Linköping Studies in Science and Technology
Dissertation No. 1160

Channel-Coded Feature Maps for Computer Vision and Machine Learning

Erik Jonsson

Department of Electrical Engineering
Linköpings universitet, SE-581 83 Linköping, Sweden

Linköping February 2008

Channel-Coded Feature Maps for Computer Vision and Machine Learning

Erik Jonsson

Department of Electrical Engineering
Linköping University
SE-581 83 Linköping
Sweden

Linköping Studies in Science and Technology
Dissertation No. 1160

Copyright © 2008 Erik Jonsson

ISBN 978-91-7393-988-1
ISSN 0345-7524

Back cover illustration by Nikolina Orešković

Printed by LiU-Tryck, Linköping 2008

To Helena

for patience, love and understanding

Abstract

This thesis is about channel-coded feature maps applied in view-based object recognition, tracking, and machine learning. A channel-coded feature map is a soft histogram of joint spatial pixel positions and image feature values. Typical useful features include local orientation and color. Using these features, each channel measures the co-occurrence of a certain orientation and color at a certain position in an image or image patch. Channel-coded feature maps can be seen as a generalization of the SIFT descriptor with the options of including more features and replacing the linear interpolation between bins by a more general basis function.

The general idea of channel coding originates from a model of how information might be represented in the human brain. For example, different neurons tend to be sensitive to different orientations of local structures in the visual input. The sensitivity profiles tend to be smooth such that one neuron is maximally activated by a certain orientation, with a gradually decaying activity as the input is rotated.

This thesis extends previous work on using channel-coding ideas within computer vision and machine learning. By differentiating the channel-coded feature maps with respect to transformations of the underlying image, a method for image registration and tracking is constructed. By using piecewise polynomial basis functions, the channel coding can be computed more efficiently, and a general encoding method for N-dimensional feature spaces is presented.

Furthermore, I argue for using channel-coded feature maps in view-based pose estimation, where a continuous pose parameter is estimated from a query image given a number of training views with known pose. The optimization of position, rotation and scale of the object in the image plane is then included in the optimization problem, leading to a simultaneous tracking and pose estimation algorithm. Apart from objects and poses, the thesis examines the use of channel coding in connection with Bayesian networks. The goal here is to avoid the hard discretizations usually required when Markov random fields are used on intrinsically continuous signals like depth for stereo vision or color values in image restoration.

Channel coding has previously been used to design machine learning algorithms that are robust to outliers, ambiguities, and discontinuities in the training data. This is obtained by finding a linear mapping between channel-coded input and output values. This thesis extends this method with an incremental version and identifies and analyzes a key feature of the method – that it is able to handle a learning situation where the correspondence structure between the input and output space is not completely known. In contrast to a traditional supervised learning setting, the training examples are groups of unordered input-output points, where the correspondence structure within each group is unknown. This behavior is studied theoretically and the effect of outliers and convergence properties are analyzed.

All presented methods have been evaluated experimentally. The work has been conducted within the cognitive systems research project COSPAL funded by EC FP6, and much of the contents has been put to use in the final COSPAL demonstrator system.

Populärvetenskaplig Sammanfattning

Datorseende (computer vision) är en ingenjörsvetenskap som går ut på att skriva datorprogram som efterhärmar det mänskliga synsinnet – som kan känna igen föremål, ansikten, orientera sig i ett rum och liknande. Inom maskininlärning (machine learning) försöker man istället få datorer att härma människans förmåga att lära sig av sina erfarenheter. Dessa två forskningsområden har mycket gemensamt, och inom datorseende lånar man ofta tekniker från maskininlärning.

Som ett första steg i att få en dator att känna igen en bild krävs att intressanta områden i bilden beskrivs matematiskt. Ofta studerar man särdrag (features) så som färg och riktning hos strukturer i bilden, till exempel gränser mellan olikfärgade områden. Denna avhandling handlar om en speciell sorts matematisk beskrivning av sådana egenskaper som kallas kanalkodade särdragskartor (channel-coded feature map). En sådan beskrivning består av ett antal *kanaler*, där varje kanal mäter förekomsten av en viss färg och en viss strukturorientering i närheten av en viss position i bilden. Detta har vissa likheter med hur man tror att information representeras i den mänskliga hjärnan.

En stor del av avhandlingen handlar just om att känna igen föremål. Den teknik som används kallas vybaserad igenkänning. Detta innebär att man tränar systemet genom att visa så kallade *träningsbilder* på föremålet sett från olika vinklar. Man låter bli att försöka tillverka någon slags tredimensionell modell av föremålet, utan nöjer sig med att jämföra tvådimensionella bilder med varandra. En fördel med denna approach är att den är relativt enkel, men en nackdel är att ett stort antal träningsbilder kan behövas.

Efter att ett objekt har detekterats kan det vara bra att finjustera objektets uppskattade position så noga som möjligt. För detta används i avhandlingen en matematisk optimeringsprocedur som bygger på derivator. Dessa derivator beskriver hur snabbt de olika kanalerna ökar och minskar i värde när man flyttar runt dem i bilden, och i avhandlingen beskrivs hur derivatorna till de kanalkodade featurekartona plockas fram.

Förutom inom objektigenkänning kan kanalkodning användas i mer allmänna maskininlärningsproblem. Här utgår man från exempel av indata och utdata, så kallade *träningsexempel*, och systemets uppgift är att generalisera utifrån dessa exempel. I avhandlingen studeras bland annat just fallet med vybaserad objektigenkänning som beskrivits ovan, där indata och utdata är en uppsättning vinklar och en kanal-kodad särdragskarta som beskriver föremålets utseende. Efter inlärningsfasen kan systemet generalisera och förutsäga föremålets utseende även för nya indata (vinklar) som systemet aldrig stött på förut. I avhandlingen studeras även en ny typ av inlärningsproblem, *korrespondens-fri inlärning*. Här finns det inte längre någon tydlig struktur bland träningsexemplen, så systemet vet inte riktigt vilken indata som ska höra ihop (korrespondera) med vilken utdata. Det visar sig att denna typ av problem går att lösa med hjälp av kanalkodningstekniker.

Alla metoder som presenteras har utvärderats experimentellt. Arbetet har utförts inom det EU-finansierade forskningsprojektet COSPAL (Cognitive Systems using Perception-Action Learning), och mycket av innehållet har använts i COSPALs slutgiltiga teknikdemonstrator.

Acknowledgments

I want to thank...

all members of the Computer Vision Laboratory in Linköping, most notably

- Gösta Granlund, for giving me the opportunity to work here, for lots of great ideas, and for having patience with me despite occasional differences of opinion.
- Michael Felsberg, for being a great supervisor in terms of technical expertise, availability, devotion and friendliness.
- Johan Hedborg, for implementing the multi-dimensional convolutions in the the piecewise polynomial encoding and helping with the cover photography.
- Fredrik Larsson, Johan Sunnegårdh and Per-Erik Forssén, for proof-reading parts of the manuscript and giving valuable comments.

all members of the COSPAL consortium, most notably

- Jiří Matas and Václav Hlaváč, for letting me stay at the Center for Machine Perception, Prague Technical University for three weeks.
- Florian Hoppe, Eng-Jon Ong, Alexander Shekovtsov and Johan Wiklund for good teamwork in putting the COSPAL demonstrator together during sometimes dark hours in Kiel. Always remember.. Fensterklappe bitte schließen bei Benutzung der Verdunkelung.

all friends and family, most notably

- Helena, for having patience with me living in the wrong city for three and a half years, and for making me happy.
- My parents, for infinite support in all matters.

Arrest this man, he talks in maths
[Karma Police, Radiohead]

Contents

Chapter 1

Introduction

No one believes that the brain uses binary code to store information. Yet, binary code is the unrivaled emperor of data representation in computer science as we know it. This has produced computers that can multiply millions of large numbers in a second, but can hardly recognize a face – that can create photorealistic virtual environments faster and better than any human artist, but can not tell a cat from a dog.

Perhaps the entire computer architecture is to blame? Perhaps even binary coding is not the best option when the goal is to create artificial human-like systems? This motivates exploring different kinds of information representations. The ultimate long-term goal with this research is to study the possibility of creating artificial cognitive systems based on a completely new computing platform, where *channel-coding* is the primitive representation of information instead of binary code. The more realistic short-term goal is to find good engineering applications of channel-coding using the computing platforms of today.

How the human brain works is a mystery, and this thesis makes no attempts to solve this. I do not make any claims that any algorithm presented in this thesis accurately mimics the brain. Rather, this thesis is about finding good engineering use of an idea that originates from biology.

The essence of my PhD project can be summarized in one sentence: *Explore the properties of and find good uses for channel-coding in artificial learning vision systems.* This quest has lead me on a journey in pattern recognition and computer vision, visiting topics such as Markov random fields, probability density estimation, robust statistics, image feature extraction, patch tracking, incremental learning, pose estimation, object recognition and object-oriented programming. At several occasions, related techniques have been discovered under different names. In some cases, dead ends have been reached, but in other cases all pieces have fallen into place. The quest eventually ended up with object recognition being the application in focus.

1.1 Thesis Overview

This section gives a short overview of the thesis. I also describe the dependencies between different chapters as a guide to a reader who does not wish to read the entire thesis.

2: Channel-Coded Scalars. The first part of this chapter describes the main principles of channel-coding and is required for all that follows. The part about continuous reconstruction is a prerequisite of Chapter 6 but not for the rest of the thesis.

3: Channel-Coded Feature Maps. Describes the concept of channel-coded feature maps (CCFM), which are very central to the thesis. Derivatives of CCFMs with respect to image transformations are derived and applied in patch tracking. These derivatives are used again in Chapter 11.

4: Channel Coding through Piecewise Polynomials. Describes an efficient method of constructing channel-coded feature maps when the basis functions are piecewise polynomials. This method is used in all experiments, but the details are rather involved and are not required for later chapters.

5: Distance Measures on Channel Vectors. Describes some distance measures between histogram data and compares them experimentally on the COIL database. The experiments in Chapter 10 relate to these results.

6: Channels and Markov Models. Discusses what happens if the hard discretizations commonly used in Markov models are replaced with channel vectors. The resulting algorithm turns out to be impractical due to the computational complexity. This direction of research is rather different from the rest of the thesis and can be skipped without loss of continuity.

7: Associative Networks. Gives an overview of previous work on associative networks (linear mappings between channel-coded domains). Different incremental updating strategies and the relationship to Hough transform methods is discussed. This chapter is tightly connected to Chapter 8 and referred to from Chapter 10, but is not a prerequisite for the pose estimation and tracking in Chapter 10 and 11.

8: Correspondence-Free Learning. Describes a new type of machine learning problems, where the common assumption of corresponding training data is relaxed. Tightly connected to Chapter 7, but not a prerequisite for later chapters.

9: Locally Weighted Regression. Reviews the LWR method by Atkeson and Schaal and extends this method by deriving an exact expression for the Jacobian. This chapter is rather self-contained and can be read even without the introductory chapters on channel coding. The LWR method is used in Chapter 10 and 11, but the analytical Jacobian derivation is rather technical and not essential for

understanding the applications.

10: Linear Interpolation on CCFMs. Discusses different view interpolation strategies for pose estimation. Uses the locally weighted regression method from Chapter 9. Should be read before Chapter 11.

11: Simultaneous View Interpolation and Tracking. Ties together the tracking from Chapter 3 with the view interpolation from Chapter 10 and formulates a single problem in which both the pose and image position of an object are optimized simultaneously.

12: A Complete Object Recognition Framework. Discusses aspects like object detection, preprocessing and object-oriented system design. These are not central to the thesis but still required in order to implement a complete object recognition system.

1.2 Contributions and Previous Publications

This thesis consists to a large extent of material adapted from previous publications by the author.

The continuous reconstruction in Chapter 2 has been adapted from [56], while the rest of Chapter 2 is a summary of previous work by others. The algorithm presented in Chapter 4 has been submitted as a journal article [55], and the Soft Message Passing algorithm in Chapter 6 is adapted from [58].

A previous version of Chapter 7 is available as a technical report [60]. The contribution of this chapter is mainly in the application of common least-squares techniques to the associative network structure.

Chapter 8 is essentially the ICPR contribution [57] but has been extended with more experiments. The contributions here are both in the problem formulation, solution and theoretical analysis of the correspondence-free learning method.

The contents of [59] has for the sake of presentation been split into several parts. The channel-coded feature maps with derivatives in Chapter 3 and the simultaneous pose estimation and tracking procedure in Chapter 11 both originate from this paper. Both these chapters have been extended with new experiments.

Except for the contents of these publications, the differentiation of the LWR model in Chapter 9 is believed to be novel. The discussion about view interpolation in Chapter 10 uses some relatively common least-squares concepts, but to my best knowledge, these issues have not been addressed in connection with view interpolation on channel-coded feature maps elsewhere.

Chapter 12 is not considered as a theoretical contribution, but as a piece of condensed experience from my work on implementing an object recognition system for the COSPAL demonstrator.

1.3 The COSPAL Project

This work has been conducted within the COSPAL project (Cognitive Systems using Perception-Action Learning), funded by the European Community (Grant IST-2003-004176) [1]. This project was about artificial cognitive systems using the following key ideas:

- *Combining continuous learning and symbolic AI.* For low-level processing, visual information and actions are best represented in a continuous domain. For high-level processing, actions and world state is best described in symbolic terms with little or no metric information.

- *Link perception and action from the lowest level.* We do not need a full symbolic description of the world before actions can be initiated. Rather, the symbolic processing builds upon low-level primitives involving both perception and action. A typical example of such a low-level perception-action primitive is visual servoing, where the goal is for example to reach and maintain an alignment between the robot and some object.

- *Use minimum prior assumptions of the world.* As much as possible should be learned by the system, and as little as possible should be built-in.

Much of the project philosophy originates from [38, 41]. Channel coding techniques [42] were predicted as an algorithmic cornerstone, and one entire work package was devoted to exploring different aspects of channel coding. This thesis is not about the COSPAL project in general and will not address issues like overall system structure. However, the project goals have more or less determined my direction of research, and references will be made to the COSPAL philosophy at various places in the thesis.

