Universität Karlsruhe (TH)
Interactive Systems Laboratory
Prof. Dr. Alexander Waibel

Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Lab
Prof. Dr. Trevor Darrell

Human Body Tracking with Rank Priors for Continuous Non-Linear Dimensionality Reduction

Diploma Thesis

Andreas Geiger

July 24th, 2008

Supervised by:

Prof. Dr. Alexander Waibel
Dr. Rainer Stiefelhagen
Dipl.-Inform. Kai Nickel

Universität Karlsruhe (TH)

Prof. Dr. Trevor Darrell
Dr. Raquel Urtasun

MIT CSAIL, UC Berkeley EECS

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Karlsruhe, den 24. Juli 2008 .....

Abstract

Non-linear dimensionality reduction methods are powerful techniques to deal with high-dimensional datasets. However, they often are susceptible to local minima and perform poorly when initialized far from the global optimum, even when the intrinsic dimensionality is known a priori. In this work a prior over the dimensionality of the latent space is introduced, which allows for simultaneously optimizing both the latent space and its intrinsic dimensionality. Ad-hoc initialization schemes are unnecessary with this approach; the latent space is initialized to the observation space and the latent dimensionality is automatically inferred using an optimization scheme that drops dimensions in a continuous fashion. The prior is applied to various tasks involving probabilistic non-linear dimensionality reduction, and it is shown that this method can outperform graph-based dimensionality reduction techniques as well as previously suggested ad-hoc initialization strategies. Results are reported for artificial datasets which cause problems to nearest neighbor-based spectral methods and for a real-world example, where the task consists of tracking a person performing activities in a kitchen scenario.

Zusammenfassung

Eine Vielzahl unterschiedlicher Sensoren erlaubt heute die Registrierung von Daten auf modernen Rechenmaschinen. Allerdings ist die Dimension D der Daten (zum Beispiel $D = 1310720$ im Falle einer 1280×1024-Videokamera) oft zu hoch um sie direkt weiter verarbeiten zu können. Daher sind Methoden für nichtlineare Dimensionsreduktion unerlässlich. Solche Methoden versuchen, durch eine geeignet gewählte Abbildung (in einen Raum niederer Dimension), die Datenmenge zu reduzieren ohne dabei die für die Aufgabe relevante Beziehungen in den Daten zu zerstören. Dabei wird angenommen dass die Daten auf oder nahe einer Mannigfaltigkeit niederer Dimension verteilt sind, welche in einen hochdimensionalen Raum eingebettet ist. Je nach Methodik kann dabei auch a-priori Vorwissen über den Prozess mit eingearbeitet werden, sofern dieses zur Verfügung steht.

Verschiedene Verfahren zur Reduktion der Dimension sind in den letzten Jahren entwickelt worden. Ein besonderes Augenmerk fällt dabei auf die nicht-linearen Verfahren, da diese oft eine kompaktere Repräsentation erzeugen können als lineare Verfahren [Pea01]. Dies ist im Besonderen dann der Fall, wenn die Daten Abhängigkeiten aufweisen, die sich nicht in Korrelationen äussern. Einige dieser Verfahren bieten geschlossene Lösungen an [RS00, TSL00, TB99, WS06, BN03, ZZ03], reagieren aber oft empfindlich auf die Wahl der Parameter oder die Datenverteilung.

Andere basieren auf der Optimierung nicht-konvexer Funktionen [Law05, LQC06] und sind daher anfällig für lokale Minima im Optimierungsprozess.

Wieder andere setzen a-priori Wissen ein [WFH08, UD07, UFG+08], welches allerdings, je nach Aufgabe, nicht immer zur Verfügung steht. Zudem wird eine geeignete Wahl der Dimension bei den meisten Verfahren als bekannt angenommen. Diese muss zunächst, zum Beispiel durch aufwändige Vergleichsprüfung (*Cross-Validation*) oder andere Verfahren [mFSA07, LB05] ermittelt werden.

In dieser Diplomarbeit wird eine neue Methode vorgestellt, welche auf dem *Gaussian Process Latent Variable Model (GPLVM)* aufbaut. Durch einen *Rank Prior über die Anzahl der Dimensionen* wird die Dimensionalität kontinuierlich (während des Optimierungsprozesses) reduziert. So lassen sich lokale Minima des GPLVM vermeiden und die Anzahl der Dimensionen wird automatisch ermittelt. Die resultierende kompakte Repräsentation bietet zudem eine probabilistische Abbildung vom niederdimensionalen (latenten) Raum in den hochdimensionalen (Daten-) Raum. Dadurch lassen sich Aussagen über die Unsicherheit eines abgefragten Datums treffen.

Im Abschnitt *Experiments* wird das entwickelte Verfahren auf 3 Fälle angewandt: Zunächst wird illustriert, wie sich mit der Methode eine *Schweizer Rolle* trotz spärlicher Abtastung noch zuverlässig entfalten lässt. Danach wird die Methode auf Beispiele angewandt, in denen es die geeignete Dimension zu bestimmen gilt. Abschliessend werden Bewegungen eines artikularen menschlichen Körpers eingelernt. Diese werden dann verwendet um eine Person, welche Aktivitäten in der Küche (wie beispielsweise Auswellen, Mahlen, Schneiden) durchführt, erfolgreich zu verfolgen und die Bewegungen korrekt zu klassifizieren.

Contents

1 Introduction

Many real-world problems involve high dimensional datasets that are computationally challenging to handle. In such cases it is desirable to reduce the dimensionality of the data while preserving the original information in the data distribution, allowing for more efficient learning and inference. Linear dimensionality reduction methods (e.g., *PCA*) are efficient but miss important structure in non-linear data. Graph-based techniques, e.g., *LLE* [RS00] and *ISOMAP* [TSL00], capture non-linear dependencies but require highly dense and homogeneously sampled manifolds for accurate modeling.

Non-linear dimensionality reduction techniques can be applied to more complex data, but generally suffer from local minima. Choosing the dimensionality of the latent space is non-trivial, and existing methods typically rely on cross-validation. Even when given the correct latent dimensionality, these techniques often do not succeed in practice when initialized far from the global minimum [UFG+08]. Factors which contribute to this include the distortion introduced by the initialization and the non-convexity of the optimization: when optimization is performed in a low dimensional space the model may not have the requisite degrees of freedom to avoid local minima.

In this work we develop a prior on the dimensionality of the set of latent coordinates, encouraging low dimensional representations. Our *Rank Prior* enforces a penalty on the non-sparsity of the singular values of the matrix of latent variables, and automatically discovers the latent space and its dimensionality using a continuous optimization that drops dimensions on the fly. By initializing the latent coordinates to the original space no distorsion is introduced; since we decrease the dimensionality slowly (starting from a high-dimensional space) extra flexibility is gained which allows the method to avoid local minima during optimization. To our knowledge, ours is the first non-linear dimensionality reduction technique that penalizes the latent space rank and simultaneously optimizes the structure of the latent space as well as its intrinsic dimensionality.

We demonstrate the effectiveness of our approach when learning probabilistic latent variable models with the *Gaussian Process Latent Variable Model (GPLVM)* [Law05], a generalization of Probabilistic PCA to the non-linear case that models the mapping from the latent space to the data space as a Gaussian process. The GPLVM has proven successful in many applications, but initialization, knowledge of the latent dimensionality, and/or additional prior knowledge were assumed (e.g.,

[GMHP04, SUF08, UFG+08]). Incorporating our prior with the GPLVM objective results in an optimization problem that allows us to discover the latent space and the intrinsic dimensionality of artificial and real datasets.

We chose *articulated body tracking* as an application of our technique. Since a full human body model consists of many *degrees of freedom (DOFs)*, reducing the dimensionality is key to performing robust and reliable tracking. Since the observation likelihood for high-dimensional models is highly non-convex, tracking in the original space is a hard problem, especially when using a small number of cameras or a simple observation likelihood. Furthermore the need for many evaluations of the likelihood function inhibits a reasonable tracking speed. Consider for example a particle filter where the number of particles needed grows exponentionally with the number of dimensions. Using a prior model created via motion capturing data helps in two ways: While making the algorithm computationally tractable it helps disambiguating ambiguous observations. We thus demonstrate the effectiveness of our approach in tracking and classifying complex articulated human body motions from video by using such a prior model which is explained in more details in this thesis.

2 Dimensionality Reduction

A fundamental challenge of artificial intelligence is to develop useful internal representations of the external world, ideally by introducing as little prior knowledge as possible. The human brain excels in tasks like memorizing objects or faces by extracting features from a vast amount of input information perceived by sensors like the eyes, for instance. Since visual information is usually high-dimensional in raw form (consider the image of a 10 Megapixels consumer camera), reducing the dimensionality is key for further using the data by processing it on a computer. Only with a good representation and relevant features is higher-level decision making possible (a high-level decision in an image could be a single binary feature, such as the answer to the question if a given object is present in the image or not).

While features for visual input can be designed in problem-specific fashion (e.g. by building bag-of-word features or by looking for strong gradients, corners and using methods like SIFT [Low99]) signal data may be on hand in many different forms and an automatic feature retrieval method is eligible.

From a statistical view, *dimensionality reduction* (also called *dimension reduction*) is the process of reducing the number of random variables under consideration. This can be further divided into feature selection and feature extraction. While feature selection tries to find a subset of the original variables which is most representative, feature extraction relates the high dimensional data to a lower dimensional space while trying to maintain the information contained in the input data. The projection can be either linear or non-linear, both approaches are presented within this thesis.

2.1 Notation

In order to avoid confusion the following notation has been retained throughout the paper:

- N: Number of training points
- D: Dimensionality of the high dimensional input (=data) space $\mathbf{Y}$
- Q: Dimensionality of the low dimensional latent space $\mathbf{X}$

- $\mathbf{y}_i$: input (=data) point i, $\mathbf{y}_i \in \Re^D$
- $\mathbf{x}_i$: latent point i, $\mathbf{x}_i \in \Re^Q$
- $\mathbf{Y} = (\mathbf{y}_1, ..., \mathbf{y}_N)^T$: Collection of input (=data) points, $\mathbf{Y} \in \Re^{N \times D}$
- $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)^T$: Collection of latent points, $\mathbf{X} \in \Re^{N \times Q}$
- Y_{ij} is the entry of matrix $\mathbf{Y}$ at row i and column j
- X_{ij} is the entry of matrix $\mathbf{X}$ at row i and column j
- $y_{ij} = Y_{ji}$ is the entry of the transposed matrix $\mathbf{Y}^T$ at row i and column j
- $x_{ij} = X_{ji}$ is the entry of the transposed matrix $\mathbf{X}^T$ at row i and column j
- $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$: $\mathbf{x}$ follows a Gaussian distribution
- $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$: $f(\mathbf{x})$ follows a Gaussian Process distribution
- $\overline{\mathbf{x}}$: mean of $\mathbf{x}$: $\overline{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$
- $\overline{\mathbf{y}}$: mean of $\mathbf{y}$: $\overline{\mathbf{y}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i$

2.2 Linear Dimensionality Reduction

Linear dimensionality reduction finds a linear projection from a low dimensional subspace $\mathbf{X}$ to a higher dimensional space $\mathbf{Y}$. For example *Principal Component Analysis (PCA)* [Pea01, DHS00], sometimes also called the Karhunen-Loève transform, seeks an optimal affine subspace $\mathbf{X}$ of $\mathbf{Y}$ which maximizes variance (along the dimensions of $\mathbf{X}$) and minimizes the reprojection error when projecting back from $\mathbf{X}$ to $\mathbf{Y}$. More formally let the projection from the latent space $\mathbf{X}$ to the data space $\mathbf{Y}$ be

$$\hat{\mathbf{y}}_i = \overline{\mathbf{y}} + \sum_{j=1}^{Q} x_{ij} \mathbf{e}_j \tag{2.1}$$

where $\{\mathbf{e}_j, ...\mathbf{e}_Q\}$ spans a Q-dimensional orthonormal basis in the data space $\mathbf{Y}$. Then the *least squares reprojection error* is defined as

$$\mathcal{E} = \sum_{i=1}^{N} \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|^2 = \sum_{i=1}^{N} \|\overline{\mathbf{y}} + \sum_{j=1}^{Q} x_{ij} \mathbf{e}_j - \mathbf{y}_i\|^2.$$

Considering that $\{\mathbf{e}_j, ...\mathbf{e}_Q\}$ forms a orthonormal basis, we can expand $\mathcal{E}$ to

$$\mathcal{E} = \sum_{i=1}^{N} \left[\sum_{j=1}^{Q} \left[x_{ij}^2 + 2 x_{ij} \mathbf{e}_j^T (\overline{\mathbf{y}} - \mathbf{y}_i) \right] + \|\overline{\mathbf{y}} - \mathbf{y}_i\|^2 \right]. \tag{2.2}$$

Minimizing $\mathcal{E}$ with respect to x_{ij} can be done in closed form and leads to

$$\frac{\partial \mathcal{E}}{\partial x_{ij}} = 2x_{ij} + 2\mathbf{e}_j^T(\overline{\mathbf{y}} - \mathbf{y}_i) \stackrel{!}{=} 0$$
$$\Rightarrow x_{ij} = -\mathbf{e}_j^T(\overline{\mathbf{y}} - \mathbf{y}_i).$$

Applying this result to equation 2.2 we have

$$\mathcal{E} = \sum_{i=1}^{N}\left[-\sum_{j=1}^{Q} x_{ij}^2 + \|\overline{\mathbf{y}} - \mathbf{y}_i\|^2\right] \tag{2.3}$$
$$= -\sum_{j=1}^{Q}\mathbf{e}_j^T\mathbf{S}\mathbf{e}_j + \sum_{i=1}^{N}\|\overline{\mathbf{y}} - \mathbf{y}_i\|^2 \tag{2.4}$$

where $\mathbf{S}$ is the *scatter matrix* of $\mathbf{Y}$, defined as

$$\mathbf{S} = \sum_{i=1}^{N}(\overline{\mathbf{y}} - \mathbf{y}_i)(\overline{\mathbf{y}} - \mathbf{y}_i)^T.$$

The constraint $\|\mathbf{e}_i\| = 1$ is enforced by introducing *Lagrange Multipliers* λ into the target equation:

$$\mathcal{E}' = -\sum_{j=1}^{Q}\mathbf{e}_j^T\mathbf{S}\mathbf{e}_j + \sum_{i=1}^{N}\|\overline{\mathbf{y}} - \mathbf{y}_i\|^2 + \sum_{j=1}^{Q}\lambda_j(\mathbf{e}_j^T\mathbf{e}_j - 1) \tag{2.5}$$

Minimizing $\mathcal{E}'$ with respect to $\mathbf{e}_j$ is straightforward

$$\frac{\partial \mathcal{E}'}{\partial \mathbf{e}_j} = -2\mathbf{S}\mathbf{e}_j + 2\lambda\mathbf{e}_j \stackrel{!}{=} 0$$

leading to the eigenvalue problem

$$\mathbf{S}\mathbf{e}_j = \lambda_j\mathbf{e}_j.$$

Since

$$\mathbf{e}_j^T\mathbf{S}\mathbf{e}_j = \lambda_j\mathbf{e}_j^T\mathbf{e}_j$$

$\mathcal{E}'$ is minimized by selecting $\mathbf{e}_j$ as the top Q eigenvectors of $\mathbf{S}$. Once the basis $\{\mathbf{e}_j, ...\mathbf{e}_Q\}$ has been calculated, projections in both directions (from $\mathbf{X}$ to $\mathbf{Y}$ and vice versa) are readily given (since we assumed linearity in equation 2.1) as

$$\mathbf{x}_i = \mathbf{E}^T(\mathbf{y}_i - \overline{\mathbf{y}}) \quad \text{and} \quad \mathbf{y}_i = \mathbf{E}\mathbf{x}_i + \overline{\mathbf{y}}$$

where $\mathbf{E}$ is a $D \times Q$ matrix comprising the basis vectors $\mathbf{e}_j$ as columns.

2.3 Non-linear Dimensionality Reduction

Linear Dimensionality Reduction (LDR) is often used due to its simplicity and closed form solution. When data dependencies are mostly expressed in terms of correlations this leads to reasonable results. Considering real world data (and also very simple artificial examples), linearity cannot be assumed in general. Consider for example a one dimensional manifold[1] coiled up to a spiral when embedded in two dimensions (see figure 3.2). Applying PCA in this case leads to a bad solution since points which are far away in the two dimensional input space $\mathbf{Y}$ are projected to nearby points on the one dimensional subspace $\mathbf{X}$ (in the figure, this means that red points (circles) and yellow points (crosses) are not separated after applying 1D PCA).

To handle non-linear dependencies one has to resort to *Non-Linear Dimensionality Reduction (NLDR)* techniques which are discussed in this section of the thesis.

2.3.1 Graph Based Methods

Graph based methods allow for non-linear dimensionality reduction by taking into consideration a k-nearest-neighbor graph. They often offer closed-form solutions and have proven successful in real-world applications like classification of digits, for example. A short overview of such techniques is given next.

ISOMAP [TSL00] finds a projection that preserves the global, nonlinear geometry of the data by preserving the geodesic manifold distances between points. It first approximates the geodesic distances (by means of a nearest neighbor graph) and then runs multidimensional scaling to find a projection that preserves these distances.

[1] A manifold is an abstract mathematical space in which every point has a neighborhood which resembles Euclidian space, but in which the global structure may be more complicated.

Laplacian Eigenmaps [BN03] calculates the embedding map by constructing a weighted graph (based on nearest neighbors or ϵ-balls) and calculating the eigenvectors of the graph Laplacian, which is an approximation to the Laplace Beltrami operator

$$\mathcal{L}f = -div\nabla(f)$$

The weights are chosen either by the heat kernel

$$w_{ij} = \exp\left(-\frac{\|\mathbf{y}_i - \mathbf{y}_j\|^2}{t}\right)$$

or by setting $w_{ij} = 1$ if an edge exists and $w_{ij} = 0$ otherwise.

Locally Linear Embeddings (LLE) [RS00] assume that the data points $\mathbf{Y}$ can be locally reconstructed in the latent space $\mathbf{X}$. First, weights are calculated by minimizing the cost function

$$\mathcal{E} = \sum_{i=1}^{N} \|\mathbf{y}_i - \sum_{j\in\eta_i} w_{ij}\mathbf{y}_j\|^2$$

where η_i denotes the set of indices of nodes in the neighborhood of $\mathbf{y}_i$. The invariance constraint $\sum_{j=1}^{|\eta_i|} w_{ij}$ ensures that the weights sum up to one. Finally the latent embedding is found by minimizing

$$\mathcal{E}' = \sum_{i=1}^{N} \|\mathbf{x}_i - \sum_{j\in\eta_i} w_{ij}\mathbf{x}_j\|^2$$

with respect to x while keeping the weights fixed. It turns out that - by removing the translational and rotational degree of freedom - this leads to a well posed problem and can be solved by a standard eigenproblem solver.

Local Tangent Space Alignment (LTSA) [ZZ03] represents the local geometry of the manifold by fitting an affine subspace in a neighborhood of each data point. The tangent spaces are then aligned to give the internal global coordinates of the data points with respect to the underlying manifold by way of a partial eigendecomposition of the neighborhood connection matrix. To do so, a smooth function

$$\mathbf{y}_i = f(\mathbf{x}_i) + \epsilon$$

which generates the data points $\mathbf{y}_i$ from latent points $\mathbf{x}_i$ is assumed. Thus a first order Taylor expansion

$$f(\overline{\mathbf{x}}) = f(\mathbf{x}) + J_f(\mathbf{x})(\overline{\mathbf{x}} - \mathbf{x}) + O(\|\overline{\mathbf{x}} - \mathbf{x}\|^2)$$

is used to reconstruct the data points locally by means of the tangent space spanned by the columns of the Jacobi matrix $J_f(\mathbf{x})$. Here $\overline{\mathbf{x}}$ is taken as each

local neighborhood's mean, respectively. [ZZ03] shows that finding the best Q-dimensional affine subspace approximation leads to a singular value problem. Global coordinates are then extracted by minimizing the overall reconstruction error, leading to an eigenvalue problem.

