
3D Object Reconstruction from Partial Views

3D Objekt-Rekonstruktion aus Teilansichten
Master-Thesis von Jan Reubold aus Frankfurt a. M.
Februar 2014



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computerscience
Intelligent Autonomous Systems

3D Object Reconstruction from Partial Views
3D Objekt-Rekonstruktion aus Teilansichten

Vorgelegte Master-Thesis von Jan Reubold aus Frankfurt a. M.

1. Gutachten: Jan Peters
2. Gutachten: Herke v. Hoof

Tag der Einreichung:

Erklärung zur Master-Thesis

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

Darmstadt, den 17. Februar 2014

(J. Reubold)

Abstract

The problem to reconstruct the shape of an object given a single view of the object is an under-constraint problem. Current approaches try to solve the problem by exploiting primitive geometric properties of the objects, rely on user interaction or have strong assumptions on the application field. More advanced approaches in this field make use of machine learning techniques to learn the rules for the reconstruction from the data instead of using predefined ones. One such approach [8] achieves state-of-the-art results by using a Gaussian Process Latent Variable Model (GP-LVM) for learning the rules, but also only works on a single category.

In this thesis we build a flexible algorithm which works on data of multiple categories. We make use of a variant of the GP-LVM, called subspace GP-LVM, to extract the shape characteristics of the data. To relax the assumptions on the input data we group objects of similar shapes and learn the shape characteristics of each of these groups. So we have a multiclass reconstruction algorithm and can still make use of the flexible subspace GP-LVM.

The algorithm works with two observations, the frontal- and back view of the object and does not rely on user interaction. For the reconstruction it decides automatically, depending on the input data by using a classification algorithm which model to use.

We show that the algorithm achieves reasonable results on a dataset of 259 objects from 10 categories and provide discussions on the crucial design solutions with extensive evaluations.

Das Problem, mit Hilfe einer einzelnen Teilansicht, die Form eines Objektes zu rekonstruieren, ist ein unterdeterminiertes Problem. Aktuelle Arbeiten auf dem Gebiet versuchen dieses Problem zu lösen, indem Sie z.B. primitive geometrische Eigenschaften der Objekte ausnutzen oder mit Hilfe von Benutzereingaben oder durch starke Beschränkungen des Anwendungsgebietes das Problem vereinfachen. Modernere Arbeiten auf dem Gebiet lernen, mit Hilfe machineller Lerntechniken, die Regeln für das Rekonstruieren der Objekte direkt von den Daten, anstatt auf vorher festgelegte Regeln zurückzugreifen. Ein solcher Ansatz [8] erzielte bisher nicht erreichte Resultate mit Hilfe eines sogenannten GP-LVM (Gaussian Process Latent Variable Model), welches benutzt wurde, um die Regeln für die Rekonstruktion zu lernen. Dieser Algorithmus kann jedoch auch nur für eine einzelne Kategorie von Objekten gelernt werden.

In dieser Thesis werden wir einen flexiblen Algorithmus entwickeln, welcher auch Daten mit Objekten aus unterschiedlichen Kategorien verarbeiten kann. Um die Regeln aus den Daten zu lernen, benutzen wir eine Variante des GP-LVM, genannt subspace GP-LVM. Um die Beschränkungen des Anwendungsgebietes zu entspannen, gruppieren wir Objekte mit ähnlicher Form und lernen die Regeln zur Rekonstruktion für jede dieser Gruppen seperat. Damit erreichen wir, dass wir einen Algorithmus entwickeln können, der mit verschiedenen Kategorien gleichzeitig zurechtkommt und können trotzdem auf das flexible subspace GP-LVM zurückgreifen.

Der Algorithmus arbeitet auf Basis von zwei Observationen, der Front- und der Rückseite des Objektes und kommt ohne jegliche Benutzereingaben zurecht. Bei der Rekonstruktion klassifiziert der Algorithmus automatisch das Eingabeobjekt und weist es einer Gruppe zu. Er wählt das zu der Gruppe dazugehörige Model (subspace GP-LVM) für die eigentliche Rekonstruktion aus.

Wir zeigen an einem Datensatz mit 259 Objekten aus 10 Kategorien, dass der Algorithmus hochwertige Ergebnisse liefert. Desweiteren stellen wir Diskussionen über die wichtigsten Designentscheidungen, sowie diesbezügliche ausführliche Evaluationen zur Verfügung.

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Thesis Goal	2
1.3	Approach	3
1.4	Related Work	4
1.4.1	Restrictions and Image Cues	4
1.4.2	Machine Learning Approaches	5
1.5	Overview	7
2	Model Learning Algorithms	8
2.1	Principal Component Analysis (PCA)	9
2.2	Probabilistic PCA	11
2.3	Dual Probabilistic PCA	12
2.4	Benefits of Nonlinear Kernel Functions	13
2.5	Gaussian Process Latent Variable Model (GP-LVM)	15
3	Methods	19
3.1	Learning Stage	20
3.1.1	Depth Map Processing	20
3.1.2	Clustering Algorithm	21
3.1.3	Gaussian Fitting	22
3.1.4	Model Learning	23
3.2	Reconstruction Stage	25
4	Experiments	27
4.1	Clustering	27
4.1.1	K-Means	30
4.1.2	Hierarchical Clustering	32
4.1.3	Spectral Clustering	34
4.2	Reconstruction	36
5	Conclusion	39
5.1	Dataset	39
5.2	Clustering	39
5.3	Future Work	40
5.3.1	Unlearned Categories	40
5.3.2	Cluster Merging	40

Symbols and Notation

Symbol: Meaning:

x, y	scalars
$\lceil x \rceil$	value of x rounded up
$\lfloor x \rfloor$	value of x rounded down
\mathbf{x}, \mathbf{y}	vectors (bold lower case letters)
\mathbf{X}, \mathbf{Y}	matrices (bold capital letters)
x_i	the i -th element of the vector \mathbf{x}
\mathbf{X}_{ij}	the element in the i -th row and j -th column of matrix \mathbf{X}
\mathbf{x}_i	the i -th column vector of the matrix \mathbf{X}
\mathbf{x}^T	the transpose of vector \mathbf{x}
\mathcal{GP}	Gaussian process
f^Y	Gaussian process to the observation space Y
\mathbf{K}	covariance/kernel matrix
k	kernel function
$\phi(\mathbf{x})$	mapping function ϕ maps the vector \mathbf{x} into another space
\mathbf{I}	the identity matrix
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian (normal) distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
$\text{tr}(\mathbf{X})$	trace of square matrix \mathbf{X}
Φ	hyperparameters of the kernel

1 Introduction

1966 Marvin Minsky, a renowned artificial intelligent scientist at the MIT, gave his undergraduate student Gerald Jay Sussman the task to “spend the summer linking a camera to a computer and getting the computer to describe what it saw” [6]. Nearly 50 years later, we know this problem to be slightly more difficult.

But how could Marvin Minsky underestimate the field of computer vision so heavily? Think of the ease we humans perceive the structure of the world around us. When looking at an image you can identify each person in the picture easily, even if most of the person is somehow occluded. Given a set of images (like in Fig. 1) you can effortlessly identify the object contained in each image although the given representations range from a painting to a photo, a cartoon, or even a collage.

It looks like an easy to solve task for a computer. But why is it not? Why is a task like identifying all images containing some representation of an specific object currently not feasible for a computer?

A big problem with computer vision approaches is that they are inverse problems, where you try to recover some unknowns with too little information to fully specify a solution. You have to make use of models (e.g. geometrical, physical, statistical, or learning theoretical) to get a more specific solution.

Nevertheless, due to our developments in technology and the corresponding growth of digital content (e.g. photos on flickr, videos on youtube), the need for computer vision algorithms has grown. And so more and more algorithms in the field of computer vision are developed.

What is already possible today? Are there computer vision approaches applied in real world applications?



Figure 1: Different representations of an elephant, ranging from a real elephant to a collage or a tattoo of one.¹

¹ images taken from:
http://www.filmweb.no/bilder/migration_catalog/article605252.ece/representations/i635w/Babar
http://www.elephantparade.com/wp-content/uploads/2013/07/punkt-stretch-elefant_berg_caceis_lux.jpg
<http://www.hf.uio.no/ikos/english/research/projects/animals/photos/SJOLAN~1.JPG>
<http://4.bp.blogspot.com/-ts5ZtNSUUus/Ugc6i0ibitI/AAAAAAAAAS8g/jYAzQuf-MaU/s1600/bm-image-750898.png>
<http://fayedodgeszombies.com/files/2010/11/Elephant-Collage.jpg>
http://www.mrsbrownart.com/artwork/elmer_800.jpg

1.1 Motivation

Today, computer vision algorithms are used in a wide range of real world applications, e.g., our postal offices use optical character recognition (OCR) algorithms for reading handwritten postal codes [16]. Or there are companies using machine inspection algorithms for detecting defects in their products [26]. Some of today's cars are equipped with obstacle detection algorithms for detecting unexpected obstacles [24] such as pedestrians on the street (see Fig. 2). Today computer vision algorithms are also widely used in medical imaging applications [12].



Figure 2: Obstacle detection system in a car for detecting pedestrians and other moving obstacles to help avoiding collisions.²

In robot grasping tasks computer vision algorithms have already been applied [33] to gather information (e.g. distance, position, size, orientation) about the object. We want to extend its use in this field and additionally want to recover the shape of the back side of an object to allow a more secure grasping of objects of an unknown shape. But what are the challenges of such an approach?

Single view 3D shape reconstruction of objects is still one of the challenging problems in today's computer vision research. The ability to reconstruct the shape of an object given only one partial view of the scene strongly depends on accumulated knowledge about objects and their shapes. Because it is a so highly under-constrained problem, many of today's single view 3D reconstruction algorithms rely on hard constraints and strong assumptions about the input. [2, 9, 15]

1.2 Thesis Goal

In this thesis we will develop a multiclass single view 3D object shape reconstruction algorithm to learn the shape of the back side of a given object. The rules of how to reconstruct the back side of a given object should be learned from data. Further the algorithm should provide reasonable predictions of the shape of instances from unlearned classes.



Figure 3: Data acquiring setup. Two Microsoft Kinects positioned opposite of each other for taking the frontal and back view of the object in the middle simultaneously.

The algorithm takes a single depth map, which represents the frontal view of the object as an input and returns a depth map as the result of the reconstruction process representing the back view of the object.

The dataset is acquired by two kinects (see Fig. 3), placed opposite of each other with the object in their middle, capturing the frontal- and back view of the object. If a more complete reconstruction is desired, more observation spaces/kinects can be added to the algorithm.

² image taken from: <http://2.bp.blogspot.com/-EeynYldZkNY/UfgIYSudw6I/AAAAAAAAAdE/7t1Pg24Y01E/s1600/volvo2.jpg>

1.3 Approach

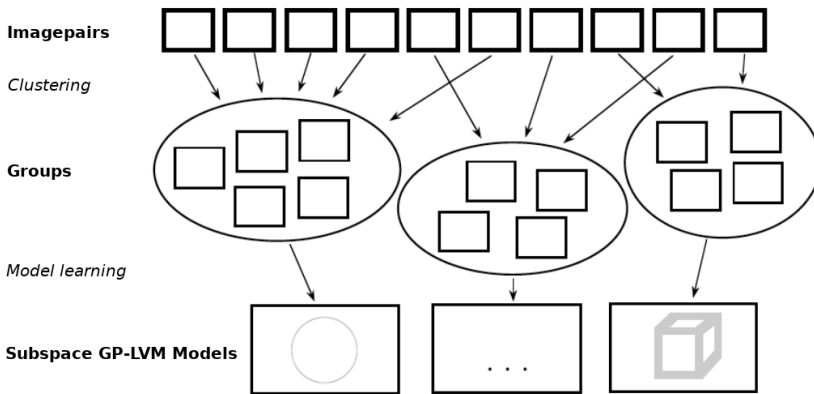


Figure 4: Learning Stage: A spectral clustering algorithm processes the image pairs and splits them into groups with respect to their shapes. Then a model is learned on each group, which represents the most significant shape characteristics of the objects in the corresponding group.

Our algorithm can be divided into two parts, the learning- and the reconstruction stage. In the learning stage (see Fig. 4) we cluster our dataset of paired depth maps into groups with respect to their shapes.

For each group of the clustered dataset, we then learn a model, which represents our prior knowledge of the shape of the objects in the group. We do not require the object to be symmetric so we use an algorithm for learning the model, which tries to find the significant shape characteristics of the group, instead of trying to just exploit the symmetric properties of the object.

In the reconstruction stage (see Fig. 5) we take as input a single depth map of the object. If the object is an instance of an object category for which a model is already learned, the classification algorithm finds the cluster it belongs to. Otherwise, if the category is not already learned, our algorithm tries to make a prediction by inferring the shape of the object by using the model with the most similar shape. With the depth map of the object assigned, we can take the corresponding model and synthesize the data for the back view of the object.

For the models representing the prior knowledge we make use of so called Gaussian Process Latent Variable Models (GP-LVMs), or more precisely an extension of these GP-LVMs called subspace Gaussian Process Latent Variable Models (subspace GP-LVMs). GP-LVMs utilize gaussian process regression to automatically extract the unknown low dimensional information of a given

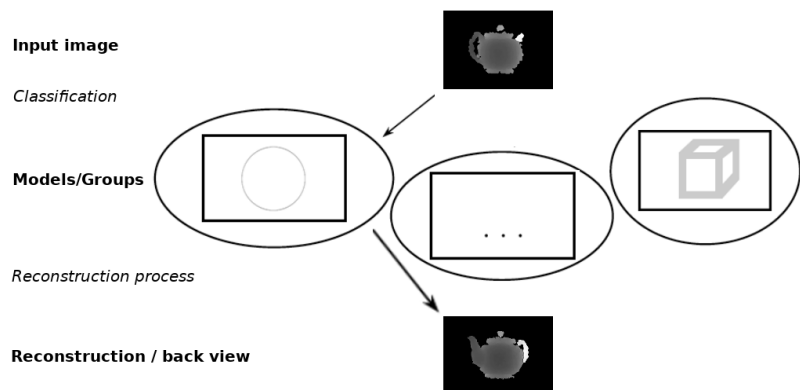


Figure 5: Reconstruction Stage: For each group the probability that the input image belongs to this group is computed. The group with the highest probability is selected and the subspace GP-LVM of this group is applied to predict the corresponding back view of the object.

high dimensional observation. Additionally, the variant we make use of, subspace GP-LVM, learns the shared low dimensional information of multiple corresponding sets of observations.

1.4 Related Work

The approach to reconstruct the shape of an object given only a single image as input, is an under-constrained problem. The lack of cues, e.g. vanishing lines, in comparison to reconstruction algorithms working with multiple views of an object, is for example one reason that makes it more difficult.

Often, work in this field relies on user interaction [17] or hard constraints on the objects or application fields [2]. Many of these approaches beside having strong assumptions on the input data also only reconstruct the shape of the part of the object visible on the input image [2, 9, 15].

1.4.1 Restrictions and Image Cues

A group of algorithms restrict the application field to just planar outdoor architecture scenes where e.g. [9] segment the image into ground, sky and verticals to then build a coarse pop-up reconstruction depending on the segmentations.



Figure 6: The modeling pipeline. Jiang et al. first calibrate the camera according to the users specified frustum vertices and reconstruct a set of 3D points. The architecture components (i.e. walls and roofs) are then interactively decomposed and modeled. Shape details can be added if necessary. Lastly, the final model is textured with their texture enhancement technique. *Content taken from [17].*

Other works try to solve the problem by exploiting image cues still present in single view reconstruction problems. Some make use of primitive geometrical characteristics of the objects [17, 20] (see Fig. 6), or they focus on curved objects to exploit other image cues. For instance in [37], Zang et al. introduce several user constraints (e.g. normal map, etc) to finally formulate the reconstruction problem as a linearly constrained quadratic optimization problem, which has a closed form solution. Prasad et al. [30] use wireframes to reconstruct the shapes of curved objects and also rely on user interaction to reduce the complexity of 3D object topology.

Han et al. [14] restrict the application field to polyhedral objects, grass and trees, where they then apply Bayesian reconstruction.

Ben Amor et al. [20] use the symmetric properties of the objects to recover the complete shape. In their work they use a two-step algorithm, where they first find the symmetry candidates with the highest votes via a voting system and then determine the corresponding linear- or rotational extrusion.

Black et al. [13, 35] restrict the application field to human body shapes, which allows them to use application field specific solutions (e.g. parametric morphable models). In [13] they also incorporate shading cues for the single view reconstruction.