1.4 Notation

1.4.1 Basic Notation

s	Scalars
$\mathbf{u}$	Vectors (always column vectors)
$\mathbf{C}$	Matrices
$s(\mathbf{x}), I(x, y)$	Scalar-valued functions
$\mathbf{p}(\mathbf{x}), \mathbf{r}(t)$	Vector-valued functions
$\mathcal{X}$	Sets and vector spaces
$\mathbf{1}$	Vector of ones, with dimensionality determined by the context
$\mathbf{A}^{\mathrm{T}}$	Matrix and Vector Transpose
$\langle \mathbf{A}, \mathbf{B} \rangle_{\mathrm{F}}$	Frobenius matrix product (sum of elementwise product)
$\|\mathbf{u}\|$	Euclidean vector norm
$\|\mathbf{M}\|_{\mathrm{F}}$	Frobenius matrix norm

$\mathrm{diag}(\mathbf{u})$	Extension of a vector to a diagonal matrix
$\mathrm{diag}(\mathbf{A})$	Extraction of the diagonal of a matrix to a vector
$\theta(x)$	Heaviside step function ($\theta(x) = 0$ for $x < 0$ and 1 for $x > 0$)
$\delta(x)$	Dirac distribution (sometimes denoted sloppily as a function)
$\Pi(x)$	Box function (1 for $-0.5 < x \leq 0.5$, zero otherwise)

1.4.2 Elements of Matrices and Vectors

Vectors and matrices are viewed as arrays of real numbers – the abstract notion of vectors existing on their own independently of any fixed coordinate system is not used. Elements of a vector or matrix are usually denoted with subscripts:

$$\mathbf{u} = [u_1, u_2, \ldots, u_N]^{\mathrm{T}}$$

A subscript on a boldface symbol means the n'th vector or matrix in a collection, and not the n'th element:

$\mathbf{u}_n$	The n'th vector in a set of vectors
$\mathbf{A}_k$	The k'th matrix in a set of matrices

Sometimes, in order to avoid too cluttered sub- and superscripts, the C/Java-inspired bracket [ ] is used to denote elements of a matrix or vector:

$\mathbf{u}[n]$	The n'th element of $\mathbf{u}$
$\mathbf{u}_k[n]$	The n'th element of vector $\mathbf{u}_k$
$\mathbf{A}_k[i + 2j, j]$	Element at row $i + 2j$, column j of matrix $\mathbf{A}_k$

1.4.3 Common Quantities

I have tried to keep the notation consistent such that common quantities are always referred to using the same symbols. This is a list of all symbols commonly (but not exclusively) used to denote the same thing throughout the thesis. Use it as a quick reference.

$\mathbf{c}$	Channel vector or N-dimensional channel grid
$\mathbf{I}$	Discrete image, with pixels accessed as $\mathbf{I}[x, y]$
$\mathbf{h}$	Filter kernel, usually with $\mathbf{h}[0, 0]$ in the center
$B(x)$	Basis function for channel encoding
$[s, \alpha, x, y]^{\mathrm{T}}$	Similarity frame (scale, rotation, translation)
(θ, ϕ)	Pose angles

Most of the time, I use a lowercase-uppercase combination to denote a 1-based integer index and its maximal value. For reference, the most common ones are

n, N	Channel index, number of channels
t, T	Training example index, number of training examples
l, L	Class label, number of class labels
i, I	point index, sample size (within one channel vector)

1.4.4 Miscellaneous

In this thesis, I will often use differential notation for derivatives. Appendix A contains an introduction to differentiating linear algebra expressions with a description of my notation and some basic relations.

By default, I will skip the bounds on sums and integrals where the range is the entire definition domain of the involved variables. Since this is the most common case, it saves a lot of writing and makes the non-default cases stand out more.

Part I

Channel Coding

Chapter 2

Channel-Coded Scalars

...where we get to meet the Channel Vector for the first time, together with one of its applications. We will start with a piece of cross-platform science in order to get a deeper understanding of the motivation and purpose of the thesis. Hold on – we will soon be back in the comfortable world of equations and algorithms.

2.1 Background

Try to temporarily forget all you know about computer science, and in particular about binary numbers. Instead, we will consider other representations of information. In order to really go back to the beginning, we must start with the concept of *representation* itself. In principle, some basic image processing could be done using a set of mirrors, physical color filters, lights and so on. It is possible to change the intensity of, rotate, stretch, skew, mirror and color adjust an image using a purely passive, optical device. In this case, we do not need any representation of the physical phenomenon we are processing other than the actual physical signal itself.

The abstraction starts once we use something else as a placeholder for the physical signal. This "something else" may be an electrical current or some combination of chemical and electrical signals depending on if we are looking at a typical man-made device or a typical biological information processing system (i.e. a central nervous system of some living being). The simplest abstraction is to use an *analog* signal, where one measurable quantity is represented directly by another measurable quantity. In analog audio equipment, the air pressure level is directly represented by an electrical voltage or current. The representation behaves *analogously* to the phenomena which we are describing. The reason for switching to some abstract representation of the physical quantity is usually that the representation is simpler to manipulate.

Analog representations have shown to be susceptible to noise and are not very well-suited for advanced processing. This is why today more and more information

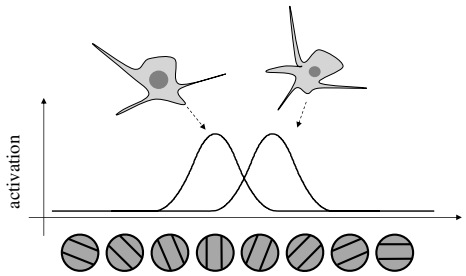

Figure 2.1: Illustration of the sensitivity profile for two neurons measuring local image orientation.

processing is performed using digital representations, with binary numbers as the basic cornerstone. However, despite all advances in digital computers, we are far away from building anything that can compete with the human brain when it comes to intelligence, learning capabilities, vision and more.

To begin to understand some principles of information representation in the brain, consider for example the case of local orientation of image structures. It is known that different neurons are activated for different orientations [35]. However, the switch between different neurons is not discrete – rather, as the orientation is slightly changed, the activation of some neurons is reduced and the activation of other neurons is increased, as illustrated in Fig. 2.1. By capturing the essence of this behavior and describing it as a general way of representing real numbers, we end up with the *channel representation* [42].

In the thesis, these biological aspects will not be stressed much further. Instead, the focus is shifted towards that of an engineer. This chapter will formalize the channel coding principle mathematically and study some basic properties of it. The rest of the thesis will then explore different aspects and applications of channel coding, with the objective of finding competitive computer vision and learning algorithms rather than as a way of understanding the human brain. In particular, I do not make any claims that any part of this thesis explains what is actually going on in the brain – for my research, I consider biology as a source of inspiration rather than as a scientific goal on its own.

2.2 Channel Coding Basics

This section gives a brief overview of channel coding in order to reach the first application as soon as possible. In Sect. 2.3, a more thorough treatment on the different options and algorithms is given.

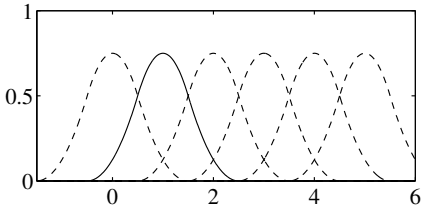

Figure 2.2: A regular grid of channels, with one basis function highlighted.

2.2.1 Channel Vectors

A *channel vector* $\mathbf{c}$ is constructed from a scalar x by the nonlinear transformation

$$\mathbf{c} = [B(x - \tilde{x}_1), \ldots, B(x - \tilde{x}_N)]^{\mathrm{T}} \ . \tag{2.1}$$

Here, B is a symmetric non-negative basis function with compact support. The values $\tilde{x}_n, n \in [1, N]$ are called *channel centers*. The simplest case is where the channel centers are located at the integers, as illustrated in Fig. 2.2. The process of constructing a channel vector from a scalar is referred to as *channel coding* or simply *encoding* the value x.

Given a channel vector $\mathbf{c}$, it is possible to reconstruct the encoded value x. This backwards procedure is referred to as *decoding*. If the channel vector is affected by noise, the decoding may not be exact, but a desirable property of any decoding algorithm is that the encoded value should be reconstructed perfectly in the noise-free case. A detailed decoding algorithm is presented in Sect. 2.4.

2.2.2 Soft Histograms

Assuming that we have I samples x_i of some variable, each sample can be encoded and the channel vectors for different i's summed or averaged. This produces a *soft histogram* - a histogram with partially overlapping and smooth bins:

$$\mathbf{c}[n] = \frac{1}{I} \sum_i B(x_i - \tilde{x}_n) \ . \tag{2.2}$$

The basis function B can here be thought of as a *bin function*. In a regular histogram, each sample is simply put in the closest bin, and we cannot expect to locate peaks in such histograms with greater accuracy than the original bin spacing. In a soft histogram constructed according to (2.2), the bins are overlapping, and samples are weighted relative to their distance to the bin center. This makes it possible to locate peaks in the histogram with sub-bin accuracy. The construction and decoding of a soft histogram is illustrated in Fig. 2.3. If all samples are almost similar, the soft histogram resembles a single encoded value, and a decoding procedure as mentioned before will find the peak almost perfectly.

Many methods in computer vision achieve robustness from a voting and clustering approach. The simplest example is the Hough transform [6, 85], but the same

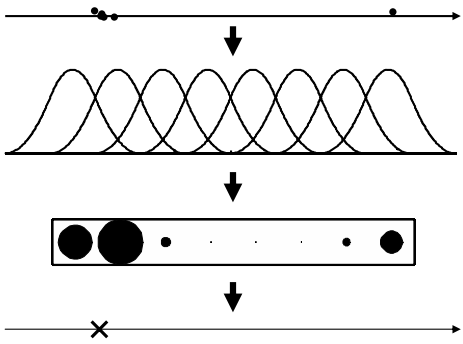

Figure 2.3: Illustration of a number of points encoded into a soft histogram. The resulting channel vector is then decoded in order to find the cluster center.

principle is found in view matching [66] and object recognition using local features [52, 78]. The idea is that a number of local measurements are gathered, and each measurement makes a vote for what kind of object is present. In the Hough transform, the local measurements are edge pixels that vote for lines. In object recognition, the local measurements can for example be patches cut out around some interest points voting for object identity and position. A large number of votes is then expected to be found in a cluster close to the correct hypothesis.

The Hough transform uses a 2D histogram to represent the votes and find peaks. This becomes impractical in higher dimensionalities, since the number of bins required grows exponentially in the number of dimensions. Using soft histograms, the number of bins required could be reduced without impairing the accuracy of the peak detection. The relation between channel coding and voting-type methods will touched again in Sect. 7.3.

2.2.3 An Initial Example - Channel Smoothing

At this point we can already study the first application, which is edge-preserving image denoising. The use of channel coding in this application was studied in [30] and further developed in [25]. In [25], an improved decoding algorithm was presented and the method was compared to other popular edge-preserving filtering techniques, including bilateral filtering and mean-shift filtering.

The idea is to encode the intensity of each pixel in a grayscale image. If the original image was of size $X \times Y$, this produces a three-dimensional data set of size $X \times Y \times N$, where N is the number of channels used. This can be seen as a stack of N parallel images – one for each graylevel channel. Each of these N images is convolved with a smoothing kernel, e.g. a Gaussian. Each voxel in the $X \times Y \times N$ volume now gives a measure of the number of pixels in a certain neighborhood having a certain gray level. Equivalently, the channel vector at image position (x, y) is a soft histogram of the gray levels in a neighborhood around (x, y).

The final step in the algorithm is to decode each of the $X \times Y$ channel vectors, i.e. find a peak of each soft local graylevel histogram. The resulting output image

Figure 2.4: An image with Gaussian and salt & pepper noise restored using channel smoothing.

consists of these decoded values. An example of a restored noisy image is given in Fig. 2.4. As can clearly be seen, the output is a blurred version of the input, but where the edges have been preserved and outliers (salt & pepper noise) been removed.

The key to this behavior is that graylevels which are close together will be averaged while graylevels that are sufficiently different will be kept separate. In fact, the combination of encoding, averaging and decoding is roughly equivalent to applying a robust estimator on the original distribution of pixel graylevels in the neighborhood. This will be explained more thoroughly in Sect. 2.4.

2.3 Channel Coding - Details

At this point, I hope that the reader has a rough idea of what channel coding is all about. In this section, more detail will be filled in. Various choices of basis functions and channel layout strategies will be treated, the representation will be extended to higher dimensionalities, and the relationship between channel vectors and density functions will be established.

2.3.1 Channel Basis Functions

In the definition (2.1), we are free to use a wide range of basis functions B. The basis function used in [52, 42, 44] was a truncated $\cos^2$ function. Another option, used for example in [30, 25] is the second-order B-spline (see Appendix B). The Gaussian kernel is also described in [30] together with a decoding method. In [28], techniques related to first order splines are also considered. By using the zeroth order B-spline (or box function) as basis function, the channel vector becomes effectively a regular histogram. This will be referred to as a *hard histogram*. This basis function is important because it ties together hard histograms and channel vectors.

The B-splines will be the primary choice in this thesis because the fact that they are piecewise polynomials can be exploited in order to construct efficient algorithms. The first example of this will be the second-order B-spline decoding in Sect. 2.4.2. Later, the entire chapter 4 builds upon this piecewiseness. Note however that this choice is mostly motivated by some computational tricks. The results of any method are not expected to depend much on the exact shape of the basis functions, and it is definitely not likely that any biological system utilizes the piecewise polynomial structure in the same way as is done in this thesis. Once again, keep in mind that this thesis aims at engineering artificial systems rather than explaining biological systems.