Maximum Variance Unfolding (MVU) [WS06] assumes that nearby inputs match the distances between nearby outputs. It places *rigid* connections between neighboring nodes by enforcing hard constraints. The optimizing problem is formulated as a quadratic program:

Maximize

$$\sum_{ij} \|\mathbf{x}_i - \mathbf{x}_j\|^2$$

subject to

$$(1) \quad \|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{y}_i - \mathbf{y}_j\|^2 \quad \text{for all} \quad (i,j) \quad \text{with} \quad w_{ij} = 1 \tag{2.6}$$

$$(2) \quad \sum_{i=1}^{N} \mathbf{x}_i = 0 \tag{2.7}$$

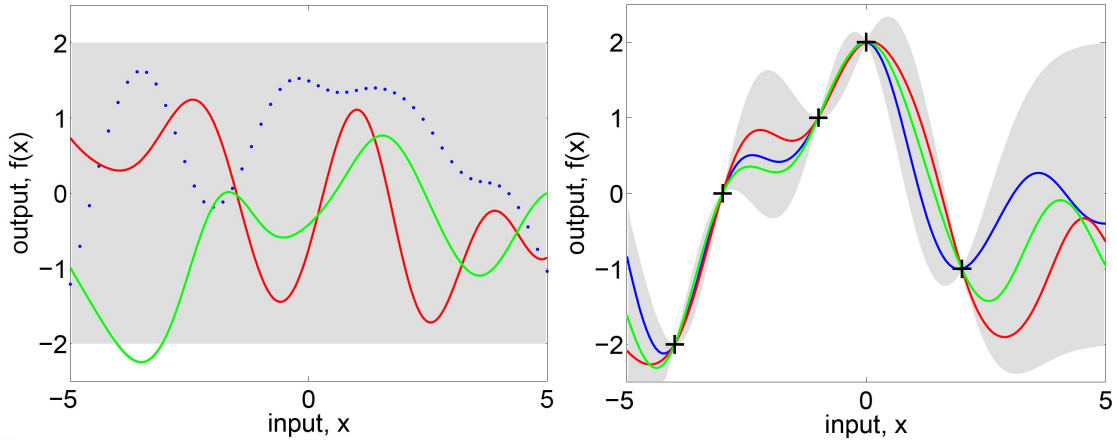
where $w_{ij} = 1$ if an edge in the neighborhood graph exists and $w_{ij} = 0$ otherwise. The problem is made tractable by reformulating the optimization as a *semidefinite program* (SDP) resulting in a *linear program*.

All spectral NLDR methods described above rely on a local neighborhood structure where the neighborhood size k is a parameter and the dimensionality Q is assumed to be known. Predicting a new data point $\mathbf{y}_i$ often requires to re-run the algorithm again. Furthermore they do not in general provide a measure of uncertainty about a given point, i.e. they are non-probabilistic.

In the following section *probabilistic Latent Variable Models* are discussed with a focus on the *Gaussian Process Latent Variable Model*, to which the prior proposed in this thesis will be applied in section 3.3.

2.3.2 Latent Variable Models

Latent Variable Models (LVMs), e.g., *MDS* or *Probabilistic PCA* [TB99] (a probabilistic extension of PCA and a special case of the Gaussian Process Latent Variable Model), assume that the data $\mathbf{Y}$ has been generated by some latent (which means unobserved) variables $\mathbf{X}$ that lie on or close to a low-dimensional manifold. Probabilistic LVMs relate the latent variables to a set of observed variables via a *probabilistic mapping*. Since we focus on the GPLVM, the next section introduces Gaussian Processes first.



(a) Prior $p(f)$ (b) Posterior $p(f|\mathbf{X}, \mathbf{Y})$

Figure 2.1: **Sampling from a Gaussian Process.** (a) shows 3 functions drawn at random from a GP prior. (b) depicts 3 functions drawn at random from the posterior, i.e. the prior conditioned on the five noise free observations indicated. The gray areas show the uncertainty for each point x respectively. This figure was taken from [RW06].

Gaussian Processes

Gaussian Processes (GPs) [RW06] underwent a revival in the Machine Learning community over the past ten years. Since the implementation is simple they became useful tools for probabilistic regression and classification. Here GPs are introduced in the context of regression and we give a short illustrative example.

A Gaussian Process is a stochastic process and can be seen as an infinite dimensional Gaussian distribution.

Definition 1 *A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution. It is completely specified by its mean function $m(\boldsymbol{x})$ and its covariance function $k(\boldsymbol{x}, \boldsymbol{x'})$ and we write a Gaussian Process as*

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'}))$$

where

$$m(\boldsymbol{x}) = \mathcal{E}(f(\boldsymbol{x})) \tag{2.8}$$

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \mathcal{E}\left[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x'}) - m(\boldsymbol{x'}))\right] \tag{2.9}$$

The specification of the covariance function implies a distribution over functions. To see this, one can draw samples from this distribution, as done in figure 2.1.

Note that the definition of GPs automatically implies a *consistency requirement*, also known as the *marginalization property*, since the covariance function $k(\mathbf{x}, \mathbf{x'})$ specifies entries of the covariance matrix K. The covariance function is also called a *kernel function*. This property means that if the GP (e.g.) specifies $(y_1, y_2) \sim \mathcal{N}(\mu, \Sigma)$, then it must also specify $y_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$ and $y_2 \sim \mathcal{N}(\mu_2, \Sigma_{22})$.

As for section 2.3.1, let $\mathbf{Y} = [\mathbf{y}_1, \cdots, \mathbf{y}_N]^T$ be the set of observations $\mathbf{y}_i \in \Re^D$, and let $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]^T$ be the set of latent variables $\mathbf{x}_i \in \Re^Q$, with $Q \ll D$. We further assume a mapping from $\mathbf{X}$ to $\mathbf{Y}$ as following:

$$y = f(\mathbf{x}) + \eta \tag{2.10}$$

where y differs from the function values $f(\mathbf{x})$ by additive noise, which is assumed to be independent, identically and Gaussian distributed:

$$\eta \sim \mathcal{N}(0, \sigma_n^2)$$

By putting a GP prior over the function space, marginalization of f leads to the marginal likelihood

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X}) d\mathbf{f} \tag{2.11}$$

where $p(\mathbf{f}|\mathbf{X}) \sim \mathcal{N}(0, K)$ and $\mathbf{f}$ is a vector containing the function values of $\mathbf{X}$. Since the likelihood $p(\mathbf{y}|\mathbf{f}, \mathbf{X})$ *and* the prior $p(\mathbf{f}|\mathbf{X})$ are Gaussian, the marginal likelihood is again Gaussian distributed. More insight can be gained by comparing the *weight space view* to the *function space view*. Consider the decomposition of f into non-linear basis functions φ_i

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} \theta_i \varphi_i(\mathbf{x})$$

where θ_i are normally distributed weights $\theta_i \sim \mathcal{N}(0, \sigma_\theta^2)$. Marginalizing the weights in the weight space view corresponds to marginalizing the functions in the function space view and is straightforward, due to the conjugate property of the prior: Since $f(\mathbf{x})$ is a linear combination of Gaussian distributed weights θ_i, the resulting distribution over f is again Gaussian. Considering an instance of the GP for a finite number of *training points* N, the mean and the covariance are straightforward to calculate:

$$\mu_k = \mathcal{E}\left(\sum_{i=1}^{\infty}\theta_i\varphi_i(\mathbf{x}_k)\right) = \sum_{i=1}^{\infty}\mathcal{E}\left(\theta_i\right)\varphi_i(\mathbf{x}_k) = 0 \tag{2.12}$$

$$\Sigma_{kl} = \mathcal{E}\left(\left(\sum_{i=1}^{\infty}\theta_i\varphi_i(\mathbf{x}_k)\right)\left(\sum_{i=1}^{\infty}\theta_i\varphi_i(\mathbf{x}_l)\right)\right) \tag{2.13}$$

$$= \mathcal{E}\left(\sum_{i=1}^{\infty}\theta_i\varphi_i(\mathbf{x}_k)\theta_i\varphi_i(\mathbf{x}_l)\right) \tag{2.14}$$

$$= \sum_{i=1}^{\infty}\mathcal{E}\left(\theta_i\varphi_i(\mathbf{x}_k)\theta_i\varphi_i(\mathbf{x}_l)\right) \tag{2.15}$$

$$= \sigma_\theta^2\sum_{i=1}^{\infty}\varphi_i(\mathbf{x}_k)\varphi_i(\mathbf{x}_l) \tag{2.16}$$

$$= \sigma_\theta^2 k(\mathbf{x}_k, \mathbf{x}_l) \tag{2.17}$$

Since we are operating in an inner product space, we make use of the *kernel trick* (in the last equality): A non-degenerate kernel like the *radial basis function (RBF)*

$$k(\mathbf{x}_i, \mathbf{x}_j) = \theta_1 \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\theta_2^2}\right)$$

allows for implicitly making use of an infinite dimensional space ($i = 1, ..., \infty$) without evaluating it explicitly (which would be intractable). Here the hyperparameters $\Theta = \{\theta_1, \theta_2\}$ stand for the RBF lengthscale (or kernel height) θ_1 and the kernel width θ_2.

Putting it all together we arrive at a simple expression for equation 2.10. If we restrict the number of training points to be finite, $f(\mathbf{x})$ is Gaussian distributed. Thus we have a conjugate prior $p(\mathbf{f}|\mathbf{X})$ and y is normally distributed, too. This results in

$$p(\mathbf{y}|\mathbf{X}) \sim \mathcal{N}(0|\mathbf{K})$$

with $\mathbf{K}$ the covariance matrix defined by a *RBF + noise* kernel:

$$\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \theta_1 \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\theta_2^2}\right) + \theta_3\delta_{ij}. \tag{2.18}$$

Here $\Theta = \{\theta_1, \theta_2, \theta_3\}$ are the hyperparameters, where θ_1 is the RBF lengthscale, θ_2 the kernel width, and θ_3 the noise. The latter not only models the observation noise, but also contributes to a numerically stable inversion of the covariance $\mathbf{K}$.

The *GP key predictive equations* provide the mean $\mu(\mathbf{f}_*)$ and the covariance $\sigma^2(\mathbf{f}_*)$ for a given test point $\mathbf{x}_*$ given the training points $\{\mathbf{X}, \mathbf{Y}\}$. They are readily derived from the Gaussian conditional distribution as

$$p(\mathbf{f}_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}(\mu(\mathbf{f}_*), \sigma^2(\mathbf{f}_*))$$

where

$$\mu(\mathbf{f}_*) = \mathbf{K}_{*,f}\left[\mathbf{K}_{f,f} + \sigma_n^2\mathbf{I}\right]^{-1}\mathbf{y} \tag{2.19}$$

$$\sigma^2(\mathbf{f}_*) = \mathbf{K}_{*,*} - \mathbf{K}_{*,f}\left[\mathbf{K}_{f,f} + \sigma_n^2\mathbf{I}\right]^{-1}\mathbf{K}_{f,*} \tag{2.20}$$

Here $\mathbf{K}_{f,f}$ denotes the covariance matrix of the training points, $\mathbf{K}_{*,f}$ is the *Kernel Density Estimate (KDE)* with respect to the test point $\mathbf{f}_*$ and $\mathbf{K}_{*,*}$ is the variance of the test point. Note that Gaussian Processes always represent a projection from $\Re^Q$ to $\Re^1$, whereas this limitation is overcome by the GPLVM, introduced in section 2.3.2.

To illustrate GP regression we might first consider a collection of only 2 random variables which gives rise to a 2 dimensional Gaussian distribution as an instance of the full GP. To illustrate the shape of a Gaussian, figure 2.2 shows face plots of 2 dimensional Gaussian distributions generated from

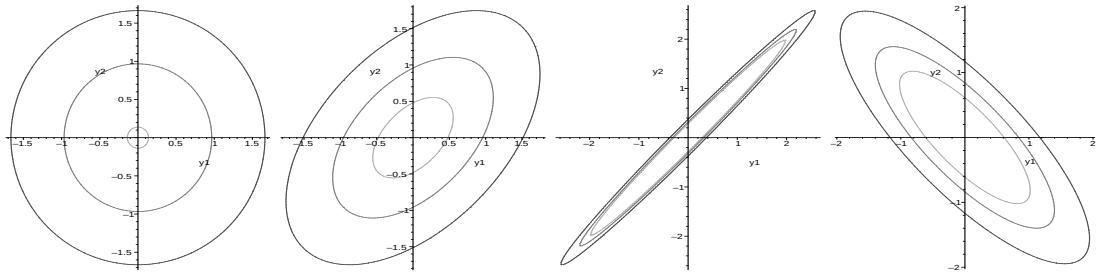
$$\mathcal{N}(0, \begin{pmatrix} 1 & \Upsilon \\ \Upsilon & 1 \end{pmatrix})$$

with different off-diagonal elements Υ. $\Upsilon = \mathbf{K}_{12} = \mathbf{K}_{21} = 0.9$ is assumed for figure 2.3. Given the training point y_1, the mean and variance for y_2 can be calculated easily using equation 2.19 and 2.20.

A slightly more complex example is shown in figure 2.4. 3 input points are given along with the covariance matrix where $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ measures the distance along the abscissa between two points. This figure shows clearly that the uncertainty increases with the distance of the test points from the training points.

Free GP toolkits for visualization of GPs, regression and classification (MATLAB and C) are available online[2].

[2]http://www.gaussianprocess.org/, http://www.rainsoft.de/projects/gausspro.html



(a) (b) (c) (d)

Figure 2.2: **Face plots of 2D Gaussians.** Here face plots of 2 dimensional Gaussians are shown for different parameters of the covariance matrix. While (a) is uncorrelated ($\Upsilon = 0$), (b) is correlated with $\Upsilon = 0.5$. (c) exhibits almost perfect linear coherence ($\Upsilon = 0.99$) whereas (d) is negatively correlated ($\Upsilon = -0.8$).

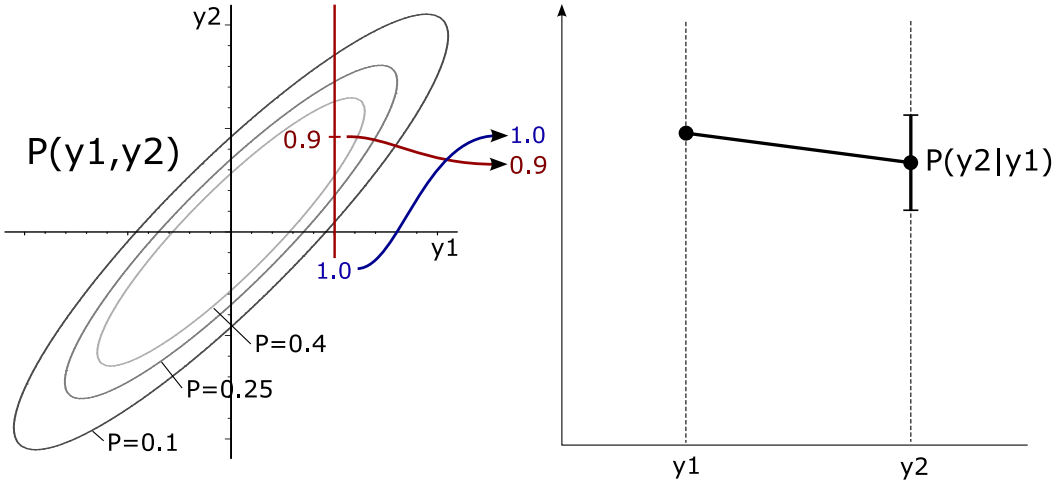


Figure 2.3: **2D illustration of Gaussian Process regression.** Here y_1 is given and the correlation between y_1 and y_2 is assumed to be known by means of a *kernel function* $k(\mathbf{x}_i, \mathbf{x}_j)$. Since y_1 and y_2 are correlated, the mean of the conditional distribution $p(y_2|y_1)$ is similar to the mean y_1 itself while the small variance indicates only little uncertainty.

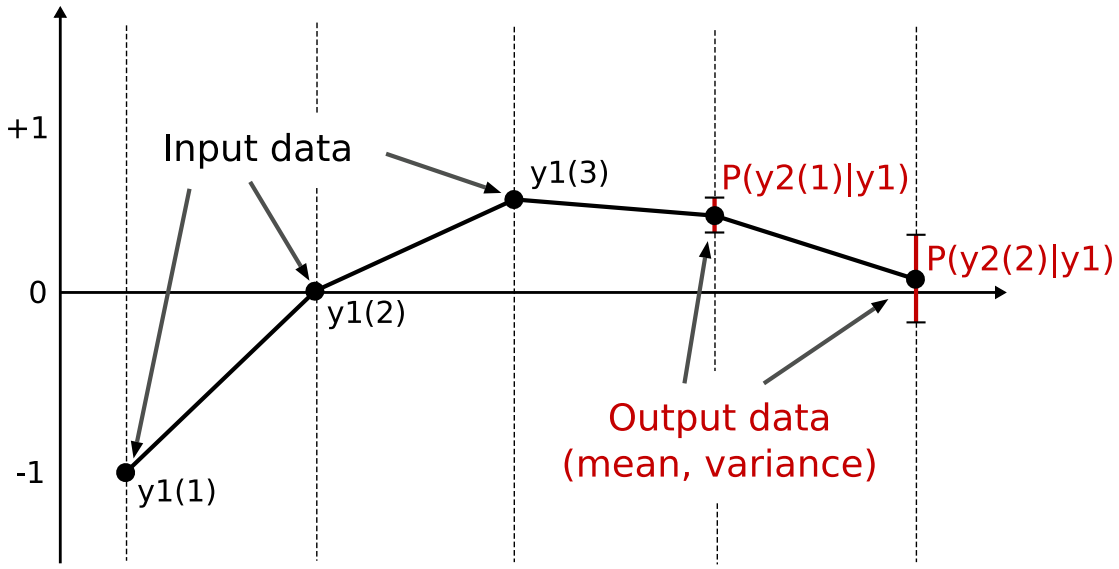


Figure 2.4: **5D illustration of Gaussian Process regression.** Here 3 training points are given compared to only one in figure 2.3. Since we have 2 test points, the regression results in a 5 dimensional Gaussian distribution. Note that uncertainty increases with the distance of a test point from the training data.

Gaussian Process Latent Variable Model

While the training points for GP regression include $\mathbf{Y}$ as well as $\mathbf{X}$, the GPLVM assumes that the latent variables $\mathbf{X}$ are unknown and optimizes the model likelihood with respect to $\mathbf{X}$ and the hyperparameters Θ in an unsupervised fashion in order to learn the latent space. Furthermore it allows for a D-dimensional data space $\mathbf{Y}$ by multiplying D single Gaussian Processes, one for each data dimension, resulting in the final GPLVM likelihood. As it turns out, starting from a good initial guess is key to avoid local minima, since the target function is non-convex.

More formally, let

$$y^{(d)} = f^{(d)}(\mathbf{x}) + \eta$$

with $y^{(d)}$ the d-th coordinate of $\mathbf{y}$, and $\eta \sim \mathcal{N}(0, \theta_3)$ iid Gaussian noise. The *Gaussian Process Latent Variable Model (GPLVM)* [Law05] places a Gaussian process prior (see section 2.3.2) over the space of mapping functions f. Marginalizing over the functions f and assuming conditional independence of the output dimensions given the latent variables results in the GPLVM likelihood

$$p(\mathbf{Y}|\mathbf{X}) = \prod_{d=1}^{D} p(\mathbf{y}^{(d)}|\mathbf{X}) = \prod_{d=1}^{D} \mathcal{N}(\mathbf{y}^{(d)}|\mathbf{0}, \mathbf{K}) \tag{2.21}$$

where $\mathbf{y}^{(d)}$ is the d-th column in $\mathbf{Y}$, and $\mathbf{K}$ is the covariance matrix, defined in terms of a *kernel function*. Here we use the RBF + noise kernel, defined in equation 2.18, since it allows for a variety of smooth, non-linear mappings using only a limited number of hyperparameters ($|\Theta| = 3$).

The log-likelihood of equation 2.21 is given as

$$\ln p(\mathbf{Y}|\mathbf{X}) = -\frac{DN}{2} \ln 2\pi - \frac{D}{2} \ln |\mathbf{K}| - \frac{1}{2} \text{trace}(\mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^T). \tag{2.22}$$

While the first term is a constant, the second term can be interpreted as a regularizer, which prevents $\mathbf{K}$ from getting too big (for example by means of high noise or a large kernel height) and the third term is the data-fitting term, which tries to maintain the relationship between points in the input space.

Learning the GPLVM is performed by maximizing the posterior

$$p(\mathbf{X}|\mathbf{Y}) \propto p(\mathbf{Y}|\mathbf{X})p(\mathbf{X})$$

with respect to the latent variables $\mathbf{X}$ and the kernel hyperparameters Θ. Here $p(\mathbf{X})$ encodes prior knowledge about the latent space $\mathbf{X}$. In [Law05] the redundancy between the overall scale of the matrix $\mathbf{X}$ and the *kernel width* is removed by chosing a product of isotropic Gaussian priors $p(\mathbf{X}) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{x}_i|0, \mathbf{I})$ which leads to a Wishart distributed scatter matrix $\mathbf{X}^T\mathbf{X}$. In practice however, this can lead to degenerate solutions (like $\mathbf{X} = \mathbf{0}$). To prevent from this behaviour, we applied constrained optimization instead and used another prior, which penalizes the dimension, as described in section 3.3. While [Law05] uses *SCG (Scaled Conjugate Gradients)* [Mol93] we made use of *SNOPT (Software for Large-Scale Non-Linear Programming)*, a constrained non-linear optimizer [GMS02] based on sequential quadratic programming.