1.4.2 Machine Learning Approaches

In [36] (see Fig. 7) Sun et al. proposed a more advanced semi-automatic approach for single view reconstruction, which also recovers the shape of the object part which is not visible on the input image. They split the process into two parts. First they used Depth-Encoded Hough Voting (DEHV), a voting scheme that incorporates depth information into the process where they learn distributions of image features for an object category. With the so learned probabilistic models of object categories they can not only localize and recognize the object in an image, but they also partially recover the shape (viewpoint limited) of it. In the second step to complete the shape recovery process, they use 3D shape exemplars from a database of 3D CAD models.

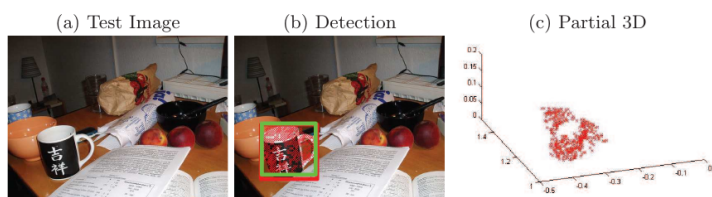


Figure 7: Illustration of the key steps in Sun et al's [36] method. Given a single (previously) unseen test image (panel a), their DEHV (Depth-Encoded Hough Voting-based) scheme is used to detect objects (panel b). The ground truth bounding box is shown in red. Their detection is shown in green. The centers of the image patches which cast votes for the object location are shown in red crosses. During detection, their method simultaneously infers the depth map of the detected object (panel c). This allows the estimation of the partial 3D shape of the object from a single image. *Content taken from [36].*

Although this work showed some promising results, it is not suitable for our goals. For one thing, our algorithm should work autonomously, but the algorithms presented by Sun et al. relies on user interaction. But more crucial, the shape recovery process is not generative. Sun et al. used previously collected 3D CAD models to recover the complete shape by finding the best fit for the object reconstruction.

Approaches like Sun et al's learn from data rather than exploiting image cues. They are more suitable when the algorithm should be used not only on a specific category, but should work on arbitrary categories. A downside of these approaches is that they often are significantly more expensive with regard to the computation time because of the required learning process. Due to the additional flexibility you get with such an approach we focused our research on these types of algorithms.

Methods which try to infer the shape of the object by just exploiting image cues, can be seen as a human without a memory or intelligence. The human receives rules from someone, which he has to follow. These rules depend on relations of image cues and the shape of objects and help the human to infer the objects shapes. But he never learns from the data and never makes his own rules. In an alternative scenario, he could for example divide the images into groups, where the members of a group have similar shapes. Then, he could learn for himself what the significant shape characteristics of these groups are and so more accurately reconstruct new objects with similar shapes.

That is exactly what some of the methods which learn from data and make use of machine learning techniques do. These methods learn priors on the object shapes from data, which can be seen as the attempt to copy the learning process of humans.

An approach using machine learning techniques was proposed by Rother et al. [32]. They learn shape priors on crude voxel representations, which are then applied to segmentation, recognition and shape reconstruction. Their approach is limited to simple and rigid objects like cups and plates.

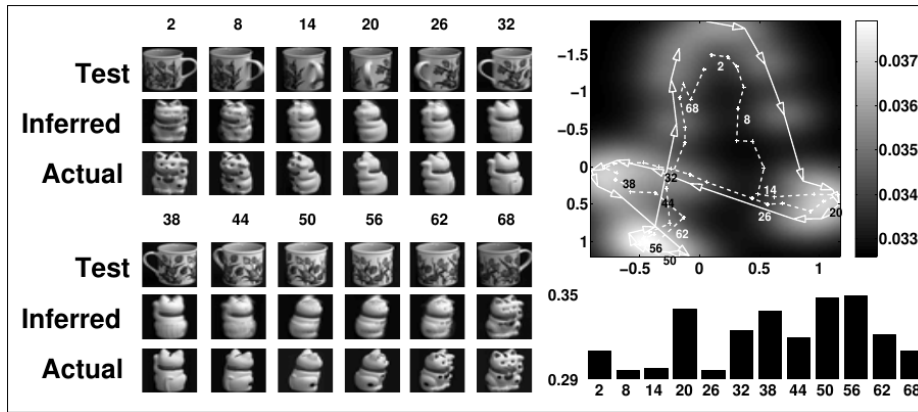


Figure 8: Shon et al’s approach: Synthesis of novel views using a shared latent variable model: After training on 24 paired images of a mug with a cat figurine (out of 72 total paired images), Shone et al. ask the model to infer the remaining 48 poses of the cat, given 48 novel views of the mug. The system uses an inverse Gaussian process model to infer a 2D latent representation for each of the 48 before unseen mugs, then synthesizes a corresponding view of the cat figurine. At the left, the before unseen mug images (‘test’), the synthesized cat images (‘inferred’), and the actual views of the cat figurine from the database (‘actual’) are plotted. In the upper right the models uncertainty in the latent space is plotted. The 24 latent coordinates from the training data are plotted as arrows, while the 48 novel latent points are plotted as crosses on a dashed line. At lower right the certainty for each latent point prediction is plotted. Note the low certainty for the blurry inferred images labeled 8, 14, and 26. *Content taken from [34].*

One approach for more general use achieving state-of-the-art results was proposed by Chen et al. [8]. They concentrate on learning the prior knowledge of the shape of a specific category from datasets with machine learning techniques. In their paper Chen et al. applied their framework to a setup, where they took 2D silhouettes of human bodies as input and then reconstructed the complete shape of the body. Unlike many other approaches we already mentioned in this section, they do not rely on heuristic regularities or predefined parametrical models which are then only suitable for particular scenes. Instead they assume that the latent factors of the object shapes are unknown in advance and are then obtained automatically during the learning process. These latent factors are learned with a GP-LVM, which is used because it automatically extracts the unknown low dimensional embedded information of the high dimensional object observations. The approach does not require any user interaction and can be generalized to reconstruct the shape of various categories of objects.

We took this approach of solving the single view reconstruction problem, especially the utilization of the GP-LVM algorithm, as the basis of our research and found an extension of the GP-LVM proposed by Shon et al. [34] which is more suitable for our task. They developed an algorithm (see Fig. 8) that learns the shared embedded low dimensional information of multiple (heterogenous) observation spaces. It can then synthesize new data from learned correspon-

dences. PCA is first applied separately to each of the sets of observations and then the average of these solutions is used to initialize the GP-LVM. A second set of $\mathcal{G}\mathcal{P}$ s are learned, which map back from the observation spaces to the latent space. That means Shon et al. assume that the generative mappings have a smooth inverse.

The algorithm used in this thesis builds up on the same idea as Shon et al's approach. Ek proposed two extensions to the GP-LVM [11], called shared GP-LVM and subspace GP-LVM, which also build a shared latent space. A downside of the shared model of Shon et. al is that they assume that the observations work in spaces with equal dimensions or modes of variability. With the development of the shared GP-LVM Ek relaxed this assumption by introducing a back-constraint to the shared model. This model can be applied in situations where you are interested in inferring the item of one observation space given the corresponding item of the other observation space, but not the other way around.

In this thesis we will use subspace GP-LVM, where the generalization is established by introducing a factorized latent space, consisting of a shared- and private subspace. Other than with shared GP-LVMs, where the model encourage the latent space to fully explain the correlated variance and to handle the non-correlated variance as noise, in subspace GP-LVMs the non-correlated variance is modeled separately by additional private latent spaces. Instead of just by our Gaussian noise model the non-correlated variance is here also explained by a $\mathcal{G}\mathcal{P}$.

Therefore, the subspace GP-LVM is not only flexible as shown in [11], but is also applicable to almost any set of corresponding observations.

1.5 Overview

After the introduction and an outline of existing related work in the field of single view reconstruction, the following chapters will explain the algorithms we make use of, provide evaluations of our work and give a summary and a conclusion at the end.

In Chapter 2 the algorithm for learning the model is explained. Therefore, we first explain what Principle Component Analysis (PCA) is and then review step-by-step the benefits of extensions to PCA, its drawbacks and the algorithms themselves in detail. In that way we will take a look at probabilistic PCA and its dual, arriving at the GP-LVM.

In Chapter 3 we explain our algorithm and the rest of the algorithms we make use of in detail. The chapter will be divided into two parts, where we describe the learning- and the reconstruction stage.

In Chapter 4 we provide extensive evaluations on the decision which clustering algorithm to use and evaluations on the performance of our algorithm.

In Chapter 5 the properties of the algorithm are summarized and ideas for further improvements are proposed.

2 Model Learning Algorithms

The core part of our approach is the model representing the prior knowledge of the object shapes. As explained in the related work section, we decided to make use of a machine learning algorithm for learning the model. The benefit of a machine learning algorithm is that it learns from data. For example, machine learning algorithms are often applied when the exact rules of how the algorithm should work are unknown. In some cases you can create input/output pairs, but you do not know the exact relationships between input and output. They are also used in data mining programs, where they can find connections and meanings in huge datasets.

For our algorithm we make use of the subspace Gaussian Process Latent Variable Model (subspace GP-LVM) to extract the shared low dimensional information of the two observations.

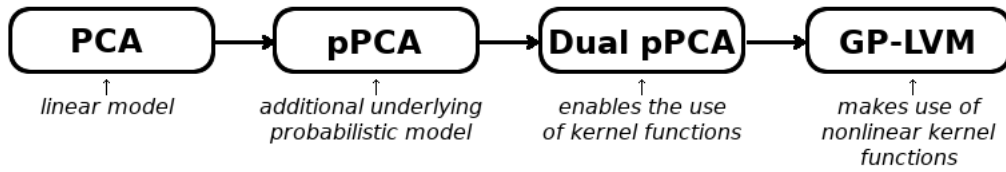


Figure 9: From PCA to GP-LVM

This chapter will explain the core part of the applied machine learning algorithm, we make use of for the model learning, in detail (see Tab. 1). Therefore, we start by explaining Principal Component Analysis (PCA) [19] and then take a look at PCA from two different perspectives allowing for a probabilistic view on PCA. The first of the two interpretations, called probabilistic PCA (pPCA) [3], integrates the latent variables out of the likelihood function to optimize over the parameters, whereas the second interpretation, called dual probabilistic PCA (Dual pPCA), optimises over the latent variables instead and integrates over the parameters. We will see that Dual pPCA additionally allows us to make use of nonlinear mappings. The probabilistic version of PCA using nonlinear mappings is called Gaussian Process Latent Variable Model (GP-LVM) (see Fig. 9). At the end of the Chapter we will learn about two extensions to GP-LVM, where one of them is the subspace GP-LVM.

	$Y \rightarrow X$	$X \rightarrow Y$	Nonlinear	Probabilistic	Convex
PCA	Y	Y	N	N	Y
pPCA	Y	Y	N	Y	Y
Dual pPCA	Y	Y	N	Y	Y
GP-LVM	N	Y	Y	Y	N

Table 1: Overview of the described algorithms. A 'Y' indicates the algorithm exhibits that property, a 'N' indicates the contrary. The characteristics of the algorithm are: $Y \rightarrow X$: does the method lead to a mapping from data to embedded space? $X \rightarrow Y$: does the method lead to a mapping from the embedded to the data-space? Nonlinear: does the method allow for nonlinear embeddings? Probabilistic: is the method probabilistic? Convex: algorithms that are considered convex have a unique solution, for the others local optima can occur. *Content taken from [22].*

2.1 Principal Component Analysis (PCA)

PCA can be applied when we have obtained measures of a high number of variables and we want to create a smaller number of artificial variables (principal components). The artificial variables should capture most of the information (variances) contained in the original variables. So PCA is a dimensionality reduction algorithm (see Fig. 10), we can apply when we have a high dimensional dataspace and the variables are correlated with each other.

We assume that we have a dataset $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N] \in \mathbb{R}^{M \times N}$ and want to find the corresponding latent points $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$. Each of the N column vectors \mathbf{y}_i , $i, j, k \in \{1 \dots N\}$ represents one measure of all variables. \mathbf{y}_i is M -dimensional and can also be expressed as:

$$\mathbf{y}_i = \sum_{j=1}^M x_{j,i} \mathbf{w}_j, \text{ where } \mathbf{w}_j^T \mathbf{w}_k = \delta_{j,k}, \text{ with } \delta_{j,k} \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise.} \end{cases}$$

We now want to find a mapping into a D -dimensional space to map all $\mathbf{y}_i \rightarrow \mathbf{x}_i$ with minimal information loss and $D \ll M$:

$$\mathbf{y}_i = \underbrace{\sum_{j=1}^D x_{j,i} \mathbf{w}_j}_{\text{Approx. } \tilde{\mathbf{y}}_i} + \underbrace{\sum_{k=D+1}^M x_{k,i} \mathbf{w}_k}_{\text{Error}}$$

where $x_{j,i}$ can be interpreted as $x_{j,i} = \mathbf{w}_j^T \mathbf{y}_i$ and the squared error is given by:

$$\mathbf{E}(\mathbf{W}) = \sum_{i=1}^N \|\mathbf{y}_i - \tilde{\mathbf{y}}_i\|^2.$$

To show that when the empirical mean of the data is zero, minimizing the squared error is equivalent to maximizing the variance of the projection, we rewrite the error assuming a single basis vector:

$$\begin{aligned} \mathbf{E}(\mathbf{w}) &= \sum_{i=1}^N \|\mathbf{y}_i - \tilde{\mathbf{y}}_i\|^2 \\ &= \sum_{i=1}^N \|\mathbf{y}_i - (\mathbf{w}^T \mathbf{y}_i) \mathbf{w}\|^2 \\ &= \sum_{i=1}^N \|\mathbf{y}_i\|^2 - 2(\mathbf{w}^T \mathbf{y}_i)^2 + (\mathbf{w}^T \mathbf{y}_i)^2 * \mathbf{w}^T \mathbf{w} \\ &= \sum_{i=1}^N \|\mathbf{y}_i\|^2 - (\mathbf{w}^T \mathbf{y}_i)^2 = \sum_{i=1}^N \|\mathbf{y}_i\|^2 - (x_i)^2. \end{aligned}$$

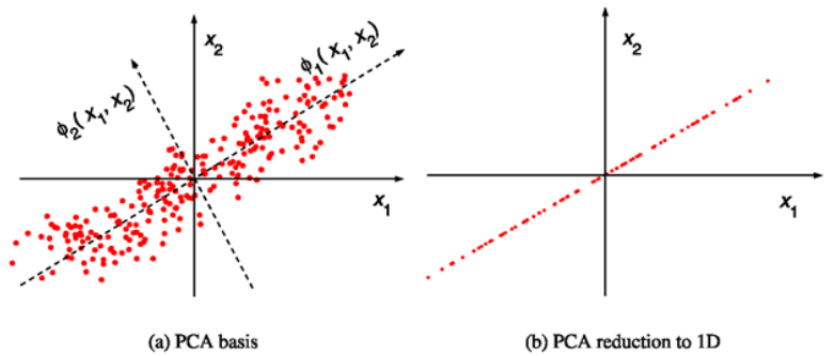


Figure 10: PCA from 2D to 1D: seeks the principal axis where the variance of the data is maximized.³

³ image acquired (2012) from: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcrJoxuitHxwngqL68n0MtSfjDGiyZ_lXPEnN

So first the data has to be centered by setting the empirical mean to zero.

$$\widehat{\mathbf{Y}} = \mathbf{Y} - [\bar{\mathbf{y}}, \dots, \bar{\mathbf{y}}] \quad , \quad \bar{\mathbf{y}} = \frac{1}{N} \sum_{i=1}^N \mathbf{y}_i.$$

We have to maximize the variance of the projection:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \bar{\mathbf{x}})^2 = \mathbf{w}^T \mathbf{C} \mathbf{w} \quad , \text{with} \quad \bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i = \mathbf{w}^T \bar{\mathbf{y}}.$$

When maximizing σ^2 (Langrangian, with the constraint $\|\mathbf{w}\| = 1$), we get $\mathbf{C} \mathbf{w} = \lambda \mathbf{w}$. So we have to compute the eigenvectors and eigenvalues of the covariance matrix \mathbf{C} .