2.3.2 Channel-Coded Vectors

The representation (2.1) can be extended into higher dimensions in a straightforward way by letting x and $\tilde{x}_n$ be vectors, and letting B accept a vector input:

$$\mathbf{c}[n] = B(\mathbf{x} - \tilde{\mathbf{x}}_n) \ . \tag{2.3}$$

The most common special case is where the channel centers are located on a regular multi-dimensional grid, and the basis function is separable. Consider a case where $\mathbf{x} = [x, y, z]$ and $B(\mathbf{x}) = B_x(x)B_y(y)B_z(z)$. The channel-coded $\mathbf{x}$ can then be written as

$$\mathbf{c}[n_x, n_y, n_z] = B_x(x - \tilde{x}_{n_x})B_y(y - \tilde{y}_{n_y})B_z(z - \tilde{z}_{n_z}) \ . \tag{2.4}$$

This can be seen as encoding x, y, z separately using a single-dimensional encoding and then taking the outer product:

$$\mathbf{c}_x = \mathrm{enc}(x) \tag{2.5}$$
$$\mathbf{c}_y = \mathrm{enc}(y) \tag{2.6}$$
$$\mathbf{c}_z = \mathrm{enc}(z) \tag{2.7}$$
$$\mathbf{c} = \mathbf{c}_x \otimes \mathbf{c}_y \otimes \mathbf{c}_z \ . \tag{2.8}$$

The final encoding $\mathbf{c}$ can be seen as a 3D array indexed by n_x, n_y, n_z or as a vector indexed by a single index n constructed by combining n_x, n_y, n_z into a linear index.

2.3.3 Channel Centers

A channel center is the position in input space at which a certain channel responds maximally. Depending on the application, these centers can be selected in different ways. Different patterns have been considered, e.g. a foveal layout in 2D [42]. In this thesis, as well as in much previous work, the default choice will be to have a regular channel spacing. When this is the case, the presentation can often be simplified by using channels located on the integers. This can be assumed without loss of generality since it is only a matter of rescaling the input space. There are different strategies for where to put the channel centers relative to the bounds of the data that is to be encoded. Two common strategies are described here.

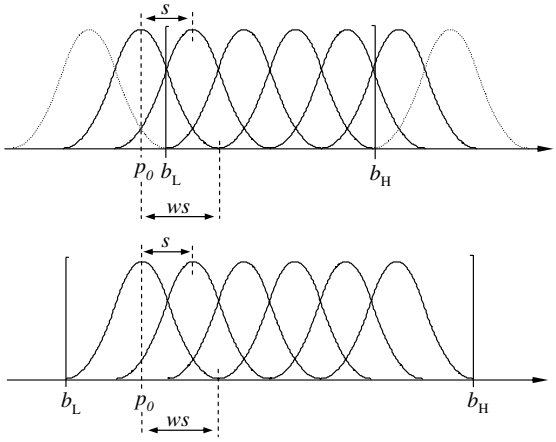

Figure 2.5: Illustration of channel layouts. Top: exterior. Bottom: interior.

Assume that the values to encode are in the range $[b_L, b_H]$. The position of the first channel is p_0 and the spacing between the channels is s. Each of the N channels has a support of size $2ws$, where w expresses the amount of overlap between the channels (see Fig. 2.5). If $w = 0.5$, the support size equals the channel spacing, meaning that there is no overlap between channels. In Fig. 2.5, $w = 1.5$.

· **Exterior Layout**
This layout places the channels equally spaced such that exactly k channels are active for any input. This is only possible if w is a multiple of 0.5. Assume that the parameters N, b_L, b_H, w are given, and we want to find p_0, s. The situation is illustrated in Fig. 2.5. The dashed channels in the figure are completely outside the interval $[b_L, b_H]$ and should not be included. From the figure, we see that the center of these non-included channels are $p_0 - s$ and $p_0 + sN$. These channels end exactly at the region bounds, such that $p_0 - s + ws = b_L$ and $p_0 + sN - ws = b_H$. Solving for p_0 and s gives

$$p_0 = (b_H + b_L N - w(b_H + b_L))/(N + 1 - 2w) \tag{2.9}$$
$$s = (b_H - b_L)/(N + 1 - 2w) \ . \tag{2.10}$$

· **Interior Layout**
This mode places the channels equally spaced such that no included channel is active outside the region bound. The first and last channel should be within the region, and be zero exactly at the bounds such that $p_0 - ws = b_L$ and $p_0 + s(N - 1) + ws = b_H$. Solving for p_0 and s gives

$$p_0 = b_L + ws \tag{2.11}$$
$$s = (b_H - b_L)/(N - 1 + 2w) \ . \tag{2.12}$$

· **Modular Layout**
The last mode considered is for the case of encoding modular domains like

orientation angles or hue (angular color component of the HSV color space). Consider a modular domain where b_{L} is identified with b_{H} (e.g. $b_{\mathrm{L}} = 0$, $b_{\mathrm{H}} = 2\pi$). The channels wrap around the region bounds such that a channel placed at b_{L} is activated also by values near b_{H}. In order to get N uniformly located channels in the entire domain, use

$$p_0 = b_{\mathrm{L}} \tag{2.13}$$

$$s = (b_{\mathrm{H}} - b_{\mathrm{L}})/N \ . \tag{2.14}$$

2.3.4 Expected Value of Channel Vectors

Assume that a soft histogram has been constructed from a set of points x_i according to (2.2), repeated here for convenience:

$$\mathbf{c}[n] = \frac{1}{I} \sum_i B(x_i - \tilde{x}_n) \ . \tag{2.15}$$

If the samples x_i are drawn from a probability distribution with probability density function (PDF) $p(x)$, the expected value of $\mathbf{c}[n]$ is

$$\mathrm{E}\{\mathbf{c}[n]\} = \int p(x) B(x - \tilde{x}_n) \, \mathrm{d}x \ . \tag{2.16}$$

This means that the channels estimate some linear features of the PDF p. Viewed in another way, since B is symmetric, (2.16) can be written

$$\mathrm{E}\{\mathbf{c}[n]\} = \int p(x) B(\tilde{x}_n - x) \, \mathrm{d}x \ = (p * B)(\tilde{x}_n) \ . \tag{2.17}$$

This means that the expected value of the channel vector is in fact the PDF p, convolved with B and sampled at the channel centers. This relationship was derived in [25] in connection with second order B-spline decoding, which will be reviewed in Sect. 2.4.2.

Channel vectors are very much related to *kernel density estimation* (KDE) [12]. In fact, the channel vector can be seen as a regularly sampled kernel density estimate. The main difference is that this sampling is relatively coarse in the channel case. The purpose of KDE is to estimate a continuous density function, while we most of the time are satisfied with the channel vector. In Sect. 2.5 however, we will see how a continuous density function can be obtained from a channel vector in a way which is different from a standard kernel density estimate.

2.4 Decoding

There are several decoding methods [25, 30] capable of perfect reconstruction of a single encoded value x. The quality of a decoding can be measured in terms of quantization effects, i.e. the dependence of the absolute values of the encoded samples relative to the channel centers. When only a single point is encoded, it is

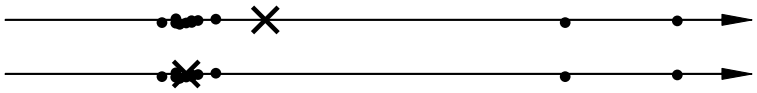

Figure 2.6: Illustration of how outliers can affect the mean value of a set of points. Top: Mean value. Bottom: Robust mean value.

always possible to find the exact value of the point, but when a number of points are encoded, as in Fig. 2.3, the detected mode might be slightly shifted towards or away from the closest channel center. An efficient method for decoding channel vectors based on second-order B-splines called *virtual shift decoding* was introduced in [25] and is briefly reviewed in this section. Understanding this method requires some knowledge about robust estimation.

2.4.1 Robust Estimation

A robust mean value is a mean value which is insensitive to outliers. A regular mean value of a number of points x_i can be seen as the point minimizing a quadratic error function:

$$\mu = \arg\min_x \frac{1}{I} \sum_i (x_i - x)^2 = \frac{1}{I} \sum_i x_i \ . \tag{2.18}$$

The mean value is the point x where the sum-of-squares error (distance) to all input points is minimal. A problem with this is that the presence of a few outliers can change the mean value dramatically, as illustrated in in Fig. 2.6. In order to be robust against outliers, we can change (2.18) to

$$\mu = \arg\min_x \frac{1}{I} \sum_i \rho(x_i - x) \ . \tag{2.19}$$

where $\rho(x)$ is a robust error norm, i.e. a function that looks like a quadratic function close to $x = 0$, but which saturates for large values of x, as illustrated in Fig. 2.7. The exact shape of this error norm is usually not very important, as long as these properties are fulfilled [13]. The function we are minimizing is referred to as the risk function $\mathcal{E}$:

$$\mathcal{E}(x) = \frac{1}{I} \sum_i \rho(x_i - x) \ . \tag{2.20}$$

For x near the true cluster of Fig. 2.7, all inliers fall within the quadratic part of the error norm, and the outliers fall within the constant part. The value and minimum of the error function is now independent of the exact position of the outliers. Only the position of the inliers and the *number* of outliers is significant. The minimization problem (2.19) looks for an x producing a low number of outliers *and* a centralized position among the inliers.

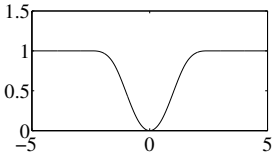

Figure 2.7: The error norm implicitly used in second-order B-spline decoding.

The expected value of the risk function can be obtained by considering a random variable ξ distributed with density $p(\xi)$ and a fixed x. We then have that

$$\mathrm{E}\left\{\mathcal{E}(x)\right\} = \int \rho(x-\xi)p(\xi)\,\mathrm{d}\xi \;= (\rho * p)(x) \;\; . \tag{2.21}$$

2.4.2 Virtual Shift B-spline Decoding

We now consider channel decoding as an instance of robust estimation. The concept of spline interpolation and B-splines is reviewed in Appendix B and is a prerequisite for this section.

Let $\mathbf{c}$ be the soft histogram of samples x_i, constructed using the second-order B-spline as basis function B. For simplicity, assume that the channel centers are located at the positive integers. We want to find a robust mean of the encoded samples by finding the minimum of the risk function

$$\mathcal{E}(x) = \frac{1}{I}\sum_i \rho(x_i - x) \;\; , \tag{2.22}$$

where ρ is some robust error norm. It turns out that an error norm ρ with derivative

$$\rho'(x) = B(x-1) - B(x+1) \;\; , \tag{2.23}$$

leads to an efficient method. This is the error norm shown in Fig. 2.7. To find extrema of $\mathcal{E}(x)$, we seek zeros of the derivative

$$\mathcal{E}'(x) = \frac{1}{I}\sum_i \rho'(x_i - x) = \frac{1}{I}\sum_i B(x_i - x - 1) - B(x_i - x + 1) \;\; . \tag{2.24}$$

We now construct a new set of coefficients $c'_n = c_{n+1} - c_{n-1}$ and have that

$$\begin{aligned} c'_n &= c_{n+1} - c_{n-1} = \\ &= \frac{1}{I}\sum_i B(x_i - (n+1)) - \frac{1}{I}\sum_i B(x_i - (n-1)) = \\ &= \mathcal{E}'(n) \;\; . \end{aligned} \tag{2.25}$$

This means that the sequence c'_n is actually the derivative of the error function sampled at the integers. To find the zero crossings of $\mathcal{E}'$, we can construct a continuous function $\tilde{\mathcal{E}}'$ from c'_n using second-order B-spline interpolation as described in

Appendix B. The interpolated $\tilde{\mathcal{E}}'$ is then a piecewise second-order polynomial, and the exact position of its zero crossings can be determined analytically by solving a second-order polynomial equation.

In practice, a recursive filtering is used to interpolate between the c'-coefficients, and the analytical solution for the zero crossings is only determined at positions where the original channel encoding has large values from the beginning, which leads to a computationally efficient method. I will not go into more detail about this, but refer to [25].

2.5 Continuous Reconstruction

In previous sections, we have studied the relationship between the channel vector and density functions, and have seen how peaks of the underlying PDF can be detected. Here, we will go one step further and attempt to reconstruct the entire underlying continuous PDF from a channel vector. From these continuous reconstructions we can extract modes, and in Sect. 2.6 some properties of these modes are compared to those detected by the previous virtual shift decoding.

Obtaining accurate peaks is not the sole purpose of studying these continuous reconstructions. They can also be used to help us derive the channel vector equivalent of multiplying two PDFs. This lets us use channel vectors for message passing in Bayesian networks, which will be treated in Chapter 6.

Reconstructing a continuous distribution from a finite-dimensional channel vector is clearly an underdetermined problem, and some regularization has to be used. The natural regularizer for density functions is the *entropy*, which measures the information content in a distribution, such that the distribution with maximal entropy is the one *least committed to the data*, or *containing a minimum amount of spurious information* [9].

The maximum entropy solution turns out to be expensive to compute. In Sect. 2.5.3, a computationally more efficient but less statistically motivated linear method for density reconstruction is presented.

2.5.1 Problem Formulation

Using the *Maximum Entropy Method* (MEM), the problem becomes the following: Given a channel vector $\mathbf{c}$, find the distribution p that maximizes the (differential) entropy

$$H(p) = -\int_{-\infty}^{\infty} p(x) \log p(x) \, \mathrm{d}x \tag{2.26}$$

under the constraints

$$\int_{-\infty}^{\infty} p(x) B(x - \tilde{x}_n) \, \mathrm{d}x = \mathbf{c}[n], \quad 1 \leq n \leq N \tag{2.27}$$

$$\int_{-\infty}^{\infty} p(x) \, \mathrm{d}x = 1 \ . \tag{2.28}$$

The first set of constraints is motivated by (2.16). Using variational calculus, it can be shown that the solution is of the form [9, 47]

$$p(x) = k \exp\left(\sum_{n=1}^{N} \lambda_n B(x - \tilde{x}_n)\right) \quad , \tag{2.29}$$

where k and the λ_n's are determined by the constraints (2.27)-(2.28). This is an example of a probability density in the *exponential family* [12]. In the exponential family framework, the parameters λ_n are called *natural parameters* (or *exponential parameters*), and the channel vector elements are the *mean parameters* of the distribution. The mean parameters are also a *sufficient statistic* for the exponential parameters, meaning that they capture all information from a sample set $\{x_i\}_i$ needed for estimating the λ_n's.

In general, there are often analytical solutions available for estimating mean parameters from exponential parameters, but in this case I am not aware of a closed-form solution. Instead, we have to resort to numerical methods like the Newton method.

