsymbol{\tau})$. For the ProMPs this variance is obtained by

$$\begin{aligned}\text{Var}(\boldsymbol{\tau}) =&\text{Var}_{\boldsymbol{w}}(\boldsymbol{\Phi}\boldsymbol{w})\\ =&\boldsymbol{\Phi}\text{Var}_{\boldsymbol{w}}(\boldsymbol{w})\boldsymbol{\Phi}^{\intercal}.\end{aligned}$$

Similarly, the movement-variance can be calculated using

$$\begin{aligned}\text{Var}(\boldsymbol{\tau}) =&\text{Var}_{\boldsymbol{\alpha}}(\boldsymbol{\Phi}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{M}^{\intercal}\boldsymbol{\alpha})\\ =&\text{Var}_{\boldsymbol{\alpha}}(\boldsymbol{\Phi}\boldsymbol{M}^{\intercal}\boldsymbol{\alpha})\\ =&\boldsymbol{\Phi}\boldsymbol{M}^{\intercal}\text{Var}_{\boldsymbol{\alpha}}(\boldsymbol{\alpha})\boldsymbol{\Phi}^{\intercal}\boldsymbol{M}.\end{aligned}$$

The respective variances for $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ can be extracted from both parameters' covariance matrices. In Figure 3.9 we compare the calculated variance of both methods by plotting the 95 % quantile of the corresponding standard deviation $\sqrt{\text{Var}(\boldsymbol{\tau})}$ of some robot joints.