But because the covariance matrix is quickly getting too large to be handled efficiently, we cannot do this on the direct way. For example for a 256×256 image, the covariance matrix has 2^{32} entries and has the size of 2^{35} Bytes (32GB). That means, the dimensions we work in are high, most of the time higher than the amount of examples we have. Also we do not need all the eigenvectors and -values of the covariance matrix, the largest ones would be sufficient. Let us look at the covariance matrix of the data and express it differently:

$$\mathbf{C} = \frac{1}{N} \sum_{i=1}^N (\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^T = \frac{1}{N} \widehat{\mathbf{Y}} \widehat{\mathbf{Y}}^T.$$

When we now apply singular value decomposition (SVD) to the centered data matrix $\widehat{\mathbf{Y}}$,

$$\frac{1}{N} \widehat{\mathbf{Y}} \widehat{\mathbf{Y}}^T = \frac{1}{N} \mathbf{U} \mathbf{S} \mathbf{V}^T (\mathbf{U} \mathbf{S} \mathbf{V}^T)^T = \mathbf{U} \left(\frac{1}{N} \mathbf{S}^2 \right) \mathbf{U}^T = \mathbf{U} \Lambda \mathbf{U}^T,$$

the left-singular vectors are the eigenvectors of the covariance matrix and with the singular values we can compute the eigenvalues of the covariance matrix:

$$\lambda_i = \frac{1}{N} s_i^2.$$

Now we can compute the largest eigenvectors and eigenvalues of the covariance matrix without the need to ever build and store the covariance matrix itself.

$$\mathbf{x}_i = \mathbf{W}^T (\mathbf{y}_i - \bar{\mathbf{y}}) \quad , \text{with} \quad \mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_D],$$

$$\tilde{\mathbf{y}}_i = \bar{\mathbf{y}} + \mathbf{W} \mathbf{x}_i.$$

So with this method PCA is efficiently computable. But what are the drawbacks of this method?

For one PCA is a simple linear transformation and does not incorporate an underlying probabilistic model for the data, which is necessary if you want to apply Bayesian methods or want to deal with missing data values. Additionally, due to the fact that it is linear, it is not as flexible as nonlinear algorithms. We will take a closer look at this later. Let us first inspect a variant of PCA, called probabilistic PCA, which solves the first mentioned drawback of PCA.

2.2 Probabilistic PCA

Probabilistic PCA (pPCA) is a latent variable model (see Fig. 11). It is an interpretation of PCA, which incorporates an underlying probabilistic model. It takes a probabilistic, generative view of the data. The maximum likelihood solution for a pPCA model can be found by applying SVD to the data's covariance matrix.

In pPCA we express the data matrix $\hat{\mathbf{Y}}$ and the matrix of the latent points \mathbf{X} differently than in PCA:

$$\hat{\mathbf{Y}} = [\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_N]^T \in \mathbb{R}^{N \times M} \quad \mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^T \in \mathbb{R}^{N \times D}.$$

pPCA assumes that the underlying latent variables \mathbf{x}_i have Gaussian distributions and are i.i.d.:

$$p(\mathbf{X}) = \prod_{i=1}^N \mathcal{N}(\mathbf{x}_i | 0, \mathbf{I}),$$

and that a linear relation exists between latent- and observed variables:

$$\hat{\mathbf{y}}_i = \mathbf{W}\mathbf{x}_i + \mathbf{e}, \text{ with } \mathbf{W} \in \mathbb{R}^{M \times D},$$

where \mathbf{W} specifies the linear relationship between data- and latent space and \mathbf{e} is noise sampled from a spherical Gaussian distribution, $\mathbf{e} = \mathcal{N}(\mathbf{e} | 0, \sigma^2 \mathbf{I})$.

The likelihood of a data point is:

$$p(\hat{\mathbf{y}}_i | \mathbf{x}_i, \mathbf{W}) = \mathcal{N}(\hat{\mathbf{y}}_i | \mathbf{W}\mathbf{x}_i, \sigma^2 \mathbf{I}).$$

It is not possible to optimize the likelihood by integrating out both, the hidden variable \mathbf{x}_i and the parameters \mathbf{W} together. So we integrate out the hidden variable \mathbf{x}_i :

$$p(\hat{\mathbf{y}}_i | \mathbf{W}) = \int p(\hat{\mathbf{y}}_i | \mathbf{x}_i, \mathbf{W}) p(\mathbf{x}_i) d\mathbf{x}_i$$

Because of the independence of the data points, we can express the marginal likelihood of the complete dataset as:

$$p(\hat{\mathbf{Y}} | \mathbf{W}) = \prod_{i=1}^N \mathcal{N}(\hat{\mathbf{y}}_i | 0, \mathbf{C}) \text{ , with } \mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2 \mathbf{I}. \quad (1)$$

The corresponding log-likelihood is then given by:

$$\mathcal{L} = -\frac{N}{2} (M \ln(2\pi) + \ln |\mathbf{C}| + \text{tr}(\mathbf{C}^{-1} \mathbf{S})) \text{ , with } \mathbf{S} = \frac{1}{N} \hat{\mathbf{Y}}^T \hat{\mathbf{Y}}.$$

It can be shown [3] that the log-likelihood is maximized when:

$$\mathbf{W}_{\text{ML}} = \mathbf{U}_D (\Lambda_D - \sigma^2 \mathbf{I})^{1/2} \mathbf{R}, \quad (2)$$

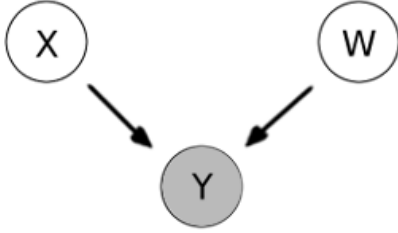


Figure 11: Graphical representation of probabilistic PCA and its Dual. In pPCA the latent variables \mathbf{X} are integrated out and the likelihood function is optimized over \mathbf{W} , where as in Dual pPCA it is the other way around, integrated over \mathbf{W} and optimizing \mathbf{X} .

where the columns of the $M \times D$ matrix \mathbf{U}_D consist of the principal eigenvectors of \mathbf{S} with the corresponding eigenvalues $\lambda_1, \dots, \lambda_D$ in the $D \times D$ matrix Λ_D . \mathbf{R} is an arbitrary $D \times D$ dimensional orthogonal rotation matrix. Additionally, when $\mathbf{W} = \mathbf{W}_{ML}$, the maximum-likelihood estimator of σ^2 is given by:

$$\sigma_{ML}^2 = \frac{1}{M-D} \sum_{j=D+1}^M \lambda_j. \quad (3)$$

After applying SVD to the covariance matrix \mathbf{S} , as we have already done in PCA for \mathbf{C} , we can compute σ_{ML}^2 with Eq. (3) and then \mathbf{W}_{ML} with Eq. (2).

The log-likelihood is optimized when \mathbf{W} spans the principal sub-space of the data $\hat{\mathbf{Y}}$ (see Eq. (2)). This is the same like in the classical PCA, where the optimal parameters can be found by applying SVD to the data $\hat{\mathbf{Y}}$.

2.3 Dual Probabilistic PCA

pPCA is a PCA variant which incorporates an underlying probabilistic model. Now we will see how we can unlock the model to work with nonlinear embeddings.

We look again at the pPCA, but set a prior on the parameters \mathbf{W} rather than on \mathbf{X} ,

$$p(\mathbf{W}) = \prod_{i=1}^M \mathcal{N}(\mathbf{w}_i | 0, \mathbf{I}), \text{ with } \mathbf{w}_i = \mathbf{W}_{i,:} \in \mathbb{R}^{1 \times D}.$$

In Bayesian frameworks parameters are seen as random variables, over which we can define a prior.

That means in the Dual of probabilistic PCA (Dual pPCA) we can marginalise over \mathbf{W} instead of \mathbf{X} :

$$p(\hat{\mathbf{Y}} | \mathbf{X}) = \prod_{i=1}^M \mathcal{N}(\hat{\mathbf{y}}_{:,i} | 0, \mathbf{K}), \text{ with } \mathbf{K} = \mathbf{X}\mathbf{X}^T + \sigma^2 \mathbf{I}. \quad (4)$$

Note that $\hat{\mathbf{y}}_{:,i}$ is not one observation of all dimensions/variables, but all observations of one single dimension/variable.

The log-likelihood of Eq. (4) is:

$$\mathcal{L} = -\frac{MN}{2} \ln(2\pi) - \frac{M}{2} \ln(|\mathbf{K}|) - \text{tr}(\mathbf{K}^{-1} \hat{\mathbf{Y}}\hat{\mathbf{Y}}^T), \quad (5)$$

where its gradients with respect to \mathbf{X} can be found as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{X}} = \mathbf{K}^{-1} \hat{\mathbf{Y}}\hat{\mathbf{Y}}^T \mathbf{K}^{-1} \mathbf{X} - M \mathbf{K}^{-1} \mathbf{X}.$$

Setting the equation to zero, dividing by M and pre-multiplying by \mathbf{K} gives:

$$\mathbf{S} \mathbf{K}^{-1} \mathbf{X} = \mathbf{X}, \text{ with } \mathbf{S} = M^{-1} \hat{\mathbf{Y}}\hat{\mathbf{Y}}^T.$$

It can be shown [22] that the objective function in Eq. (5) can be optimized, when:

$$\mathbf{X} = \mathbf{U}\mathbf{L}\mathbf{V}^T, \quad (6)$$

where $\mathbf{U} \in \mathbb{R}^{N \times D}$ are the eigenvectors of \mathbf{S} with the corresponding eigenvalues $\lambda_1, \dots, \lambda_D$. The entries on the diagonal of the diagonal matrix $\mathbf{L} \in \mathbb{R}^{D \times D}$ are $l_i = (\lambda_i - \sigma^2)^{1/2}$ and $\mathbf{V} \in \mathbb{R}^{D \times D}$ is an arbitrary rotation matrix.

pPCA and Dual pPCA are interpretations of PCA. With pPCA we benefit of an underlying probabilistical model, which e.g. permits the application of Bayesian methods and allows us to deal with missing data values. Dual pPCA is a reformulation of pPCA which further allows us to replace the inner product $\widehat{\mathbf{Y}}\widehat{\mathbf{Y}}^T$ with a kernel and thereby unlocks the use of nonlinear embeddings.

But what exactly are the benefits of using nonlinear kernels and therefore mappings?

2.4 Benefits of Nonlinear Kernel Functions

In general a dataset of N data points cannot be linearly separated in a space of dimension D , where $D < N$. Data points not linearly separable in the data space could possibly, if mapped into another space, we will call feature space, be linearly separated (see Fig. 12). If the dimensionality of the feature space is $F \geq N$, the data points would almost always be linearly separable. Therefore, let us define a mapping function $\phi(\mathbf{x})$, which maps \mathbf{x} into another space.

So, if we would project the data into an infinite dimensional feature space, all datasets could almost always be linearly separated. But how can we do computations in such a space? Would it not be too costly?

There exists a trick called kernel-trick [11] that can be applied, when input vectors to a function only occur as an inner product with each other. So let us define a function, we call kernel function, that maps vectors into another space and there computes the inner product of the mapped vectors:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j).$$

In Dual pPCA we had the case that the input vectors only occurred as inner products with each other:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j + \sigma^2 \delta_{ij}.$$

When we now apply the mapping function $\phi(\mathbf{x})$ to the input vectors, the inner product of the projected input vectors could be computed by deriving a kernel function, without the necessity to ever map them into the feature space.

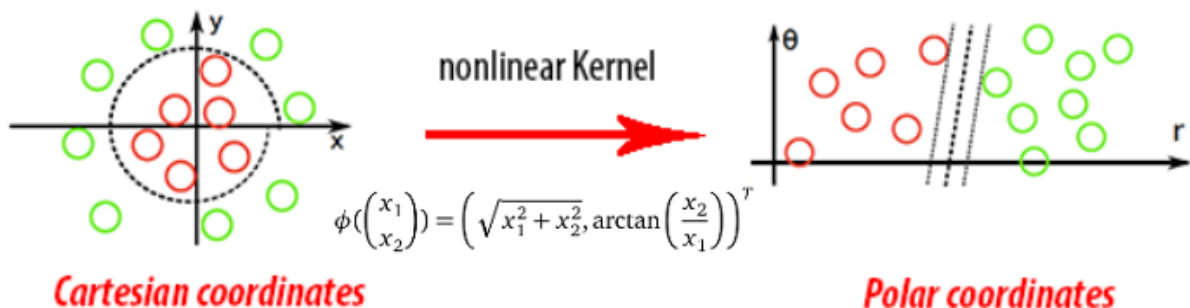


Figure 12: Benefits of a nonlinear kernel: Data that is not linearly separable in the cartesian space, is, if mapped to the polar space, linearly separable.⁴

⁴ image taken from: Peter Gehler - Lecture on Machine Learning

To give an example, consider the polynomial kernel of degree 2:

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^2, \text{ where } \mathbf{x} = (x_1, x_2), \mathbf{y} = (y_1, y_2) \in \mathbb{R}^2,$$

for better readability $\mathbf{x}_i = \mathbf{x}$ and $\mathbf{x}_j = \mathbf{y}$. With the corresponding kernel function:

$$k(\mathbf{x}, \mathbf{y}) = (x_1 y_1 + x_2 y_2)^2 = x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2.$$

The feature map ϕ for $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$ can be expressed as:

$$\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2),$$

mapping \mathbf{x} and \mathbf{y} from a 2D- into a 3D space, with the inner product of the mapped input vectors:

$$\phi(\mathbf{x})^T \phi(\mathbf{y}) = x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2.$$

As you can see the kernel function yields the same result as the inner product computed between the projected input vectors.

Our algorithm makes use of the popular RBF-kernel [7] (see Fig. 13), which maps the input vectors into an infinite dimensional space:

$$k(\mathbf{x}, \mathbf{y}) = \alpha \exp\left(-\frac{\gamma}{2}(\mathbf{x} - \mathbf{y})^T(\mathbf{x} - \mathbf{y})\right)$$

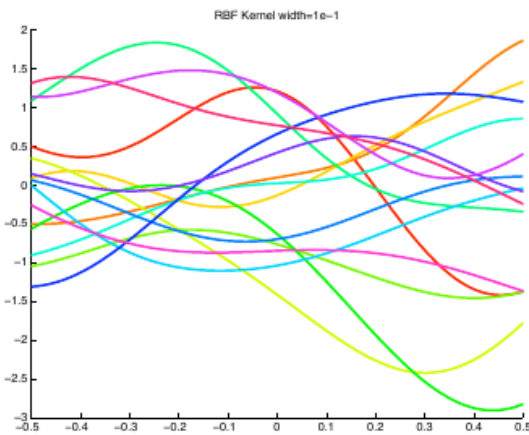


Figure 13: A RBF with kernel width of 0.1. ⁵

To verify that the RBF-kernel maps into an infinite dimensional space, we take a look at a simplified kernel for points in \mathbb{R}^2 :

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= \exp(-(\mathbf{x} - \mathbf{y})^2) \\ &= \exp(-(\mathbf{x}_1 - \mathbf{y}_1)^2 - (\mathbf{x}_2 - \mathbf{y}_2)^2) \\ &= \exp(-\mathbf{x}_1^2 + 2\mathbf{x}_1 \mathbf{y}_1 - \mathbf{y}_1^2 - \mathbf{x}_2^2 + 2\mathbf{x}_2 \mathbf{y}_2 - \mathbf{y}_2^2) \\ &= \exp(-|\mathbf{x}|^2) \exp(-|\mathbf{y}|^2) \exp(2\mathbf{x}^T \mathbf{y}), \end{aligned}$$

using a Taylor series it can be written as:

$$k(\mathbf{x}, \mathbf{y}) = \exp(-|\mathbf{x}|^2) \exp(-|\mathbf{y}|^2) \sum_{n=0}^{\infty} \frac{\exp(2\mathbf{x}^T \mathbf{y})^n}{n!}$$

When using the RBF-kernel, any combination of \mathbf{x} and \mathbf{y} results in a non-zero inner product. Therefore, the feature space F needs to be infinite dimensional.

Let us see how we can use the powerful nonlinear RBF-kernel in combination with our latent variable model.

⁵ image taken from: Ek, Torr and Lawrence - Talk on Shared Gaussian Latent Variable Models (2007)

2.5 Gaussian Process Latent Variable Model (GP-LVM)

When looking at the likelihood function of the Dual pPCA, in Eq. (4), it can be interpreted as M independent Gaussian processes mapping from the latent- into the data space sharing the same linear kernel \mathbf{K} .

To obtain the nonlinear latent variable model GP-LVM (see Fig. 14), the covariance-/kernel matrix \mathbf{K} has to be substituted by a covariance function that allows for nonlinear functions, such as the RBF-kernel.

Gaussian Process:

Gaussian processes (\mathcal{GP} s) are probabilistic models which specify distributions over function spaces. You can think of a \mathcal{GP} as the extension of a multivariate Gaussian distribution to infinite dimensionality. \mathcal{GP} s generate data on a domain so that any finite subset of the range follows a multivariate Gaussian distribution, allowing to view distribution over the function space by only focussing on instantiated points.

\mathcal{GP} s are completely determined by their mean- and covariance function, where the mean is often set to zero and the \mathcal{GP} and its properties (e.g. smoothness, stationarity, local independence) are determined by its covariance function k .