2.5.2 Newton Method

To make the notation more compact, I introduce a combined notation for the constraints (2.27)-(2.28) by defining feature functions f_n and residuals

$$f_n(x) = B(x - \tilde{x}_n) \quad \text{for } 1 \le n \le N \tag{2.30}$$

$$f_{N+1}(x) \equiv 1 \tag{2.31}$$

$$\mathbf{d} = \begin{bmatrix} \mathbf{c} \\ 1 \end{bmatrix} \tag{2.32}$$

$$r_n = \int_{-\infty}^{\infty} p(x) f_n(x)\, \mathrm{d}x \; - d_n \quad \text{for } 1 \le n \le N+1 \ . \tag{2.33}$$

In this notation, (2.29) becomes

$$p(x) = \exp\left(\sum_{n=1}^{N+1} \lambda_n f_n(x)\right) \quad , \tag{2.34}$$

and the problem to solve is $\mathbf{r} = 0$. Note that the factor k from (2.29) is replaced by λ_{N+1}. Let us now apply a Newton method on this system. Differentiating $p(x)$ with respect to λ_n gives

$$\frac{\mathrm{d}p(x)}{\mathrm{d}\lambda_n} = f_n(x) p(x) \ . \tag{2.35}$$

In differentiating r_i with respect to λ_j, we can exchange the order of differentiation and integration to obtain

$$\frac{\mathrm{d}r_i}{\mathrm{d}\lambda_j} = \int_{-\infty}^{\infty} \frac{\mathrm{d}p(x)}{\mathrm{d}\lambda_j} f_i(x)\,\mathrm{d}x = \\ = \int_{-\infty}^{\infty} f_i(x) f_j(x) p(x)\,\mathrm{d}x \quad . \tag{2.36}$$

The Jacobian then becomes

$$\mathbf{J} = \left[\frac{\mathrm{d}r_i}{\mathrm{d}\lambda_j}\right]_{ij} = \left[\int_{-\infty}^{\infty} f_i(x) f_j(x) p(x)\,\mathrm{d}x\right]_{ij} \quad . \tag{2.37}$$

The update in the Newton method is $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \mathbf{s}$, where the increment $\mathbf{s}$ in each step is obtained by solving the equations $\mathbf{J}\mathbf{s} = -\mathbf{r}$. When evaluating the integrals in (2.37) and (2.32), I use the exponential form (2.34) for p using the current estimate of the λ_n's.

Since our feature functions f_n are localized functions with compact support, most of the time f_i and f_j will be non-zero at non-overlapping regions such that $f_i f_j \equiv 0$, making $\mathbf{J}$ band-shaped and sparse, and hence relatively cheap to invert. The main workload in this method is in the evaluation of the integrals, both for $\mathbf{J}$ and $\mathbf{r}$. These integrals are non-trivial and must be evaluated numerically. The entire method usually requires around 10 iterations.

2.5.3 Minimum-Norm Reconstruction

A density function is by definition positive. If this requirement is relaxed, we can replace the maximum-entropy regularization with a *minimum-norm* (MN) regularization, which permits the use of linear methods for the reconstruction. This is not motivated from a statistical point of view, and may even lead to a negative density function $p(x)$, but is included for comparison since it is the simplest way of obtaining continuous reconstructions.

For the derivations, we consider the vector space $L_2(\mathbb{R})$ of real square-integrable functions [19], with scalar product

$$\langle f, g\rangle = \int_{-\infty}^{\infty} f(x) g(x)\,\mathrm{d}x \tag{2.38}$$

and norm $\|f\|^2 = \langle f, f\rangle$. The minimum norm reconstruction problem is now posed as

$$p_* = \arg\min_p \|p\| \quad \text{subject to } \langle p, f_n\rangle = d_n \quad \text{for } 1 \le n \le N+1 \ , \tag{2.39}$$

where the feature functions f are defined as in the previous section. Reconstructing p from the d_n's resembles the problem of reconstructing a function from a set of frame coefficients [69]. The reconstruction p_* of minimum norm is in the space $Q_1 = \mathrm{span}\{f_1, \ldots, f_{N+1}\}$, which can easiest be seen by decomposing p_* into $p_* =$

$q_1 + q_2$, where $q_1 \in Q_1$ and $q_2 \in Q_1^\perp$. Since $q_1 \perp q_2$, we have $\|p_*\|^2 = \|q_1\|^2 + \|q_2\|^2$. But $q_2 \perp f_n$ for all feature functions f_n, so q_2 does not affect the constraints and must be zero in order to minimize $\|p_*\|^2$. Hence $p_* = q_1 \in Q_1$, which implies that p_* can be written as

$$p_*(x) = \sum_n \alpha_n f_n(x) \ . \tag{2.40}$$

To find the set of α_n's making p_* fulfill the constraints in (2.39), we write

$$d_n = \langle p_*, f_n \rangle = \left\langle \sum_k \alpha_k f_k \ , \ f_n \right\rangle = \sum_k \alpha_k \langle f_k, f_n \rangle \ , \tag{2.41}$$

giving the α_n's as a solution of a linear system of equations. In matrix notation, this system becomes

$$\boldsymbol{\Phi}\boldsymbol{\alpha} = \mathbf{d} \ , \tag{2.42}$$

where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_{N+1}]^T$ and $\boldsymbol{\Phi}$ is the Gram matrix $\boldsymbol{\Phi} = [\langle f_i, f_j \rangle]_{ij}$. Note that since $\boldsymbol{\Phi}$ is independent of our feature values $\mathbf{d}$, it can be computed analytically and inverted once and for all for a specific problem class. The coefficients $\boldsymbol{\alpha}$ can then be obtained by a single matrix multiplication.

A theoretical justification of using the minimum norm to reconstruct density functions can be given in the case where p shows just small deviations from a uniform distribution, such that $p(x)$ is defined on $[0, K]$, and $p(x) \approx K^{-1}$. In this case, we can approximate the entropy by linearizing the logarithm. The first terms of the Taylor expansion around K^{-1} gives $\log p(x) \approx \log K^{-1} + (p(x) - K^{-1})/K^{-1}$, and

$$\begin{aligned} H(p) &= -\int_0^K p(x) \log p(x) \, \mathrm{d}x \\ &\approx -\int_0^K p(x) \big(\log K^{-1} + Kp(x) - 1 \big) \, \mathrm{d}x \ = \\ &= -(\log K^{-1} - 1) \int_0^K p(x) \, \mathrm{d}x \ - K \int_0^K p(x)^2 \, \mathrm{d}x \ . \end{aligned} \tag{2.43}$$

Since $\int_0^K p(x) \, \mathrm{d}x \ = 1$, maximizing this expression is equivalent to minimizing $\int_0^K p(x)^2 \, \mathrm{d}x \ = \|p\|^2$. This shows that the closer $p(x)$ is to being uniform, the better results should be expected from the minimum-norm approximation.

2.6 Decoding and Reconstruction Experiments

In this section, I experimentally analyze the behavior of the continuous reconstruction methods and the B-spline decoding from Sect. 2.4.2. For the numerical evaluation of the integrals in the MEM method, the PDFs were discretized using 400 samples per unit distance. As a channel coefficient $\mathbf{c}[n]$ gets closer to zero,

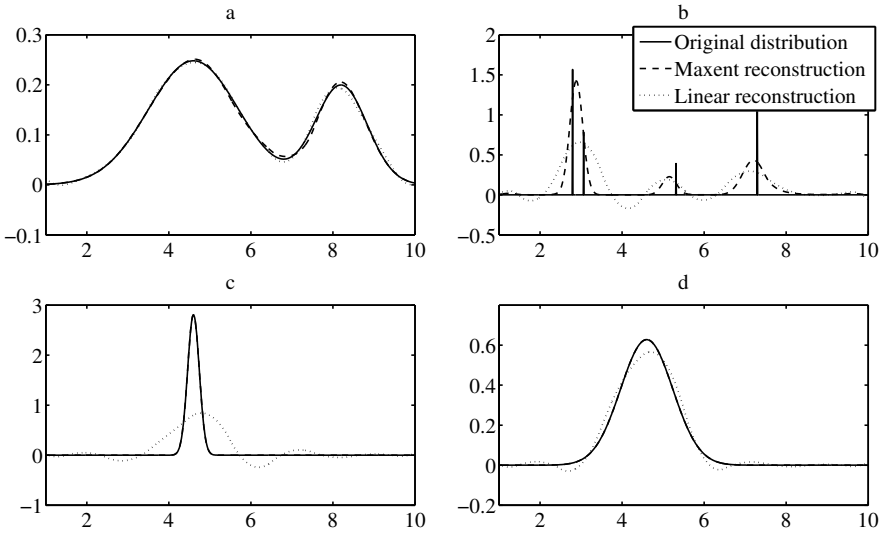

Figure 2.8: The MEM and MN reconstruction of (a) Sum of two Gaussians, (b) 4 Diracs of different weights, (c-d) Single Gaussians with different variance.

the corresponding λ_n from the MEM tends towards $-\infty$, leading to numerical problems. To stabilize the solution in these cases, a small background DC level was introduced (ϵ-regularization).

2.6.1 Qualitative Behavior

In Fig. 2.8, the qualitative behavior of the MEM and MN reconstructions is examined. The feature vector $\mathbf{d}$ was calculated for some different distributions with the channel centers located at the integers. The two Gaussians (c-d) were reconstructed almost perfectly using MEM, but rather poorly using MN. In (b), the two leftmost peaks were mixed together, but even the rightmost peaks were close enough to influence each other, and all methods failed to find the exact peak locations. For the slowly varying continuous distribution (a), both methods performed quite well.

2.6.2 Quantitative Behavior

To make a more quantitative comparison, I focused on two key properties; the *discrimination threshold* [84] and the *quantization effect* [24] of channel decoding. These properties can be measured both on the continuous reconstructions and on the virtual shift B-spline decoding.

Recall that the virtual shift decoding does not look for a maximum of the PDF directly, but rather for a minimum of an error function equivalent to the PDF convolved with some robust error norm. In order to estimate a similar error function minimum from the continuous reconstructions, our estimated p should

Method		$\Delta x_0 = 0$	$\Delta x_0 = 0.5$
MEM	p	0.34	0.57
	$B * p$	0.53	0.71
	$B_{\mathrm{VS}} * p$	1.00	1.00
MN	p	0.57	0.71
	$B * p$	0.64	0.81
	$B_{\mathrm{VS}} * p$	0.95	1.00
Virtual Shift		1.00	1.00

Table 2.1: Discrimination thresholds.

likewise be convolved with some kernel prior to the maximum detection.

From (2.23), a robust error norm ρ is implicitly defined up to an additive constant, which can be selected arbitrarily. Let $B_{\mathrm{VS}} = (\max_x \rho(x)) - \rho(x)$. This is the kernel implicitly used in the virtual shift decoding. For all continuous reconstruction experiments, peaks were detected from the raw reconstruction p as well as from $B * p$ (with the second-order B-spline kernel B) and from $B_{\mathrm{VS}} * p$. Note that B_{VS} is wider than B.

To measure the discrimination threshold, two values $x_0 \pm d$ were encoded. The discrimination threshold in this context is defined as the minimum value of d which gives two distinct peaks in the reconstruction. As the background DC level increases, the distribution becomes closer to uniform, and the performance of the MEM and MN methods is expected to become increasingly similar. To keep this DC level low but still avoid numerical problems, we chose a regularization level corresponding to 1% of the entire probability mass. The discrimination threshold was calculated for both reconstruction methods and the different choices of convolution kernels, and the results are summarized in Table 2.1. These values were evaluated both for x_0 at a channel center ($\Delta x_0 = 0$) and in the middle between two centers ($\Delta x_0 = 0.5$).

With the *quantization effect*, we mean the fact that two distributions p differing only in shift relative to the grid of basis functions are reconstructed differently. To measure this effect, two distinct impulses of equal weight located at $x_0 \pm d$ with d below the discrimination threshold were encoded. Ideally, the peak of the reconstructed distribution would be located at x_0, but the detected peak m actually varies depending on the location relative to the channel grid. In Fig. 2.9, the difference between m and x_0 is plotted against the offset $z = x_0 - \tilde{x}$ (the position of x_0 relative to a channel center $\tilde{x}$). Figure 2.10 shows the quantization effect for the virtual shift decoding algorithm from Sect. 2.4. Note the different scales of the plots. Also note that as the error becomes small enough, the discretization of $p(x)$ becomes apparent.

Achieving a quantization error as low as 1% of the channel spacing in the best case in a very nice result, but keep in mind that this error is only evaluated for the special case of two Diracs. In general, the results may be dependent on the exact distribution of the samples. It is not obvious how to measure this effect in a more general way, without assuming some specific form of the distribution.

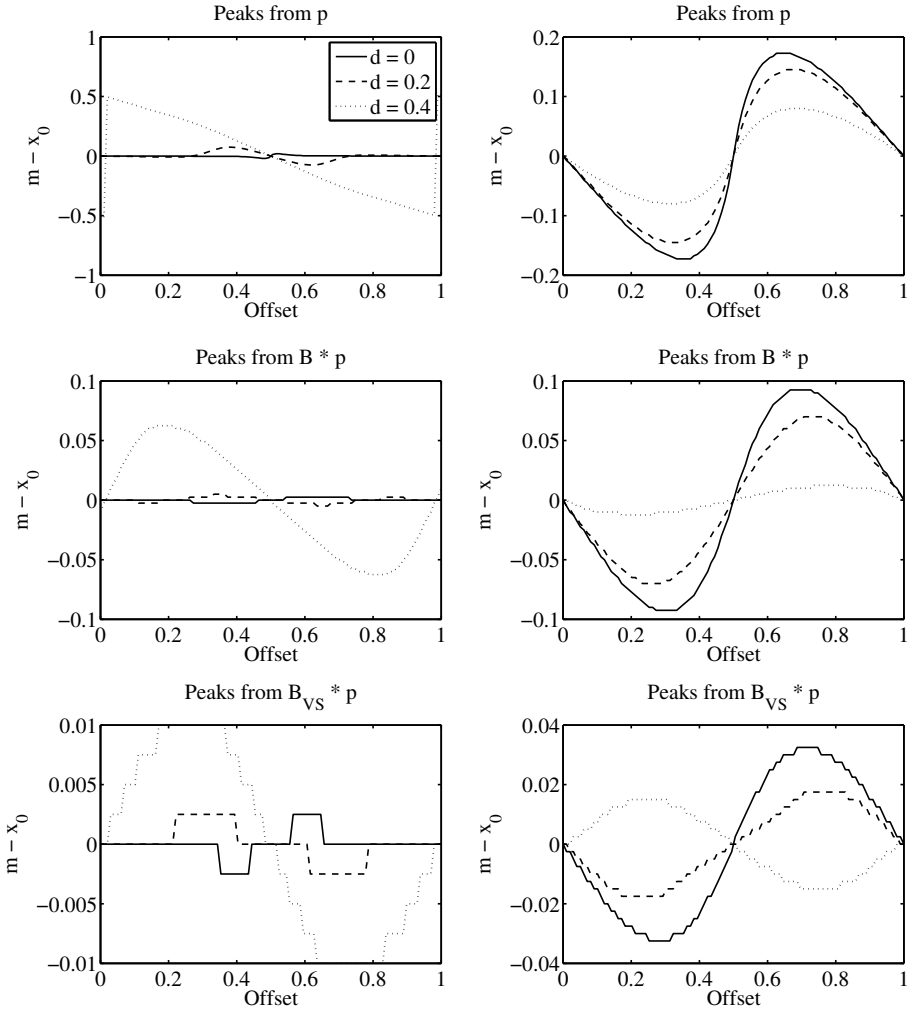

Figure 2.9: The quantization effect for continuous reconstructions. Left: Maximum entropy. Right: Minimum norm.