PCA and graph-based techniques are commonly used to initialize the latent space in GPLVM-based dimensionality reduction; both offer closed-form solutions. However, PCA [Pea01] cannot capture non-linear dependencies, LLE [RS00] gives a good initialization *only* if the data points are uniformly sampled in the manifold, and ISOMAP [TSL00] has difficulties with non-convex datasets [Har07]. Generally, when initialized far from the true minimum, the GPLVM optimization can get stuck in local minima [Law05, UFG+08].

To avoid this problem, different priors over the latent space have been developed (see chapter 3).

A further issue evolves from the fact that GPLVMs rely on the calculation of the inverse of the covariance matrix, which prohibits the use of a large number of training points and will be addressed in the next section.

2.3.3 Sparsification

One of the main limitations of using Gaussian Processes in practical applications is the computational complexity of learning a model - like for many kernel methods in general. Since learning involves the inversion of a $N \times N$-matrix, computational cost[3] is approximately $O(N^3)$. This restricts applicability to cases with $N < 5000$.

While several methods for Gaussian Process sparsification have been developed [SNS06, YDD05, UD08, SW02, QCR05], it is a bit more subtle to overcome the time limitiations for GPLVMs. This is due to the fact that in each iteration the inverse of the covariance matrix $\mathbf{K}^{-1}$ is needed for calculating the gradient of equation 2.22 with respect to $\mathbf{X}$. One way of increasing the speed is to simply remove points from the training set by subsampling or applying information criterions like the one proposed in the *Informative Vector Machine (IVM)* [LSH02]. If further speed-up is needed without rejecting training points from the dataset, one has to resort to more sophisticated techniques, like the ones described in the following.

Sparse Gaussian Process Approximations

A unifying view, which includes many existing probabilistic sparse approximations for Gaussian Process regression has been developed in [QCR05]. The approach relies on expressing several methods in terms of the so-called *effective prior*. Rather than the interpretation *"approximate inference with exact prior"* the methods are reinterpreted as *"exact inference with an approximated prior"*.

Let us first consider inference for GP regression. Since

$$p(\mathbf{f}, \mathbf{f}_*|\mathbf{y}) = \frac{p(\mathbf{f}, \mathbf{f}_*)p(\mathbf{y}|\mathbf{f}_*, \mathbf{f})}{p(\mathbf{y})} \quad \text{and} \quad p(\mathbf{y}|\mathbf{f}_*, \mathbf{f}) = p(\mathbf{y}|\mathbf{f})$$

inference can be done by marginalizing out the unwanted training variables

$$p(\mathbf{f}_*|\mathbf{y}) = \int p(\mathbf{f}, \mathbf{f}_*|\mathbf{y})d\mathbf{f} = \frac{1}{p(\mathbf{y})} \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}, \mathbf{f}_*)d\mathbf{f} \tag{2.23}$$

[3]The *Strassen algorithm* can solve the matrix inversion problem in $O(N^{2.807})$. The fastest known algorithm - derived by *Coppersmith and Winograd* - lies in $O(N^{2.376})$, but has huge constants hidden in the O notation and is therefore not used in practice.

where $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma_\eta^2 \mathbf{I})$ is the likelihood, $p(\mathbf{f}, \mathbf{f}_*)$ is the conjugate joint prior and $\frac{1}{p(\mathbf{y})}$ is a normalization constant. The predictive distribution $p(\mathbf{f}_*|\mathbf{y})$ is Gaussian with the first and second central moment given by equation 2.19 and equation 2.20.

In order to reduce computational requirements from 2.23, several approximations to the joint prior $p(\mathbf{f}_*, \mathbf{f})$ are reviewed in [QCR05]. The prior has been rewritten by introducing another set of M latent variables $\mathbf{u} = [u_1, ..., u_M]^T$ which are called *inducing variables*. They are values of the Gaussian process, corresponding to a set of input locations $X_\mathbf{u}$, which are called *inducing inputs*. Since GPs exhibit the marginalization property, the joint GP prior $p(\mathbf{f}_*, \mathbf{f})$ can be rewritten as the marginal

$$p(\mathbf{f}_*, \mathbf{f}) = \int p(\mathbf{f}_*, \mathbf{f}, \mathbf{u}) d\mathbf{u} = \int p(\mathbf{f}_*, \mathbf{f}|\mathbf{u}) p(\mathbf{u}) d\mathbf{u}$$

where $p(\mathbf{u}) = \mathcal{N}(0, \mathbf{K}_{\mathbf{u},\mathbf{u}})$. The approximation is then done by assuming that $\mathbf{f}_*$ and $\mathbf{f}$ are conditionally independent given $\mathbf{u}$, such that

$$p(\mathbf{f}_*, \mathbf{f}) \simeq \int p(\mathbf{f}_*|\mathbf{u}) p(\mathbf{f}|\mathbf{u}) p(\mathbf{u}) d\mathbf{u}.$$

The different computationally efficient algorithms proposed in the literature correspond to different *additional* assumptions about the two approximate inducing conditionals $p(\mathbf{f}_*|\mathbf{u})$ and $p(\mathbf{f}|\mathbf{u})$. Applying sparse Gaussian Process approximation reduces the computational complexity from $O(N^3)$ to $O(NM^2)$ where N is the number of training points and M is the number of inducing variables. Further details as well as the individual approximation schemes are discussed in [QCR05].

In [Law07] the method reviewed in [QCR05] was transferred to the Gaussian Process Latent Variable Model and the optimization was performed with respect to $\mathbf{X}$, $\mathbf{X}_u$ and Θ. While being an elegant way to reduce computational complexity, introducing additional variables ($\mathbf{X}_u$) into the model also makes the optimization prone to overfitting and introduces local minima. In practice, the inducing variables also tend to be placed at non-intuitive locations after optimization which has to be further investigated.

Nyström Method

The *Nyström Method* [WS01] computes an approximation to the eigendecomposition of the Gram matrix $\mathbf{K}$. It is done by carrying out an eigendecomposition on a smaller system of size $M < N$ and then expanding the result back to N dimensions. The computational complexity therefore reduces to $O(NM^2)$.

Considering the *kernel function*

$$k(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{R} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{y})$$

where $R \leq \infty$ and λ_i are the eigenvalues to the eigenfunctions ϕ_i, such that

$$\int k(\mathbf{x}, \mathbf{y}) \phi_i(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} = \lambda_i \phi_i(\mathbf{y})$$

where $p(\mathbf{x})$ denotes the probability density of the input vector $\mathbf{x}$. The integral over $p(\mathbf{x})$ is replaced by an empirical average (which is called the *Nyström Method*) to obtain

$$\frac{1}{q} \sum_{k=1}^{q} k(\mathbf{y}, \mathbf{x}_k) \phi_i(\mathbf{x}_k) \approx \lambda_i \phi_i(\mathbf{y}).$$

This motivates the matrix eigenproblem

$$\mathbf{KU} = \mathbf{U\Lambda}.$$

Computational costs of computing $\mathbf{K}$ can now be cut down by using the eigendecomposition $\mathbf{K} = \mathbf{U\Lambda U}^T$ where $\mathbf{U}$ is orthonormal and $\mathbf{\Lambda}$ a diagonal matrix with the eigenvalues of $\mathbf{K}$. Williams et al approximate $\mathbf{K} + \sigma\mathbf{I}$ by $\mathbf{U\Lambda U}^T + \sigma\mathbf{I}$. Furthermore they show that the eigenvalues and eigenvectors which are needed can be approximated by

$$\lambda_i^{(\mathbf{N})} \approx \frac{\mathbf{N}}{\mathbf{M}} \lambda_i^{(\mathbf{M})} \tag{2.24}$$

$$\mathbf{u}_i^{(\mathbf{N})} \approx \sqrt{\frac{\mathbf{M}}{\mathbf{N}}} \frac{1}{\lambda_i^{(\mathbf{M})}} \mathbf{K}_{\mathbf{N},\mathbf{M}} \mathbf{u}_i^{(\mathbf{M})} \tag{2.25}$$

where $\left\{(\lambda_i^{(\mathbf{N})}, \mathbf{u}_i^{(\mathbf{N})})\right\}_{i=1}^{N}$ are eigenvalues and eigenvectors of the full kernel matrix $\mathbf{K}$ and $\left\{(\lambda_i^{(\mathbf{M})}, \mathbf{u}_i^{(\mathbf{M})})\right\}_{i=1}^{M}$ are eigenvalues and eigenvectors of a $M \times M$ submatrix of $\mathbf{K}$.

If the eigenvalues of $\mathbf{K}$ decay rapidly, good approximations can be obtained by using $M \ll N$. However there is no clear answer to the choice of the subset of points which give rise to the $M \times M$ submatrix of the $N \times N$ matrix $\mathbf{K}$.

Furthermore, using our datasets, it turned out that the eigenvalues of $\mathbf{K}$ didn't decay as quickly as needed for using the Nyström method, mainly because of a large kernel width.

Therefore we employed the simple method described next to speed up learning of larger motion datasets.

Product of local and global Gaussian Processes

The method proposed here is simple to implement and based on the assumption that topological knowledge about the latent points is given. More precisely, it is sufficient to know local neighborhoods. Thus this method can be applied when dealing with motion capture data, for example.

Instead of mapping the full dataset from the latent space $\mathbf{X}$ to the data space $\mathbf{Y}$, different subsets of $\mathbf{X}$ are mapped to different subsets of $\mathbf{Y}$. This results in a product of products of Gaussian Processes, or a product of GPLVMs (see equation 2.21). The likelihood of the sparse GP can be written as

$$p(\mathbf{Y}|\mathbf{X}) \approx \prod_{s \in L \cup G} \prod_{d=1}^{D} \mathcal{N}(\mathbf{y}_s^{(d)}|\mathbf{0}, \mathbf{K}_s) \tag{2.26}$$

where L is a set of indices capturing overlapping local neighborhoods along the motion trajectory and G is a set of indices which puts points into global relation to each other. The idea is illustrated in figure 2.5. The resulting computational complexity is $O(M^3)$ with $M = \max_{s \in L \cup G} |s|$. Note that this is not a proper sparsification technique in a probabilistic sense, but rather serves as a reasonable ad-hoc approximation to the gradients needed for the GPLVM.

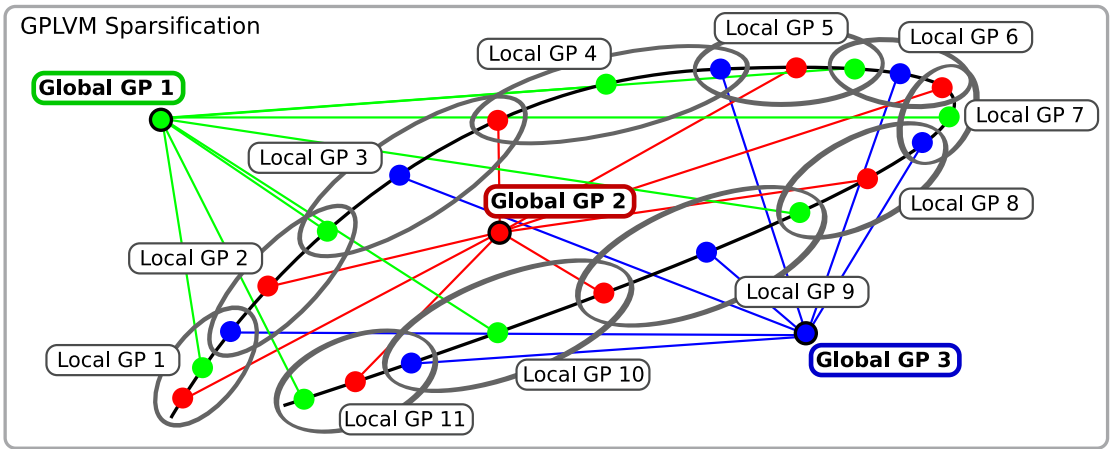


Figure 2.5: **Sparsification: Global and local GPs.** This figure illustrates the sparsification method used for motion datasets with a large number of training points. The colored points are training points, lying on a motion trajectory (black) in the data space. Instead of inverting the full Gaussian Process covariance matrix (which takes $O(N^3)$, with N the number of training points), we make use of prior knowledge about the nature of motion data, i.e. we assume smoothness over time (or frames). Thus only a set of smaller *local GPs* (which overlap with neighboring sets, shown in gray) is calculated, reducing the computation time dramatically. To also keep track of global relationships, each point in a local GP set is linked to one point in each other local GP set via a *global GP* (which thus again consists of a very limited number of training points, contributing to a speed-up in learning the model). In this example 3 global GPs are depicted (red, green and blue).

3 Priors for Latent Variable Model

To overcome the problem of local minima and to learn smooth latent spaces for human motion, different priors over the latent space have been developed. In [WFH08] a prior was introduced in the form of a Gaussian process over the dynamics in the latent space (see section 3.1). This results in smoother manifolds but performs poorly when learning stylistic variations of a motion or multiple motions. Urtasun et al. [UFG+08] proposed a prior over the latent space, inspired by the LLE cost function, that encourages smoothness and allows the introduction of prior knowledge, e.g., topological information about the manifold (see section 3.2). However, such prior knowledge is not commonly available, reducing considerably the applicability of their technique. In contrast, the method developed in section 3.3 below introduces a generic prior that requires no specific prior knowledge, directly penalizing the dimensionality of the latent space to learn effective low-dimensional representations.

3.1 Dynamical Priors

Dynamical Priors, as introduced in the *Gaussian Process Dynamical Model (GPDM)* [WFH08], assume knowledge about the dynamics in the latent space, i.e. that data points have been recorded consecutively with a constant frame rate. They thus employ a model where an additional mapping within the latent space captures a first-order Markov dynamics

$$\mathbf{x}^{(t+1)} = g(\mathbf{x}^{(t)}) + \boldsymbol{\eta}_{\mathbf{x}}^{(t)} \tag{3.1}$$

$$\mathbf{y}^{(t)} = f(\mathbf{x}^{(t)}) + \boldsymbol{\eta}_{\mathbf{y}}^{(t)} \tag{3.2}$$

where t represents a discrete time index. The prior $p(\mathbf{X})$ is then given as

$$p(\mathbf{X}) = p(\mathbf{x}^{(1)}) \int \prod_{t=2}^{N} p(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}, \mathbf{g}) p(\mathbf{g}) d\mathbf{g}.$$

In [WFH08] a GP prior and a *linear + RBF* kernel was used for modeling the dynamics mapping. They show that incorporating the dynamics prior into the GPLVM likelihood results in smoother latent trajectories which can be used for better motion generation or articulated body tracking. However, when the training data contains large stylistic variations and multiple motions, the generic GPDM and the back-constrained GPDM [UFG+08] do not produce useful models.

3.2 Topological Priors

In [UFG+08] we introduced *Topologically-Constrained Latent Variable Models* which incorporate a prior based on knowledge about walk cycle phases to align motions and separate style (e.g. weak vs. exaggerated arm movement) from content (the walking cycle itself). Furthermore when switching between styles (e.g. walking vs. running) a transition is restricted to be possible only when at least one foot touches the ground. The application is graphics animation, where an animator specifies foot constraints, for example, and a motion is generated from a specific model with respect to the specified constraints.

In order to do so we inspired ourselves by the *Locally Linear Embedding (LLE)* NLDR method (see section 2.3.1). Our proposed prior is given by

$$p(\mathbf{X}|\mathbf{W}) \propto \exp\left(-\frac{1}{\sigma^2}\sum_{i=1}^{N}\sum_{j=1}^{D}\|X_{ij}-\sum_{k\in\eta_i} w_{ik}^{(j)} X_{ij}\|^2\right)$$

where $\mathbf{W}$ is the weight matrix, σ^2 represents a global scaling of the prior and η_i is the local neighborhood of data point i. In LLE the weights are computed by solving $\forall j \forall k \in \eta_i$

$$\sum_l C_{lk} w_{ik}^{(j)} = 1$$

where $C_{lk}^{sim} = (\mathbf{y}_i - \mathbf{y}_l)^T(\mathbf{y}_i - \mathbf{y}_k)$ if $l, k \in \eta_i$, and 0 otherwise.

To incorporate transitions, the covariance matrix C can be modified as

$$C_{ij}^{trans} = 1 - \left[\delta_{ij}\exp\left(-\zeta(t_i - t_j)^2\right)\right]$$

with ζ a constant, and $\delta_{ij} = 1$ if t_i and t_j are in the same set $\left\{\hat{t}_k\right\}_{k=1}^{M}$ and otherwise $\delta_{ij} = 0$. Here $\left\{\hat{t}_k\right\}_{k=1}^{M}$ is defined as a subset of frames where transitions are possible. This covariance encourages the points at which transitions are physically possible to be close together in the latent space.

Cyclic topologies can be enforced by introducing covariances based on the phase, specifying different covariances for each latent dimension:

$$C_{jk}^{cos} = (cos(\phi_i) - cos(\phi_j))(cos(\phi_i) - cos(\phi_k)) \quad (3.3)$$

$$C_{jk}^{sin} = (sin(\phi_i) - sin(\phi_j))(sin(\phi_i) - sin(\phi_k)) \quad (3.4)$$

We also show, that it is possible to use the *back-constrained GPLVM* [LQC06] to incorporate prior knowledge by modifying the kernel function. Experiments show, that the learned models are smooth and exhibit desirable properties for

character animation, for example. However, the prior knowledge needed is not always available. While the cyclic assumption is reasonable for walking or running motions, motions in general do not exhibit this structure, thus violating the assumption (consider for example arbitrary motions performed in a kitchen scenario as discussed in section 4.3 of this thesis, see figure 4.14). The *Rank Prior* introduced in the next section seeks for a viable embedding without making additional assumptions about the dynamics or the topology of the dataset.

3.3 Rank Priors

Here we introduce a novel method for probabilistic non-linear manifold learning which avoids the initial distortion induced by ad-hoc initialization. The idea is illustrated in the figures 3.1 and 3.2. While figure 3.1 describes the general idea motivated by a spring-force system, figure 3.2 shows the actual algorithm when applied to a spiral manifold embedded in two dimensions.

In our method we initialize the latent space to the high-dimensional observation space and define a *Rank Prior* which favors latent spaces with low dimensionality. Minimizing the dimensionality of the latent space is equivalent to minimizing $\mathcal{R}$, the rank of the unbiased covariance estimate (along the dimensions) of $\tilde{\mathbf{X}}$

$$\mathcal{R} = rank\left(\frac{1}{N-1}\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}\right) \tag{3.5}$$

where $\tilde{\mathbf{X}}$ denotes the mean subtracted latent variables $\tilde{\mathbf{X}} = \mathbf{X} - \overline{\mathbf{X}}$ and $\overline{\mathbf{X}}_{ij} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{X}_{nj}$.

3.3.1 Rank Prior Formulation

The cost function in Eq. (3.5) is discrete and thus difficult to minimize. However, it can be expressed in terms of singular values and singular vectors of the covariance matrix along the dimensions. Let the singular value decomposition (SVD) of $\tilde{\mathbf{X}}$ be $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are matrices containing the left and right singular vectors, and $\mathbf{\Sigma}$ comprises the singular values $\{\sigma_1(\tilde{\mathbf{X}}), ..., \sigma_D(\tilde{\mathbf{X}})\}$ on its diagonal. Then $\tilde{\mathbf{X}}^T\tilde{\mathbf{X}} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$, and $\mathcal{R}$ can be minimized by minimizing the number of non-zero singular values of $\tilde{\mathbf{X}}$. This connection can also be drawn by comparing the null spaces of $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}$, leading to the insight that $rank\left(\tilde{\mathbf{X}}\right) = rank\left(\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}\right)$. Reducing the dimensionality of $\tilde{\mathbf{X}}$ is equivalent to reducing the rank of $\tilde{\mathbf{X}}$ or the number of non-zero singular values.