To optimize the GP-LVM with respect to the latent variables \mathbf{X} , we first have to compute the gradients of Eq. (5) with respect to the kernel \mathbf{K} :

$$\frac{\partial \mathcal{L}}{\partial \mathbf{K}} = \mathbf{K}^{-1} \widehat{\mathbf{Y}} \widehat{\mathbf{Y}}^T \mathbf{K}^{-1} - \mathbf{M} \mathbf{K}^{-1}, \quad (7)$$

which is independent of the choice of kernel matrix. After that, Eq. (7) has to be combined with $\frac{\partial \mathbf{K}}{\partial x_{i,j}}$ via chain rule.

As for nonlinear kernels typically no closed form solution exists and we have to deal with an objective function with likely many local optima, solving an eigenvalue problem is no longer sufficient and therefore optimizing the objective function is more difficult. We have to use a gradient based method because the log-likelihood function is a highly nonlinear function of the embeddings and the parameters and no closed form solution exists. Therefore, we apply a Scaled Conjugated Gradient approach (SCG) [28] to the gradients in combination with Eq. (5) for the optimization. To optimize the kernel parameters, gradients with respect to these parameters have to be computed and then used to jointly optimize the latent variables and the parameters of the kernel.

After training a GP-LVM, how can we determine the latent point \mathbf{x}_* and the likelihood of a new previously unseen data point \mathbf{y}_* ?

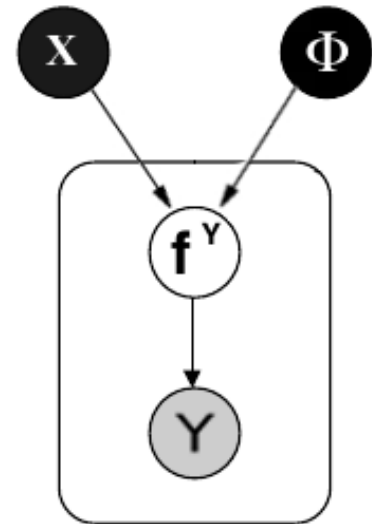


Figure 14: Graphical representation of a GP-LVM, with the Gaussian process as a latent variable model.

The joint distribution of the training data points and the unseen data point is:

$$\begin{bmatrix} \widehat{\mathbf{Y}} \\ \mathbf{y}_* \end{bmatrix} \sim \left(0, \begin{bmatrix} \mathbf{K}_{\mathbf{I},\mathbf{I}} & \mathbf{k}_{\mathbf{I},*} \\ \mathbf{k}_{\mathbf{I},*}^T & k(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix} \right),$$

where $\mathbf{K}_{\mathbf{I},\mathbf{I}}$ is the kernel matrix developed from the latent points of the training data. $\mathbf{k}_{\mathbf{I},*}$ is a column vector containing the elements of the kernel matrix computed between the latent points of the training data and the latent representation \mathbf{x}_* of the new data point \mathbf{y}_* .

Again, it is not as easy as with PCA and its interpretations. To compute the likelihood of a new data point \mathbf{y}_* , we first have to find a MAP solution for the corresponding latent point \mathbf{x}_* . When the variance over each output dimension is constant, we can approximate the likelihood. The likelihood is given by:

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

where

$$\mu = \mathbf{Y}^T \mathbf{K}_{\mathbf{I},\mathbf{I}}^{-1} \mathbf{k}_{\mathbf{I},*}.$$

The variance is given by:

$$\sigma^2 = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_{\mathbf{I},*}^T \mathbf{K}_{\mathbf{I},\mathbf{I}}^{-1} \mathbf{k}_{\mathbf{I},*}.$$

With the likelihood function given we can now find the MAP solution for \mathbf{x}_* using gradient techniques. To approximate the likelihood of \mathbf{y}_* we have to project \mathbf{x}_* back into the data space and compute the probability of \mathbf{y}_* under the resulting distribution.

A problem again comes with \mathbf{X} possibly being multi-modal with respect to \mathbf{x}_* . Because of this, it is not guaranteed that the solution is unique. In some cases sampling methods are needed to evaluate the likelihood [22].

This is the theory behind a GP-LVM, but as the optimization problem is nonlinear and high dimensional (ND interdependent parameters/latent variables before also taking the kernel parameters into account), Lawrence [22] provides an approximation, which relies on forced sparsification of the model. The sparsification is done by selecting a subset \mathbf{A} , called active set, of the original set. The algorithm used for selecting the subset, called informative vector machine (IVM), was also developed by Lawrence et al. [23]. Given the sparsification, the optimization of the points is faster than the optimization of the full dataset. The likelihood of the active set can be computed as:

$$p(\mathbf{Y}_{\mathbf{A}}) = \frac{1}{(2\pi)^{D/2} |\mathbf{K}_{\mathbf{A},\mathbf{A}}|^{1/2}} \exp\left(-\frac{1}{2} \text{tr}(\mathbf{K}_{\mathbf{A},\mathbf{A}}^{-1} \mathbf{Y}_{\mathbf{A}} \mathbf{Y}_{\mathbf{A}}^T)\right). \quad (8)$$

To take the inactive points \mathbf{J} into account again, we have to optimize them too, but luckily given the fixed active set, the points are not interdependent anymore, so we can optimize them independently:

$$p(\mathbf{y}_j | \mathbf{x}_j) = \mathcal{N}(\mathbf{y}_j | \mu_j, \sigma_j^2 \mathbf{I}), \quad (9)$$

where the mean μ_j is:

$$\mu_j = \mathbf{Y}^T \mathbf{K}_{\mathbf{A},\mathbf{A}}^{-1} \mathbf{k}_{\mathbf{A},j},$$

Algorithm 1 A practical algorithm for GP-LVMs

Require: A size for the active set, d_A . A number of iterations T .

Initialise \mathbf{X} through PCA.

for T iterations **do**

 Select a new active set \mathbf{A} using the IVM algorithm.

 Optimise Eq. (8) with respect to the parameters of \mathbf{K} (and optionally the latent positions \mathbf{X}_A) using scaled conjugated gradients.

 Select a new active set \mathbf{A} .

for each point j of the points not in the active set, **do**

 Optimise Eq. (9) with respect to \mathbf{x}_j using scaled conjugated gradients.

end for

end for

Content taken from [22].

and the variance is given by:

$$\sigma_j^2 = k(\mathbf{x}_j, \mathbf{x}_j) - \mathbf{k}_{A,j}^T \mathbf{K}_{A,A}^{-1} \mathbf{k}_{A,j},$$

The full algorithm is summarized in Alg. (1) with more details provided by Lawrence in [22].

A GP-LVM is a powerful model, but for the thesis we need a model allowing for working with multiple corresponding observations.

We require a model, which is not only able to learn a low dimensional representation of the higher dimensional observation spaces. It should additionally learn a mapping from one observation space to the corresponding observation space(s) and vice versa, so that when given a new observation we can synthesise the corresponding observation(s).

Ek et al. [11] developed two variants of the GP-LVM, called shared- and subspace GP-LVM. Both work with multiple corresponding observations, where the observation spaces are generated from the same latent variable \mathbf{X} .

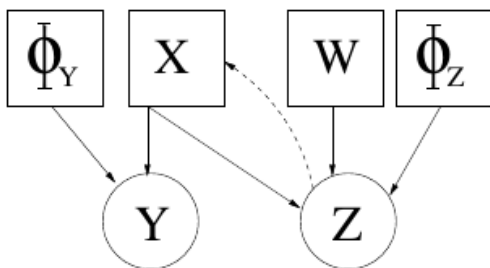


Figure 15: Graphical representation of a shared GP-LVM, representing the back-mapping constraining the latent space by the dashed line. Content taken from [11].⁶

A downside of shared models in general is that they assume that the observations work in spaces with equal dimensions or modes of variability [11]. With the development of shared GP-LVMs (see Fig. 15), Ek relaxed this assumption by introducing a back-constraint to the shared model. This model is applicable in a situation where you are interested in inference in only one direction.

Here we will focus on subspace GP-LVM, in which the assumption is relaxed further. Instead of the model in shared GP-LVMs, which encourages the latent space to fully explain the correlated variance and to handle the non-correlated variance as noise, in subspace GP-LVMs the non-correlated variance is modeled separately by additional latent spaces. So the non-correlated variance is here also explained

⁶ Image taken from: Carl Henrik Ek - Talk on Shared Gaussian Process Latent Variable Models (2007)

by a \mathcal{GP} instead of just by our Gaussian noise model. This model is explained in more detail in the following chapter.

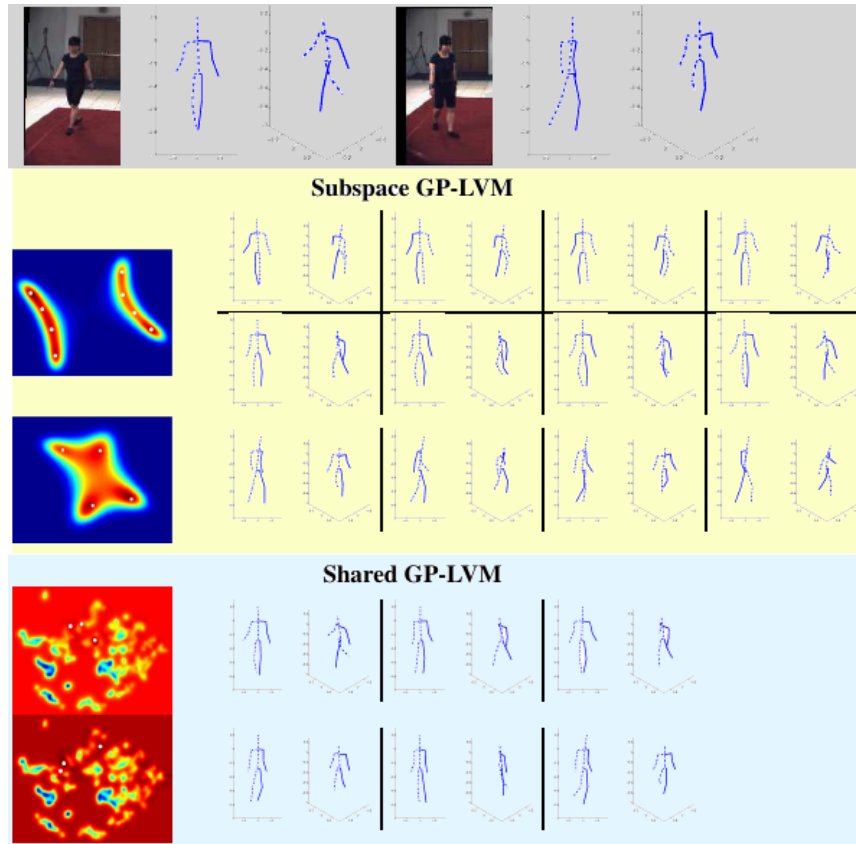


Figure 16: The gray row on the top shows two example images from the training data. The rows of the yellow area show results from inferring the pose using a subspace GP-LVM, whereas the rows of the green area show results when using a shared GP-LVM. In these areas the first column shows the likelihood sampled over the pose specific latent space constrained by the image features, the remaining columns shows the modes associated with the locations of the white dots over the pose specific latent space.

Subspace GP-LVM (yellow): In the 1st row where the results of the reconstruction of the first test image are shown, the position of the leg and the heading angle cannot be determined in a robust way from the image features. This is reflected by two elongated modes over the latent space representing the two possible headings. The poses along each mode represent different leg configurations. The top row of the 1st column shows the poses generated by sampling along the right mode and the bottom row along the left mode. In the 2nd row, where the results of the second test image are shown, the position of the leg and the heading angle is still ambiguous to the feature, however here the ambiguity is between a discrete set of poses indicated by four clear modes in the likelihood over the pose specific latent space.

Shared GP-LVM (green): Here both rows show the results of doing inference using the shared GP-LVM. Even though the most likely modes found are in good correspondence to the ambiguities in the images the latent space is cluttered by local minima, where the optimization can get stuck in. *Content taken from [11].*

3 Methods

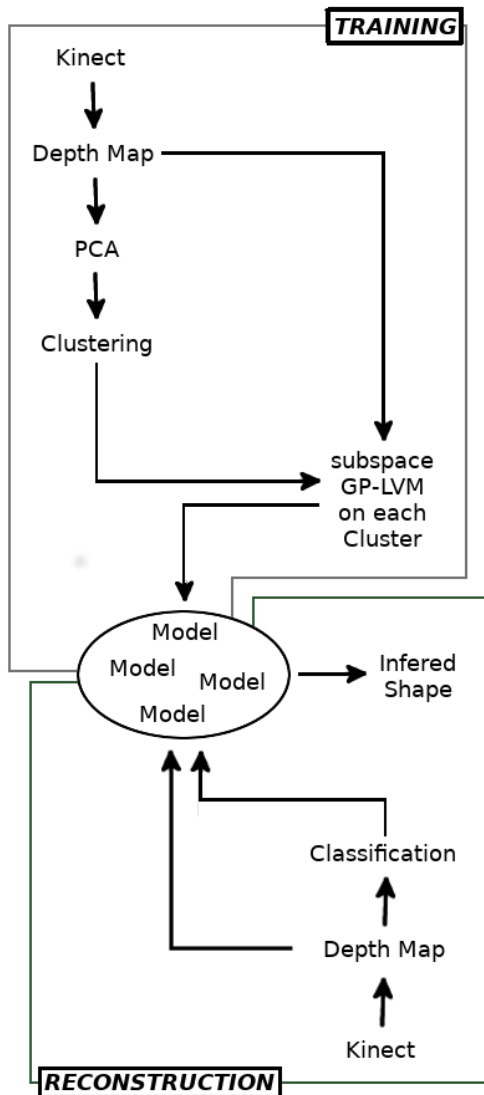


Figure 17: Outline of our algorithm with the training stage viewed in the upper box and the reconstruction stage viewed in the box below. In the training stage the depth map is processed (centered, etc) and projected onto a low dimensional space via PCA. The resulting representation is used for the spectral clustering algorithm. The obtained clusters are then used to train the different models. In the reconstruction stage the depth map is equally processed and then assigned to the group with the most similar objects. The corresponding model is then used to infer the observation of the back view of the object.

Here we will present our algorithm in detail (see Fig. 17). The algorithm consists of a learning- and a reconstruction stage. After giving an outline of the full algorithm we will focus on the two stages and explain each in depth.

For building the dataset for the learning stage where each item consists of two corresponding observations, we take two depth maps from two Microsoft Kinect cameras (see Fig. 3). The Kinects were positioned opposite of each other on a planar surface. Then the object is placed on the planar surface positioned in the middle of the two Kinects. When correctly positioned the depth maps are recorded by the Kinects.

The images for testing are also obtained in that way. Where later one depth map is used as the input image and the corresponding depth map as the ground truth for the reconstruction evaluation.

To train the subspace GP-LVM, the algorithm we use to model the shared low dimensional embedded information of the two observations, we take the average of the two depth maps contained in each image-pair.

For the next step we evaluated several clustering algorithms (K-Means, Hierarchical Clustering, Spectral Clustering) in combination with different data representations (raw data, data projected via PCA, features extracted from data with Kernel Descriptors) to find the combination which works best with our algorithm (evaluations can be found in the next chapter).

The result of the evaluations is that a combination of a spectral clustering algorithm applied to the projected data via PCA works best. So, to obtain a low dimensional representation of the aver-

aged image pairs, they have to be projected onto a lower dimensional space via PCA by using the eigenvectors corresponding to the highest eigenvalues. These representations are then used to cluster the dataset into groups of objects of similar shapes, using a spectral clustering algorithm.

For each of these groups we learn a subspace GP-LVM to extract the unknown low dimensional embedded information of the high dimensional observations of the group. Furthermore, for each group we fit a Gaussian to all the depth maps contained in the group. These Gaussians are later used in the reconstruction stage for the classification.

In the reconstruction stage we have the Gaussians fitted to the groups and the learned models available. We get a depth map of a previously unseen object as input. After the input data point is processed, the probability that the input image belongs to the group is evaluated for each group. We take the model of the group with the highest probability and let the model infer the back view of the object. Examples of the flexibility of this algorithm are presented in the next chapter.

3.1 Learning Stage

Here we will focus on the single parts of this stage. In the Experiments Chapter we give details of our evaluation process to find the best clustering algorithm (K-Means, Hierarchical Clustering, Spectral Clustering) in combination with the image representation (raw data, PCA, Kernel Descriptors).