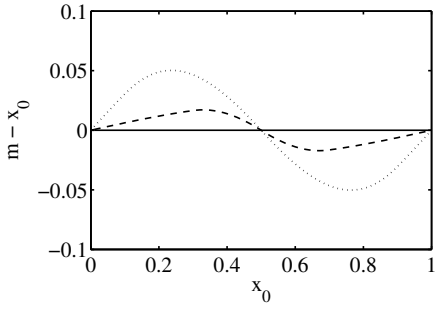

Figure 2.10: The quantization effect for the B-spline decoding.

2.7 Discussion

This chapter has given an overview of the channel coding idea and discussed the possibility of reconstructing continuous density functions from the channel vectors. The maximum entropy reconstruction is theoretically appealing, since it provides a natural way of reconstructing density functions from partial information. In most applications however, the exact shape of the density function is not needed, since we are merely interested in locating modes of the distribution with high accuracy. Efficient linear methods for such mode detection can be constructed, but generally perform worse than the MEM method in terms of quantization error and discriminating capabilities.

In general, for wider convolution kernels in the mode extraction, we get better position invariance but worse discriminating capabilities. Thus, there is a trade-off between these two effects. The possibility of achieving as little quantization error as $\pm 1\%$ of the channel spacing is promising for using channel-based methods in high-dimensional spaces. In e.g. Hough-like ellipse detection, the vote histogram would be 5-dimensional, and keeping a small number of bins in each dimension is crucial. Unfortunately, it is hard to turn the MEM reconstruction into a practical and efficient mode seeking algorithm due to the computational complexity and the necessity to evaluate integrals numerically.

Chapter 3

Channel-Coded Feature Maps

...where we visit the spatio-featural space for the first time. In this space we meet our old friend the SIFT descriptor, but also his brothers and cousins that you perhaps have never seen before. We get the pleasure of computing derivatives of the whole family, and begin to suspect that dressing up as piecewise polynomials might be the latest fashion.

Channel-coded feature maps (CCFMs) are a general way of representing image features like color and gradient orientation. The basic idea is to create a soft histogram of spatial positions and feature values, as illustrated in Fig. 3.1. This is obtained by viewing an image as a number of points in the joint space of spatial position and feature values, the *spatio-featural* space, and channel coding these points into a soft histogram. The most well-known variation of this type of representations is the SIFT descriptor [66], where position and edge orientation are encoded into a three-dimensional histogram. SIFT uses linear interpolation to assign each pixel to several bins, while channel-coded feature maps can be constructed using any basis function.

If we wish to adjust the position in an image at which the CCFM is computed e.g. in order to track an object in time, the derivatives of the CCFM with respect to scale change, rotation and translation are needed. These derivatives are more well-behaved if the basis functions are smooth.

Apart from the SIFT descriptor, other forms of feature histograms are rather common in object recognition and tracking. The *shape contexts* used in [8] are log-polar histograms of edge point positions. Since they are only used as a cue for point matching, no attempt of computing their derivatives with respect to image transformations is made. In [16], objects are tracked using single (global) color histograms, weighted spatially with a smooth kernel. This can be seen as using a channel-coded feature map with only a single spatial channel. The gradient-based optimization is restricted to translations - scale changes are handled by testing a number of discrete scales exhaustively, and rotation is not handled at all. In [94],

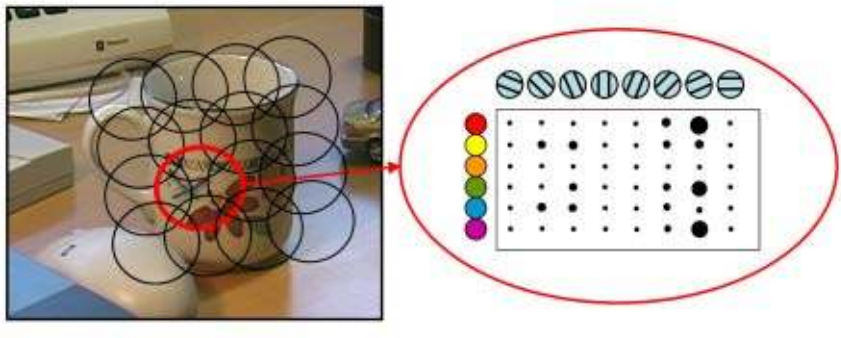

Figure 3.1: Illustration of a Channel-coded feature map. For each spatial channel, there is a soft histogram of chromacity and orientation, giving in total a 4D histogram.

orientation histograms and downsampled color images are constructed efficiently by box filters using integral images [92]. This is possible since rotation of the tracked object is not considered. Usually when SIFT features are used in tracking (e.g. in [82]), the descriptors are computed at fixed positions without attempting to fine-tune the similarity parameters. The channel-coded feature maps generalize all these approaches, allow for arbitrary basis functions, and support derivatives with respect to rotation, translation and scale changes.

In this chapter I describe the main idea of channel-coded feature maps, differentiate them with respect to similarity transforms, and show how to apply the theory in a tracking experiment.

3.1 Channel-Coded Feature Maps

3.1.1 Definition and Notation

A channel-coded feature map can be constructed using an arbitrary number of features. You can think about color and local orientation for a concrete example. First, let $\{\mathbf{x}_i\}_i$ be a set of points in a spatio-featural space $\mathcal{F}$, where each $\mathbf{x}_i$ corresponds to one image pixel. The first two elements of each $\mathbf{x}_i$ are the spatial pixel positions, denoted as $\mathbf{u}_i = [u_i, v_i]^\mathrm{T}$, and the rest are feature values, denoted as $\mathbf{z}$. Since the feature values are a function of the image coordinate, we can write

$$\mathbf{x}_i = \begin{pmatrix} \mathbf{u}_i \\ \mathbf{z}(\mathbf{u}_i) \end{pmatrix} \quad . \tag{3.1}$$

Let $\tilde{\mathbf{x}} = [\tilde{\mathbf{u}}^\mathrm{T}, \tilde{\mathbf{z}}^\mathrm{T}]^\mathrm{T} \in \mathcal{F}$ be a channel center. As in Chapter 2, we can without loss of generality assume that these centers are unit-spaced, since that is only a matter of changing the coordinate system. Furthermore, we let $\mathbf{u} = [0, 0]^\mathrm{T}$ be the center of the encoded image or patch (see Fig. 3.2). The channel-coded feature map is

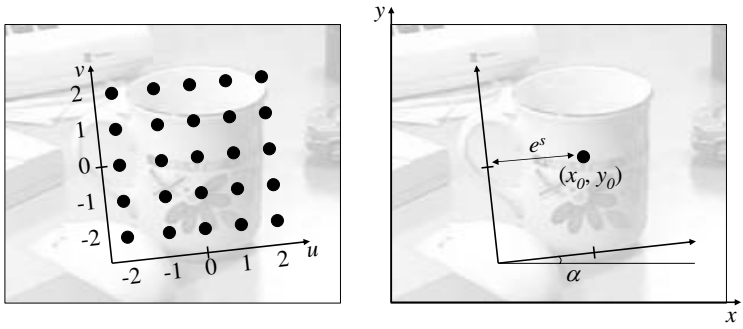

Figure 3.2: Left: Intrinsic channel coordinate system. The dots indicate channel centers. Right: Similarity parameters governing the location of the patch in the image.

now a multi-dimensional array

$$\mathbf{c}[\tilde{\mathbf{x}}] = \frac{1}{I}\sum_i w_i B(\mathbf{x}_i - \tilde{\mathbf{x}}) = \frac{1}{I}\sum_i w_i B(\mathbf{u}_i - \tilde{\mathbf{u}}, \mathbf{z}(\mathbf{u}_i) - \tilde{\mathbf{z}}) \ . \tag{3.2}$$

The weights w_i can be selected based on the confidence of the feature extraction, such that e.g. homogeneous regions get a low weight since the orientation estimates in these regions are unreliable.

When working with derivatives with respect to image transformations, it will be more convenient to use a continuous formulation. Assume that the image coordinates $\mathbf{u}_i$ are taken from a regular grid. As this grid gets finer and finer, the sums above approach the integrals

$$\mathbf{c}[\tilde{\mathbf{x}}] = \int w(\mathbf{u}) B(\mathbf{u} - \tilde{\mathbf{u}}, \mathbf{z}(\mathbf{u}) - \tilde{\mathbf{z}}) \ \mathrm{d}\mathbf{u} = \int w(\mathbf{u}) B(\mathbf{x} - \tilde{\mathbf{x}}) \ \mathrm{d}\mathbf{u} \ . \tag{3.3}$$

3.1.2 Motivation

Creating a channel-coded feature map from an image is a way of obtaining a coarse spatial resolution while maintaining much information at each position. For example, we can represent the presence of multiple orientations in a region without averaging them together. A 128×128 grayscale image can be converted to a 12×12 patch with 8 layers, where each layer represents the presence of a certain orientation. This is advantageous for methods adjusting the spatial location of an image patch based on a local optimization in the spirit of the KLT tracker (see Sect. 3.3). The low spatial resolution increases the probability of reaching the correct optimum of the energy function, while having more information at each pixel improves the robustness and accuracy.

If non-overlapping box functions are used as basis functions, we get a regular hard histogram in spatio-featural space. If we use local edge orientation as a single feature, create a binary weight w_i by thresholding the gradient magnitude at 10%

of the maximal possible value, and use the first order (linear) B-spline as basis function, we get something similar to the SIFT descriptor [66]. By increasing the overlap and smoothness of the basis functions, we expect to get a smoother behavior.

The low spatial resolution and the smoothness of the basis functions make it more likely that the view representation transforms smoothly between different views of an object, which also makes it suitable for view interpolation. This idea will be explored further in Chapter 10.

3.1.3 Choice of Features

Channel-coded feature maps can be constructed from different sets of features. The primary examples in the thesis will be local orientation and color.

Local orientation can be used in different ways. First note that the local orientation information is only significant close to edges in an image. In large homogeneous regions, the orientation is usually very noisy and should perhaps not be included with equal weight in the channel coding. Consider using only the gradient direction as a local orientation measure. One option that comes to mind is to use the gradient magnitude as weights w_i in (3.2). This causes pixels with less distinct structure to contribute less to the encoding. However, often the exact value of the gradient magnitude is not a significant feature.

If color is used as a feature, any color space can be used. For example, in order to be invariant to changes in illumination, the hue and saturation channels of the HSV representation could be used. However, the hue component is very unstable for dark black and bright white colors. A more detailed discussion about practical considerations in the choice of features is given in Sect. 12.2.

3.2 Derivatives of Channel Coded Feature Maps

One issue in applications like object pose estimation, tracking and image registration is the fine-tuning of similarity parameters. The problem is to find a similarity transform that maps one image (or image patch) to another image in a way that minimizes some cost function. One way to solve this is to encode the first image or patch into a *target CCFM* $\mathbf{c}_0$. Let $\mathbf{f}(s, \alpha, x_0, y_0)$ be a function that extracts a CCFM from the second image (the query image), from a patch located at (x_0, y_0) with radius e^s and rotation α (see Fig. 3.2). We then look for the parameters that make $\mathcal{E} = \|\mathbf{f}(s, \alpha, x_0, y_0) - \mathbf{c}_0\|^2$ minimal. In Sect. 3.3, this formulation will be used for tracking, and in Chapter 11, this will be one component of a view-based pose estimation method. In order to minimize $\mathcal{E}$ with a local optimization, we need the derivatives of $\mathbf{f}$ with respect to the similarity parameters.