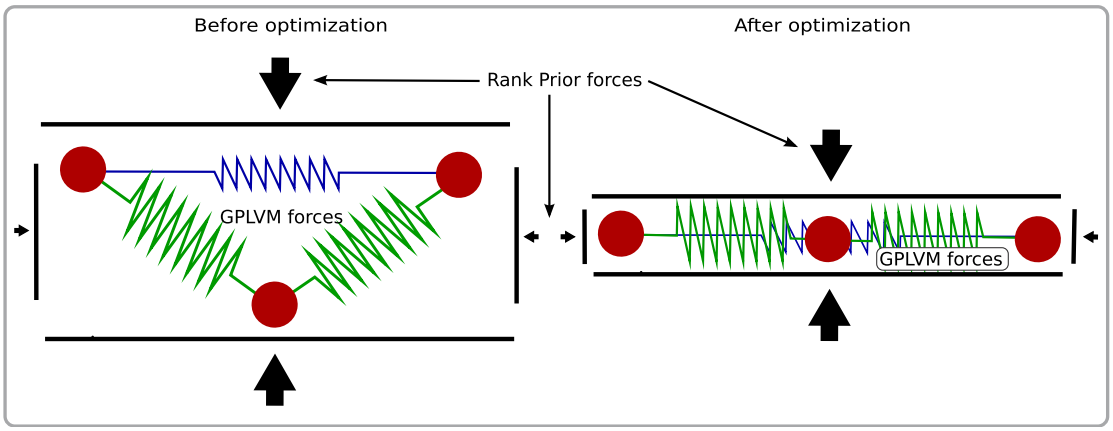


Figure 3.1: **Illustration using a spring-force system.** From a simplified viewpoint the method can be seen as a system of observation points (red) in the high-dimensional data space, connected by springs (green and blue) of strengths defined according to the distance of the two points they are connnecting, respectively. Thus the spring strength can be defined by a radial basis function, for example. Here, e.g., the blue spring is weaker than the two green springs since the data points it connects are further apart from each other. Before applying the algorithm, the springs are in their *home position*, imposing no force. The spring system stands for the GPLVM, which preserves dissimilarities. On the other hand, the pressure plates surrounding the system apply forces towards the covariance directions. They represent the Rank Prior which favors low-dimensional spaces. Here the pressure along the smaller variance is strongest since we want to get rid of the *"smaller"* dimensions first. This is achieved by using a non-linear penalty function, as described in section 3.3.3.

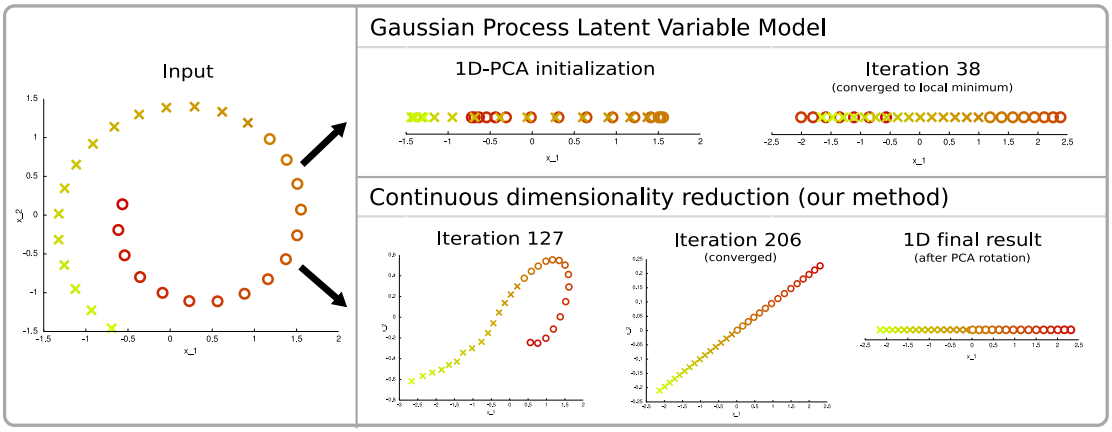


Figure 3.2: **Illustration of our Rank Prior with a GPLVM.** The goal is to recover the 1D manifold embedded in 2D. The GPLVM gets stuck in local minima very early (upper row) since PCA initialization does not capture non-linear dependencies, whereas our method decreases dimensionality gradually and recovers the correct manifold (lower row).

We transform the discrete optimization criteria in Eq. (3.5) into a continuous one by introducing a *sparsity penalty on the singular values*. In particular we introduce a prior of the form

$$p(\mathbf{X}) = \frac{1}{Z} \exp\left(-\sum_{m=1}^{D} \varphi(\hat{\sigma}_m(\tilde{\mathbf{X}})) \right) \tag{3.6}$$

where $\hat{\sigma}_m(\tilde{\mathbf{X}}) = \frac{1}{\sqrt{N-1}}\sigma_m(\tilde{\mathbf{X}})$ is the normalized singular value of $\tilde{\mathbf{X}}$, φ is a sparsity penalty function, and $Z = \int \exp\left(-\sum_{m=1}^{D} \varphi(\hat{\sigma}_m(\tilde{\mathbf{X}}))\right) d\tilde{\mathbf{X}}$ a normalization constant[1].

One might consider a variety of *sparsity penalty functions*. The identity function $\varphi(\hat{\sigma}) = \hat{\sigma}$ results in the L_1-norm since the singular values are always positive. Of particular interest to us are functions that drive small singular values faster towards 0 than larger ones. Examples of such functions are the logarithmic $\varphi(\hat{\sigma}) = \alpha \ln(1 + \beta\hat{\sigma}^2)$ and the sigmoid $\varphi(\hat{\sigma}) = \alpha(1 + \exp(-\beta(\hat{\sigma} - \gamma)))^{-1}$ functions, with α, β and γ constant parameters.

Minimizing the negative log posterior $-\ln p(\mathbf{Y}|\mathbf{X}) - \ln p(\mathbf{X})$ results in an optimization that reduces the dimensionality in a continuous fashion:

$$\min_{\mathbf{X}} \left[\frac{D}{2} \ln |\mathbf{K}(\mathbf{X}, \Theta)| + \frac{D}{2} tr(\mathbf{K}(\mathbf{X}, \Theta)^{-1}\mathbf{Y}\mathbf{Y}^T) + \sum_{m=1}^{D} \varphi(\hat{\sigma}_m(\tilde{\mathbf{X}})) \right] \quad s.t.\ \Delta_E = 0 \tag{3.7}$$

[1]Please note that the fact that this is an improper prior has no impact in the optimization since it acts as a constant when minimizing the negative log posterior.

where

$$\Delta_E = |E(\tilde{\mathbf{X}}) - E(\tilde{\mathbf{Y}})|$$

is the difference between the energies of the spectrum of the mean subtracted latent coordinates $\tilde{\mathbf{X}}$ and the mean subtracted observations, $\tilde{\mathbf{Y}} = \mathbf{Y} - \bar{\mathbf{Y}}$, with $\bar{\mathbf{Y}}$ the mean of the observations. This constraint keeps the overall energy of the system constant during optimization and thus removes the redundancy between the overall scale and the *kernel width*. The energy of the spectrum at the current iteration is defined as

$$E(\tilde{\mathbf{X}}) = \sum_{m=1}^{D} \hat{\sigma}_m^2(\tilde{\mathbf{X}})$$

while the energy of the initial spectrum equals the energy of the data points' spectrum:

$$E(\tilde{\mathbf{Y}}) = \sum_{m=1}^{D} \hat{\sigma}_m^2(\tilde{\mathbf{Y}}).$$

Equation 3.7 is straightforward to optimize, the required derivatives are given below.

3.3.2 Derivatives

The derivative of the GPLVM log likelihood (equation 2.22) with respect to the kernel matrix $\mathbf{K}$ is given by

$$\frac{\partial \ln p(\mathbf{Y}|\mathbf{X})}{\partial \mathbf{K}} = \frac{1}{2}\mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^T\mathbf{K}^{-1} - \frac{D}{2}\mathbf{K}^{-1}$$

and can be composed with

$$\frac{\partial k(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_i} = -\frac{\theta_1}{\theta_2^2}(\mathbf{x}_i - \mathbf{x}_j)\exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\theta_2^2}\right) \tag{3.8}$$

$$\frac{\partial k(\mathbf{x}_i, \mathbf{x}_j)}{\partial \theta_1} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\theta_2^2}\right) \tag{3.9}$$

$$\frac{\partial k(\mathbf{x}_i, \mathbf{x}_j)}{\partial \theta_2} = \frac{\theta_1\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\theta_2^3}\exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\theta_2^2}\right) \tag{3.10}$$

$$\frac{\partial k(\mathbf{x}_i, \mathbf{x}_j)}{\partial \theta_3} = \delta_{ij} \tag{3.11}$$

via the chain rule

$$\frac{\partial f(\mathbf{K})}{\partial X_{ij}} = trace\left[\left(\frac{\partial f(\mathbf{K})}{\partial \mathbf{K}}\right)^T \frac{\partial \mathbf{K}}{\partial X_{ij}}\right]$$

The derivative of the rank prior is also easy to calculate. Since $\tilde{\mathbf{X}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ we have $\mathbf{U}^T\tilde{\mathbf{X}}\mathbf{V} = \mathbf{\Sigma}$. Thus, since $\mathbf{\Sigma}$ is diagonal, the derivative of the rank prior with respect to the latent coordinates follows as

$$\frac{\partial}{\partial X_{ij}} \sum_{m=1}^{D} \varphi(\hat{\sigma}_m) = \frac{1}{\sqrt{N-1}} \sum_{m=1}^{D} \frac{\partial \varphi(\hat{\sigma}_m)}{\partial \hat{\sigma}_m} U_{im} V_{jm} \ , \tag{3.12}$$

where $\frac{\partial \varphi(\hat{\sigma}_m)}{\partial \hat{\sigma}_m}$ depends on the sparsity function and U_{im} and V_{jm} are elements of the respective singular vectors.

We use the SNOPT [GMS02] non-linear constraint optimizer to minimize Eq. (3.7). After optimization, we select the latent dimension as

$$Q = \text{argmax}_m \frac{\hat{\sigma}_m}{\hat{\sigma}_{m+1} + \epsilon}$$

where $\epsilon \ll 1$, and $\hat{\sigma}_1 \geq \hat{\sigma}_2 \cdots \geq \hat{\sigma}_D$. Alternatively one could also threshold the singular value spectrum.

3.3.3 Putting it all together: the Algorithm

The final steps consist of applying PCA in the optimized Q-dimensional space and optimizing $p(\mathbf{Y}|\mathbf{X}, \Theta)$ with respect to the hyperparameters Θ. Note that the mapping is still non-linear since PCA is performed in the latent space, not in the observation space, and simply rotates the data to produce the most compact Q-dimensional representation.

An overview of the complete algorithm is given in the following.

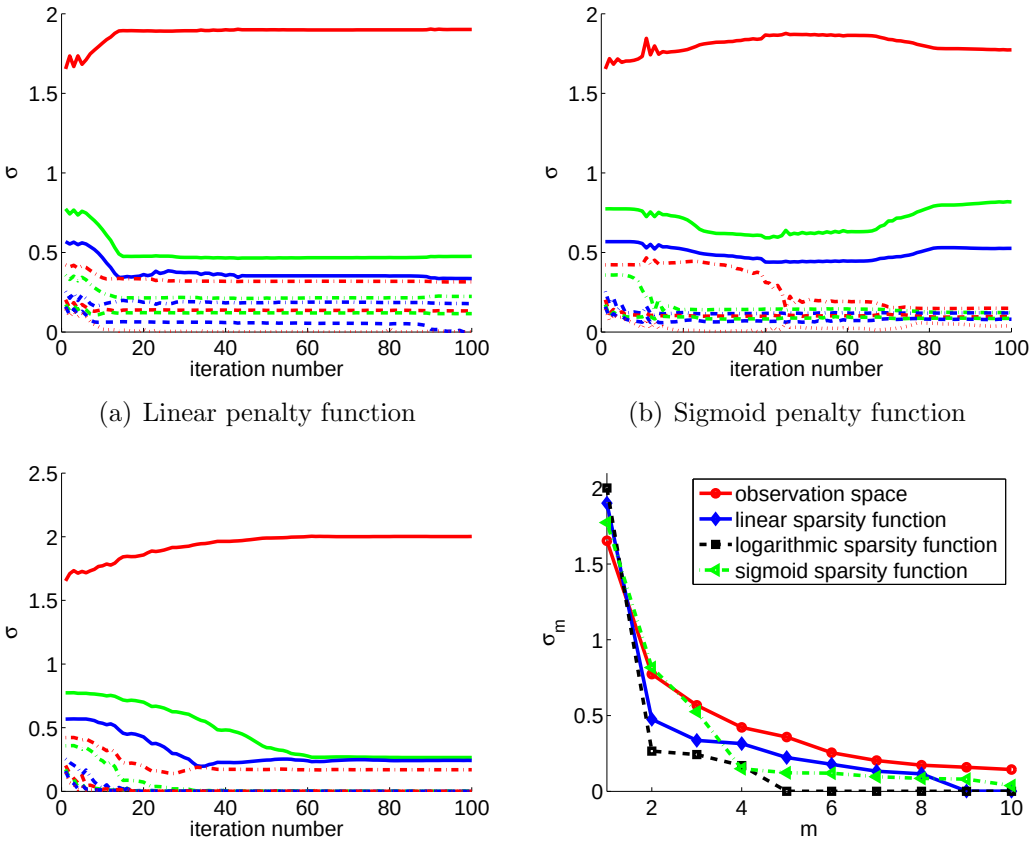
Algorithm 1 Continuous Non-Linear Dimensionality Reduction

1: Normalize $\mathbf{Y}$ and set $\mathbf{X} = \mathbf{Y}$
2: Select parameters (Θ, α, β, γ)
3: Optimize equation 3.7 with respect to $\mathbf{X}$
4: Apply PCA to $\mathbf{X}$ for final rotation (this reduces Q)
5: Optimize equation 2.22 with respect to Θ while keeping $\mathbf{X}$ fixed

Figure 3.2 compares the GPLVM (initialized with PCA) with the result of optimizing Eq. (3.7) on a toy example where a 1D manifold is embedded in 2D space. PCA provides a non-optimal initialization, and the GPLVM gets trapped in local minima whereas our method recovers the correct structure. Note that our final

PCA projection rotates the latent space and results in a 1D manifold. In this example, using spectral methods could lead to a successful initialization for the GPLVM. However, for more complex datasets this is not necessarily the case in general, as shown in the *examples* section in Figs. 4.1 and 4.15.

Figure 3.3 (a),(b) and (c) depicts the evolution of the first ten singular values when optimizing Eq. (3.7) with a linear, sigmoid and logarithmic sparsity penalty function respectively (for a motion database composed of 30D observations). Note how our method drops dimensions as the optimization evolves (i.e., the smallest singular values drop to zero within the first few iterations). A comparison of the spectrum of different sparsity penalty functions is shown in figure 3.3 (d). The L_1-norm results in a poor estimation of the dimensionality, while the more aggressive sigmoid and logarithmic functions are able to recover the correct dimensionality in this example. In the remainder of the paper we use the logarithmic function since it converges faster than the L_1-norm and has fewer parameters than the sigmoid. Since the final result often depends strongly on the sparsity penalty function and its parameters further investigation will be needed concerning this choice.



(a) Linear penalty function

(b) Sigmoid penalty function

(c) Logarithmic penalty function

(d) Comparing the resulting spectrum

Figure 3.3: **Spectrum of a 30D motion database.** (a),(b) and (c): Evolution of the first ten singular values as a function of the optimization iteration number for a linear, sigmoid and logarithmic sparsity penalty functions. (d): Spectrums learned after convergence by different sparsity penalty functions compared to the observation space spectrum (red).

4 Experimental results

In this section we demonstrate our approach in three different scenarios. In section 4.1 we compare our method to graph-based techniques and the GPLVM with different initializations in artificial data (a sparsely sampled swiss roll). We further illustrate our method's ability to estimate the latent space dimensionality in complex synthetic data in section 4.2. Finally we present an application of our technique to the challenging problem of tracking and classifying 3D articulated human body motion in the kitchen domain (see section 4.3).

4.1 A sparsely sampled swiss roll

The swiss roll is a widely used example of a 2D manifold which is embedded in a 3D space. Since the manifold is embedded in the data space in a non-linear fashion, PCA [Pea01] fails to recover it. However, many state-of-the-art graph-based techniques (like ISOMAP [TSL00], Laplacian Eigenmaps [BN03], Locally Linear Embedding [RS00], Local Tangent Space Alignment [ZZ03] and Maximum Variance Unfolding [WS06]), which rely on local neighborhoods and have a closed-form solution, can be used to unravel it correctly if the data is homogenously sampled, the noise is small, and the neighborhood size is selected appropriately. However, real data often violates these assumptions resulting in poor performance. Imagine for example a motion dataset: Using a high-frequency motion capturing system, many training points can be obtained from a motion along its trajectory in the data space, but the sampling is very sparse in the direction of different styles. This explains the need for more sophisticated methods than the spectral methods mentioned above when it comes to handle complex and sparse datasets like human motion.

Therefore we have created an example where only a very sparse and noisy subset of the swiss roll is assumed to be known to the algorithm (see figure 4.1a, points in black). The goal is to recover the correct underlying manifold (depicted on the left side). We then compare the reconstruction of the test points in 2D and their mean reprojection in 3D to the ground truth (figure 4.1a).

The first row in figure 4.1 (b) shows the result of applying PCA, Isomap, Laplacian Eigenmaps, LLE, LTSA and MVU (see [vdMPvdH07] for a review on these techniques). The second row depicts our technique and the result of optimizing

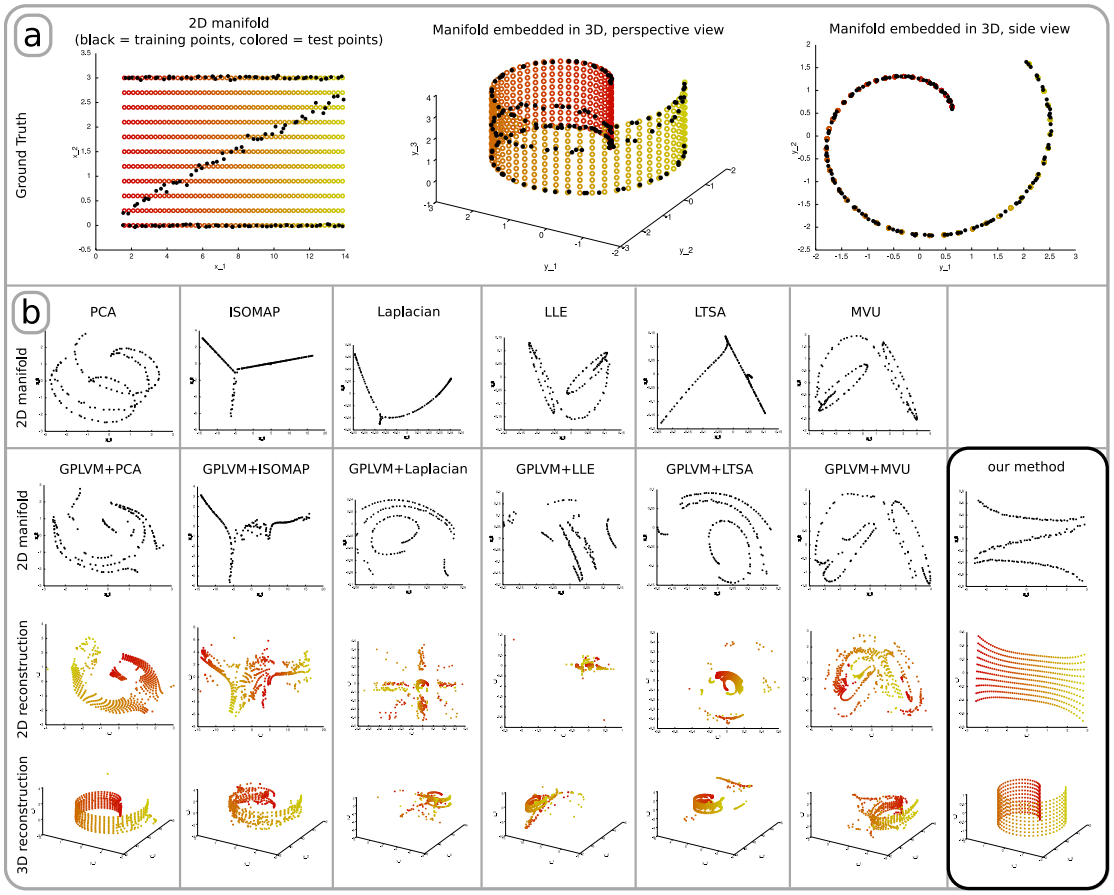


Figure 4.1: **Finding a 2D manifold in 3D space on a sparsely sampled swiss roll.** Only a sparse, noisy subset (depicted in black) of the full manifold is assumed to be known (a). (b) shows the initialization (with neighborhood size k=6), GPLVM result and 2D/3D reconstruction of the full manifold (from top to bottom).

the GPLVM with different initializations. Finally the last two rows of figure 4.1 (b) show the test data (i.e., colored samples) reconstructed in the latent space and in the original space.

The results of our method are shown in the last column of figure 4.1b. To visualize our algorithm, figure 4.2 shows how the latent space (which in our case is initialized to the data space) evolves over time.

Note that our method, unlike PCA, graph-based techniques and the GPLVM with any of the initializations, is able to recover the correct manifold. The bending effect at the boundaries of the manifold (which causes the manifold to spread at the red and yellow end) is explained by the fact that the GPLVM preserves global distances (the inner points are close to the outer points) rather than only local relationships between data points next to each other.