3.1.1 Depth Map Processing

The raw depth maps $\mathbf{R} \in \mathbb{R}^{480 \times 640}$ taken from the Microsoft Kinects have a resolution of 640×480 , which we subsample to a resolution of 400×300 . Because of the setup, the depth map values we are interested in, the values which contain the information about the object, range from 0.5 to 0.9, so all values above and below are set to 0:

$$\mathbf{R}_{i,j} = \begin{cases} \mathbf{R}_{i,j} & \text{if } 0.5 < \mathbf{R}_{i,j} < 0.9 \\ 0 & \text{otherwise.} \end{cases}$$

We get depth maps, which only contain information about the object. For the learning process, these depth maps are then centered and vectorized, resulting in the vectorized depth maps of the observation space of the front view $\mathbf{m}_i^F \in \mathbb{R}^{120000 \times 1}$ and the vectorized depth maps of the corresponding observation space of the back view $\mathbf{m}_i^B \in \mathbb{R}^{120000 \times 1}$, where $i, j \in [1, N]$.

For centering we search for the positions $\mathbf{c}_b \in (x, y)^T$ of non-zero values (object information) in the depth map \mathbf{R} . With the non-zero value matrix $\mathbf{C} \in [\mathbf{c}_1, \dots, \mathbf{c}_Q]$, where Q equals the number of non-zero values. The positions of the outermost non-zero values nearest to the borders of \mathbf{X} can be found with:

$$\begin{aligned} pos_{min}^X &= \min(\mathbf{C}_{1,:}) \\ pos_{max}^X &= \max(\mathbf{C}_{1,:}) \\ pos_{min}^Y &= \min(\mathbf{C}_{2,:}) \\ pos_{max}^Y &= \max(\mathbf{C}_{2,:}). \end{aligned}$$

The size of the rectangle containing the object is computed by:

$$\begin{aligned} l^X &= pos_{max}^X - pos_{min}^X \\ l^Y &= pos_{max}^Y - pos_{min}^Y. \end{aligned}$$

To center the object, the rectangle containing the object information is copied into the middle of an empty 400×300 matrix:

$$\tilde{\mathbf{R}}((300 - l^X)/2, (400 - l^Y)/2) = \mathbf{R}(pos_{min}^X : pos_{max}^X, pos_{min}^Y : pos_{max}^Y)$$

After the objects in the depth maps are centered we vectorize the depth maps to obtain the dataset of the observation space of the frontal view $\mathbf{M}^F = [\mathbf{m}_1^F, \dots, \mathbf{m}_N^F]^T$ and of the back view $\mathbf{M}^B = [\mathbf{m}_1^B, \dots, \mathbf{m}_N^B]^T$.

For the learning process the average of the two depth maps contained in each image-pair is computed:

$$\widehat{\mathbf{m}}_i = \frac{1}{2}(\mathbf{m}_i^F + \mathbf{m}_i^B).$$

This is done to obtain a single representation of each image-pair for the clustering process. The single representation contains the information of both depth maps and also maintains the position of the information. Before giving the matrix $\widehat{\mathbf{M}}$ into the clustering algorithm we first extract the low dimensional representation of it:

$$\tilde{\mathbf{M}} = \widehat{\mathbf{M}} * \mathbf{W}_{:,1:50},$$

where \mathbf{W} are principal components obtained by applying PCA to the averaged depth maps $\widehat{\mathbf{M}}$.

3.1.2 Clustering Algorithm

We make use of a spectral clustering algorithm according to Ng et al. [29]. The advantage of such a clustering algorithm in comparison to traditional clustering algorithms based on generative models, e.g. K-means, is that it has no strong assumptions on the data, as for instance, that the data points in a cluster have to follow a multivariate Gaussian. Additionally, with some generative clustering methods you can get bad clustering results, because you get stuck in a local optimum when optimizing the log likelihood. In spectral clustering the similarity matrix between the data points is computed and the algorithm works on the eigenvectors of this matrix.

First the similarity graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is created, where $\mathbf{v}_i \in \mathbf{G}$ are the input vectors $\widehat{\mathbf{m}}_i$. To determine the edges \mathbf{E} of the graph \mathbf{G} , we have to define a similarity function $s_{i,j}$ between the vertices \mathbf{v}_i . For the thesis we use the so called ρ -nearest neighbor similarity function, where the goal is to connect each vertex with its ρ nearest neighbors. The edges are not directed and are created between vertices when \mathbf{v}_i is one of the nearest neighbors of \mathbf{v}_j or if \mathbf{v}_j is one of the nearest neighbors of \mathbf{v}_i . $\mathbf{N}_\rho(i)$ defines the ρ nearest neighbors of \mathbf{v}_i and the adjacency matrix \mathbf{A} of \mathbf{G} is:

$$a_{ij} = \begin{cases} s_{ij} + \alpha & \text{if } \mathbf{v}_i \in \mathbf{N}_\rho(j) \vee \mathbf{v}_j \in \mathbf{N}_\rho(i) \\ 0 & \text{otherwise,} \end{cases}$$

where α is a small constant that helps avoiding degenerated instances with no connections at all.

Next the Laplacian of the graph is computed as:

$$\mathbf{L} = \mathbf{D} - \mathbf{A},$$

where \mathbf{D} is a diagonal matrix with:

$$d_{ii} = \sum_{j=1}^N \mathbf{A}_{ij}.$$

The normalized Laplacian of the graph is given by:

$$\hat{\mathbf{L}} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2}.$$

The number of clusters is automatically defined with the help of the ascending sorted eigenvalues λ_i of the normalized Laplacian, where we determine the number of clusters k with the eigengap heuristic:

$$k = \operatorname{argmax}_i (\lambda_{i+1} - \lambda_i).$$

Next up we select the k eigenvectors of the normalized Laplacian corresponding to the k largest eigenvalues and put them into the matrix $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_k] \in \mathbb{R}^{N \times k}$. With \mathbf{U} defined we build the matrix $\mathbf{T} \in \mathbb{R}^{N \times k}$ by normalizing the rows of \mathbf{U} :

$$t_{ij} = u_{ij} / \left(\sum_{g=1}^k u_{ig}^2 \right)^{1/2}.$$

The input vectors $\tilde{\mathbf{m}}_i$ correspond to the i -th row of \mathbf{T} . To get the k final clusters $\mathbf{G}_1, \dots, \mathbf{G}_k$, we apply K-means to the matrix \mathbf{T} . The indices of the groups are contained in the clusters $\mathbf{A}_1, \dots, \mathbf{A}_k$, with $\mathbf{A}_i = \{j | \mathbf{y}_j \in \mathbf{G}_i\}$.

3.1.3 Gaussian Fitting

For each group we now fit a Gaussian to the data points which is later used for classification in the reconstruction stage. We first reshape the processed vectorized depth maps \mathbf{m}_i^F and \mathbf{m}_i^B resulting in the matrices \mathbf{M}_i^F and \mathbf{M}_i^B to then subsample them by a factor of 0.09 and vectorize the resulting matrices again. For fitting the Gaussians we first compute the centers of each group with:

$$\mu_i = \frac{1}{H_i} \sum_{j=1}^{H_i} \mathbf{m}_j^F + \mathbf{m}_j^B,$$

where H_i is the number of data points in group \mathbf{A}_i . The covariance matrix of the Gaussian we get with:

$$\sigma_i^2 = \left(\frac{1}{H_i} \sum_{j=1}^{H_i} (\mathbf{m}_j - \mu_i)(\mathbf{m}_j - \mu_i)^T \right) + \alpha \mathbf{I},$$

where α is a small constant added to the diagonal of the covariance matrix to ensure that the covariance matrix is invertible. For each cluster \mathbf{A}_i we save the mean μ_i , the covariance matrix σ_i^2 , the determinante of the covariance matrix and its inverse together with the number of files contained in the cluster H_i .

3.1.4 Model Learning

To train the subspace GP-LVM on each cluster, we make use of the processed depth maps \mathbf{M}^F and \mathbf{M}^B again. This time we subsample them by a factor of 0.25 and vectorize the resulting matrices again.

For the subspace GP-LVM we make use of the scaled conjugated gradient method [28] for optimization. How to define the size of the dimensions for the latent subspaces is explained in the Experiments Chapter, it depends on the input data and how flexible the subspace GP-LVM has to be.

We put the submatrices $\mathbf{M}^F_{\{A_i\},:}$ and $\mathbf{M}^B_{\{A_i\},:}$ of the two matrices \mathbf{M}^F and \mathbf{M}^B containing the subsampled vectorized depth maps of both observation spaces into the subspace GP-LVM. After the model is learned, we save it for later use.

To explain how the subspace GP-LVM works (see Fig. 18) and is optimized, we will, for better readability, substitute our two datasets of the two observation spaces, the dataset of the front view \mathbf{M}^F and the one of the back view \mathbf{M}^B with $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]^T$ and $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_N]^T$, where $\mathbf{y}_i \in \mathbb{R}^{D_{high}^Y}$ and $\mathbf{z}_i \in \mathbb{R}^{D_{high}^Z}$. We assume that these observations had been generated from low dimensional spaces by:

$$\mathbf{y}_i = f^Y(\mathbf{u}_i^Y) + \mathbf{e}_Y \quad \mathbf{z}_i = f^Z(\mathbf{u}_i^Z) + \mathbf{e}_Z,$$

where $\mathbf{e}_Y \sim \mathcal{N}(0, \sigma_Y^2 \mathbf{I})$ and $\mathbf{e}_Z \sim \mathcal{N}(0, \sigma_Z^2 \mathbf{I})$ are the additive Gaussian noises and $\mathbf{u}_i^Y \in \mathbb{R}^{D_{low}^Y}$ and $\mathbf{u}_i^Z \in \mathbb{R}^{D_{low}^Z}$ are the low dimensional representations of the data points in the latent space with $D_{low}^Y < D_{high}^Y$ and $D_{low}^Z < D_{high}^Z$.

The two latent sub-spaces are defined as:

$$\mathbf{U}^Y = [\mathbf{X}^S; \mathbf{X}^Y] \quad \text{and} \quad \mathbf{U}^Z = [\mathbf{X}^S; \mathbf{X}^Z],$$

and we assume that $\mathbf{X}^S \subset \mathbf{U}^Y$, $\mathbf{X}^S \subset \mathbf{U}^Z$ and $\mathbf{X}^S \neq 0$. Therefore, the two latent spaces \mathbf{U}^Y and \mathbf{U}^Z can be parametrized in such a way that they have a common non-empty shared subspace \mathbf{X}^S , with:

$$\mathbf{x}_i^S = g^Y(\mathbf{y}_i) = g^Z(\mathbf{z}_i).$$

Hence we call \mathbf{X}^S shared subspace, \mathbf{X}^Y and \mathbf{X}^Z private subspaces, where we further assume that the relationship between the observed data and the corresponding private space representation is a smooth mapping, with:

$$\begin{aligned} \mathbf{x}_i^Y &= h^Y(\mathbf{y}_i) \\ \mathbf{x}_i^Z &= h^Z(\mathbf{z}_i). \end{aligned}$$

$\mathbf{X} = [\mathbf{X}^S; \mathbf{X}^Y; \mathbf{X}^Z]$ then is the combined latent representation obtained by concatenating the shared subspace with the private subspaces, representing the shared- and non-shared variance in a factorized form. Given this latent structure the shared- and the non-shared variance can be modeled separately.

To construct a subspace GP-LVM and respecting the factorized latent structure, two separate Gaussian processes (\mathcal{GPs}) f^Y and f^Z are learned. The input space of f^Y is \mathbf{U}^Y , learning a mapping to generate the observations \mathbf{Y} . The input space of f^Z is \mathbf{U}^Z and generating \mathbf{Z} . Because

the shared- and non-shared variance are modeled separately, the mappings f^Y and f^Z can only be used on the subspaces associated with the corresponding observations.

The model is optimized by applying a gradient based method (in our case SCG) to the objective function:

$$\begin{aligned} \{\mathbf{X}, \Phi_Y, \Phi_Z\} &= \operatorname{argmax}_{\mathbf{X}, \Phi_Y, \Phi_Z} p(\mathbf{Y}, \mathbf{Z} | \mathbf{X}, \Phi_Y, \Phi_Z) \\ &= \operatorname{argmax}_{\mathbf{X}^S, \mathbf{X}^Y, \mathbf{X}^Z, \Phi_Y, \Phi_Z} p(\mathbf{Y} | \mathbf{X}^S, \mathbf{X}^Y, \Phi_Y) p(\mathbf{Z} | \mathbf{X}^S, \mathbf{X}^Z, \Phi_Z), \end{aligned}$$

As shown in [11], because of the use of a gradient based method, for a good solution the initialization of the model is crucial. In general a GP-LVM should only be applied when there exists an analogous spectral algorithm for the initialization of the latent space. A GP-LVM can be seen as a generative algorithm improving the solution given by a spectral algorithm.

In our case we need a spectral dimensionality reduction algorithm that finds the factorized latent spaces, which separately represent the shared- and private information of the corresponding observation spaces.

In contrast to shared GP-LVMs where the model can be initialized through PCA, with subspace GP-LVMs the initialization with PCA is not sufficient. Therefore, Ek developed [11] an extension to the feature selection algorithm Canonical Correlation Analysis [1] (CCA) called Non Consolidating Component Analysis (NCCA), a spectral dimensionality reduction algorithm for finding factorized latent spaces, where shared variance and private variances are represented separately.

Ek proposes that the shared latent subspace \mathbf{X}^S is initialised using CCA for finding directions in the two observation spaces, which are maximally correlated. Before CCA is used, PCA is applied to each dataset separately to avoid low variance solutions [21]. For each observation space, CCA finds a set of basis vectors, in our case \mathbf{B}_Y and \mathbf{B}_Z , so that the correlation ρ between the projections is maximized:

$$\rho = \frac{\operatorname{tr}(\mathbf{B}_Y^T \mathbf{Y}^T \mathbf{Z} \mathbf{B}_Z)}{(\operatorname{tr}(\mathbf{B}_Z^T \mathbf{Z}^T \mathbf{Z} \mathbf{B}_Z) \operatorname{tr}(\mathbf{B}_Y^T \mathbf{Y}^T \mathbf{Y} \mathbf{B}_Y))^{1/2}},$$

subject to unit variance along each direction, $\mathbf{B}_Y^T \mathbf{Y}^T \mathbf{Y} \mathbf{B}_Y = \mathbf{I}$ and $\mathbf{B}_Z^T \mathbf{Z}^T \mathbf{Z} \mathbf{B}_Z = \mathbf{I}$.

With the basis vectors \mathbf{B}_Y and \mathbf{B}_Z explaining the shared variance, Ek makes use of the NCCA algorithm to acquire the basis vectors explaining the private non-shared variance \mathbf{P}^Y and \mathbf{P}^Z of both observation spaces \mathbf{Y} and \mathbf{Z} . The basis Ek is looking for should be directions of maximum variance and orthogonal to the shared bases. Each observation space is processed separately searching for the first direction with:

$$\hat{\mathbf{p}} = \operatorname{argmax}_{\mathbf{p}} \mathbf{p}^T \mathbf{C} \mathbf{p},$$

constrained by:

$$\begin{aligned} \mathbf{p}^T \mathbf{p} &= 1 \\ \mathbf{p}_i^T \mathbf{B} &= 0, \end{aligned} \tag{10}$$

where \mathbf{B} are the bases vectors of the shared variance and \mathbf{C} the covariance matrix of the feature space.

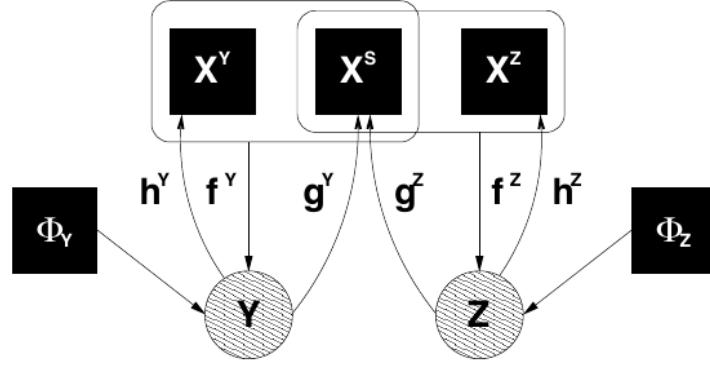


Figure 18: Graphical representation of a subspace GP-LVM, where the two observation spaces Y and Z are generated from the latent variable X factorized into subspaces X^Y , X^Z representing the private information and X^S modeling the shared information. Φ_Y and Φ_Z are the hyper-parameters of the Gaussian processes modeling the generative mappings f_Y and f_Z . Content taken from [11]⁷

By formulating the Lagrangian of the problem the first orthogonal direction can be found. To find further consecutive directions the found directions have to be appended to the orthogonality constraint in Eq. (10). To find the m -th direction, the orthogonality constraint looks like:

$$\mathbf{p}_m^T[\mathbf{B}; \mathbf{p}_1, \dots, \mathbf{p}_{m-1}] = 0.$$

With the bases \mathbf{B} , \mathbf{P}^Y and \mathbf{P}^Z we can initialise the subspace GP-LVM and start improving the latent variables.