3.2.1 Derivation

The starting point for this derivation is the definition in (3.3). We focus on a certain channel coefficient $c = \mathbf{c}[\tilde{\mathbf{x}}]$ for a given fixed $\tilde{\mathbf{x}}$. To make the notation more

compact, we define

$$h(\mathbf{x}) = B(\mathbf{x} - \tilde{\mathbf{x}}) = B(\mathbf{u} - \tilde{\mathbf{u}}, \mathbf{z}(\mathbf{u}) - \tilde{\mathbf{z}}) \ . \tag{3.4}$$

Furthermore, we ignore the weight function $w(\mathbf{u})$ for a moment. This produces a shorter version of (3.3) as

$$c = \int h(\mathbf{u}, \mathbf{z}(\mathbf{u}))\, \mathrm{d}\mathbf{u} \ . \tag{3.5}$$

Since the expressions get rather lengthy anyway, this will be more convenient to work with. The weights will be considered again in Sect. 3.2.2. Let us now see what happens when the channel grid is rotated, scaled and translated according to

$$c = \int h(\mathbf{u}, \mathbf{z}(\mathbf{A}\mathbf{u} + \mathbf{b}))\, \mathrm{d}\mathbf{u} \ , \tag{3.6}$$

where

$$\mathbf{A} = e^s \mathbf{R} = e^s \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} b_u \\ b_v \end{pmatrix} \ . \tag{3.7}$$

Positive b_u, b_v, s, α correspond to a positive shift and rotation and an increase of size of the channel grid in the (u, v) coordinate system (Fig. 3.2). Substituting $\mathbf{u}' = \mathbf{A}\mathbf{u} + \mathbf{b}$ gives $\mathbf{u} = \mathbf{A}^{-1}(\mathbf{u}' - \mathbf{b})$ and

$$c = |\mathbf{A}^{-1}| \int h(\mathbf{A}^{-1}(\mathbf{u}' - \mathbf{b}), \mathbf{z}(\mathbf{u}'))\, \mathrm{d}\mathbf{u}' \ . \tag{3.8}$$

We can now rename $\mathbf{u}'$ as $\mathbf{u}$ again. Note that $|\mathbf{A}^{-1}| = e^{-2s}|\mathbf{R}^{-1}| = e^{-2s}$, where $|\mathbf{R}^{-1}| = 1$ since $\mathbf{R}$ is orthonormal. This gives

$$c = e^{-2s} \int h(\mathbf{A}^{-1}(\mathbf{u} - \mathbf{b}), \mathbf{z}(\mathbf{u}))\ \mathrm{d}\mathbf{u} \ . \tag{3.9}$$

We want to differentiate (3.9) with respect to α, s, b_u, b_v and start with α. We can replace the order of the integration and differentiation and get

$$\frac{\mathrm{d}c}{\mathrm{d}\alpha} = e^{-2s} \int \frac{\mathrm{d}}{\mathrm{d}\alpha} [h(\ldots)]\ \mathrm{d}\mathbf{u} = e^{-2s} \int h'_{\mathbf{u}}(\ldots) \frac{\mathrm{d}\mathbf{A}^{-1}}{\mathrm{d}\alpha} (\mathbf{u} - \mathbf{b})\, \mathrm{d}\mathbf{u} \ , \tag{3.10}$$

where

$$\frac{\mathrm{d}\mathbf{A}^{-1}}{\mathrm{d}\alpha} = e^{-s} \begin{bmatrix} -\sin\alpha & \cos\alpha \\ -\cos\alpha & -\sin\alpha \end{bmatrix} \tag{3.11}$$

$$h'_{\mathbf{u}} = [h'_u,\ h'_v] \ . \tag{3.12}$$

For compactness, the arguments to h and its derivatives have been left out. These arguments are always as in (3.9). The differentiation with respect to $\mathbf{b}$ proceeds similarly. We get

$$\frac{\mathrm{d}c}{\mathrm{d}\mathbf{b}} = e^{-2s} \int \frac{\mathrm{d}}{\mathrm{d}\mathbf{b}} [h(\ldots)]\ \mathrm{d}\mathbf{u} = -e^{-2s} \int h'_{\mathbf{u}}(\ldots) \mathbf{A}^{-1}\ \mathrm{d}\mathbf{u} \ . \tag{3.13}$$

In differentiating with respect to s, the product rule gives us

$$\begin{aligned}\frac{\mathrm{d}c}{\mathrm{d}s} =& \frac{\mathrm{d}(e^{-2s})}{\mathrm{d}s}\int h(\ldots)\,\mathrm{d}\mathbf{u} \;+\; e^{-2s}\int \frac{\mathrm{d}}{\mathrm{d}s}\left[h(\ldots)\right]\mathrm{d}\mathbf{u} = \\ =& -2e^{-2s}\int h(\ldots)\,\mathrm{d}\mathbf{u} \;+e^{-2s}\int h'_{\mathbf{u}}(\ldots)\frac{\mathrm{d}\mathbf{A}^{-1}}{\mathrm{d}s}(\mathbf{u}-\mathbf{b})\,\mathrm{d}\mathbf{u} = \\ =& -e^{-2s}\int 2h(\ldots)+h'_{\mathbf{u}}(\ldots)\mathbf{A}^{-1}(\mathbf{u}-\mathbf{b})\,\mathrm{d}\mathbf{u}\quad . \end{aligned} \tag{3.14}$$

If we evaluate these derivatives for $s=0, \alpha=0, \mathbf{b}=0$, we get $\mathbf{A}^{-1}=\mathbf{I}$, and (3.10), (3.13) and (3.14) become

$$\frac{\mathrm{d}c}{\mathrm{d}b_u} = -\int h'_u(\mathbf{u},\mathbf{z}(\mathbf{u}))\,\mathrm{d}\mathbf{u} \tag{3.15}$$

$$\frac{\mathrm{d}c}{\mathrm{d}b_v} = -\int h'_v(\mathbf{u},\mathbf{z}(\mathbf{u}))\,\mathrm{d}\mathbf{u} \tag{3.16}$$

$$\frac{\mathrm{d}c}{\mathrm{d}s} = -\int 2h(\mathbf{u},\mathbf{z}(\mathbf{u})) + uh'_u(\mathbf{u},\mathbf{z}(\mathbf{u})) + vh'_v(\mathbf{u},\mathbf{z}(\mathbf{u}))\,\mathrm{d}\mathbf{u} \tag{3.17}$$

$$\frac{\mathrm{d}c}{\mathrm{d}\alpha} = \int vh'_u(\mathbf{u},\mathbf{z}(\mathbf{u})) - uh'_v(\mathbf{u},\mathbf{z}(\mathbf{u}))\,\mathrm{d}\mathbf{u}\quad . \tag{3.18}$$

3.2.2 Weighted Data

In the previous section, the weights from (3.2) were not considered. By introducing these weights again, the results are similar. Since the weights are defined for each pixel in the feature image, they transform with the features such that (3.6) in the weighted case becomes

$$c = \int h(\mathbf{u},\mathbf{z}(\mathbf{A}\mathbf{u}+\mathbf{b}))w(\mathbf{A}\mathbf{u}+\mathbf{b})\,\mathrm{d}\mathbf{u}\quad . \tag{3.19}$$

After the variable substitution, we have

$$c = |\mathbf{A}^{-1}|\int h(\mathbf{A}^{-1}(\mathbf{u}-\mathbf{b}),\mathbf{z}(\mathbf{u}))w(\mathbf{u})\,\mathrm{d}\mathbf{u}\quad . \tag{3.20}$$

In this expression, the weighting function is independent of the transformation parameters $\alpha, s, \mathbf{b}$ and is left unaffected by the differentiation. The complete expressions for the derivatives in the weighted case are just (3.15)-(3.18) completed with the multiplicative weight $w(\mathbf{u})$ inside the integrals.

3.2.3 Normalization

An option is to normalize the channel vectors using $\tilde{\mathbf{c}} = \mathbf{c}/\|\mathbf{c}\|$, where $\|\cdot\|$ is the L_2 norm. In this case, we should change the derivatives from previous section accordingly. From Appendix A, we have

$$\mathrm{d}\tilde{\mathbf{c}} = \|\mathbf{c}\|^{-1}(\mathbf{I}-\tilde{\mathbf{c}}\tilde{\mathbf{c}}^{\mathrm{T}})\,\mathrm{d}\mathbf{c}\quad . \tag{3.21}$$

We consider first the derivative with respect to rotation and write

$$\mathrm{d}\tilde{\mathbf{c}} = \|\mathbf{c}\|^{-1}(\mathbf{I} - \tilde{\mathbf{c}}\tilde{\mathbf{c}}^{\mathrm{T}})\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}\alpha}\,\mathrm{d}\alpha \quad , \tag{3.22}$$

giving the explicit derivative as

$$\frac{\mathrm{d}\tilde{\mathbf{c}}}{\mathrm{d}\alpha} = \|\mathbf{c}\|^{-1}(\mathbf{I} - \tilde{\mathbf{c}}\tilde{\mathbf{c}}^{\mathrm{T}})\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}\alpha} \quad . \tag{3.23}$$

The derivatives with respect to the other similarity parameters are handled in a similar way.

Another option is to normalize using the L_1 norm. For $\tilde{\mathbf{c}} = (\mathbf{1}^{\mathrm{T}}\mathbf{c})^{-1}\mathbf{c}$, Appendix A gives us

$$\mathrm{d}\tilde{\mathbf{c}} = (\mathbf{1}^{\mathrm{T}}\mathbf{c})^{-1}(\mathbf{I} - \tilde{\mathbf{c}}\mathbf{1}^{\mathrm{T}})\,\mathrm{d}\mathbf{c} \quad , \tag{3.24}$$

which also applies to all four similarity derivatives by changing the differentials to derivatives like above. The pros and cons of different normalizations will be discussed in later sections.

From an implementation perspective, I found it useful to have one function that computes non-normalized CCFMs and derivatives. The normalization is done by a second function which is completely general, accepting simply a vector and a number of derivatives without knowing anything about how this vector was computed or which variables the derivatives are with respect to.

3.2.4 Using the Sum Notation

On a digital computer, images are usually represented as a discrete array of numbers, and when computing the derivatives, the integrals (3.15)-(3.18) are approximated by sums. Returning to our original notation where the channel centers are explicitly written out, the discrete approximations of the derivatives are

$$\frac{\mathrm{d}\mathbf{c}[\tilde{\mathbf{x}}]}{\mathrm{d}b_u} = -\frac{1}{I}\sum_i B'_u(\mathbf{x}_i - \tilde{\mathbf{x}}) \tag{3.25}$$

$$\frac{\mathrm{d}\mathbf{c}[\tilde{\mathbf{x}}]}{\mathrm{d}b_v} = -\frac{1}{I}\sum_i B'_v(\mathbf{x}_i - \tilde{\mathbf{x}}) \tag{3.26}$$

$$\frac{\mathrm{d}\mathbf{c}[\tilde{\mathbf{x}}]}{\mathrm{d}s} = -\frac{1}{I}\sum_i \big(2B(\mathbf{x}_i - \tilde{\mathbf{x}}) + u_i B'_u(\mathbf{x}_i - \tilde{\mathbf{x}}) + v_i B'_v(\mathbf{x}_i - \tilde{\mathbf{x}})\big) \tag{3.27}$$

$$\frac{\mathrm{d}\mathbf{c}[\tilde{\mathbf{x}}]}{\mathrm{d}\alpha} = \frac{1}{I}\sum_i \big(v_i B'_u(\mathbf{x}_i - \tilde{\mathbf{x}}) - u_i B'_v(\mathbf{x}_i - \tilde{\mathbf{x}})\big) \tag{3.28}$$

This is the form that will be most convenient to work with in the next chapter, where an efficient method of constructing CCFMs with derivatives is presented, based on the piecewiseness of B-splines.

3.3 Tracking

3.3.1 Main Algorithm

To give a first application of channel-coded feature maps, we reconsider the example from the introduction of Sect. 3.2 in order to build a patch tracker. From the first frame of an image sequence, we cut out an image patch of a given radius s at some given position (x_0, y_0) and rotation α and create a target CCFM $\mathbf{c}_0$ from this patch. We then want to track this patch through subsequent frames, handling translation as well as rotation and scale changes.

Let $\boldsymbol{\psi} = (s, \alpha, x_0, y_0)$ be a *similarity frame* defining the position, scale and rotation of a patch in an image, and let $\mathbf{f}(\boldsymbol{\psi})$ be a function creating a CCFM from the similarity frame $\boldsymbol{\psi}$ in the next image of the sequence. The tracking must find

$$\boldsymbol{\psi}_* = \arg\min \|\mathbf{f}(\boldsymbol{\psi}) - \mathbf{c}_0\|^2 \ . \tag{3.29}$$

This is a non-linear least-squares problem which can be solved using e.g. a Gauss-Newton method. One interpretation of Gauss-Newton is that it in each iteration replaces $\mathbf{f}(\boldsymbol{\psi})$ with a linearized version $\mathbf{f}(\boldsymbol{\psi}+\mathbf{s}) \approx \mathbf{f}(\boldsymbol{\psi})+\mathbf{J}\mathbf{s}$, where $\mathbf{J}$ is the Jacobian of $\mathbf{f}$. We can then find the optimal update $\mathbf{s}$ by minimizing $\|\mathbf{f}(\boldsymbol{\psi}) + \mathbf{J}\mathbf{s} - \mathbf{c}_0\|^2$. This is a linear least-squares problem with the explicit solution

$$\mathbf{s} = (\mathbf{J}^\mathrm{T}\mathbf{J})^{-1}\mathbf{J}^\mathrm{T}(\mathbf{f}(\boldsymbol{\psi}) - \mathbf{c}_0) \ . \tag{3.30}$$

This requires an initial guess $\boldsymbol{\psi}$ used as the point of linearization. After solving for $\mathbf{s}$, this estimate is refined using $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} + \alpha\mathbf{s}$, where α is the step length.

In order to guarantee convergence, the step length α should be determined by a line search, such that $\mathbf{s}$ merely determines the direction of the update. In practice, a fixed $\alpha < 1$ often works well. Choosing a smaller α can improve robustness but increases the number of iterations required. For a detailed treatment about convergence requirements and line search methods, see [77]. A less rigorous introduction to non-linear optimization can be found in [48]. The entire tracking algorithm is summarized as Alg. 3.1.

Note that the derivatives from previous section are relative to the (u, v) coordinate system. In order to get the derivatives with respect to (x_0, y_0), note that a translation of the channel grid by (b_u, b_v) in the (u, v) system corresponds to a translation (b_x, b_y) in the (x, y) system, with

$$b_u = k\big(\ \cos(\alpha)b_x + \sin(\alpha)b_y\big) \tag{3.31}$$

$$b_v = k\big(-\sin(\alpha)b_x + \cos(\alpha)b_y\big) \ . \tag{3.32}$$

The constant k represents the scale change between the coordinate systems. If the (x, y) system is measured in image pixels, k^{-1} is *pixels per channel-spacing unit*. The chain rule now give the transformed derivatives as

$$\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}b_x} = k\left(\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}b_u}\cos(\alpha) - \frac{\mathrm{d}\mathbf{c}}{\mathrm{d}b_v}\sin(\alpha)\right) \tag{3.33}$$

$$\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}b_y} = k\left(\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}b_u}\sin(\alpha) + \frac{\mathrm{d}\mathbf{c}}{\mathrm{d}b_v}\cos(\alpha)\right) \ . \tag{3.34}$$

Algorithm 3.1 Tracking using channel-coded feature maps.