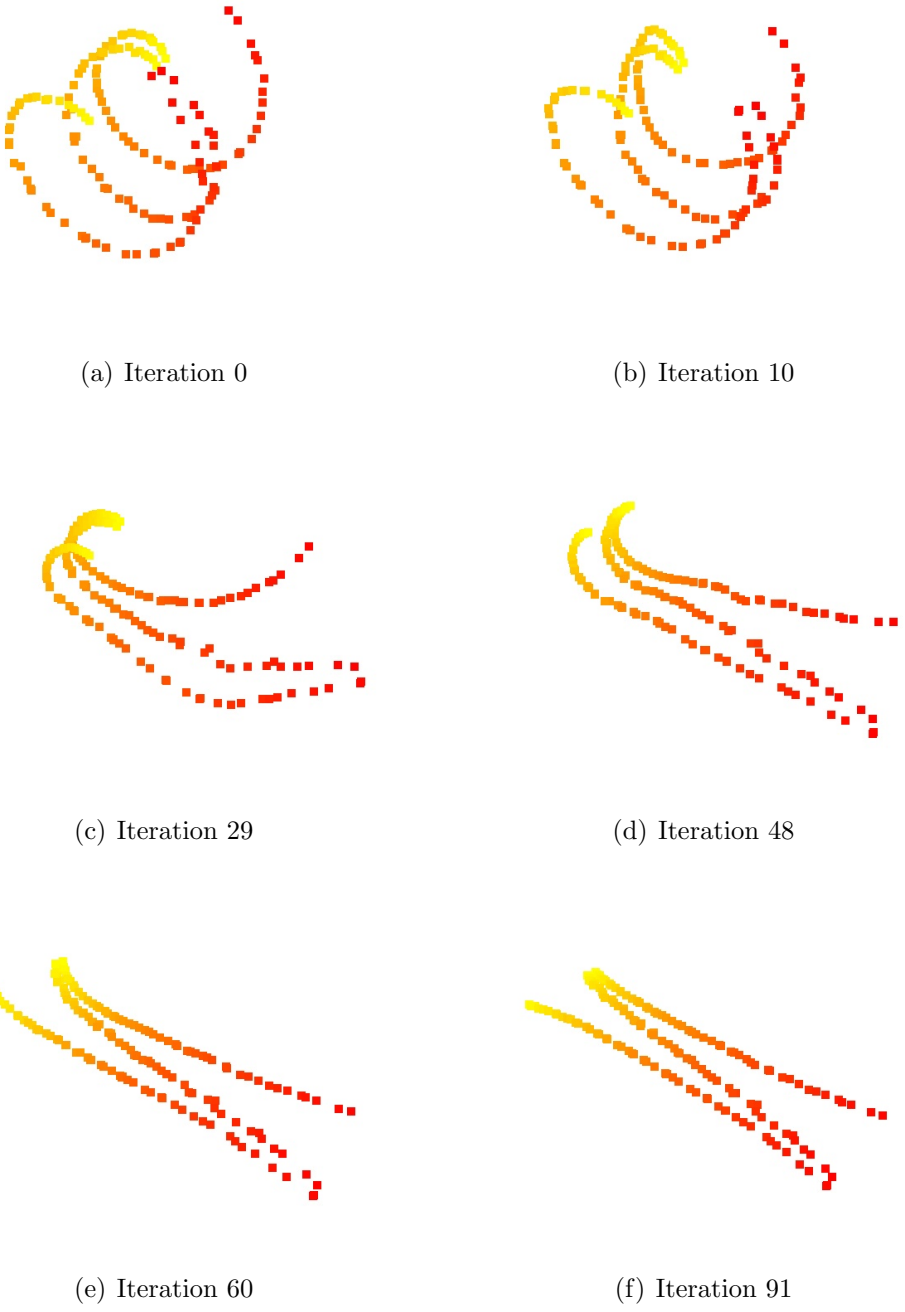
We evaluate the performance of the different algorithms on this example computing a global and a local measure of accuracy. The *reconstruction error* is a global measure of the ability to generalize, and was obtained by first finding the latent coordinates $\mathbf{x}^*$ of the test data $\mathbf{y}^*$ by maximizing $p(\mathbf{x}^*|\mathbf{y}^*, \mathbf{X}, \mathbf{Y})$, and then computing the average mean prediction error

$$\frac{1}{N_t} \sum_i \|\mu(\mathbf{x}_i^*) - \mathbf{y}_i^*\|_2$$

with N_t the number of test data. The *relationship error*, R_{error}, measures how well local neighborhoods are preserved and is defined as

$$R_{error} = \sum_{i=1}^{N_t} \sum_{j \in \eta_i} \left(\Gamma_{ij} - \bar{\Gamma}_{ij}\right)^2$$

where η_i is the set of neighbors of the i-th test data, $\Gamma_{ij} = \frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2}{\|\mathbf{y}_i - \mathbf{y}_j\|_2}$ is the ratio between the distance in the latent space and the distance in the observation space for two neighbors, and $\bar{\Gamma}_{ij}$ is the mean ratio in the local neighborhood. figure 4.3 and figure 4.4 depict these two error measures when performing the experiment in figure 4.1 averaged over 20 random partitions of the data. We use a local neighborhood of size 4 to compute the relationship error in all experiments, and a logarithmic sparsity function with $\alpha = 10$, $\beta = 10$, and $\Theta = \{0.5, 1.5, 0.01\}$. The hyperparameters were optimized for the GPLVM baselines. Note that our method outperforms the baselines independent of the initialization used for the GPLVM.



(a) Iteration 0

(b) Iteration 10

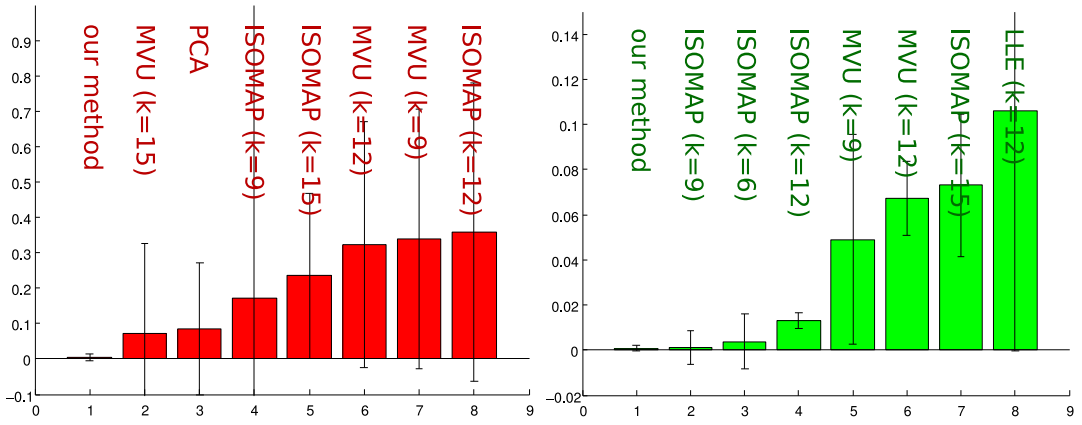
(c) Iteration 29

(d) Iteration 48

(e) Iteration 60

(f) Iteration 91

Figure 4.2: **Learning a sparsely sampled Swiss Roll.** Here snapshots of the learning process are shown at different iterations during optimization using our method. One can clearly see how the roll is unraveled correctly. The final manifold has 2 dimensions.



(a) Reconstruction Error (b) Relationship Error

Figure 4.3: **Example: A sparsely sampled swiss roll.** Reconstruction Error (a) and Relationship Error (b) for the experiment in figure 4.1 averaged over 20 random partitions of the data for the 8 best performing dimensionality reduction techniques. A more detailed result is given in figure 4.4.

	Reconstruction Error		Relationship Error	
	mean	**stddev**	**mean**	**stddev**
our method	**0.0041**	**0.0107**	**0.0008**	**0.0013**
PCA init	0.0845	0.1860	0.4232	0.1915
ISOMAP init (k=6)	0.3813	0.1794	0.0038	0.0120
ISOMAP init (k=9)	0.1720	0.8453	0.0011	0.0076
ISOMAP init (k=12)	0.3583	0.4221	0.0130	0.0035
ISOMAP init (k=15)	0.2350	0.2326	0.0731	0.0314
Laplacian init (k=6)	3.7027	2.6553	9.2735	2.5540
Laplacian init (k=9)	2.3136	1.3150	0.6426	8.1716
Laplacian init (k=12)	1.6038	0.2878	7.9784	1.5029
Laplacian init (k=15)	0.5561	2.0321	2.2400	0.0508
LLE init (k=6)	3.5514	2.1871	7.5591	1.4807
LLE init (k=9)	2.6175	2.5141	8.5485	1.4881
LLE init (k=12)	1.6235	1.1386	0.1058	0.1064
LLE init (k=15)	2.7300	4.5488	1.7820	8.9215
LTSA init (k=6)	2.6377	0.7273	1.2636	1.6498
LTSA init (k=9)	3.5149	3.8835	2.0575	2.4038
LTSA init (k=12)	2.1734	3.2099	2.3700	3.0718
LTSA init (k=15)	4.0950	3.6681	2.1819	4.5150
MVU init (k=6)	0.3783	0.5381	0.1238	0.0055
MVU init (k=9)	0.3383	0.3665	0.0491	0.0465
MVU init (k=12)	0.3228	0.3477	0.0672	0.0164
MVU init (k=15)	0.0706	0.2560	0.5856	2.8057

Figure 4.4: **Example: A sparsely sampled swiss roll.** Reconstruction and Relationship Error for the experiment in figure 4.1 averaged over 20 random partitions of the data. Here more detailed results are shown, including PCA and graph-based methods with different neighborhood sizes. The 8 best performing techniques are depicted in 4.3.

4.2 Discovering the correct dimensionality

Here we illustrate our method's ability to discover the intrinsic dimensionality of the underlying manifold in 5 complex synthetic examples, which all are embedded in 3D space ($D = 3$), but exhibit a different intrinsic dimensionality ($Q = \{1, 2, 3\}$). In figure 4.5a a spiral with a wide separation between rings is reduced to a 1D manifold. When the distance between the different rings decreases, the intrinsic manifold dimensionality changes from 1D to 2D (see figure 4.5b), since relationships between points that have the same phase are considered. In figure 4.5c the underlying 2D manifold from a cut-off sphere sampled along longitudinal lines is discovered. The manifold in figure 4.5d is intrinsically 3D and thus cannot be reduced (as discovered by our method), while the manifold in figure 4.5e can be reduced to 2D. The learning process is illustrated in figure 4.6, 4.7, 4.8, 4.9, 4.10, 4.11 and 4.12.

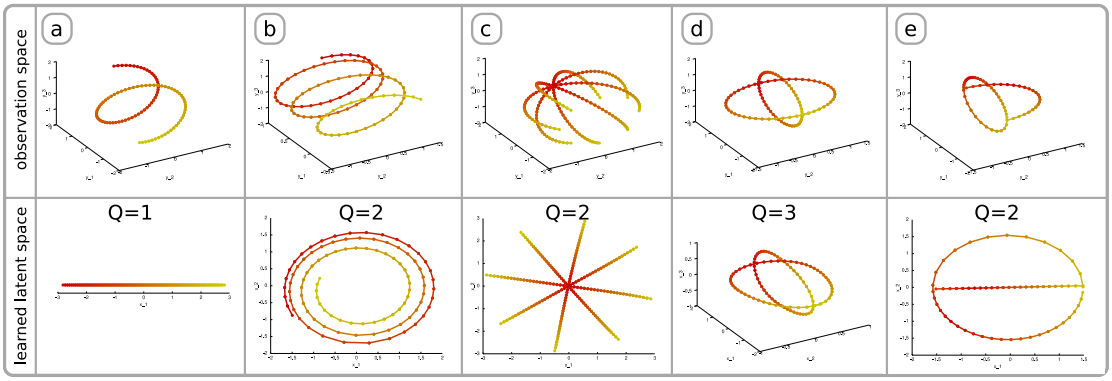
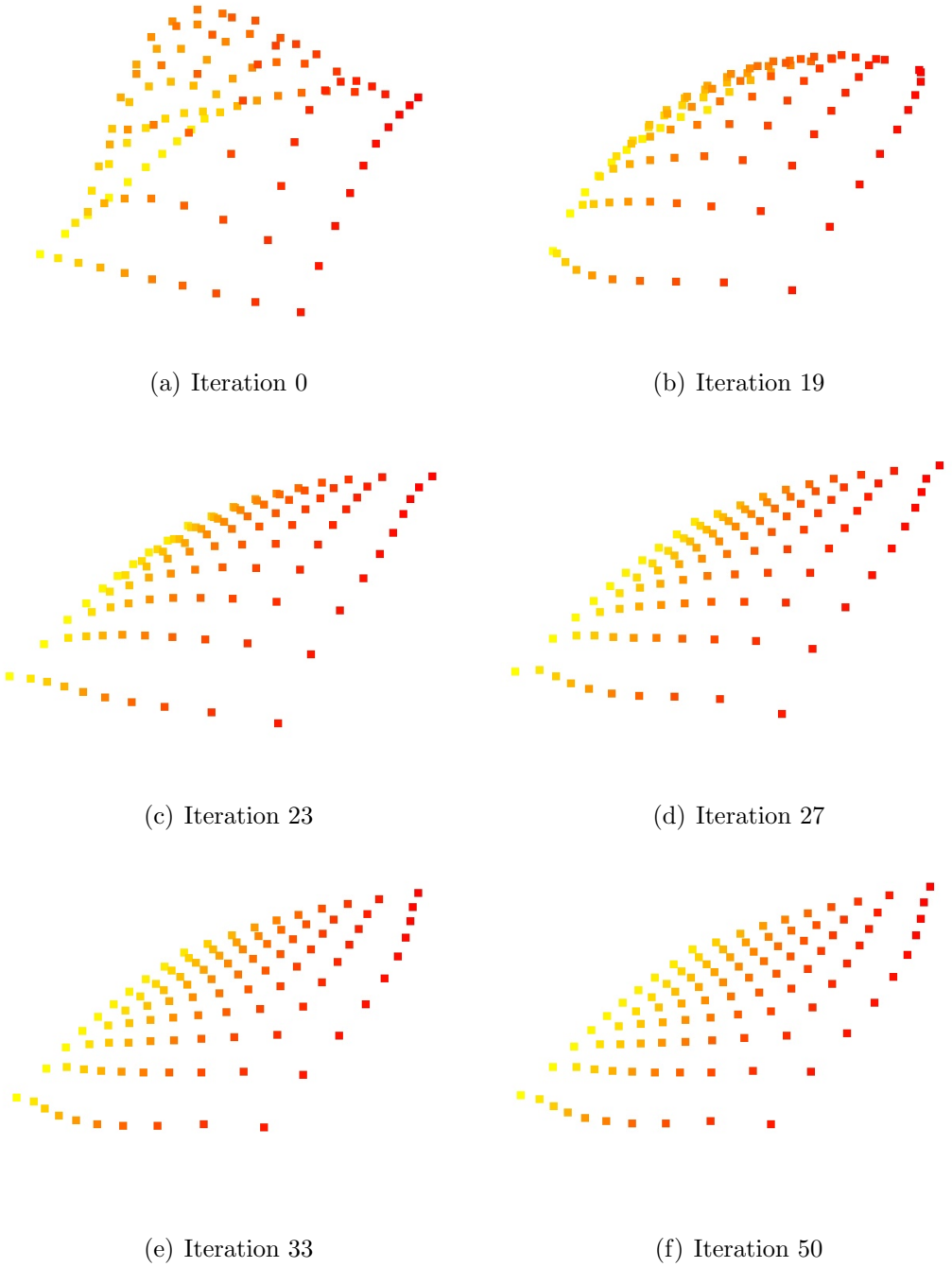


Figure 4.5: **Dimensionality estimation.** (Top) Five 2D manifolds embedded in 3D. (Bottom) Latent spaces and intrinsic dimensionalities Q learned using our continuous dimensionality reduction method.



(a) Iteration 0

(b) Iteration 19

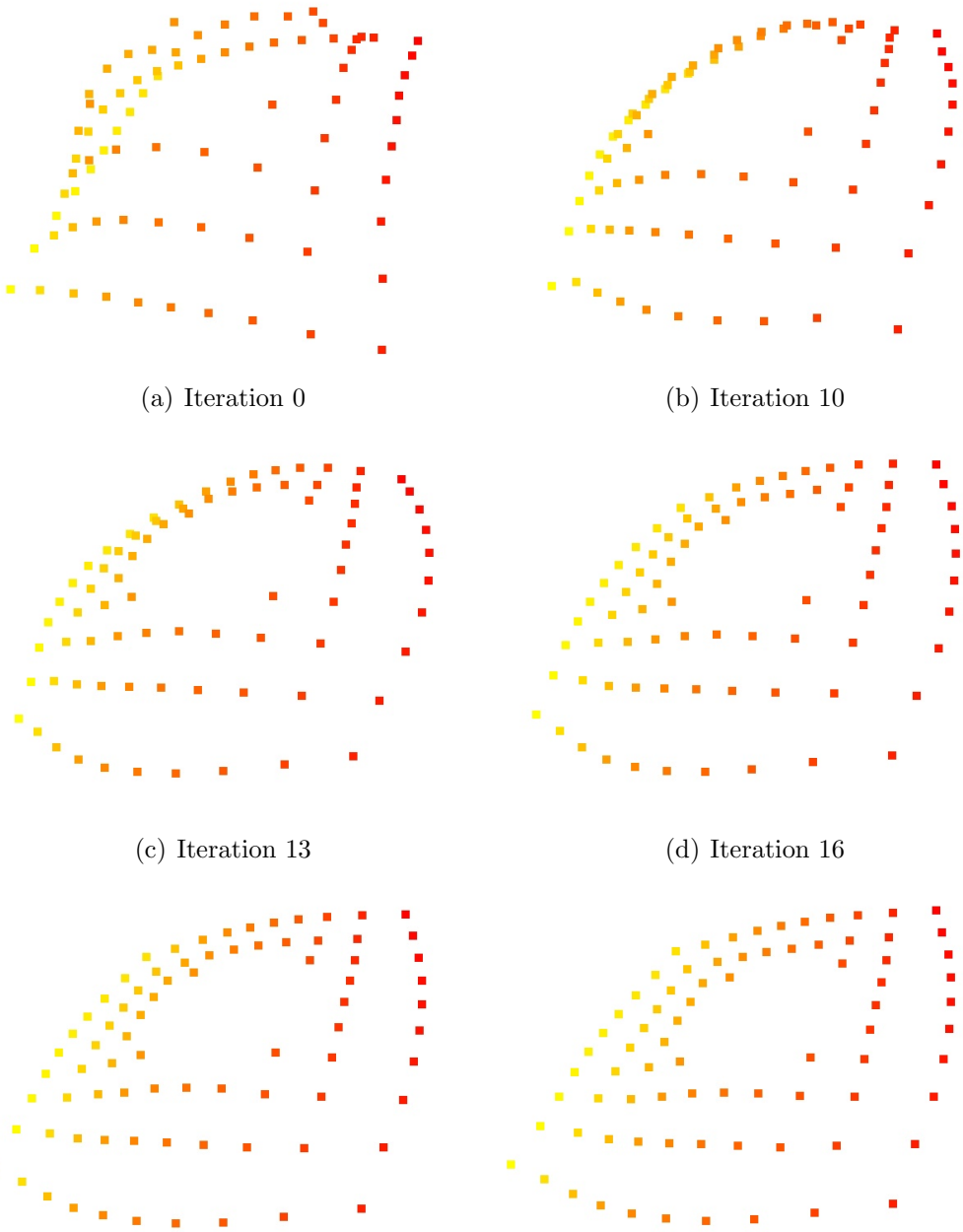
(c) Iteration 23

(d) Iteration 27

(e) Iteration 33

(f) Iteration 50

Figure 4.6: **Learning a hat shaped manifold generated by sine functions.** Snapshots of the learning process are shown at different iterations during optimization using our method. The discovered dimensionality Q is 2.