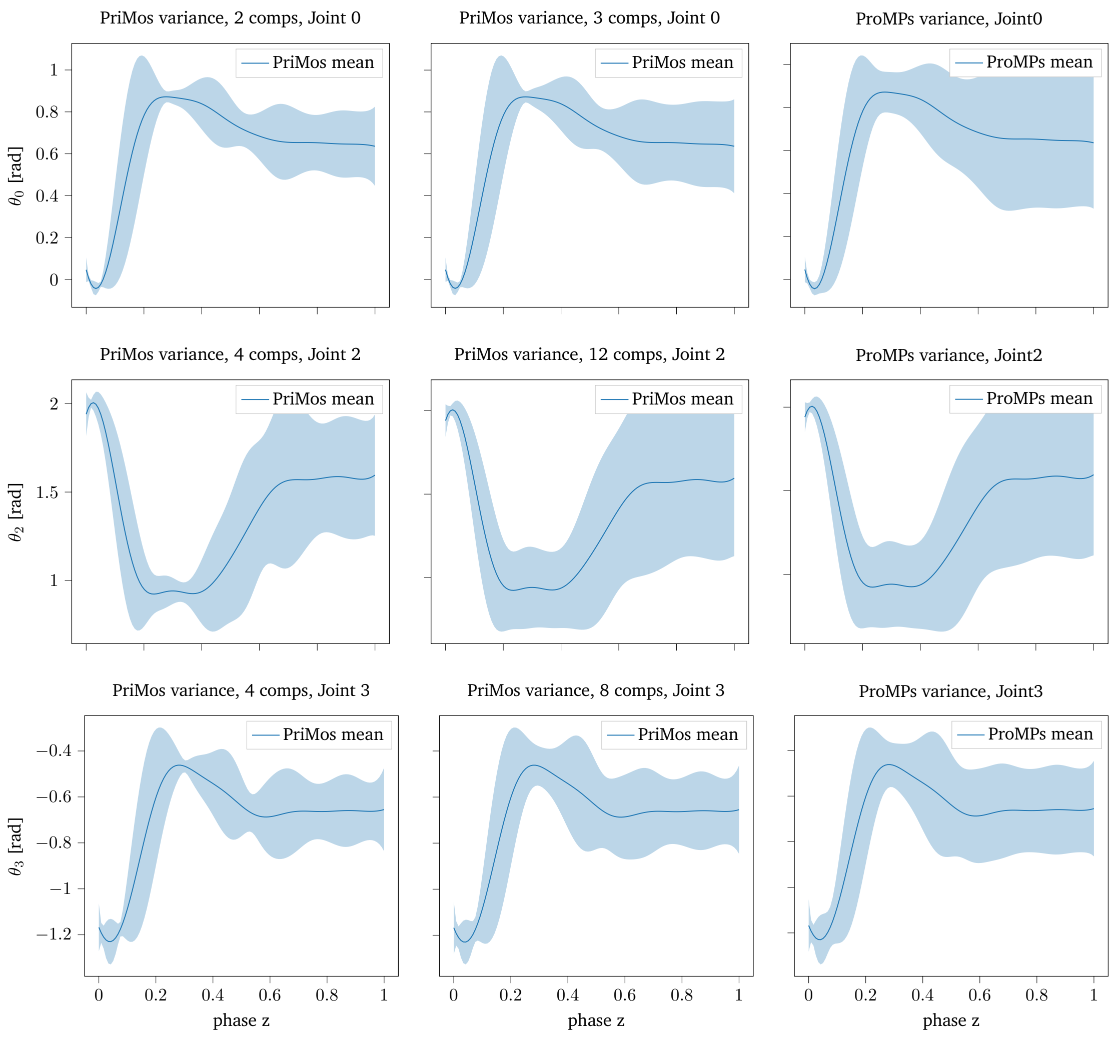

Figure 3.8: Comparison of the calculated standard deviation of PriMos and ProMPs evaluated on the `Darias` dataset. Each row represents one of the robot's joints. In each row, the first two plots from left to right are obtained by using PriMos to generate the standard deviation with an ascending number of components used. The rightmost plot of each row shows the standard deviation of the ProMPs. Every plot was created using 12 basis functions f.

Figure 3.9 indicates that the ProMPs have the largest variance. This implies that sampling randomly from ProMPs a larger set of movements can be generated. This can be undesirable because not every trajectory created in this way is guaranteed to be safe for the robot to execute. At worst, unsafe movements could damage the environment or the robot itself. By contrast, the standard deviation of PriMos is smaller, especially while using a low number of components c, as can be observed in the first column of Figure 3.9. Any generated trajectory will, therefore, be more similar to the mean movement. These trajectories are in general safer. With an increasing number of components the standard deviation of the PriMos increases. If $c = f$, the PriMos' standard deviation will be identical to that of the ProMPs, as can be seen in the second row of Figure 3.9. As a result, the probabilistic sampling using PriMos is a versatile approach since the number of components can be selected according to the task at hand.

4 Discussion

4.1 Conclusion

We introduced Principal Movement Primitives and derived its analytical solution. We showed that this approach is able to parameterize movements in a very compact way that is especially advantageous in a setting that uses a set of similar movement behaviors. The repeated execution of robot manipulation tasks, for example, would be a possible use case for PriMos. We evaluated PriMos on different datasets. As a set of high-dimensional data, we used humanoid movements obtained by motion capture techniques. Additionally, we gathered low-dimensional with high movement-correlation by defining and executing a robot task. Our experiments on these datasets suggested that PriMos are able to produce comparable errors using a lower number of parameters compared to a joint-space-reduction approach. Furthermore, we compared different methods of dimensionality reduction with PriMos and discussed the results. We described how to employ PriMos in a probabilistic context and showed the possibility of generating random trajectories using a compact representation of its covariance matrix. Finally, we presented a setup that would allow PriMos to be used together with RL agents so they can make use of concise action representation and therefore work in a computationally more efficient way.

4.2 Future Work

Although we used PCA throughout this work, the PriMos framework is not limited to this method of dimensionality reduction. There is a large number of algorithms and approaches that are concerned with dimensionality reduction problems and it would be interesting to see other methods used in PriMos instead of the standard PCA. One alternative approach to reducing the data's dimensionality would be by using an Autoencoder [8, 4]. These neural networks may be able to find advantageous subspaces since they are nonlinear

approaches.
Furthermore, the combination of PriMos and RL algorithms remains to be evaluated. PriMos could be used for the agent's action representation, so the RL-agent only handles low dimensional computation. The setup could be an episodic RL task, like grasping or placing an object. A suitable RL algorithm like Relative Entropy Policy Search [24] could be used to optimize a suboptimal initial policy, that has been obtained using imitation learning. The agent then tries to perform an action that is better than the human demonstration it has been presented.
Likewise, a step-based RL task could be considered by constructing a problem that would require the agent performing sequential actions. In this setting an off-policy algorithm like NOPG [31] could be employed in order to make use of a sample efficient policy gradient method.
Both RL experiments could be evaluated on their sample efficiency as well as the computational efficiency using PriMos and other methods of dimensionality reduction.

Bibliography

[1] Heni Ben Amor et al. “Generalization of human grasping for multi-fingered robot hands”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 2043–2050.

[2] Dana H Ballard. “Modular Learning in Neural Networks.” In: *AAAI*. 1987, pp. 279–284.

[3] Hamid Benbrahim and Judy A Franklin. “Biped dynamic walking using reinforcement learning”. In: *Robotics and Autonomous Systems* 22.3-4 (1997), pp. 283–302.

[4] Nutan Chen et al. “Efficient movement representation by embedding dynamic movement primitives in deep autoencoders”. In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2015, pp. 434–440.