For the thesis we need a bidirectional mapping, but work with observation spaces of same dimensionality and same modes of variability. Therefore, the subspace GP-LVM is more flexible than we need it, e.g. it allows for observation spaces with different dimensions.

Still, we decided to make use of subspace GP-LVMs, because it not only gives us what we need but also allows for further development.

3.2 Reconstruction Stage

In the reconstruction stage we are given the parameters of the Gaussians fitted to the clusters and the learned models. When receiving a previously unseen depth map $\mathbf{R}_* \in \mathbb{R}^{480 \times 640}$ as input, we first process \mathbf{R}_* according to the procedure explained in 3.1.1 and so obtain the processed vectorized input data point $\mathbf{m}_* \in \mathbb{R}^{10800 \times 1}$.

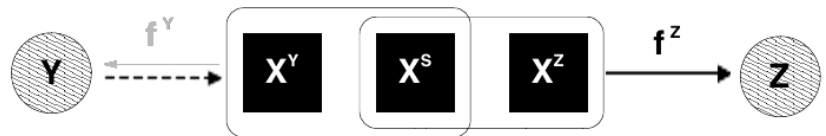


Figure 19: Inferring the shape of the back view of the object: 1) find the MAP solution for the latent representation of the data point in Y (dotted line) 2) synthesize the corresponding position of the point in Z ($\mathcal{G}^P f^Z$).

⁷ image taken from: Ek, Torr and Lawrence - Talk on Shared Gaussian Latent Variable Models (2007)

After that, we compute the log-posterior of \mathbf{m}_* with regard to each Gaussian, where the prior is given by the percentage of the data contained in the corresponding cluster.

In that way the posterior probability of the input image belonging to a cluster is computed for each cluster. We take the model corresponding to the group with the highest probability for the reconstruction process. In the reconstruction process we first have to map the input data point \mathbf{m}_* into the latent space and then synthesize the data point in the corresponding observation space. Because there exists no direct mapping from the observation space to the latent space, to map the data point \mathbf{m}_* of the observation space Y into the latent space we have to find the MAP solution of \mathbf{m}_* in the latent space \mathbf{u}_*^Y with regard to the likelihood, given by:

$$p(\mathbf{y}_* | \mathbf{X}^Y, \mathbf{X}^S, \Phi_Y, \mathbf{u}_*^Y) = \mathcal{N}(\mathbf{y}_* | \mu, \sigma^2),$$

where

$$\mu = \mathbf{Y}^T \mathbf{K}_{\mathbf{I},\mathbf{I}}^{-1} \mathbf{k}_{\mathbf{I},*}.$$

The variance is given by:

$$\sigma^2 = k(\mathbf{u}_*^Y, \mathbf{u}_*^Y) - \mathbf{k}_{\mathbf{I},*}^T \mathbf{K}_{\mathbf{I},\mathbf{I}}^{-1} \mathbf{k}_{\mathbf{I},*}.$$

We use the MAP solution of the point in the latent space \mathbf{u}_*^Y in combination with the $\mathcal{GP} f^Z$ to project \mathbf{u}_*^Y into the data space Z and compute the probability of \mathbf{z}_* under the resulting distribution. The inferred shape of the object's back view is given by the optimized \mathbf{z}_* .

Additionally, we first included a mixture-of-experts approach. The purpose of this approach was to increase the quality of reconstructions of unseen objects of categories not learned in the training stage.

The idea was that, if an input object's highest posterior probability for belonging to a group is under a threshold of 50%, the algorithm selects the models with the highest probabilities for the reconstruction. In this case the models with the highest probabilities are defined as the model with the highest probability plus all models not lower than 5% of this model's posterior probability. For each of the selected models, the input object is reconstructed separately. In the end the reconstructions are combined. This combining was the reason we rejected to make use of this approach for our algorithm. In our approach we used a primitive combination method by taking the weighted average of each pixel, where we used the posterior probabilities as the weights. But as we are working on depth maps where the values in each point stand for the distance of the pixel, this combination method resulted in poor reconstructions.

We think, putting further work into developing a more suitable combination method could boost the quality of reconstructions on objects of untrained categories. In the last chapter we discuss this further.

4 Experiments

In this chapter we will present empirical evaluations on the clustering algorithms and our reconstruction algorithm. For the clustering algorithm we compare results of K-means, hierarchical clustering and spectral clustering evaluated in three different setups. The reconstruction algorithm is first evaluated on a single category to survey the influence of the dimensionality of the latent space and then evaluated on the complete dataset.

The dataset used for the evaluations consists of $D = 259$ image pairs of objects like cups, bowls, teapots, boxes and cans (see Fig. 20).

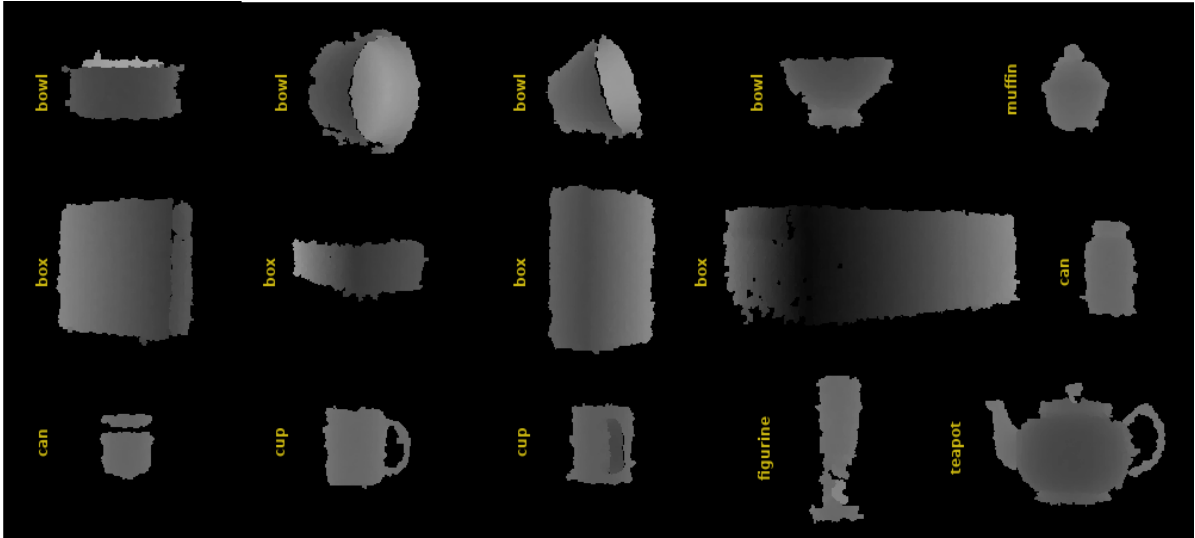


Figure 20: Sample objects of the dataset. Black values represent depth map values of $\leq 0.6m$ whereas white values represent distances of $\geq 0.9m$.

4.1 Clustering

Here we present the results of the different clustering algorithms evaluated on each of the different input representations. Clustering is a crucial part for the reconstruction process, because based on its results the particular objects for each model are determined.



Figure 21: An example of a cluster we rate as 'good', with objects of similar shape but still not overfitted.

Our goal in clustering is not to put only all the objects of one category in a group, but to group objects with similar shapes independent of the category. In this thesis we evaluate the clustering results empirically with the focus on the performance for a general use. The criteria for a candidate for our algorithm are, that the algorithm should cluster objects of similar shape

together. When such clusters exist and no objects of the same shape are missing we grade this a ‘good’ cluster. When it groups only a small fraction, under 10%, of the objects of similar shape into one cluster we grade it a ‘poor’ cluster and if it groups clusters of too distinct shape together this is also a ‘poor’ cluster. We will evaluate the algorithms regarding the ratio of ‘good’ and ‘poor’ clusters, the percentage of data points contained in the ‘good’ clusters, the minimal-, maximal- and average size of the ‘good’ clusters, as well as the average size of the ‘poor’ clusters. The resulting ‘good’ clusters should contain most of the data. Due to the properties of our dataset, they should also come up with clusters of roughly the same size. Results with the average size of ‘poor’ clusters considerably larger than the average size of ‘good’ clusters hints that it grouped distinct shapes into the same cluster, because it could not differentiate between them. With the combination of all these observations we are able to compare the algorithms and find the best clustering algorithm for our work.

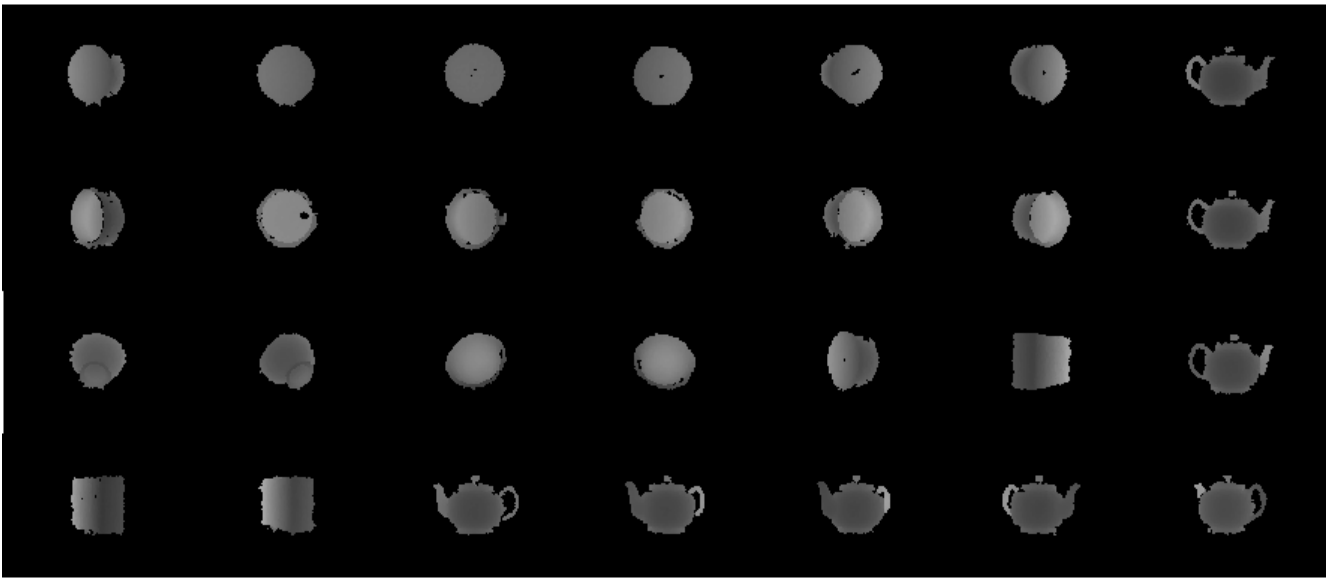


Figure 22: An example of a as ‘poor’ rated cluster, with objects of too distinct shapes (e.g. box and teapot).

We also provide a score with which we can compare the performances of each result, given by the score-function:

$$\tau = 0.3 + 0.35p_G + 0.35p_d - 0.15 \frac{\max(G_{max} - 2G_{min}, 0)}{G_{max}} - 0.15 \frac{\max(|G_{avg} - P_{avg}| - 0.2G_{all}, 0)}{\max(G_{avg}, P_{avg})} - \delta(\max(|G_{avg} - \lceil \sqrt{D} \rceil - 0.15G_{all}, 0)),$$

with the penalty function δ for the ‘good’ clusters, which punishes too small and too large average clusters, as:

$$\delta(x) = \begin{cases} 0.05 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases},$$

where p_G is the percentage of ‘good’ clusters proportional to all clusters and p_d is the percentage of all data points contained in the ‘good’ clusters. G_{min} and G_{max} are the minimal- and maximal size of ‘good’ clusters, whereas G_{avg} and P_{avg} are the average sizes of the ‘good’- and the ‘poor’ clusters and G_{all} is the size of the complete dataset.

The score function was developed, because only punishing mixed clusters, where objects of different clusters are grouped, is too simple a view. In some cases, where the objects have similar shapes, these clusters are desired. The score function gives credit to the percentage of ‘good’ clusters as well as to the percentage of objects contained in these clusters. The score is penalized, if the difference between the size of the ‘good’ group with the lowest number of members and the size of the ‘good’ group with the largest number of members exceeds 50% of the size of the largest ‘good’ group. If the average group size of the ‘good’ and the ‘bad’ clusters differ with more than 2% of the size of the complete dataset, the score function is penalized further. The function also punishes, if the average size of the ‘good’ clusters differs more than 1.5% from the square root of the size of the complete dataset.

This function is designed and tuned for our dataset only and has to be adjusted for other datasets.

For each of the different clustering algorithms, we will evaluate the results on a set of different input representations. The representations are the raw averaged image pairs themselves (Setup I), the projected average image pairs after we apply PCA on them (Setup II) and kernel descriptor features [5] extracted from the averaged image pairs (Setup III).

Kernel Descriptors [4, 5]:

For computer vision algorithms making use of image features, the design of these low-level image features is crucial. For recognition tasks for example, orientation histograms like SIFT [25] and HOG [10], or in the 3D-space Spin images [18] are popular choices.

In [4] Bo et al. developed a general framework to obtain low-level features from image patches. The work is based on the insight that the inner product of orientation histograms is a specific match kernel over image patches. Bo et al. provided a unified framework to turn any type of pixel attribute, e.g. gradient and color, into patch-level features. In [5] they further developed a set of five depth kernel descriptors for working with depth images. These kernel features capture image cues like size, 3D shape and depth edges of the depth images.

Kernel descriptors provided by Bo et al. outperform popular image features including e.g. SIFT or, in the variant for the depth images, Spin images.

4.1.1 K-Means

K-Means [27]:

K-Means is a clustering algorithm, using an iterative refinement technique to cluster n observations $\{x_1, \dots, x_n\}$ into k clusters, where each observation belongs to the cluster with the nearest mean.

The algorithm is given the number of means k and an initial set of them, $\{m_1^{(1)}, \dots, m_k^{(1)}\}$. The clustering process consists of two steps:

Assignment step: Assign each of the n observations x_j , with $j = 1, \dots, n$, to the cluster C_i , where $i = 1, \dots, k$, with the nearest mean:

$$C_i^{(t)} = \{x_j : \|x_j - m_i^{(t)}\| \leq \|x_j - m_{i^*}^{(t)}\|, \forall i^* = 1, \dots, k\}.$$

Update step: Update the position of the mean of each cluster to be the centroid of the observations in that cluster:

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j.$$

The algorithm alternates between these two steps until the assignments no longer change.

K-Means [27] has the disadvantage that we have to define the number of clusters k beforehand. For that reason it is no candidate for our algorithm. We still evaluate its performance with the data to get a wider comparison. We will determine the number of clusters depending on the eigengap heuristic in the spectral clustering algorithm.







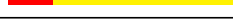
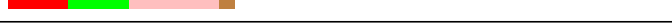





Clusters	objects (itemsize ■)
* 1	
* 2	
3	
* 4	
* 5	
6	
* 7	
8	
* 9	
* 10	
11	
12	
13	

Table 2: K-Means on the raw depth maps:

box ■ / figurine ■ / packet ■ / shampoo-bottle ■ / bottle ■ / bowl ■
 can ■ / cup ■ / tube ■ / teapot ■

Tab. 2 shows the resulting clusters when we apply K-means to the dataset of the averaged image pairs $\widehat{\mathbf{M}}$. A ‘*’ alongside the cluster number indicates ‘good’ clusters, containing objects of similar shapes. Cluster numbers without a ‘*’ indicate clusters where too distinct objects are grouped together (see Fig. 22). We will give examples for clusters without a ‘*’, where we will present the two most distinct objects in these clusters. So, the reader can relate to our decisions.

Seven clusters ($\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_4, \mathbf{A}_5, \mathbf{A}_7, \mathbf{A}_9, \mathbf{A}_{10}$) resulting of K-means on the raw data, contain objects of similar shape as was the goal. The clusters $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_5, \mathbf{A}_{10}$ contain rectangle shaped objects in different forms and orientation. They are overfitted and merging all of these clusters would be preferable. The subspace GP-LVM is flexible enough to model these differences.