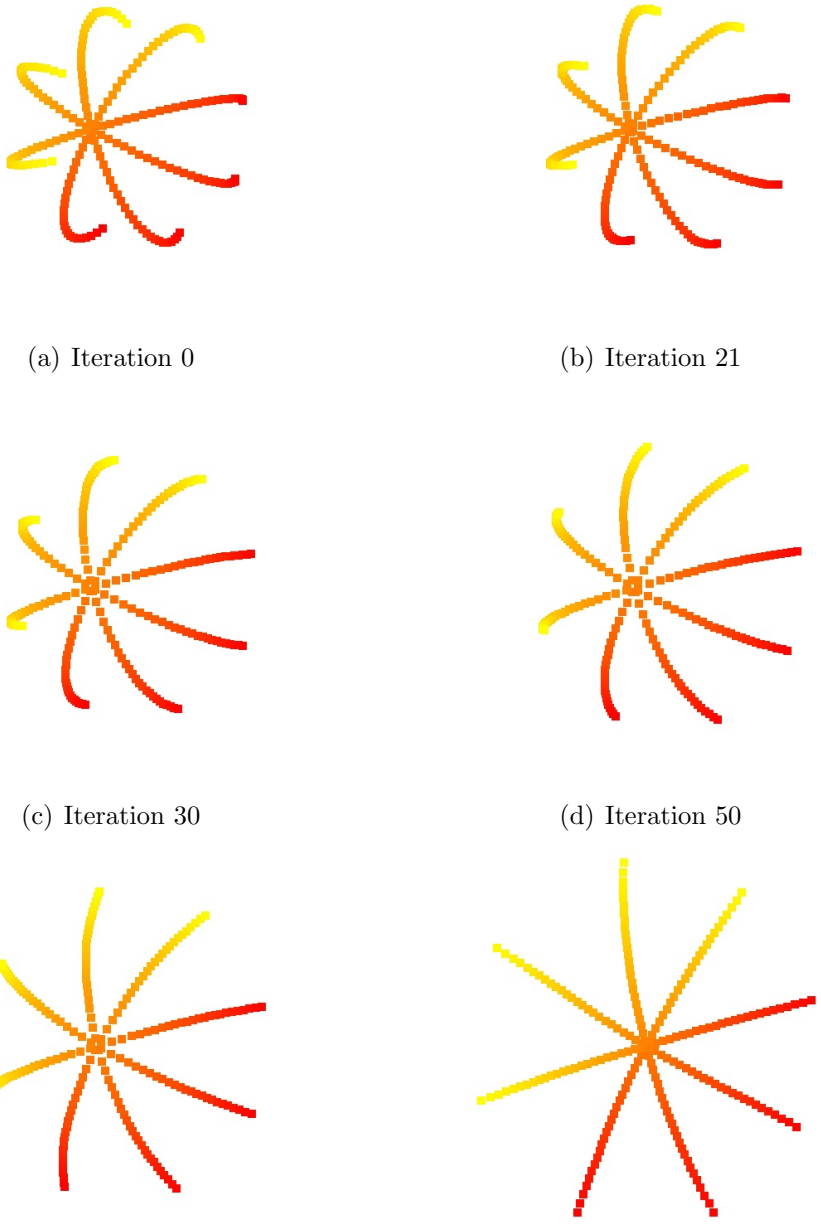


(a) Iteration 0 (b) Iteration 10

(c) Iteration 13 (d) Iteration 16

(e) Iteration 19 (f) Iteration 50

Figure 4.7: **Learning a hat shaped manifold with a hole.** Snapshots of the learning process are shown at different iterations during optimization using our method. The discovered dimensionality Q is 2.

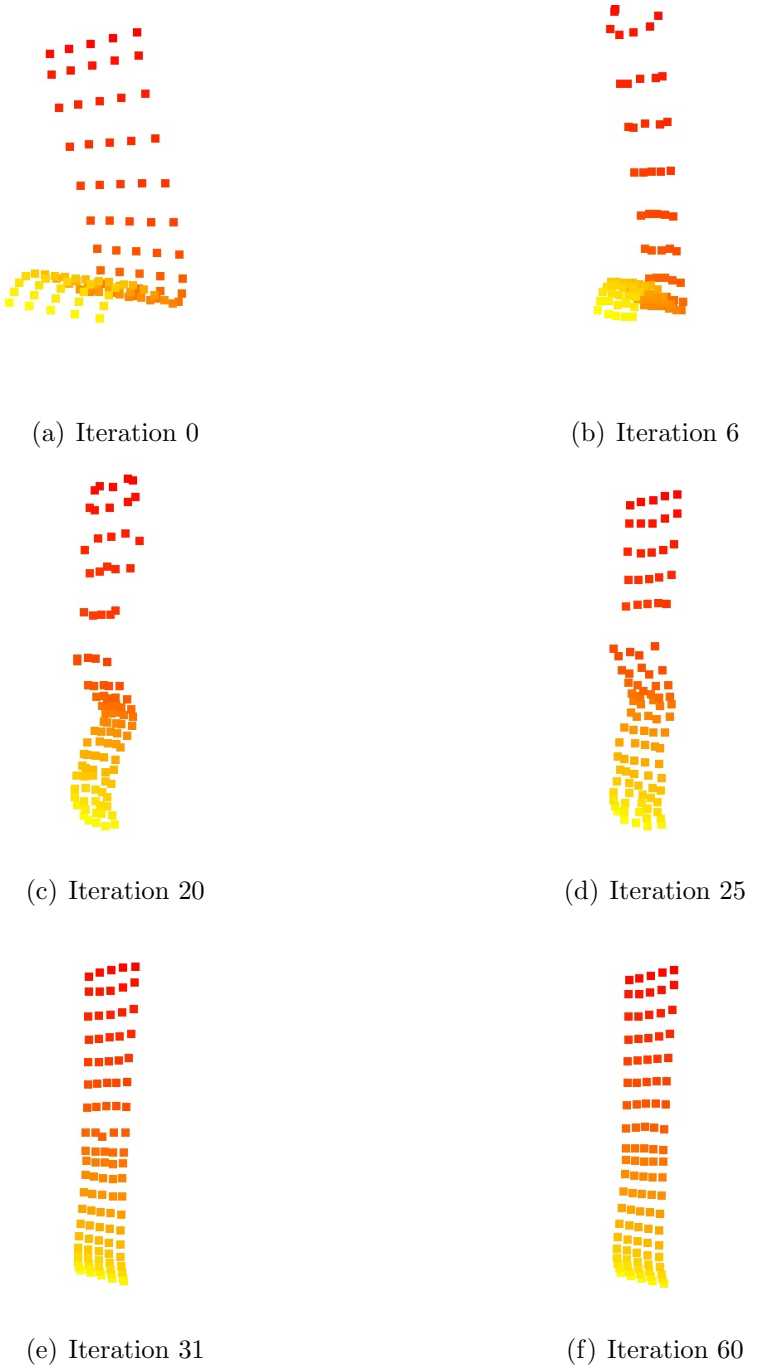


(a) Iteration 0 (b) Iteration 21

(c) Iteration 30 (d) Iteration 50

(e) Iteration 65 (f) Iteration 100

Figure 4.8: **Learning a cut-off bowl manifold sampled along longitudal lines.** Snapshots of the learning process are shown at different iterations during optimization using our method. The discovered dimensionality Q is 2.



(a) Iteration 0

(b) Iteration 6

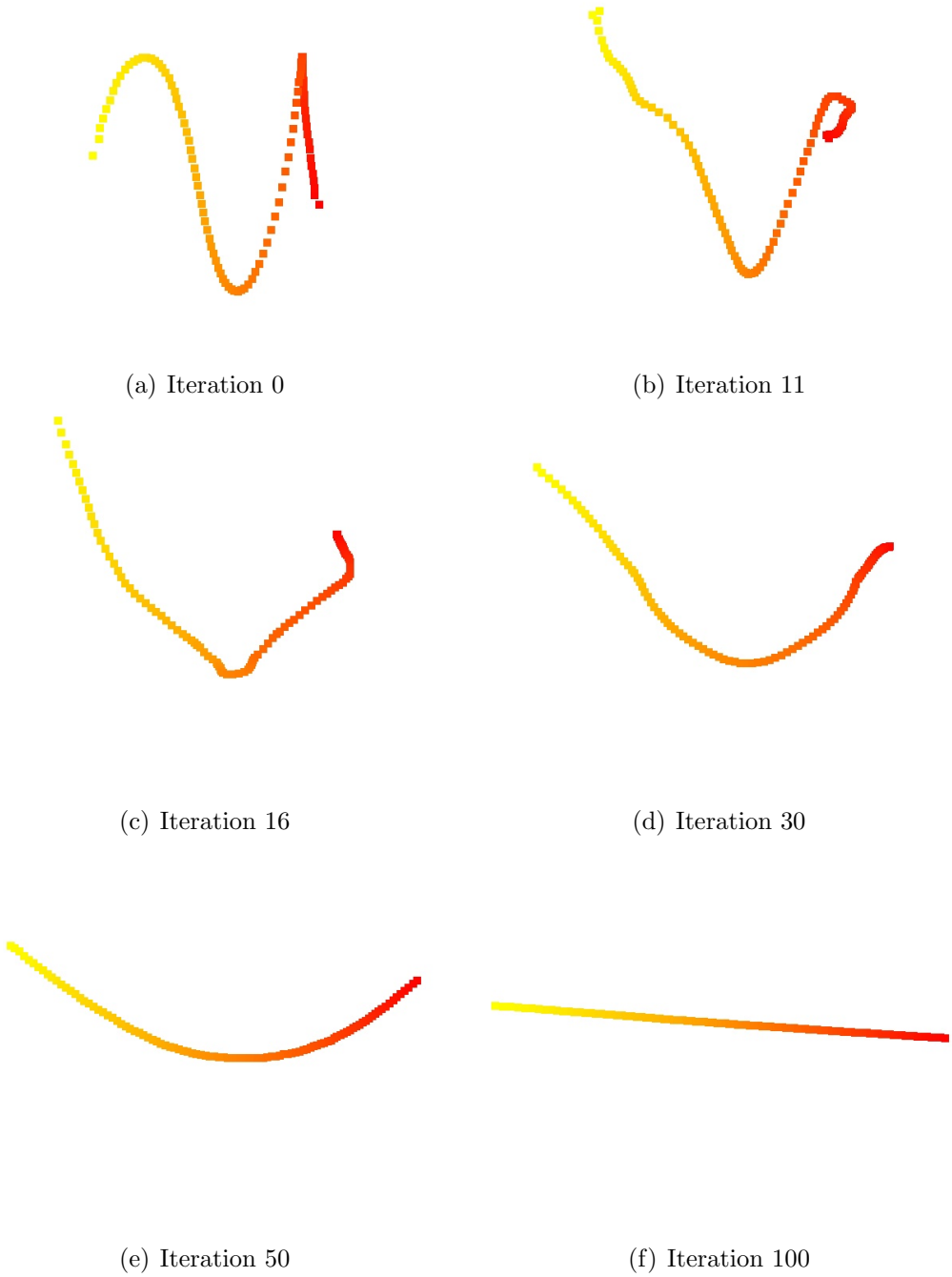
(c) Iteration 20

(d) Iteration 25

(e) Iteration 31

(f) Iteration 60

Figure 4.9: **Learning a 2D wave shaped manifold.** Snapshots of the learning process are shown at different iterations during optimization using our method. The discovered dimensionality Q is 2.



(a) Iteration 0

(b) Iteration 11

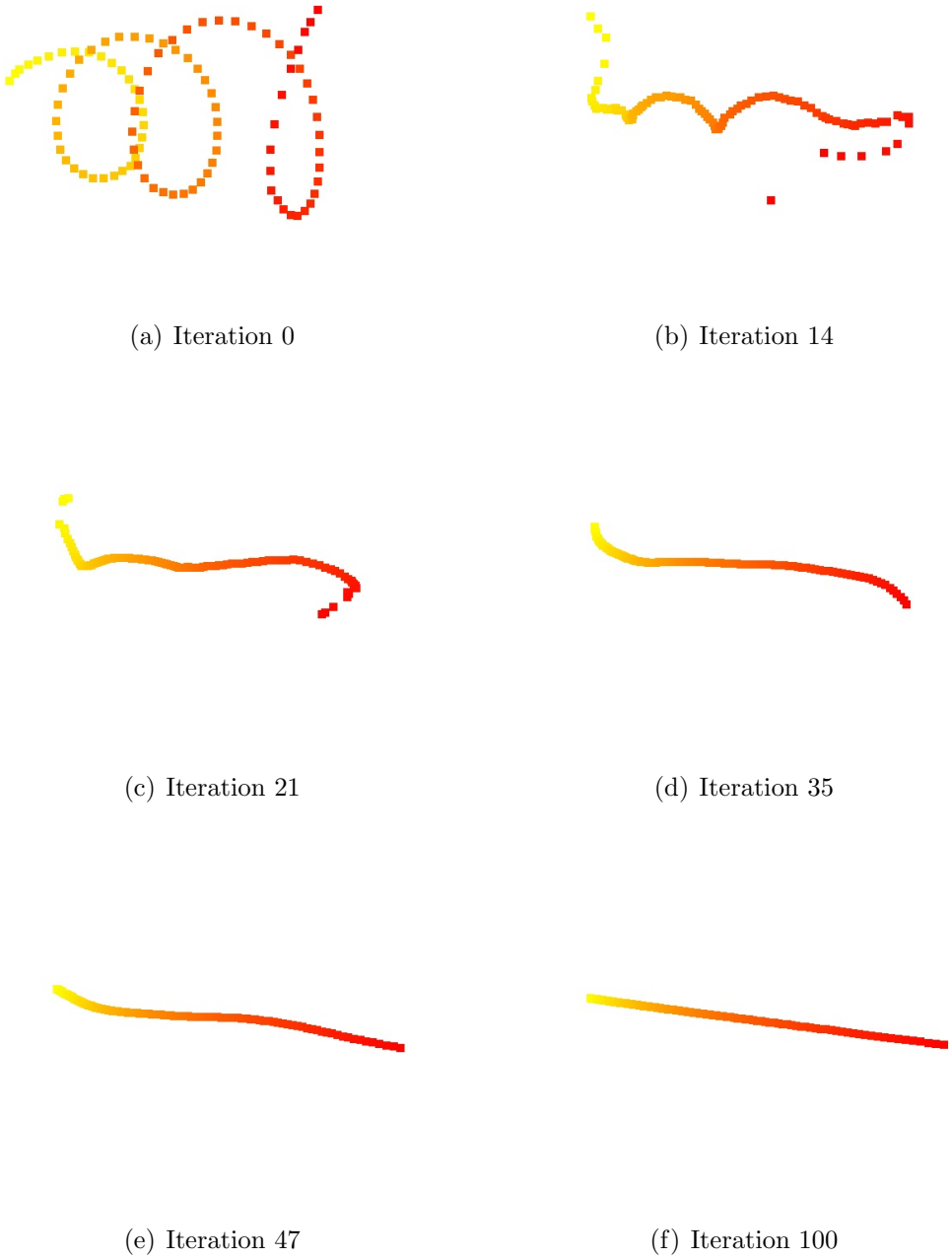
(c) Iteration 16

(d) Iteration 30

(e) Iteration 50

(f) Iteration 100

Figure 4.10: **Learning a spiral manifold embedded in 3D.** Snapshots of the learning process are shown at different iterations during optimization using our method. The discovered dimensionality Q is 1.



(a) Iteration 0

(b) Iteration 14

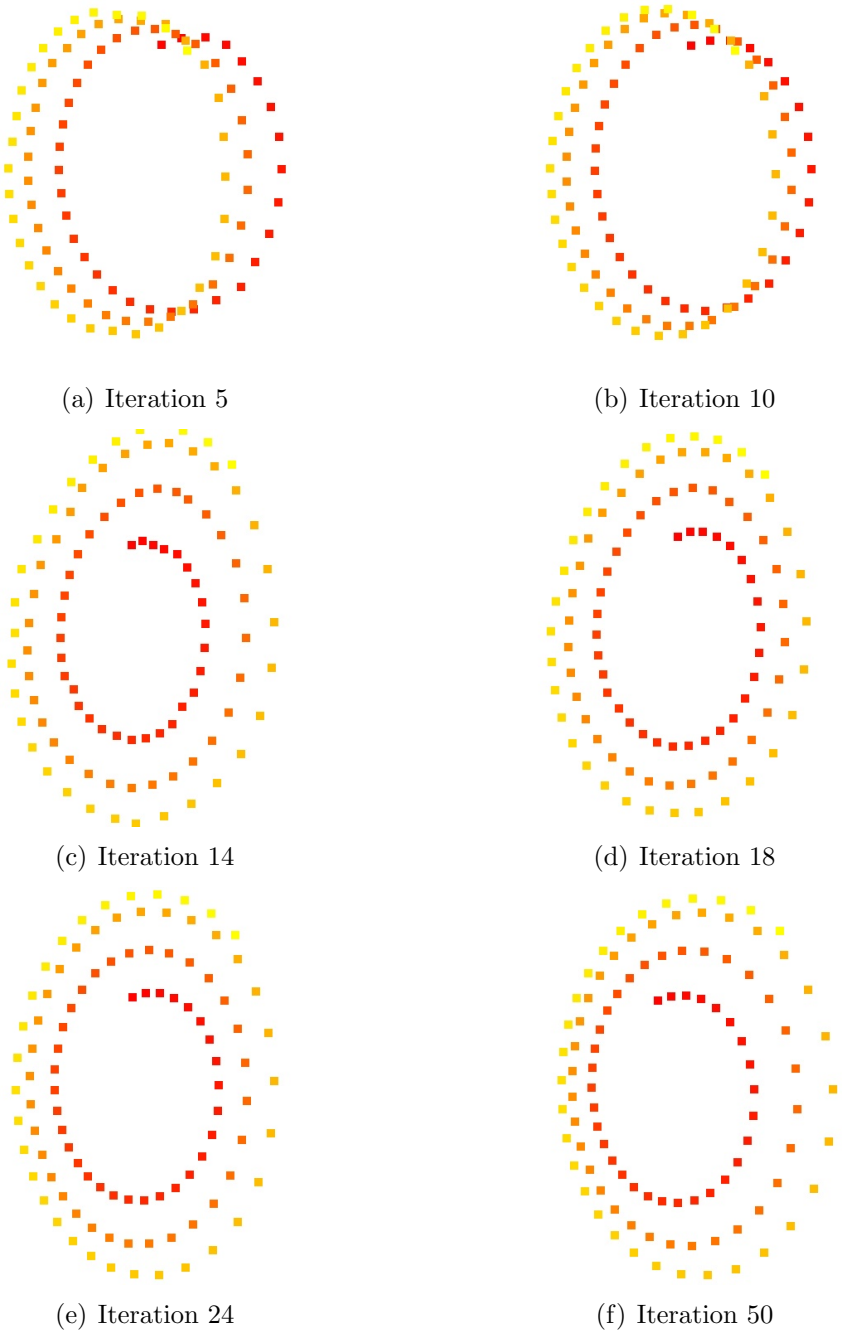
(c) Iteration 21

(d) Iteration 35

(e) Iteration 47

(f) Iteration 100

Figure 4.11: **Learning a spiral manifold embedded in 3D.** Snapshots of the learning process are shown at different iterations during optimization using our method. The discovered dimensionality Q is 1.



(a) Iteration 5 (b) Iteration 10

(c) Iteration 14 (d) Iteration 18

(e) Iteration 24 (f) Iteration 50

Figure 4.12: **Learning a narrow spiral manifold embedded in 3D.** Snapshots of the learning process are shown at different iterations during optimization using our method. The discovered dimensionality Q is 2.

4.3 Tracking and classification in the kitchen domain

An interesting real-world application of discovering low-dimensional structure in a high-dimensional space is tracking and classifying human motion from video sequences. Tracking consists of inferring the 3D locations of body joints from images. Since tracking requires good generalization abilities and latent trajectories to be smooth this is a difficult task.

The goal of this experiment is to evaluate a model learned from different motions by tracking and classifying human movements performed in a kitchen scenario. This is done by learning the model from previously captured motion capturing data and tracking the person's movements using a particle filter approach (see figure 4.17). Once tracking can be performed robustly, classification is straightforward, e.g. by assigning the class label of the nearest neighbor in the latent space to the current frame.

	Ølength	**#total**	**#learning**	**#tracking**
rolling	**18 s**	**6**	**2**	**4**
milling	**15 s**	**4**	**2**	**2**
brooming	**11 s**	**2**	**2**	**0**

Figure 4.13: **Dataset used in the tracking experiment.** Here an overview of the dataset used in the experiment is given. First, the average length per sequence is given in seconds. Next the total number of utilizable sequences, the number of sequences used for learning the model and the number of sequences used for tracking is shown.

We show how learning multiple motions in a single latent space can be done using our method since this is a difficult task in general. We use the kitchen dataset of [KPFW08], that consists of images from 2D cameras and ground truth joint angles of 3 motion types (i.e., rolling, milling, brooming), performed several times by the same subject. Unfortunately, only a subset of the entire dataset was found to be not corrupted by mislabelings and could be used for the experiments. More specifically those were 6 rolling motions, 4 milling motions and 2 brooming motions. For our experiments we learned a common latent space using the first two motions of each type and tested our algorithm on the remaining motions. Therefore no tests on the brooming motions were performed although they were included in the training set. The motion capture data was recorded using a VICON system (at 100 Hz) while the video streams were captured by four Sony VWF 500 color cameras (at 30 Hz). An overview of the available sequences is given in table 4.13 and in figure 4.14.