[5] Adri‘a Colom’e et al. “Dimensionality reduction for probabilistic movement primitives”. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 794–800.

[6] Carl De Boor et al. *A practical guide to splines*. Vol. 27. springer-verlag New York, 1978.

[7] Vijaykumar Gullapalli, Judy A Franklin, and Hamid Benbrahim. “Acquiring robot skills via reinforcement learning”. In: *IEEE Control Systems Magazine* 14.1 (1994), pp. 13–24.

[8] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[9] Arthur E Hoerl and Robert W Kennard. “Ridge regression: Biased estimation for nonorthogonal problems”. In: *Technometrics* 12.1 (1970), pp. 55–67.

[10] Harold Hotelling. “Analysis of a complex of statistical variables into principal components.” In: *Journal of educational psychology* 24.6 (1933), p. 417.

[11] Auke J Ijspeert, Jun Nakanishi, and Stefan Schaal. “Learning attractor landscapes for learning motor primitives”. In: *Advances in neural information processing systems*. 2003, pp. 1547–1554.

[12] Ian T Jolliffe. “Principal components in regression analysis”. In: *Principal component analysis*. Springer, 1986, pp. 129–155.

[13] Dorothea Koert et al. “Demonstration based trajectory optimization for generalizable robot motions”. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2016, pp. 515–522.

[14] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[15] Guilherme Maeda et al. “Active incremental learning of robot movement primitives”. In: 2017.

[16] Guilherme Maeda et al. “Learning interaction for collaborative tasks with probabilistic movement primitives”. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 527–534.

[17] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[18] Todd K Moon. “The expectation-maximization algorithm”. In: *IEEE Signal processing magazine* 13.6 (1996), pp. 47–60.

[19] Alexandros Paraschos et al. “Probabilistic movement primitives”. In: *Advances in neural information processing systems*. 2013, pp. 2616–2624.

[20] Alexandros Paraschos et al. “Using probabilistic movement primitives in robotics”. In: *Autonomous Robots* 42.3 (2018), pp. 529–551.

[21] Frank C Park and Kyoosang Jo. “Movement primitives and principal component analysis”. In: *On Advances in Robot Kinematics*. Springer, 2004, pp. 421–430.

[22] Magdalena Pawlyta and Przemysław Skurowski. “A survey of selected machine learning methods for the segmentation of raw motion capture data into functional body mesh”. In: *Conference of Information Technologies in Biomedicine*. Springer. 2016, pp. 321–336.

[23] Karl Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[24] Jan Peters, Katharina Mulling, and Yasemin Altun. “Relative entropy policy search”. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[25] Sam T Roweis and Lawrence K Saul. “Nonlinear dimensionality reduction by locally linear embedding”. In: *science* 290.5500 (2000), pp. 2323–2326.

[26] Stefan Schaal. “Dynamic movement primitives-a framework for motor control in humans and humanoid robotics”. In: *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[27] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[28] Richard S Sutton and Andrew G Barto. “Reinforcement learning: An introduction”. In: (2011).

[29] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[30] Joshua B Tenenbaum, Vin De Silva, and John C Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *science* 290.5500 (2000), pp. 2319–2323.

[31] Samuele Tosatto et al. “A Nonparametric Offpolicy Policy Gradient”. In: *arXiv preprint arXiv:2001.02435* (2020).

[32] Christopher John Cornish Hellaby Watkins. “Learning from delayed rewards”. In: (1989).

5 Appendix

5.1 Derivations

5.1.1 Ridge Regression

In the PriMos approach ridge regression is used on trajectories $\boldsymbol{y}$ of different dimensionality. Let n denote the number of data-points in $\boldsymbol{y}$. For every $\boldsymbol{y}$ n may vary by a substantial amount. Similarly, the number of parameters d might differ. The penalization of the ridge term is affected by these dimensions, so we want to determine a scaling of λ that accounts for the potential differences.
We seek a vector $\boldsymbol{w}$ so that

$$\boldsymbol{w}^*_{\text{ridge}} = \text{arg min}_{\boldsymbol{w}} \frac{1}{n} \left(\boldsymbol{\Phi w} - \boldsymbol{y}\right)^\intercal \left(\boldsymbol{\Phi w} - \boldsymbol{y}\right) + \frac{\lambda}{d} \boldsymbol{w}^\intercal \boldsymbol{w}$$

holds. To achieve this it is reasonable to form the derivative with respect to our parameters and set the result equal to zero. The formulation of the gradient can be restructured as follows