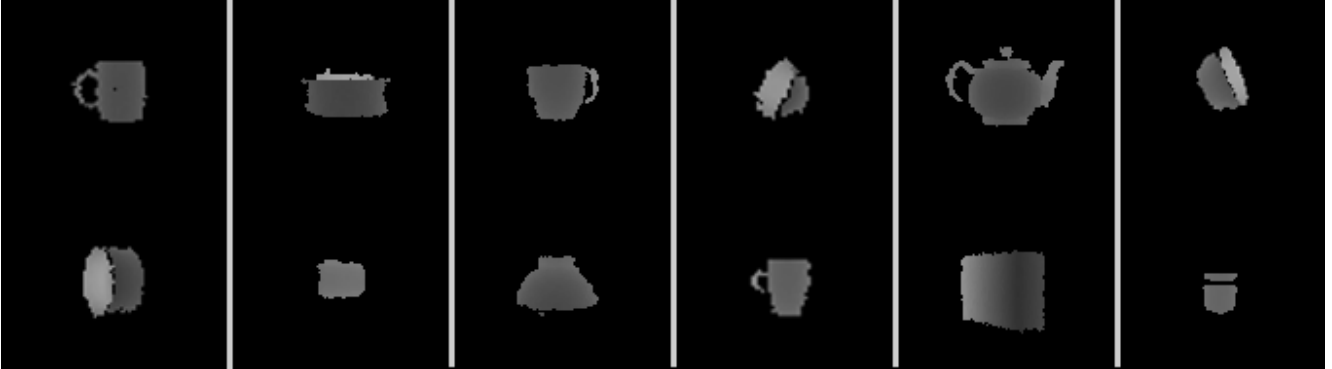


Figure 23: Each column stands for one ‘poor’ cluster and shows the most distinct objects. The clusters were obtained with K-means in Setup I.

We also get six clusters ($\mathbf{A}_3, \mathbf{A}_6, \mathbf{A}_8, \mathbf{A}_{11}, \mathbf{A}_{12}, \mathbf{A}_{13}$) where we have objects not similar but grouped nonetheless (see Fig. 23).

In the second setup the averaged image pairs are projected to a lower dimensional space by using the first 50 principal components of PCA applied on the data. Each data point is then a 50 dimensional vector. With this setup (see Tab. 3) we only have 6 ‘good’ clusters ($\mathbf{A}_1, \mathbf{A}_4, \mathbf{A}_6, \mathbf{A}_7, \mathbf{A}_9, \mathbf{A}_{12}$). Again, we also have four clusters ($\mathbf{A}_1, \mathbf{A}_4, \mathbf{A}_9, \mathbf{A}_{12}$) which would best be grouped into a single one. The rest of them are compositions, which contain objects of too distinct shapes.

Compared to the clusters achieved with K-means on the raw averaged image pairs, 5 of them are the same and the rest of them are also comparable. The clusters obtained with K-means on the raw data yield a slightly better result, with 36.68% of all data points contained in reasonable clusters, whereas in K-means on projected data only 27.03% of all data points ended up in reasonable ones. The average size of ‘good’ clusters in the first K-means setup is 13.57 objects and 27.33 objects as average for ‘poor’ clusters. In the second setup in average 11.67 objects are contained in ‘good’ clusters, whereas 27 objects are contained in average in the ‘poor’ clusters.

When K-means is applied to features extracted with the kernel descriptors algorithm from the data, we again split the data into 13 clusters (see Tab. 3). The setup clusters 37.45% of the data points into ‘good’ ones, with an average size of 13.86 objects. The ‘poor’ clusters have 27 objects per class in average. We again have 7 ‘good’ and 6 ‘poor’ clusters and comparable compositions regarding both other setups.

Overall the clustering results of all setups using K-means yield poor performance regarding it with respect to shape for the general use.















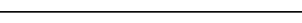











Clusters	K-means in Setup II	K-means in Setup III
(★ / ★) 1		
(- / -) 2		
(- / ★) 3		
(★ / ★) 4		
(- / -) 5		
(★ / ★) 6		
(★ / -) 7		
(- / -) 8		
(★ / -) 9		
(- / ★) 10		
(- / ★) 11		
(★ / ★) 12		
(- / -) 13		

Table 3: K-Means in Setup II and II with itemsize ■:

box ■ / figurine ■ / packet ■ / shampoo-bottle ■ / bottle ■ / bowl ■
 can ■ / cup ■ / tube ■ / teapot ■

4.1.2 Hierarchical Clustering

Hierarchical Clustering [31]:

Instead of K-Means, which result in a flat clustering, Hierarchical Clustering outputs a structured hierarchy. The benefit of such a hierarchy is, that it holds more information than flat results and further, the number of clusters is not required to be prespecified [31]. On the other hand, a drawback of hierarchical clustering algorithms is, that they are, compared to flat ones, less efficient. Where e.g. K-means has a linear complexity, most common hierarchical clustering algorithms have at least a quadratic complexity in the number of objects.

There exist two types of hierarchical clustering algorithms:

Agglomerative: This is a ‘bottom up’ algorithm, where at the start each object is treated as a singleton cluster. These are then successively merged until the algorithm results in a single one containing all objects. Some sort of distance function is needed to find the best matches in each step. With this approach clusters generated in an early stage of the process are nested in clusters generated in later stages.

Divisive: A ‘top down’ approach, which works in the opposite direction of an agglomerative approach. At the start, there is one cluster containing all objects. Step-by-step it is split up into smaller ones, until singleton clusters for all objects exist. This approach needs a method for splitting a cluster.

In hierarchical clustering [31] the algorithm seeks to build a hierarchy of clusters. We use an agglomerative approach, where each data point starts in its own cluster and is merged with another when moving up the hierarchy. The algorithm which is used to compute the distance

Clusters	Setup I	Setup II
(*/*) 1		
(-/*) 2		
(*/*) 3		
(*/*) 4		
(-/*) 5		
(*/-) 6		
(*/*) 7		
(*/*) 8		
(*/-) 9		
(*/*) 10		
(-/*) 11		
(*/*) 12		
(*/*) 13		

Table 5: Spectral Clustering on the raw- and projected averaged image pairs:
 box / figurine / packet / shampoo-bottle / bottle / bowl
 can / cup / tube / teapot

4.1.3 Spectral Clustering

We use the spectral clustering algorithm explained in the previous chapter. We still have to choose the parameter for the similarity function, which determines the number of relevant nearest neighbors for the clustering process. This has to be chosen depending on the dataset. In our case we determined the parameter empirically by evaluating the score function of the spectral clustering results with differing parameter values. The results can be seen in Fig. 24. We set the parameter to 12.

The benefits of spectral clustering compared to K-means, mentioned in the previous chapter, is confirmed by the results in our experiments. Although spectral clustering does not work that well combined with the features of the kernel descriptor in Setup III, achieving the poorest results compared to the other spectral clustering setups, it still achieves better results than the other algorithms. It results in 6 ‘good’ and 5 ‘poor’ clusters where the ‘good’ ones contain 45.27% of the data without being overfitted and are more balanced than with the other approaches.

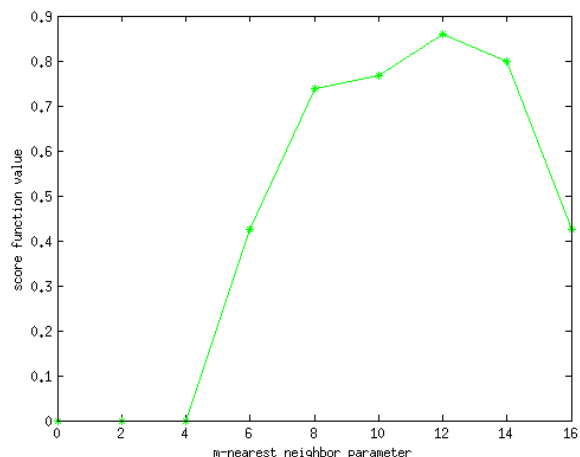


Figure 24: The plot shows the score function value depending on the parameter for the m-nearest neighbor similarity function.

Spectral clustering running with either Setup I or -II (for both see Tab. 5) achieves the best performances. With 11 ‘good’ clusters out of 13 with Setup II, the algorithm gives the best results in our experiments.

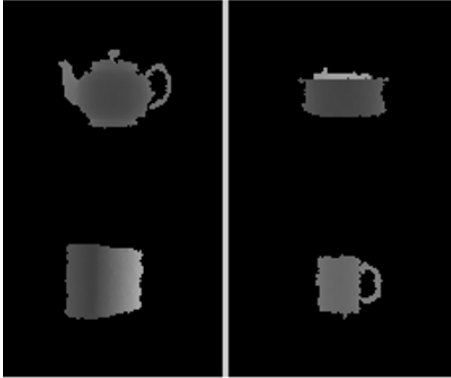


Figure 25: Example objects of the two ‘poor’ clusters of spectral clustering with Setup II, where each column shows the most distinct object shapes in both clusters.

Although, still two clusters give ‘poor’ compositions (see Fig. 25), spectral clustering with Setup II can be used for our algorithm. The results with Setup I yield only slightly worse performances, with 10 ‘good’ clusters out of 13 and covering 70.27% of the data.

Tab. 6 gives an overview of all performances of the clustering algorithms. With K-means and hierarchical clustering too many data points are clustered into groups with too distinct shapes or into clusters which are overfitted regarding the shapes. When running subspace GP-LVM on these clusters, either the resulting model would be unable to extract the commonly shared shape characteristic of the group (when object shapes are too distinct) or the flexibility of the model would be wasted (when the composition with respect to

shape in the clusters is overfitted). Still the composition of the ‘good’ groups in hierarchical clustering showed some potential for finding similar, but not overfitted clusters. It should be re-evaluated when working with very large datasets.

With spectral clustering, especially with setup II, we obtain well composed groups which are not overfitted. The clusters are well balanced regarding the size and the model only has to handle 2 clusters with poor group compositions. Based on the results of the evaluation we decided to make use of the spectral clustering in setup II for our algorithm.

Clustering Algorithm	★ % clusters	★ % of data	size				τ SCORE
			★ min	★ max	★ avg	avg. ‘bad’	
KMR	53.85%	36.68%	7	44	13.57	27.33	0.4665
KMPCA	46.15%	27.03%	7	44	11.67	27	0.3465
KMKD	53.85%	37.45%	7	52	13.86	27	0.4647
HCKD	30.77%	31.27%	1	107	20.25	56	0.2876
SCR	76.92%	70.27%	12	24	18.20	25.67	0.8007
SCPCA	84.62%	82.24%	12	29	19.36	23	0.8581
SCKD	54.55%	45.17%	12	25	19.50	28.40	0.6224

Table 6: A comparison of the different clustering algorithms with: ‘KM’ stands for K-means, ‘HC’ for hierarchical clustering and ‘SC’ for spectral clustering. The additional letters stand for the feature set used, where ‘R’ stands for raw data, ‘PCA’ for the projected data via PCA and ‘KD’ stands for the features extracted with the kernel descriptors algorithm.

4.2 Reconstruction

Here we present the results of our reconstruction algorithm. We first evaluate the influence and properties of the size of the shared latent space regarding the reconstruction results. Therefore, GP-LVMs are learned on a single category (boxes and cups separately) with differing latent space dimensions, ranging from 2D in double steps to 16D. The box dataset consists of 28 objects (see Fig 27), whereas the cups dataset consists of 68 objects. When setting the latent space to 2 dimensions the resulting depth maps of reconstructed cups yield better results than the results for the boxes. This seems to depend on the variations of shapes in each dataset. The box dataset has more distinct object shapes than the cup dataset with an average error-rate (normalized squared difference) of 0.1679 for the the cups and 0.2659 for the boxes.

In this setup the algorithm is underfitted (see Fig. 31), but already yields reasonable results for the cups. When increasing the latent space to 4 dimensions the box reconstruction results yield a slightly better performance with 0.2592 than with only 2 dimensions. Step by step when

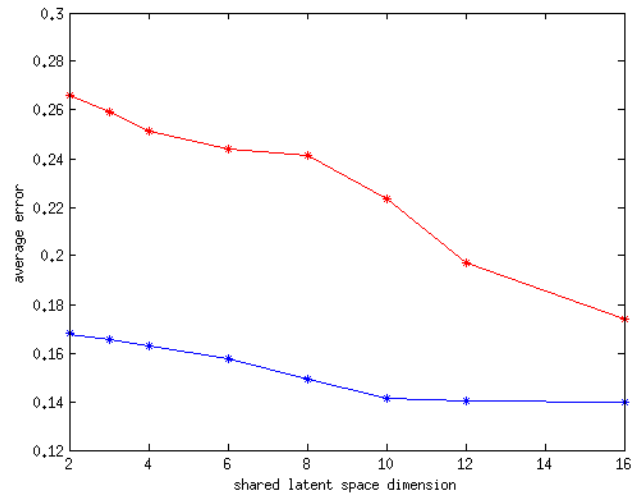


Figure 26: The plot shows the progression of the average error-rate (normalized squared difference) by increasing latent space size. The red line is for the error of the model trained on the boxes dataset, the blue for the error of the model trained on the cups dataset.

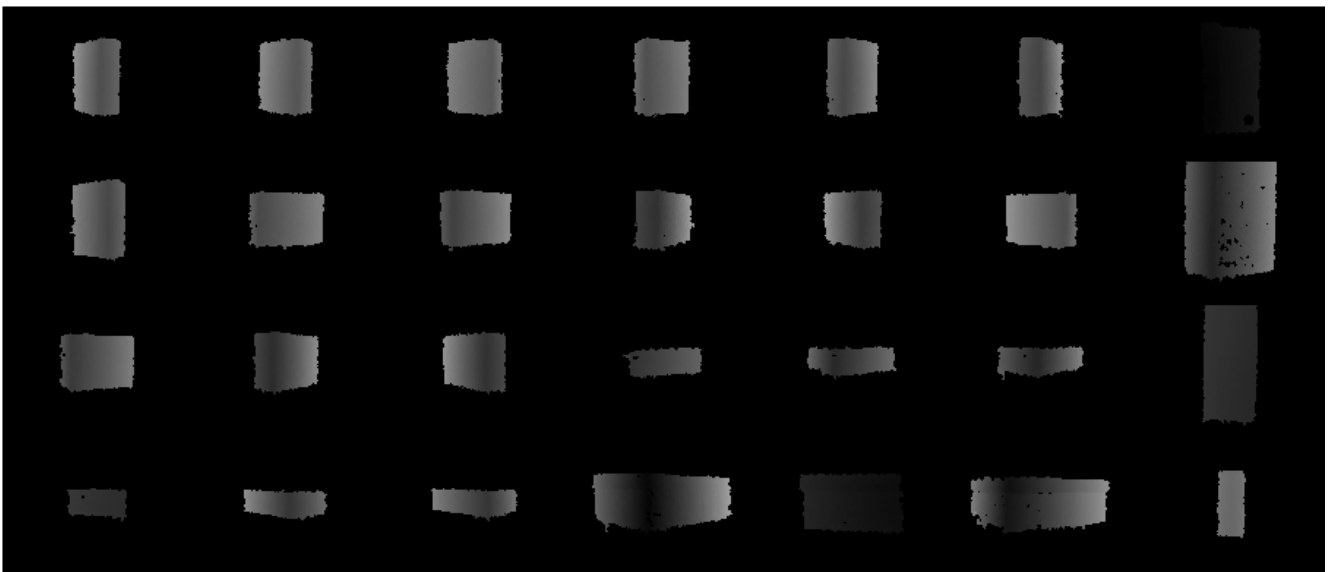


Figure 27: The complete box dataset.

increasing the shared latent space the results of the inferred reconstructions get better (see Fig. 26). With a two dimensional latent space the model for the cups is significantly better than the

result of the reconstruction of the box. When increasing the latent space dimensionality, the quality of the results approach each other.