(a) Camera 1: Rolling

(b) Camera 2: Rolling

(c) Camera 1: Brooming

(d) Camera 2: Brooming

(e) Camera 1: Milling

(f) Camera 2: Milling

Figure 4.14: **Tracking sequences.** For the tracking and classification experiments the database from [KPFW08] was used. It contains several types of motion, captured with a VICON motion capturing system and four Sony VWF 500 color cameras (resolution 640x480). In this figure snapshots of 3 motions (rolling, brooming and milling) are depicted from 2 views each.

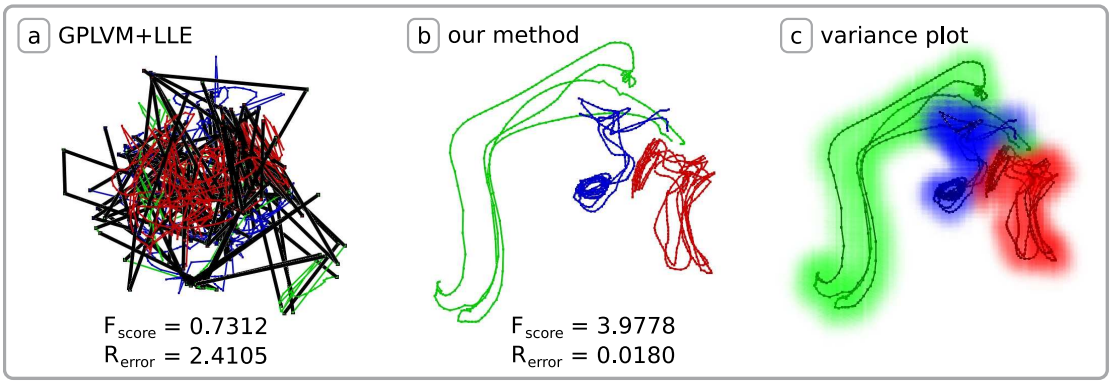


Figure 4.15: **Learning different types of motion into one single 3D latent space.** Each type is visualized with a different color (red = rolling, green = brooming, blue = milling). (a) depicts a 3D-GPLVM initialized to LLE, discontinuities are emphasized in black. (b,c) show the manifold learned by our method along with the variance.

4.3.1 Learning

We learned a single 3D latent space from 30D joint angle observations of 2 trials for each of the 3 different motions. Since learning involves the inversion of the covariance matrix K and thus scales as $O(N^3)$ where N represents the number of training examples, only a limited number of samples can be used for training the model.

Due to variation in dynamics, learning a latent space using a simple subsampling (like 1 out of 4) of the dataset leads to suboptimal results, since similar points in the latent space do not contribute to a richer model. Thus a small threshold has been employed which determines the minimum distance between 2 observation points. Observations were then added incrementally to an initially empty vector if the condition is fulfilled. Alternatively, one could also apply the *IVM* criteria, described in [LSH02]. The number of training examples which we used finally was $N = 1120$, comprising samples from 3 motions with 2 trials each.

For a further speedup the sparsification technique, as described in section 2.3.3 was employed. Altogether we were able to learn the model in 4 hours on a 2-Ghz single CPU machine.

Since articulated body motion is non-linear we used LLE initialization for the GPLVM as a baseline. Figure 4.15 shows the result of learning the latent space of the full model using a GPLVM initialized with LLE (a) and our method (b). Note that our method, unlike the GPLVM initialized with LLE, leads to a smooth (i.e., consecutive frames in time are close in the latent space) result, and separates well the different classes. As shown in the figure, smoothness implies a lower

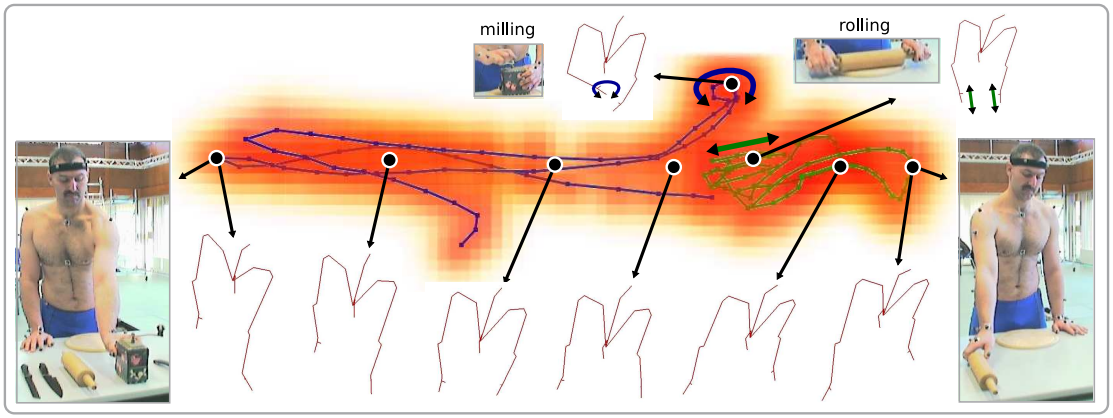


Figure 4.16: **Learning 2 motions (blue = milling, green = rolling) using our method.** The upper-body skeleton (in red) depicts mean-poses calculated at the corresponding points in the latent space. One can see that "grabbing the mill" (using the left hand, poses to the left) is well separated from "grabbing the rolling pin" (using the right hand, poses to the right).

relationship error. To quantify class separation, we also compute the Fisher score [UD07] defined as

$$F_{score} = tr\left(S_w^{-1} S_b\right)$$

where S_w is the within class matrix

$$S_w = \sum_{i=1}^{C} \frac{N_i}{N} \left[\frac{1}{N_i} \sum_{j=1}^{N_i} \left(\mathbf{x}_j^{(i)} - \overline{\mathbf{x}}^{(i)}\right) \left(\mathbf{x}_j^{(i)} - \overline{\mathbf{x}}^{(i)}\right)^T \right]$$

and S_b is the between class matrix

$$S_b = \sum_{i=1}^{C} \frac{N_i}{N} \left(\overline{\mathbf{x}}^{(i)} - \overline{\mathbf{x}}\right) \left(\overline{\mathbf{x}}^{(i)} - \overline{\mathbf{x}}\right)^T.$$

Here C denotes the number of classes (e.g. $C = 3$ in figure 4.15), N_i is the number of points in class i and N is the total number of training points. Furthermore $\mathbf{x}_j^{(i)}$ stands for the j'th point in class i, $\overline{\mathbf{x}}^{(i)}$ is the mean of class i and $\overline{\mathbf{x}}$ represents the overall mean. Note that our method performs significantly better than the GPLVM with LLE initialization in terms of both, the relationship error and the fisher score.

To gain more insight into the latent space, figure 4.16 shows the model after optimizing training data from 2 types of motion. Here the mean pose is depict for different points in the latent space.

4.3.2 Feature extraction and image likelihood

Tracking was performed using silhouette features. In contrast to [UFF06], where points in the video sequence (for example knees, hands, elbows, etc.) had to be tracked by a separate 2D tracker, this appearance-based approach has the advantage of no need to being reinitialized manually (after occlusions) or relying on a separate tracking algorithm. Disadvantages include that gradient-based refinements (for accurate tracking) are harder to incorporate and a more sophisticated likelihood had to be used than the one we used here. However, we found that with our test data reliable tracking was possible with an easy-to-implement likelihood described in this section.

To retrieve silhouettes from the input image, a small number of frames (we used 4 frames of the sequence) was labeled manually in each view (see figure 5.2). Departing from the labeled frames, a background and a color representation has been created. The background representation consists of storing all pixel values belonging to the background for each labeled frame. The color representation was created by storing a foreground and a background color histogram, where 16 bins per color (since each color channel provides 256 values, 16 values have been condensed together) have been used. Thus, in total $16 \times 16 \times 16 = 4096$ bins have been used.

The background segmentation was done by comparing all pixels of a 640×480 input image to all pixels (up to 4) in the background representation for the corresponding pixel coordinate. A pixel was then considered belonging to the foreground (or to the silhouette) if its Euclidian distance to all background pixels at this location (in the RGB color space) was larger than 18 (which was found emperically).

For the (skin) color segmentation we used the Bayesian classifier described in [PBC05] which employs a minimum cost decision rule: Classify a pixel with color c as skin (or foreground silhouette), iff: $P(skin|c) > P(\overline{skin}|c)$. Since

$$P(skin|c) = \frac{P(c|skin)P(skin)}{P(c)} \quad \text{and} \quad P(\overline{skin}|c) = \frac{P(c|\overline{skin})P(\overline{skin})}{P(c)}$$

we have

$$\frac{P(c|skin)}{P(c|\overline{skin})} > \frac{P(\overline{skin})}{P(skin)}.$$

Here $P(c|skin)$ and $P(c|\overline{skin})$ are given as counts by the previously collected color histograms:

$$P(c|skin) = \frac{\text{number of foreground pixels with same color}}{\text{total number of foreground pixels}}$$

$$P(c|\overline{skin}) = \frac{\text{number of background pixels with same color}}{\text{total number of background pixels}}$$

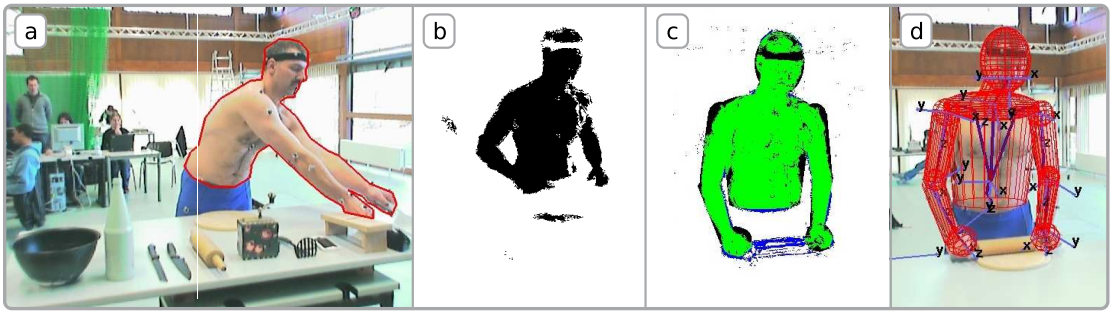


Figure 4.17: **Tracking setup.** First, some (3-5) frames of the sequence were labeled manually (a). Then a background and a color representation has been generated automatically in order to segment the complete sequence (b). The likelihood is based on a matching (green) between the generative model and the segmentation in each frame (c). Finally a particle filter was used to track the sequence (d).

For the experiments reported in this thesis $\frac{P(\overline{skin})}{P(skin)}$ was set to $= 0.3$, which was found empirically.

We finally combined the background segmented image and the color segmented image using a logical *AND* operator with respect to the foreground region. For calculating the image likelihood $p(\mathbf{z}|\mathbf{y})$ (here $\mathbf{z}$ denotes an observation, which - in our case - comes frome one or two color cameras) given a (30 dimensional) data point $\mathbf{y}$, the silhouette of the articulated body model (see figure 4.17) was generated from $\mathbf{y}$ and rendered into the back buffer of an ATI graphics card via OpenGL. By reading the graphics buffer back into memory, $p(\mathbf{z}|\mathbf{y})$ was finally obtained as $p(\mathbf{z}|\mathbf{y}) \propto \varrho$, where ϱ denotes the number of pixels (in the projected and the segmented image) belonging to the same class (foreground or background).

4.3.3 Particle filter

Tracking itself (see figure 5.3) was performed using a simplified particle filter with second order Marcov dynamics in the latent space. We assumed that all observed poses are captured by our pose model. The probabilistic framework of the tracker is described next.

Let $\mathbf{s}_t$ be the state of our system at time t and let $\mathbf{z}_t$ be the observation of the cameras at time t. To shorten notation we abbreviate $\mathbf{s}_{1:t} = \{\mathbf{s}_1, ..., \mathbf{s}_t\}$ and $\mathbf{z}_{1:t} = \{\mathbf{z}_1, ..., \mathbf{z}_t\}$.

Following [IB98] we assume that the object dynamics forms a temporal Markov chain, i.e.

$$p(\mathbf{s}_t|\mathbf{s}_{1:t-1}) = p(\mathbf{s}_t|\mathbf{s}_{t-1}).$$

Furthermore the observations $\mathbf{z}_t$ are assumed to be conditional independent and independent of the underlying dynamical process

$$p(\mathbf{z}_{1:t-1}, \mathbf{s}_t|\mathbf{s}_{1:t-1}) = p(\mathbf{s}_t|\mathbf{s}_{1:t-1}) \prod_{i=1}^{t-1} p(\mathbf{z}_i|\mathbf{s}_i)$$

which leads to the mutual conditional independence (by marginalizing $\mathbf{s}_t$)

$$p(\mathbf{z}_{1:t}|\mathbf{s}_{1:t}) = \prod_{i=1}^{t} p(\mathbf{z}_i|\mathbf{s}_i).$$

Using these assumptions, the rule for propagation of state density over time can be formulated as

$$p(\mathbf{s}_t|\mathbf{z}_{1:t}) = k_t p(\mathbf{z}_t|\mathbf{s}_t) p(\mathbf{s}_t|\mathbf{z}_{1:t-1})$$

where

$$p(\mathbf{s}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{s}_t|\mathbf{s}_{t-1}) p(\mathbf{s}_{t-1}|\mathbf{z}_{1:t-1}) d\mathbf{s}_{t-1}$$

and k_t is a normalisation constant that does not depend on $\mathbf{s}_t$. The rule can be interpreted as the Bayes' rule for inferring the posterior state density from observations for the time-varying case. The prove can be found in [IB98].

Since the observation density is non-Gaussian, the evolving state density $p(\mathbf{s}_t|\mathbf{z}_{1:t})$ is also non-Gaussian. Applying a non-linear filter to evaluate the state density cannot be done analytically. Thus we are using an approximation by evaluating a finite set of particles.

Our state vector $\mathbf{s}_t$ consists of $\mathbf{s}_t = \left\{\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}\right\}$, the estimation of the position in the latent space at time t and time $t-1$. Including the latent position at time $t-1$ is needed for drawing samples from the second order Marcov dynamics. Given the state vector $\mathbf{s}_t$ at time t we estimate the state vector $\mathbf{s}_{t+1}$ at time $t+1$ by creating P samples $\hat{\mathbf{x}}_i$ with $i = \{1, ..., P\}$ in the latent space and calculating the image likelihood of the mean projection $p\left(\mathbf{z}_{t+1}|\mu(\hat{\mathbf{x}}_i)\right)$ as described in section 4.3.2. Then $\mathbf{s}_{t+1}$ is chosen as

$$\mathbf{s}_{t+1} = \begin{pmatrix} \mathbf{x}^{(t+1)} \\ \mathbf{x}^{(t)} \end{pmatrix} = \begin{pmatrix} argmax_{\hat{\mathbf{x}}_i} \left[p\left(\mathbf{z}_{t+1}|\mu(\hat{\mathbf{x}}_i)\right)\right] \\ \mathbf{x}^{(t-1)} \end{pmatrix}.$$

The samples $\hat{\mathbf{x}}_i$ were generated from the (predictive) *Mixture-of-Gaussians* distribution (see figure 4.18)

$$p(\hat{\mathbf{x}}_i|\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}) = \frac{2}{5}\mathcal{N}\left(\mathbf{x}^{(t)}, \frac{1}{25}\mathbf{I}\right) + \frac{2}{5}\mathcal{N}\left(2\mathbf{x}^{(t)} - \mathbf{x}^{(t-1)}, \frac{1}{625}\mathbf{I}\right) + \frac{1}{5}p_{KDE}\left(\mathbf{x}^{(t)}\right)$$

where the first term is a Gaussian distribution centered at the position of $\mathbf{x}$ at time step t, the second term represents a Gaussian distribution centered on the

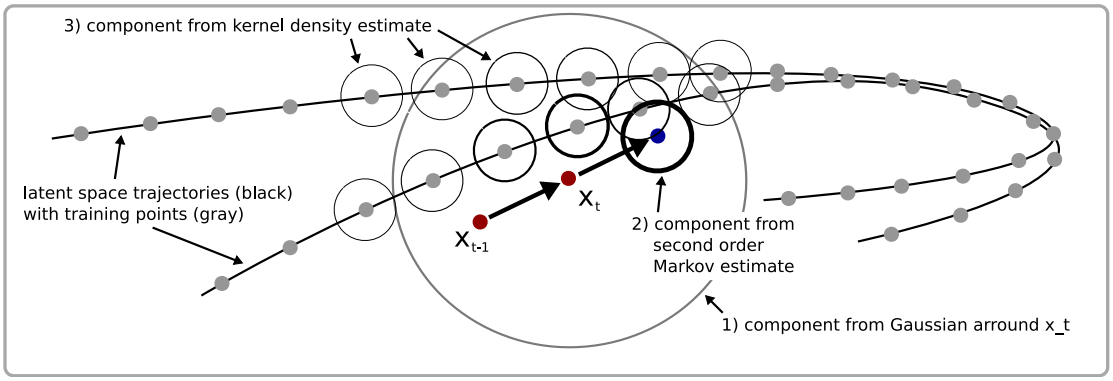


Figure 4.18: **Predictive Distribution.** This figure illustrates the dynamical model. The predictive distribution is taken as a *Mixture-of-Gaussians* with 3 components: (1) A random walk sampled directly around $\mathbf{x}_t$ to marginalize the danger of local minima, (2) a second order Marcov model which extrapolates linearly from $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ (Gaussian around blue point), and (3) Gaussians around training points, weighted by their corresponding kernel density estimate using a RBF kernel. Here 2 trajectories with their training points are shown in the latent space. Gaussians are depict by a circle. The line width of the circles indicate their respective mixture weight.

second order Marcov estimate and the third term is a *Kernel Density Estimate (KDE)* of $\mathbf{x}$ at time t, defined as

$$p_{KDE}(\mathbf{x}^{(t)}) = \frac{1}{\sum_{i=1}^{N} k(\mathbf{x}^{(t)}, \hat{\mathbf{x}}_i)} \sum_{i=1}^{N} k(\mathbf{x}^{(t)}, \hat{\mathbf{x}}_i) \mathcal{N}(\hat{\mathbf{x}}_i, \frac{1}{625}\mathbf{I})$$

where N is the number of training points in the model. Here the *inference kernel* $k(\mathbf{x}_i, \mathbf{x}_j)$ is defined as a radial basis function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \theta_1 \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\theta_2^2}\right) + \theta_3 \delta_{ij}$$

In our experiments we set the hyperparameters of the *inference kernel* to $\theta_1 = 1$, $\theta_2 = 0.2$ and $\theta_3 = 0.0001$.

Since we are dealing with Gaussian Processes and large datasets, speed becomes an issue. Calculating the mean pose $\mu(\hat{\mathbf{x}}_i)$ of a sample point $\hat{\mathbf{x}}_i$ using a full GP involves a one time inversion of the covariance matrix K which scales as $O(N^3)$. Furthermore, for each sample $\hat{\mathbf{x}}_i$, a full KDE and N-dimensional matrix-vector multiplication has to be performed which scales as $O(N^2)$. To speed up inference several approximation techniques could be used [SNS06, YDD05, UD08, SW02, QCR05]. We chose one of the most intuitive ones, which makes use of the sparseness of K. As in [UD08] we infer the mean pose $\mu(\mathbf{x}^{(i)})$ using only a subset

(we chose the 50 nearest neighbors) of the training points, which in practice lead to a sufficiently good approximation.

Improvement in tracking accuracy could be gained by further optimizing the posterior $p(\mathbf{s}|\mathbf{z})$ as done in [UFF06], which allows for tracking poses beyond the ones captured by the model. In this scenario, however, we found direct sampling from the model sufficient to obtain good tracking results (see figure 4.19).

4.3.4 Results

figure 4.19 depicts tracking and classification performance for milling and rolling motions[1] when using the models depicted in figure 4.15. We used the particle filter described in section 4.3.3, that operates in the low dimensional latent space and models the dynamics with a second order Markov model. Our image likelihood is based on low-level silhouette features (see section 4.3.2). The reason for using a simple likelihood is that it was found sufficient for the experiments shown in this thesis. This is mainly because most of the ambiguities are resolved by the strong pose prior we use. For more complicated sequences however, designing a more sophisticated likelihood can improve results. This could be done by using features based on optical flow, edges, appearance or incorporating an additional discriminative model into the tracking process [UD08].

In practice we could achieve a *sample evaluation rate* of 100 frames per second on a 2 Ghz single CPU machine with hardware graphics acceleration. This means that using 25 particles and 2 cameras (which was found sufficient for reliable tracking in our scenario, see figure 4.19) results in a tracking frame rate of 2 frames per second, for example.

The tracking error was measured by averaging the positions of 6 joints (shoulder left/right, elbow left/right, hand left/right). It is given as

$$\text{Tracking Error} = \frac{1}{6}\sum_{i=1}^{6} \|\hat{\mathbf{p}}_i - \mathbf{p}_i\|_2$$

where $\hat{\mathbf{p}}_i$ denotes the 3D position of the i'th joint as estimated by the particle filter and $\mathbf{p}_i$ stands for the ground truth.