Inputs:
- Target CCFM $\mathbf{c}_0$
- Estimated similarity frame $\boldsymbol{\psi}_{k-1}$ from previous image
- New image $\mathbf{I}_k$.

Initialize $\boldsymbol{\psi}_k = \boldsymbol{\psi}_{k-1}$
repeat
  Extract features $\mathbf{c}_k = \mathbf{f}(\boldsymbol{\psi}_k)$ with derivatives $\mathbf{J}$ from image $\mathbf{I}_k$
  $\mathbf{s} = (\mathbf{J}^{\mathrm{T}}\mathbf{J})^{-1}\mathbf{J}^{\mathrm{T}}(\mathbf{c}_k - \mathbf{c}_0)$
  Determine α by a line search in the direction $\mathbf{s}$
  $\boldsymbol{\psi}_k = \boldsymbol{\psi}_k + \alpha\mathbf{s}$
until α is small enough or max iterations reached

No coordinate system compensation is required for the rotation and scale derivatives.

The Jacobian $\mathbf{J}$ is an $N \times 4$ matrix, where N is the number of channels in the CCFM. This means that $(\mathbf{J}^{\mathrm{T}}\mathbf{J})^{-1}$ is 4×4, so the update $\mathbf{s}$ is relatively cheap to compute once the CCFM and its derivatives are computed.

The optimization step in this tracker is similar to that used in the well-known KLT tracker [68], [83], but using channel-coded feature maps instead of raw images gives it rather different properties. First of all, we do not need to think about how to compute the derivatives of the image. Instead, we keep the image unchanged and use the derivatives of the basis functions. In the next section, the robustness against occlusion and clutter will be discussed.

3.3.2 Robustness Against Occlusion

Consider the case where the tracked patch is occluded. This can be modeled by writing the observed CCFM as $\tilde{\mathbf{c}}_k = \mathbf{c}_k + \mathbf{z}$, where $\mathbf{z}$ is a disturbance term. From (3.30) we see that a requirement for the tracking to be unaffected by this disturbance is that $\mathbf{J}^{\mathrm{T}}\mathbf{z} = 0$. This is the case for example if the occluding object activates channels which are inactive for the tracked object. Note that this is rather different from the usual KLT tracker, which uses the image intensity directly in the minimization problem. Also note that this is an argument against normalizing the CCFMs. If the CCFM is normalized to unit sum, the introduction of new edges in one part of an image causes the channels in other parts of the image to decrease in value.

Figure 3.3 shows a conceptual examples of the effect of clutter. If the feature space consists of orientation and color, only clutter which is similar both in position, orientation and color will affect the optimization. Image (B) shows a situation where the disturbance has edges of an unrelated orientation. In (C), the disturbing edges have a similar orientation but an unrelated color. Finally, in (D) the disturbing edges are similar in both spatial position, color and orientation. The tracking is expected to be significantly affected only in (D).

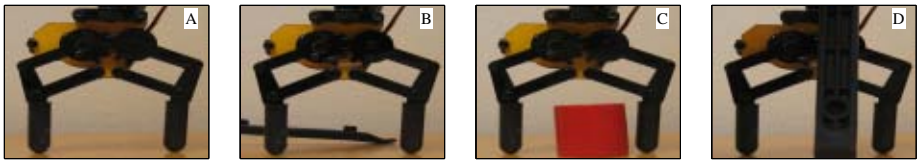

Figure 3.3: Illustration of the effect of clutter. See the text for details.

3.3.3 Experiments

Figure 3.4 shows eight subsequent frames of tracking a robotic gripper using a channel-coded feature map with $5 \times 5 \times 5 \times 5 \times 5$ channels (spatial position, local orientation and two color channels). First order B-spline channels were used, and each channel center is marked in the figure. Note the accuracy of the registration compared to the spatial spacing of the channels. Also note that the shift between two consecutive images is sometimes relatively large. In frame 2 and 3, the patch was occluded with an object of unrelated color, and we can see that the tracking is not significantly affected. On the other hand, in frame 5-8, the occluding object has a color similar to the tracked object. The tracking finally breaks down in frame 8.

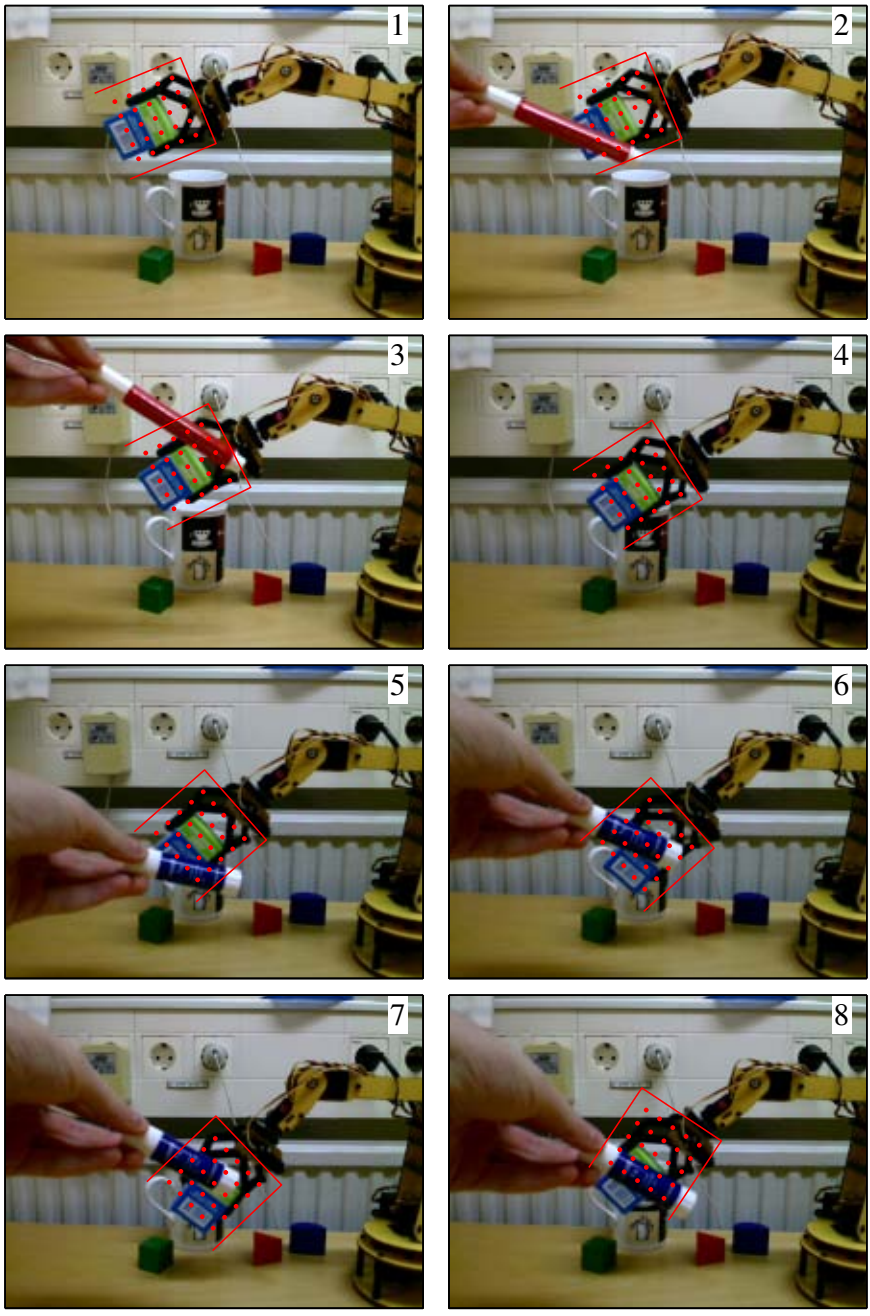

Figure 3.4: Example of tracking using CCFMs. See the text for details.

Chapter 4

Channel Coding through Piecewise Polynomials

...where we take advantage of the piecewiseness of the B-spline basis functions, almost get lost in a haze of multidimensional convolutions, but eventually manage to find our way. The motivation behind this chapter was to seek an efficient method for computing channel-coded feature maps, which turned out to be more tricky than was first anticipated. This chapter exposes the final method with all its gory details, and is a healthy challenge in multi-dimensional index book-keeping.

Channel-coded feature maps are comparably time consuming to compute, especially when several features are included. While it is possible to track a single CCFM in almost real-time even using a straight-forward encoding [59], time becomes a limiting factor when several objects are considered simultaneously. This motivates looking for efficient algorithms. When the basis functions are piecewise polynomials, so are their derivatives. In this chapter, I explore this property and present an algorithm for computing CCFMs and their derivatives using piecewise polynomials. This unfolds as a rather involved exercise in multi-dimensional convolutions and index book-keeping, and in order to write everything down in a convenient way some new notation is introduced. I compare this approach to a more straight-forward algorithm in terms of computational complexity and identify situations where the piecewise approach is beneficial.

Recall that the channel-coded feature maps can be seen as a generalization of the SIFT descriptor. Viewed in that context, the algorithm in this chapter can be used to efficiently compute SIFT descriptors and their derivatives.

4.1 Implementation Using Piecewise Polynomials

4.1.1 Monopieces and Polypuzzles

In [90], k'th order splines are uniquely characterized by an expansion in shifted k'th order B-splines. In this chapter, we take a different approach. Note that a

polynomial is characterized by a coefficient for each participating monomial, and a piecewise polynomial is characterized by a set of such coefficients for each piece. In order to express this compactly, a one-dimensional *monopiece* $P^{(p)}(x)$ of order p is introduced as

$$P^{(p)}(x) = \begin{cases} x^p & \text{if } -0.5 < x \leq 0.5 \\ 0 & \text{otherwise} \end{cases} \quad . \tag{4.1}$$

Using these monopieces, any piecewise polynomial function with unit-spaced knots can be written as

$$B(x) = \sum_p \sum_s \mathbf{K}[p, s] P^{(p)}(x + s) \quad . \tag{4.2}$$

Such a function $B(x)$ will be called a (one-dimensional) *polypuzzle*. In contrast to splines as defined in [90], this definition also support non-smooth and even non-continuous piecewise polynomials. In practice however, only continuous functions will be considered - otherwise the derivatives are not well-defined. As long as the knots are unit-spaced, all splines are polypuzzles.

The shifts s can be at the integers or offset by 0.5, which is the case e.g. for odd-order centralized B-splines. $\mathbf{K}$ is a matrix holding coefficients for each polynomial order and shift. Note that we index this matrix directly with p and s even though s is not necessarily an integer. Think of $\mathbf{K}$ as a mapping $\mathcal{P} \times \mathcal{S} \to \mathbb{R}$, where $\mathcal{P}$ is the set of polynomial orders and $\mathcal{S}$ is the set of shifts used. These sets are finite, so $\mathbf{K}$ can be represented by a matrix. Some examples will be given later.

The derivative of a continuous polypuzzle is another polypuzzle of lower order, and it is simple to state the relationship between the coefficient matrices of the two. Let $\mathbf{K}'$ be the coefficient matrix of B'. From the rules of polynomial differentiation, it follows that $\mathbf{K}'[p, s] = (p + 1)\mathbf{K}[p + 1, s]$.

4.1.2 Multidimensional Monopieces

A *multi-dimensional monopiece* is a separable function consisting of one monopiece in each dimension:

$$P^{(\mathbf{p})}(\mathbf{x}) = \prod_d P^{(\mathbf{p}[d])}(\mathbf{x}[d]) \quad . \tag{4.3}$$

The vector $\mathbf{p}$ defines the polynomial order in each dimension and will be called a *polynomial order map*. A multi-dimensional separable polypuzzle is

$$B(\mathbf{x}) = \prod_d B_d(\mathbf{x}[d]) \quad . \tag{4.4}$$

Combining (4.4) with (4.2) gives the multi-dimensional polypuzzle expressed in terms of monopieces:

$$B(\mathbf{x}) = \prod_d \sum_p \sum_s \mathbf{K}_d[p, s] P^{(p)}(\mathbf{x}[d] + s) \quad . \tag{4.5}$$

We expand this product and note that each term contains a product of one single-dimensional monopiece in each dimension according to

$$B(\mathbf{x}) = \sum_{s_1} \cdots \sum_{s_D} \sum_{p_1} \cdots \sum_{p_D} \prod_d \mathbf{K}_d[p_d, s_d] P^{(p_d)}(\mathbf{x}[d] + s_d) \ . \tag{4.6}$$

We can now gather all shift and polynomial order indices into index vectors $\mathbf{s}$ and $\mathbf{p}$. The coefficients in front of the monopieces can be combined into a multi-dimensional version of $\mathbf{K}$ defined as[1]

$$\mathbf{K}[\mathbf{p}, \mathbf{s}] = \prod_d \mathbf{K}_d[\mathbf{p}[d], \mathbf{s}[d]] \ . \tag{4.7}$$

Furthermore, the product of the monopieces can be written as a multidimensional monopiece $P^{(\mathbf{p})}(\mathbf{x} - \mathbf{s})$, which lets us write (4.6) more compactly as

$$B(\mathbf{x}) = \sum_{\mathbf{p}} \sum_{\mathbf{s}} \mathbf{K}[\mathbf{p}, \mathbf{s}] P^{(\mathbf{p})}(\mathbf{x} + \mathbf{s}) \ . \tag{4.8}$$

This equation simply states that the multi-dimensional polypuzzle can be written as a linear combination of translated multi-dimensional monopieces of different orders. The coefficients in front of the monopieces are organized in an array $\mathbf{K}$, giving the coefficient for each translation $\mathbf{s}$ and polynomial order map $\mathbf{p}$.

4.1.3 Monopiece Encodings

The proposed way of computing the channel-coded feature maps is to go via a *monopiece encoding*, defined as

$$\mathbf{c}_{\text{mp}}[\mathbf{p}, \tilde{\mathbf{x}}] = \frac{1}{I} \sum_i w_i P^{(\mathbf{p})}(\mathbf{x}_i - \tilde{\mathbf{x}}) \ . \tag{4.9}$$

This can be seen as a multi-layered CCFM where each $\mathbf{p}$-layer corresponds to one polynomial order configuration $\mathbf{p}$. Each layer is like a histogram using $P^{(\mathbf{p})}$ as basis function instead of the box function. The channels with different centers $\tilde{\mathbf{x}}$ within each layer are not overlapping, meaning that each pixel of the input image belongs to exactly one channel per layer. To compute this encoding, simply loop over the image, compute the value of each polynomial for each pixel, and accumulate the result into $\mathbf{c}$.