$$\begin{aligned}
&\nabla_{\boldsymbol{w}}\left[\frac{1}{n}\Big(\boldsymbol{\Phi w}-\boldsymbol{y}\Big)^{\intercal}\Big(\boldsymbol{\Phi w}-\boldsymbol{y}\Big)+\frac{\lambda}{d}\boldsymbol{w}^{\intercal}\boldsymbol{w}\right]\\
&=\nabla_{\boldsymbol{w}}\left[\frac{1}{n}\Big(\boldsymbol{w}^{\intercal}\boldsymbol{\Phi}^{\intercal}-\boldsymbol{y}\Big)\Big(\boldsymbol{\Phi w}-\boldsymbol{y}\Big)+\frac{\lambda}{d}\boldsymbol{w}^{\intercal}\boldsymbol{w}\right]\\
&=\nabla_{\boldsymbol{w}}\left[\frac{1}{n}\Big(\boldsymbol{w}^{\intercal}\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi w}-\boldsymbol{w}^{\intercal}\boldsymbol{\Phi}^{\intercal}\boldsymbol{y}-\boldsymbol{y}^{\intercal}\boldsymbol{\Phi w}+\boldsymbol{y}^{\intercal}\boldsymbol{y}\Big)+\frac{\lambda}{d}\boldsymbol{w}^{\intercal}\boldsymbol{w}\right]\\
&=\nabla_{\boldsymbol{w}}\left[\frac{1}{n}\Big(\boldsymbol{w}^{\intercal}\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi w}-2\boldsymbol{y}^{\intercal}\boldsymbol{\Phi w}+\boldsymbol{y}^{\intercal}\boldsymbol{y}\Big)+\frac{\lambda}{d}\boldsymbol{w}^{\intercal}\boldsymbol{w}\right]\\
&=\frac{1}{n}\left(\nabla_{\boldsymbol{w}}\left[\boldsymbol{w}^{\intercal}\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi w}\right]-2\nabla_{\boldsymbol{w}}\left[\boldsymbol{y}^{\intercal}\boldsymbol{\Phi w}\right]+\nabla_{\boldsymbol{w}}\left[\boldsymbol{y}^{\intercal}\boldsymbol{y}\right]\right)+\nabla_{\boldsymbol{w}}\left[\frac{\lambda}{d}\boldsymbol{w}^{\intercal}\boldsymbol{w}\right]\\
&=\frac{1}{n}\left(\nabla_{\boldsymbol{w}}\left[\boldsymbol{w}^{\intercal}\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi w}\right]-2\nabla_{\boldsymbol{w}}\left[\boldsymbol{y}^{\intercal}\boldsymbol{\Phi w}\right]\right)+\nabla_{\boldsymbol{w}}\left[\frac{\lambda}{d}\boldsymbol{w}^{\intercal}\boldsymbol{w}\right]\\
&=\frac{1}{n}\left(\nabla_{\boldsymbol{w}}\left[\boldsymbol{w}^{\intercal}\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi w}\right]-2\nabla_{\boldsymbol{w}}\left[\boldsymbol{y}^{\intercal}\boldsymbol{\Phi w}\right]\right)+2\frac{\lambda}{d}\mathbb{I}\boldsymbol{w}\\
&=\frac{1}{n}\left(2\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi w}-2\nabla_{\boldsymbol{w}}\left[\boldsymbol{y}^{\intercal}\boldsymbol{\Phi w}\right]\right)+2\frac{\lambda}{d}\mathbb{I}\boldsymbol{w}\\
&=\frac{2}{n}\left(\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi w}-\boldsymbol{\Phi}^{\intercal}\boldsymbol{y}\right)+2\frac{\lambda}{d}\mathbb{I}\boldsymbol{w}
\end{aligned}$$

The solution is then obtained by

$$\begin{aligned}
&\frac{2}{n}\left(\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi w}-\boldsymbol{\Phi}^{\intercal}\boldsymbol{y}\right)+2\frac{\lambda}{d}\mathbb{I}\boldsymbol{w}=0\\
&\Leftrightarrow 2\left(\frac{1}{n}\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi}+\frac{\lambda}{d}\mathbb{I}\right)\boldsymbol{w}=\frac{2}{n}\boldsymbol{\Phi}^{\intercal}\boldsymbol{y}\\
&\Leftrightarrow \boldsymbol{w}=\frac{1}{n}\left(\frac{1}{n}\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi}+\frac{\lambda}{d}\mathbb{I}\right)^{-1}\boldsymbol{\Phi}^{\intercal}\boldsymbol{y}\\
&\Leftrightarrow \boldsymbol{w}=\left(\boldsymbol{\Phi}^{\intercal}\boldsymbol{\Phi}+\frac{n}{d}\lambda\mathbb{I}\right)^{-1}\boldsymbol{\Phi}^{\intercal}\boldsymbol{y}
\end{aligned}$$