For classification purposes, we labeled the dataset using the following 7 classes:

- rest *(both hands rest on the table next to the board)*
- grasp pin *(reaching for the pin with the right hand)*
- rolling *(perform a cyclic rolling motion with the rolling pin)*
- grasp broom *(grasp the broom and the dustpan with both hands)*

[1] No results are shown for brooming since no test data was available.

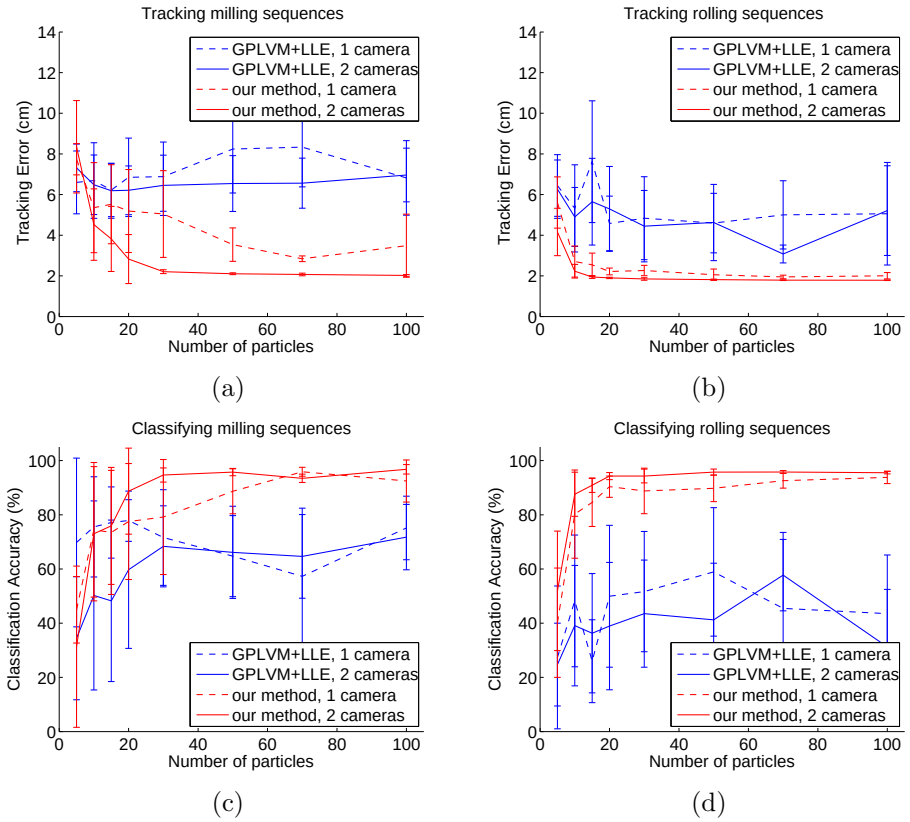


Figure 4.19: **Tracking and nearest neighbor classification performance** for milling and rolling motions using our method (red) and GPLVM initialized to LLE (blue) as a function of the number of particles used in the particle filter.

- brooming *(perform a brooming motion using the broom and the dustpan)*
- grasp mill *(grasp the mill with the left hand)*
- milling *(perform a cyclic milling motion)*

Classification was done by computing the nearest neighbor of the current pose estimate in the latent space, and assigning its class label to the current frame. The classification accuracy is then given by

$$\text{Classification Accuracy} = \frac{\text{number of correctly labeled frames}}{\text{total number of frames}} \times 100\%.$$

Our method significantly outperforms the GPLVM with LLE initialization in both tracking accuracy and NN classification performance (see figure 4.19) when using both, one or two cameras. We observed that the milling motion is more ambiguous than the rolling motion and thus can be tracked reliably *only* when using two cameras, whereas one camera proved sufficient for the rolling motion.

We furthermore observed that although the GPLVM model initialized to LLE exhibits many jumps and discontinuities in the latent space, a particle filter can still perform relatively well, depending on the neighborhood size which was used for generating the Locally Linear Embedding. This is due to the fact that in order to disambiguate motions with any of the models, the standard deviation used for creating samples can not drop below a certain value. Thus jumps in the latent space were found to be not as critical as initially assumed. In contrast, if one wants to generate motion by sampling, discontinuities start to play a much more important rule, leading to unrealistic motions when used for animation. We also expect a tracking system which further optimizes the observation likelihood together with the pose prior to behave significantly worse with a model like the one shown in figure 4.15a. Thus we think that a method like ours, which can reduce dimensionality in a non-linear way without generating jumps in the latent space can be worth further investigations.

5 Implementation

This project was implemented in *C++* using *Eclipse CDT* and *Ubuntu 8.04 - the Hardy Heron*. The following libraries were used:

- **FLTK** (Fast Light Toolkit, cross-platform C++ GUI toolkit)
- **OpenGL** (visualizing the latent space, rendering the generative model)
- **OpenCV** (import, export and undistortion of images, 2D drawing)
- **SNOPT** (constrained non-linear optimization for learning the model)
- **BLAS, LAPACK** (efficient algorithms for linear algebra, e.g. SVD)

Figures 5.1, 5.2 and 5.3 show the GUI of the program created during this thesis. First a script has been used to undistort images using the OpenCV function *cvUndistort2* with the intrinsic camera parameters found by the *MATLAB camera calibration toolbox*[1]. Furthermore errors in the motion datasets were corrected by looking for discontinuities along the latent trajectories. The errors occured during registration of the motion capture dataset due to mislabeled markers. If a small number of consecutive defective frames were detected, a linear interpolation was used to interpolate in between. If a large number of consecutive frames was defective the trial was rejected from being used for learning or tracking.

In the second step the motions were loaded into the program depict in figure 5.1 and a latent space was learned using the method described in section 3.3. This program also allows for showing the quality of the model by placing a line into the latent space and reconstructing the mean pose $\mathbf{y}$ from samples along this line. For samples close to the training points those artificially generated motions should look realistic, which indicates good interpolation and extrapolation abilities and a useful model for tracking or motion generation.

The third step consists of labeling 4 dissimilar frames of an image sequence which is about to be tracked manually, as can be seen in 5.2. This program further allows for creating the image statistics necessary for tracking and for checking the result of a segmented test image.

In the final step, another program allows for tracking the scene using the particle filter described in section 4.3.3 (see figure 5.3). The program parameters comprise the number of particles which are used for tracking as well as the number

[1] http://www.vision.caltech.edu/

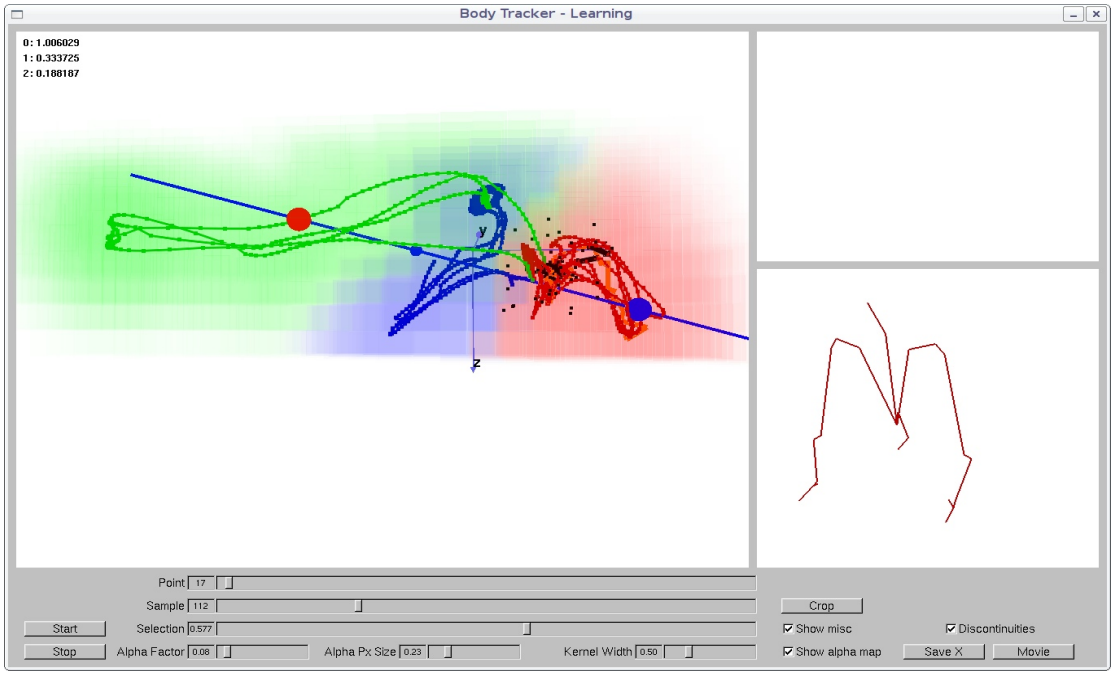


Figure 5.1: **Learning the latent space.** This application allows for loading a motion dataset and learning a latent space using different methods like GPLVM or GPLVM with Rank Prior. Initializations with graph based techniques were obtained via the MATLAB toolkit from [vdMPvdH07]. The main window shows the current latent space, projected to the first three dimensions via PCA. It also allows for displaying the covariance plot and the particle sample distribution in the latent space after a tracking experiment. The upper-right window displays the evolution of the singular values over time during an optimization. The lower-right window depicts the current state of the articulated human body model by applying the mean projection from the latent to the data space of a given test point.

of cameras. Since our test sequences behaved well, we found one or two cameras sufficient for this task. Tracking error and classification accuracy are stored in text files. Furthermore images which show the tracking and classification result from different views are created. Videos of those results can be found at *http://www.rainsoft.de*.

After tracking a sequence the particle distribution over time can be displayed in the latent space by loading the samples and the model back again into the program depict in figure 5.1.

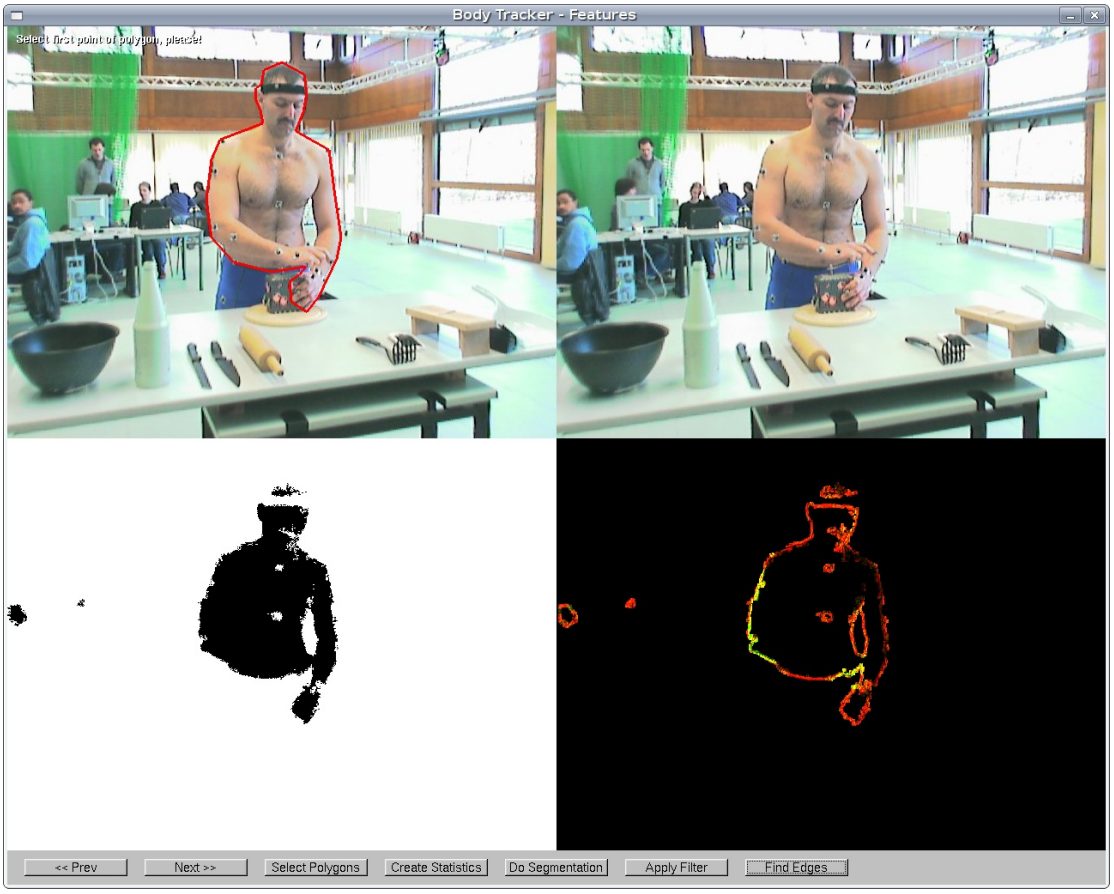


Figure 5.2: **Image likelihood and features.** Prior to tracking, this application asks the user to label the silhouette of the subject in 4 frames (upper-left). Using this information, a background- and a color-representation is created. The lower images show the result of applying those representations to a new frame (upper-right), resulting in a foreground segmentation (lower-left) and edges (lower-right). For tracking, the foreground representation was matched with a generative model of the articulated body.

Figure 5.3: **Tracking using a simple particle filter.** After learning the model and creating the feature representations this application performs the tracking itself. A particle filter can be started, which is initialized by computing the training point with the highest likelihood. The tracking step itself involves the generation of 5 to 100 particles (depending on the experiment) using a second order Markov model. After computing the mean prediction, the generative articulated body model silhuette is read from the OpenGL back buffer and matched against the observation silhouette (for 1 or 2 cameras, depending on the experiment). The sample with the highest likelihood is kept and the mean prediction is displayed to the OpenGL front buffer.

6 Acknowledgments

I want to thank everybody who made my diploma thesis possible. First of all many thanks to my supervisors[1] who agreed on having me jointly working at *Universität Karlsruhe (TH)* and *Massachusetts Institute of Technology*. They helped me in every situation and answered all of my questions. Many thanks also go to the DAAD (German Academic Exchange Service) who supported me with a scholarship for my 5-month stay in Boston, USA. Furthermore I want to thank my parents, friends and my girlfriend for their steady support and encouragement.

[1]

Prof. Dr. Trevor Darrell studied at University of Pennsylvania and at MIT where he received his Phd in 1996. After visiting Stanford University he first became assistant professor (2000) and then associate professor (2003) at MIT where he lead the Vision Interface Group. In August 2008 he moved to the University of California in Berkeley, EECS & ICSI.

Dr. Raquel Urtasun studied telecommunication engineering at University of Navarra (UPNA) in Pamplona, Spain. She hold positions at UPNA, EURECOM and ENST (France) and received her Phd in 2006 from EPFL (Switzerland). After being postdoctoral associate with Trevor Darrell at CSAIL, MIT she moved to the University of California in Berkeley, EECS & ICSI.

Prof. Dr. Alexander Waibel studied electrical engineering and computer science at MIT and received his PhD in 1986 from CMU at Pittsburgh. He currently is professor at CMU and Universität Karlsruhe (TH) where he leads the "international center for Advanced Communication Technologies" (interACT). His research interests include speech technologies and machine learning.

Dr. Rainer Stiefelhagen is assistant professor with Alexander Waibel at Universität Karlsruhe (TH). He leads the visual perception group of interACT. He studied at Universität Karlsruhe (TH) where he also received his Phd in 2002 with a work about "focus of attention". His research is about visual human perception, intelligent environments and machine learning.

Dipl.-Inform. Kai Nickel graduated from Universität Karlsruhe (TH) and now works as a research assistant with Rainer Stiefelhagen. His research interests include visual person tracking and guesture recognition. he is involved in the CHIL-project (computers in the human interaction loop) and in the SFB588 (german collaborative research center 588) "humanoid robots".

List of Figures

Index

Bibliography

[BN03] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, June 2003.

[DHS00] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification, 2nd edition*. Wiley-Interscience, 2000.

[GMHP04] K. Grochow, S. Martin, A. Hertzmann, and Z. Popovic. Style-based inverse kinematics. In *Proceedings of SIGGRAPH 2004*. ACM Press / ACM SIGGRAPH, 2004.

[GMS02] Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12(4):979–1006, 2002.

[Har07] Stefan Harmeling. Exploring model selection techniques for nonlinear dimensionality reduction. Technical report, School of Informatics, Edinburgh University, Scotland, March 2007.

[IB98] Michael Isard and Andrew Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[KPFW08] H. Koehler, M. Pruzinec, T. Feldmann, and A. Woerner. Automatic human model parametrization from 3d marker data for motion recognition. In *Proceedings of International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2008.

[Law05] Neil Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of Machine Learning Research*, 6:1783–1816, 2005.

[Law07] N. D. Lawrence. Learning for larger datasets with the gaussian process latent variable model. In *M. Meila and X. Shen (eds) Proceedings of the Eleventh International Workshop on Artificial Intelligence and Statistics*, 2007.

[LB05] Elizaveta Levina and Peter J. Bickel. Maximum likelihood estimation of intrinsic dimension. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 777–784. MIT Press, Cambridge, MA, 2005.

[Low99] David G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conference on Computer Vision ICCV, Corfu*, pages 1150–1157, 1999.

[LQC06] Neil D. Lawrence and Joaquin Quinonero-Candela. Local distance preservation in the gp-lvm through back constraints. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 513–520, New York, NY, USA, 2006. ACM.

[LSH02] Neil D. Lawrence, Matthias Seeger, and Ralf Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems 15*, pages 609–616. MIT Press, 2002.

[mFSA07] Amir massoud Farahmand, Csaba Szepesvári, and Jean-Yves Audibert. Manifold-adaptive dimension estimation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 265–272, New York, NY, USA, 2007. ACM.

[Mol93] Martin Fodslette Moller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.

[PBC05] S.L. Phung, Sr. Bouzerdoum, A., and Sr. Chai, D. Skin segmentation using color pixel classification: analysis and comparison. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(1):148–154, Jan 2005.

[Pea01] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[QCR05] Joaquin Quinonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

[RS00] Sam T. Roweis and Lawrence K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, 2000.

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[SNS06] Yirong Shen, Andrew Ng, and Matthias Seeger. Fast gaussian process regression using kd-trees. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1225–1232. MIT Press, Cambridge, MA, 2006.

[SUF08] M. Salzmann, R. Urtasun, and P. Fua. Local deformation models for monocular 3d shape recovery. In *Proceedings of the Conference in Computer Vision and Pattern Recognition (CVPR) 2008*. IEEE Computer Society, 2008.

[SW02] Matthias Seeger and Christopher K. I. Williams. Fast forward selection to speed up sparse gaussian process regression. In *Advances in Neural Information Processing Systems 14*. Cambridge, MA, 2002.

[TB99] Michael E. Tipping and Christopher M. Bishop. Probabilistic principal component analysis. *Journal Of The Royal Statistical Society Series B*, 61(3):611–622, 1999.

[TSL00] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[UD07] Raquel Urtasun and T. Darrell. Discriminative gaussian process latent variable models for classification. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, June 2007.

[UD08] Raquel Urtasun and Trevor Darrell. Local probabilistic regression for activity-independent human pose inference. In *Proceedings of the Conference in Computer Vision and Pattern Recognition (CVPR) 2008*, June 2008.

[UFF06] Raquel Urtasun, David J. Fleet, and Pascal Fua. 3d people tracking with gaussian process dynamical models. In *Proceedings of the Conference in Computer Vision and Pattern Recognition (CVPR) 2006*, pages 238–245, Washington, DC, USA, 2006. IEEE Computer Society.

[UFG+08] R. Urtasun, D. J. Fleet, A. Geiger, J. Popovic, T. Darrell, and Lawrence N.D. Topologically-constrained latent variable models. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008.

[vdMPvdH07] L. J. P. van der Maaten, E. O. Postma, and H. J. van den Herik. Dimensionality reduction: A comparative review, unpublished. Matlab implementation available for all methods., 2007.

[WFH08] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):283–298, 2008.

[WS01] Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In T. Leen, T. Dietterich, and V. Tresp, editors, *Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

[WS06] Kilian Q. Weinberger and Lawrence K. Saul. An introduction to nonlinear dimensionality reduction by maximum variance unfold-

ing. In *Proceedings of the Twenty First National Conference on Artificial Intelligence (AAAI)*, 2006.

[YDD05] Changjiang Yang, Ramani Duraiswami, and Larry Davis. Efficient kernel machines using the improved fast gauss transform. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1561–1568. MIT Press, Cambridge, MA, 2005.

[ZZ03] Zhenyue Zhang and Hongyuan Zha. Nonlinear dimension reduction via local tangent space alignment. In *IDEAL 2003: International conference on intelligent data engineering and automated learning*, pages 477–481, 2003.