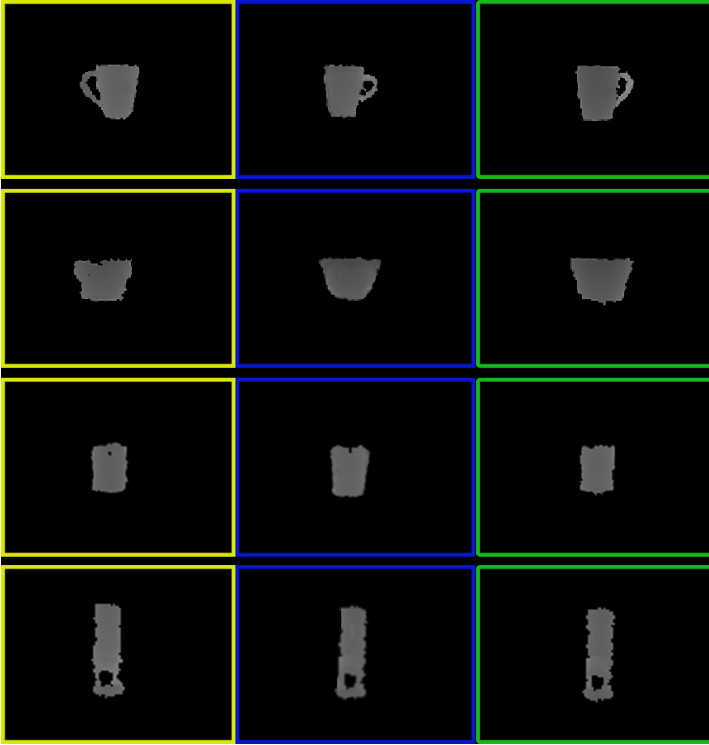


Figure 28: Examples of the results of the reconstruction process, displaying the input objects in the yellow boxes, the inferred objects in blue boxes and the ground truth in the green boxes.

compared to the iterations used by Ek, because the algorithm takes in average 5 minutes per iteration of the optimization step on a 2.67GHz quad-core cpu with 8GB internal memory. For 30 iterations the algorithm needs 150 minutes. When training 13 models, the algorithm needs in total approximately 32.5 hours.

The classification algorithm, needed for the evaluation on the full dataset, works fine (see Fig. 29) with only one out of 20 test images resulting in a wrong classification. With a reached successful classification in 95% of all cases it is a fundamental part of the good performances of the algorithm.

Due to the equally good performance of the clustering algorithm additional to the flexibility of the subspace GP-LVM the evaluation results on the full dataset yield an average error-rate of 0.1343. Only one of our testdata objects results in a completely wrong reconstruction (see Fig. 30). This example shows that the algorithm cannot handle noise surrounding the object. Because of the noise the reconstruction completely fails.

The subspace GP-LVMs are learned on a 7 dimensional shared latent space with 2 dimensions for each private latent space. We only allow for 30 iterations in the optimization process. We set the iterations to such a low value,

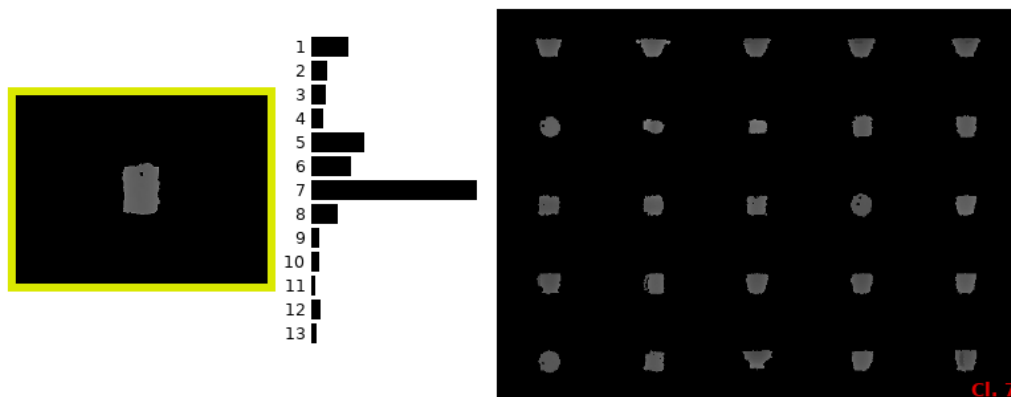


Figure 29: An illustration of the classification. The probabilities that the input object is related to each cluster is computed. Cluster 7 has a probability of 41.51%. It is displayed on the right side of the figure.

In conclusion, to classify the dataset we used, it is to say that it does not consist of too complicated objects for the reconstruction. For example we did not include non-rigid objects. It consists of 10 different object categories and is versatile. Evaluating the algorithm again with a larger, more complicated and more versatile dataset should result in a more detailed estimation of the flexibility of this algorithm.

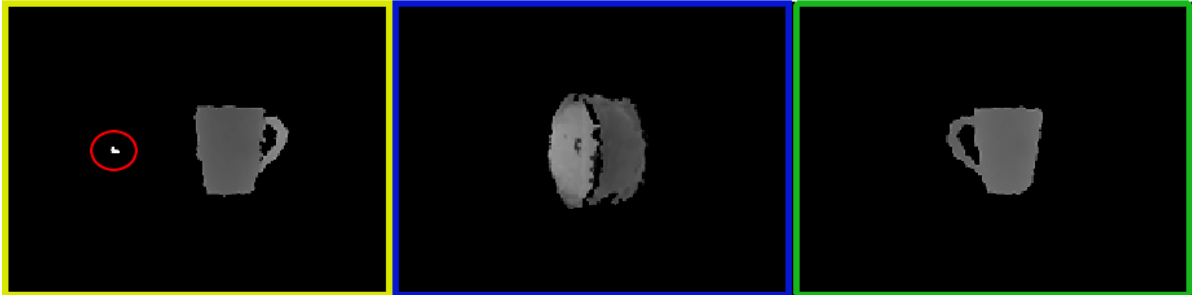


Figure 30: The input data (object in the yellow box) should be completely cleaned of noise surrounding the model shape (marked with the red circle). Otherwise the reconstruction (object in the blue box) fails. The ground truth is displayed in the green box.

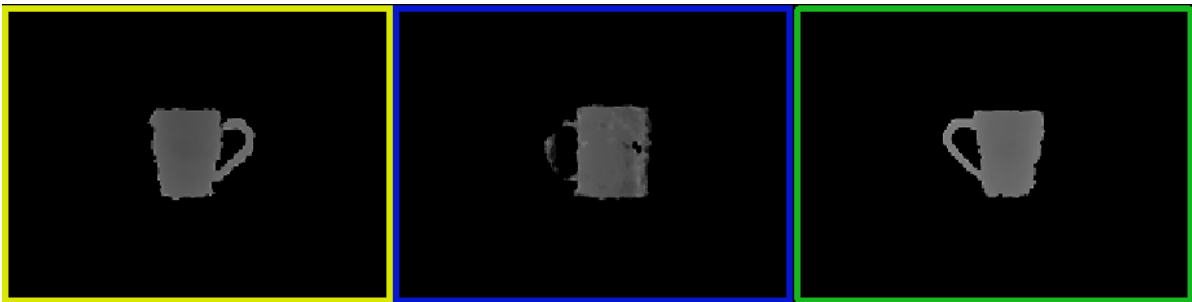


Figure 31: If the latent dimensionality is too low for the data, the model for the reconstruction is underfitted, resulting in poor reconstruction results (see reconstruction in the blue box).

5 Conclusion

In this thesis we developed a multiclass single view 3D shape reconstruction algorithm, in a setup with two corresponding observation spaces. The reconstruction algorithm mainly consists of three core parts, 1) The flexible subspace GP-LVM for learning the common low dimensional shape characteristics of the given objects, 2) the clustering algorithm to split the multiclass data into groups, where each group contains objects that have similar shapes and 3) the classification algorithm which, depending on the input data, selects the model for inferring the back side shape of the object.

The algorithm first learns the shape characteristics of the objects from data, by dividing the given dataset into groups of similar shapes and then learning a subspace GP-LVM on each group. When the algorithm is given a depth map for reconstruction of a previous unseen object, a classification algorithm assigns the given object to a group depending on the shape. The associated subspace GP-LVM is used to synthesize the corresponding representation of the other observation space (back side view of the object).

The experiments on our dataset have shown that the algorithm achieves reasonable results on the test data when provided with diversified data and a large enough latent space, which depends on the composition of the input objects.

5.1 Dataset

Early evaluations on a incomplete dataset have also shown, that the objects given into the subspace GP-LVM should have similar shapes, but should not be too equal. It means that if a model is trained, e.g. on cups, the provided objects for the learning process should not only consist of cups of the same size and form, but should contain cups of various sizes and forms, so that the model is not overfitted in the end. Otherwise the algorithm could have problems to generalize the object characteristics and end up with giving bad reconstruction results on cups of a different size or form.

The dataset we used only contains a small number of objects and an experiment with a more versatile dataset could result in further conclusions. What we did and what always should be done, if possible, is centering the data, so that the model does not have to incorporate the position of the object into the latent space variables. Without centering, the algorithm would still work, but the model would lose some of its flexibility.

5.2 Clustering

In our approach the clustering algorithm determines the composition of each group, which is used for learning a subspace GP-LVM. A clustering algorithm that does not result in overfitted groups is crucial, because otherwise we again would end up with a model having problems to generalize the object characteristics. On the other hand, the obtained clusters should not contain objects of too distinct shapes, otherwise the model would reach its limit also resulting in poor reconstruction results.

As shown in the previous Chapter the clustering algorithms k-means and hierarchical clustering are not suitable for our algorithm. The clusters obtained by k-means tend to be too overfitted or result in clusters with objects of too distinct shapes to learn a model on them. Additionally, for k-means you have to define the number of clusters the algorithm should produce, which is also contradictory to our goal to make the algorithm unsupervised. So, when summing up

the properties of k-means, it shows that it is not usable as a core part for our algorithm. With hierarchical clustering most clusters tend to be overfitted or larger than desired, containing objects of highly distinct shapes. As a consequence, the clusters are highly unbalanced, with some containing almost one third of all objects, whereas others contain only a single object.

Spectral clustering works best for splitting the data into groups depending on their shapes. The clusters obtained are not overfitted and well balanced. As shown in the experiments, our algorithm works well when the models are learned with the clusters obtained via spectral clustering.

5.3 Future Work

As shown, the algorithm presented is already highly flexible. It can be modified to work in a different setup, e.g. with other and/or more observation spaces. The biggest disadvantage of the current implementation is the computation time of the optimization of a model. Currently, to learn a model with only 30 optimization steps, the algorithm takes roughly 150 minutes on a computer with an Intel i7 920 cpu with four 2.67GHz cores and an internal memory of 8GB. An implementation using the multiple cores of a computer could speed up this process significantly.

5.3.1 Unlearned Categories

The algorithm in its current state tries to infer the shape of a previous unseen object by using the model corresponding to the cluster with the most similar objects. So if we are given an object of a category which was never learned, the algorithm uses the model with the highest similarity to infer the back side shape of the object.

In that case, to enhance the quality of the reconstruction, we tried to integrate a mixture-of-experts approach into the reconstruction process (mentioned in the Methods-Chapter). The model used a combination of clusters instead of only one to infer the shape of the object. The models for the reconstruction were chosen depending on the posterior probabilities that the input object belongs to them.

Each of the chosen models inferred the shape for the reconstruction independently, before combining the results depending on the probabilities.

The approach worked well, except for the combination method. When we combined the different results, we used a primitive weighted average of the pixels, where we used the posterior probabilities as weights. This combination method is not suitable for the task and further work has to be put in, to develop a more suitable approach.

A more convenient combination method could be, to use a grid of static size and fit it to the object of the reconstruction results. The combined result could be obtained by computing the weighted average (position and value) on each grid point.

5.3.2 Cluster Merging

The clusters, obtained with the evaluated algorithms, sometimes resulted in clusters where the shape of the objects of different clusters were similar. In these cases our algorithm learns a subspace GP-LVM on each of them. This can result in overfitted models and also increases the computation time of the learning process significantly.



Figure 32: An example of two clusters (blue and red) that could be merged to enhance the performance of the reconstruction algorithm.

Depending on the applied algorithm and the dataset, a cluster merging algorithm could be integrated to optimize them. This algorithm could help to prevent these overfitted clusters to make use of the full flexibility of the subspace GP-LVM. It would also speed up the learning process by reducing the number of models which have to be optimized.

References

- [1] F. R. Bach and M. I. Jordan. A probabilistic interpretation of canonical correlation analysis. Technical Report 688, University of California, Berkley, 2005.
- [2] O. Barinova, V. Konushin, A. Yakubenko, K. Lee, H. Lim, and A. Konushin. Fast automatic single-view 3-d reconstruction of urban scenes. *European Conference on Computer Vision*, 2008.
- [3] C. M. Bishop and M. E. Tipping. Probabilistic principal component analysis. *Journal of the Royal Statistical Society*, (61):611–622, 1999.
- [4] L. Bo, X. Ren, and D. Fox. Kernel descriptors for visual recognition. *Advances in Neural Information Processing Systems*, (23):244–252, 2010.
- [5] L. Bo, X. Ren, and D. Fox. Depth kernel descriptors for object recognition. *International Conference on Intelligent Robots and Systems*, 2011.
- [6] M. A. Boden. *Mind As Machine: A History of Cognitive Science*. Oxford University Press, 2006.
- [7] M.D. Buhmann. *Radial basis functions: theory and implementations*. Cambridge University Press, 2003.
- [8] Y. Chen and R. Cipolla. Learning shape priors for single view reconstruction. *International Conference on Computer Vision*, (12):1425–1432, 2009.
- [9] A. Criminisi, I. Reid, and A. Zisserman. Single view metrology. *International Journal of Computer Vision*, (40(2)):123–148, 2000.
- [10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Conference on Computer Vision and Pattern Recognition*, 2005.
- [11] C. H. Ek. *Shared Gaussian Process Latent Variable Models*. PhD thesis, Oxford Brookes University, 2009.
- [12] P. Golland. *Computer vision for medical imaging*. 2013.
- [13] P. Guan, A. Weiss, A. Balan, and M. Black. Estimating human shape and pose from a single image. *International Conference on Computer Vision*, 2009.
- [14] F. Han and S. Zhu. Bayesian reconstruction of 3d shapes and -scenes from a single image. *HLK'03: Proc. the 1st IEEE Int. Workshop on Higher-Level Knowledge*, 2003.
- [15] D. Hoiem, A. Efros, and M. Hebert. Automatic photo pop-up. *SIGGRAPH*, pages 577–584, 2005.
- [16] J. J. Hull, G. Krishnan, P. Palumbo, and S. N. Srihari. Optical character recognition techniques in mail sorting: A review of algorithms. Technical Report 214, University at Buffalo, 1984.
- [17] N. Jiang, P. Tan, and L.-F. Cheong. Symmetric architecture modeling with a single image. *SIGGRAPH*, 2009.

-
- [18] H. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5), 1999.
- [19] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.
- [20] O. Kroemer, H. Ben Amor, M. Ewerton, and J. Peters. Point cloud completion using symmetries and extrusions. In *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*, 2012.
- [21] M. Kuss and T. Graepel. The geometry of kernel canonical correlation analysis. Technical Report TR-108, Max Planck Institute for Biological Cybernetics, Tübingen, 2003.
- [22] N. Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of Machine Learning Research*, (6):1783–1816, 2005.
- [23] N. Lawrence, M. Seeger, and R. Herbrich. Fast sparse gaussian process methods: The informative vector machine. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 625–632. MIT Press, 2003.
- [24] G. Lefaix, É. Marchand, and P. Bouthemy. Motion-based obstacle detection and tracking for car driving assistance. *International Conference on Pattern Recognition*, 4:74–77, 2002.
- [25] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.
- [26] Y. Lu, T.-Q. Chen, J. Chen, A. Tisler, and J. Zhang. An advanced machine vision system for vfd inspection. Technical report, The University of Michigan-Dearborn, Dearborn, 1996.
- [27] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proc. of the Fifth Berkeley Symp. On Math. Stat. and Prob.*, 1:281–296, 1967.
- [28] Martin F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.
- [29] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 849–856. MIT Press, 2002.
- [30] M. Prasad, A. Fitzgibbon, A. Zisserman, and L. Gool. Finding nemo: Deformable object class modelling using curve matching. *Conference on Computer Vision and Pattern Recognition*, 2010.
- [31] N. Rajalingam and K. Ranjini. Article: Hierarchical clustering algorithm - a comparative study. *International Journal of Computer Applications*, 19(3):42–46, April 2011.
- [32] D. Rother and G. Sapiro. Seeing 3d objects in a single 2d image. *International Conference on Computer Vision*, 2009.
- [33] A. Saxena, J. Driemeyer, and A. Y. Ng. Robotic grasping of novel objects using vision. *International Journal of Robotics Research*, 2007.
- [34] A. P. Shon, K. Grochow, A. Hertzmann, and R. Rao. Learning shared latent structure for image synthesis and robotic imitation. *Conference on Neural Information Processing Systems*, 2005.

-
- [35] L. Sigal, A. Bălan, and M. Black. Combined discriminative and generative articulated pose and non-rigid shape estimation. *Conference on Neural Information Processing Systems*, 2007.
- [36] M. Sun, G. Bradski, B.-X. Xu, and S. Savarese. Depth-encoded hough voting for joint object detection and shape recovery. *European Conference on Computer Vision*, 2010.
- [37] L. Zhang, G. Dugas-Phocion, and J. Samson. Single-view modeling of free-form scenes. *The Journal of Visualization and Computer Animation*, (13):225–235, 2002.