This is a generalization of the P-channel encoding used in [28, 26], where only one constant and one linear function in each dimension were used. Here, we will use all monopieces needed for our target B-spline encoding. Which these are will be stated in Sect. 4.1.7.

[1] We reuse the symbol $\mathbf{K}$ for the multi-dimensional case. The two meanings will be disambiguated by the arguments of $\mathbf{K}$.

4.1.4 Converting Monopiece Encodings to CCFMs

Recall the definition of channel-coded feature maps from (3.2), repeated here for convenience:

$$\mathbf{c}[\tilde{\mathbf{x}}] = \frac{1}{I}\sum_i w_i B(\mathbf{x}_i - \tilde{\mathbf{x}}) \ . \tag{4.10}$$

In order to see how to get from the monopiece encoding in (4.9) to the CCFM in (4.10), we rewrite (4.10) in terms of monopieces according to (4.8), rearrange the sums, and plug in the definition of the monopiece encodings from (4.9):

$$\begin{aligned}\mathbf{c}[\tilde{\mathbf{x}}] =& \frac{1}{I}\sum_i w_i \sum_{\mathbf{p}}\sum_{\mathbf{s}} \mathbf{K}[\mathbf{p},\mathbf{s}] P^{(\mathbf{p})}(\mathbf{x}_i - \tilde{\mathbf{x}} + \mathbf{s}) = \\ =& \sum_{\mathbf{p}}\sum_{\mathbf{s}} \mathbf{K}[\mathbf{p},\mathbf{s}] \frac{1}{I}\sum_i w_i P^{(\mathbf{p})}(\mathbf{x}_i - \tilde{\mathbf{x}} + \mathbf{s}) = \\ =& \sum_{\mathbf{p}}\sum_{\mathbf{s}} \mathbf{K}[\mathbf{p},\mathbf{s}] \ \mathbf{c}_{\mathrm{mp}}[\mathbf{p}, \tilde{\mathbf{x}} - \mathbf{s}] \ .\end{aligned} \tag{4.11}$$

At this point, it is convenient to use the $(\cdot)$-notation. With $\mathbf{c}_{\mathrm{mp}}[\mathbf{p},\cdot]$, I mean a function $\tilde{\mathbf{x}} \mapsto \mathbf{c}_{\mathrm{mp}}[\mathbf{p},\tilde{\mathbf{x}}]$. Similarly, $\mathbf{K}[\mathbf{p},\cdot]$, is a function $\tilde{\mathbf{x}} \mapsto \mathbf{K}[\mathbf{p},\tilde{\mathbf{x}}]$. Since these are functions of a vector variable, we can apply a multi-dimensional discrete convolution operator according to

$$\big(\mathbf{K}[\mathbf{p},\cdot] * \mathbf{c}_{\mathrm{mp}}[\mathbf{p},\cdot]\big)(\tilde{\mathbf{x}}) = \sum_{\mathbf{s}} \mathbf{K}[\mathbf{p},\mathbf{s}] \ \mathbf{c}_{\mathrm{mp}}[\mathbf{p},\tilde{\mathbf{x}} - \mathbf{s}] \ . \tag{4.12}$$

This is recognized as the right-hand side of (4.11), which to summarize gives us

$$\mathbf{c} = \sum_{\mathbf{p}} \mathbf{K}[\mathbf{p},\cdot] * \mathbf{c}_{\mathrm{mp}}[\mathbf{p},\cdot] \ . \tag{4.13}$$

In words, in order to convert from a monopiece encoding to a CCFM, we must perform one multidimensional convolution in each $\mathbf{p}$-layer, and then sum over all layers. Note from the definition of $\mathbf{K}$ in (4.7) that the filter kernel $\mathbf{K}[\mathbf{p},\cdot]$ is separable. This means that each convolution in (4.13) can be computed as a sequence of one-dimensional convolutions with kernel $\mathbf{K}_d[\mathbf{p}[d],\cdot]$ in dimension d. The complete algorithm is summarized as Alg. 4.1.

4.1.5 Generalized Notation

In the next section, I describe how the derivatives are computed in the piecewise polynomial framework. In order to do that, we first need to further extend our notation. Recall that each multi-dimensional polypuzzle is characterized by its coefficient array $\mathbf{K}$. In order to be able to express such matrices for different functions conveniently, we define $\mathbf{K}\{f\}[\mathbf{p},\tilde{\mathbf{x}}]$ for any polypuzzle f such that

$$f(\mathbf{x}) = \sum_{\mathbf{p},\mathbf{s}} \mathbf{K}\{f\}[\mathbf{p},\mathbf{s}] P^{(\mathbf{p})}(\mathbf{x} + \mathbf{s}) \ . \tag{4.14}$$

Algorithm 4.1 Creating CCFMs from monopiece encodings.

Inputs:
- Monopiece encoding $\mathbf{c}_{\mathrm{mp}}$
- Kernel definition matrix $\mathbf{K}_d$ for each dimension d

Initialization:
- **result** = Zero N-dimensional array

for each polynom order map **p** **do**
  thislayer = $\mathbf{c}_{\mathrm{mp}}[\mathbf{p}, \cdot]$
  for each dimension d **do**
    kernel1d = $\mathbf{K}_d[\mathbf{p}[d], \cdot]$
    Reshape **kernel1d** to be along dimension d
    thislayer = **thislayer** $*$ **kernel1d** (1D convolution)
  end for
  result += **thislayer**
end for

To be strict, there should always be a *function* within the {}-brackets, but the notation is simplified slightly by allowing ourselves to write things like $\mathbf{K}\{vB'_u\}$, to be interpreted as $\mathbf{K}\{\mathbf{x} \mapsto vB'_u(\mathbf{x})\}$, where $\mathbf{x} = [u, v, z_1, z_2, \ldots]^{\mathrm{T}}$ as in the previous chapter. Furthermore, we denote different complete encodings as $\mathbf{c}\{f\}$, meaning the encoding obtained when using f as the multi-dimensional basis function. The relations (3.2) and (4.13) then generalize to

$$\mathbf{c}\{f\} = \frac{1}{I}\sum_i f(\mathbf{x}_i - \cdot) = \sum_{\mathbf{p}} \mathbf{K}\{f\}[\mathbf{p}, \cdot] * \mathbf{c}_{\mathrm{mp}}[\mathbf{p}, \cdot] \ . \tag{4.15}$$

Even though the notation is non-standard and may look confusing, this single equation compactly summarizes more or less everything said so far in this chapter. It is important to fully understand this equation before proceeding into the next section.

4.1.6 Derivatives

From (3.25) and (3.26), it is immediately clear that

$$\frac{\mathrm{d}\mathbf{c}\{B\}[\tilde{\mathbf{x}}]}{\mathrm{d}b_u} = -\mathbf{c}\{B'_u\}[\tilde{\mathbf{x}}] \tag{4.16}$$

$$\frac{\mathrm{d}\mathbf{c}\{B\}[\tilde{\mathbf{x}}]}{\mathrm{d}b_v} = -\mathbf{c}\{B'_v\}[\tilde{\mathbf{x}}] \ . \tag{4.17}$$

These can be computed by Alg. 4.1 using $\mathbf{K}\{B'_u\}$ and $\mathbf{K}\{B'_v\}$. The derivatives with respect to rotation and scale are more complicated. From (3.27), (3.28), we

Algorithm 4.2 Computing CCFMs and derivatives through monopiece.

Compute $\mathbf{c}_{\mathrm{mp}}[\mathbf{p}, \tilde{\mathbf{x}}]$ for each $\mathbf{p}$.
$\mathbf{c}\{B\} = \mathtt{convert}(\mathbf{c}_{\mathrm{mp}}, \mathbf{K}\{B\}, \mathbf{K}\{B\}, \mathbf{K}\{B\})$
$\mathbf{c}\{B'_u\} = \mathtt{convert}(\mathbf{c}_{\mathrm{mp}}, \mathbf{K}\{B'\}, \mathbf{K}\{B\}, \mathbf{K}\{B\})$
$\mathbf{c}\{B'_v\} = \mathtt{convert}(\mathbf{c}_{\mathrm{mp}}, \mathbf{K}\{B\}, \mathbf{K}\{B'\}, \mathbf{K}\{B\})$
$\mathbf{c}\{uB'_u\} = \mathtt{convert}(\mathbf{c}_{\mathrm{mp}}, \mathbf{K}\{xB'\}, \mathbf{K}\{B\}, \mathbf{K}\{B\})$
$\mathbf{c}\{uB'_v\} = \mathtt{convert}(\mathbf{c}_{\mathrm{mp}}, \mathbf{K}\{xB\}, \mathbf{K}\{B'\}, \mathbf{K}\{B\})$
$\mathbf{c}\{vB'_u\} = \mathtt{convert}(\mathbf{c}_{\mathrm{mp}}, \mathbf{K}\{B'\}, \mathbf{K}\{xB\}, \mathbf{K}\{B\})$
$\mathbf{c}\{vB'_v\} = \mathtt{convert}(\mathbf{c}_{\mathrm{mp}}, \mathbf{K}\{B\}, \mathbf{K}\{xB'\}, \mathbf{K}\{B\})$
Compute the desired derivatives using equations (4.16), (4.17), (4.22), (4.23).

see that the four sums

$$I_1 = \frac{1}{I}\sum_i u_i B'_u(\mathbf{x}_i - \tilde{\mathbf{x}}) \qquad I_2 = \frac{1}{I}\sum_i v_i B'_u(\mathbf{x}_i - \tilde{\mathbf{x}})$$
$$I_3 = \frac{1}{I}\sum_i u_i B'_v(\mathbf{x}_i - \tilde{\mathbf{x}}) \qquad I_4 = \frac{1}{I}\sum_i v_i B'_v(\mathbf{x}_i - \tilde{\mathbf{x}}) \tag{4.18}$$

are needed. If B is here a separable polypuzzle of order n, so is uB'_u and vB'_v, while vB'_u and uB'_v are of order $n+1$. This means that in order to compute the derivative with respect to image rotation we need more monopieces than are needed for the original encoding. This is a bit disappointing, but can be handled without too much extra overhead.

To see how to construct I_1, I_2, I_3, I_4 from a monopiece encoding, first rewrite I_1 as

$$I_1 = \frac{1}{I}\sum_i (u_i - \tilde{u}) B'_u(\mathbf{x}_i - \tilde{\mathbf{x}}) + \tilde{u}\frac{1}{I}\sum_i B'_u(\mathbf{x}_i - \tilde{\mathbf{x}}) = \tag{4.19}$$
$$= \mathbf{c}\{uB'_u\}[\tilde{\mathbf{x}}] + \tilde{u}\,\mathbf{c}\{B'_u\}[\tilde{\mathbf{x}}] \ . \tag{4.20}$$

By separating $uB'_u(\mathbf{x})$ into

$$uB'_u(\mathbf{x}) = uB'_1(u)\prod_{d\neq 1} B_d(\mathbf{x}[d]) \ , \tag{4.21}$$

$\mathbf{K}\{uB'_u\}$ can be separated according to (4.7) with $\mathbf{K}_1 = \mathbf{K}\{xB'\}$ and $\mathbf{K}_d = \mathbf{K}\{B\}$ for $d \neq 1$, and the encoding $\mathbf{c}\{uB'_u\}$ can be computed by Alg. 4.1.[2] The other

[2]The symbol x is used to indicate that we are talking about a one-dimensional $\mathbf{K}$. To be strict, we should write $\mathbf{K}\{xB'\}$ as $\mathbf{K}\{x \mapsto xB'(x)\}$, not to be confused with $\mathbf{K}\{\mathbf{x} \mapsto uB'_u(\mathbf{x})\}$.

sums from (4.18) can be handled in a similar way, and altogether we get

$$\begin{aligned}\frac{\mathrm{d}\mathbf{c}\{B\}[\tilde{\mathbf{x}}]}{\mathrm{d}\alpha} &= \mathbf{c}\{vB'_u\}[\tilde{\mathbf{x}}] + \tilde{v}\,\mathbf{c}\{B'_u\}[\tilde{\mathbf{x}}] - \\ &\quad \mathbf{c}\{uB'_v\}[\tilde{\mathbf{x}}] - \tilde{u}\,\mathbf{c}\{B'_v\}[\tilde{\mathbf{x}}] \ , \qquad (4.22)\\ \frac{\mathrm{d}\mathbf{c}\{B\}[\tilde{\mathbf{x}}]}{\mathrm{d}s} &= -\,\mathbf{c}\{uB'_u\}[\tilde{\mathbf{x}}] - \tilde{u}\,\mathbf{c}\{B'_u\}[\tilde{\mathbf{x}}] - \\ &\quad \mathbf{c}\{vB'_v\}[\tilde{\mathbf{x}}] - \tilde{v}\,\mathbf{c}\{B'_v\}[\tilde{\mathbf{x}}] - \\ &\quad 2\mathbf{c}\{B\}[\tilde{\mathbf{x}}] \ . \qquad (4.23)\end{aligned}$$

Each of these $\mathbf{c}$-terms can be computed from the monopiece encoding $\mathbf{c}_{\mathrm{mp}}$ using Alg. 4.1 with the corresponding $\mathbf{K}$. To see how to construct things like $\mathbf{K}\{xB'\}$, we first note that the identity function $f(x) = x$ can be written as a piecewise polynomial with $\mathbf{K}\{x\}[0, s] = s$ and $\mathbf{K}\{x\}[1, s] = 1$ for all s. This holds regardless of whether the shifts s are at the integers or at the 0.5-shifted integers. It is well-known that the product of two polynomials can be computed by a convolution of the polynomial coefficients, and for piecewise polynomials this convolution can be performed separately for each piece. In our notation this can be expressed compactly as

$$\mathbf{K}\{fg\}[\cdot, s] = \mathbf{K}\{f\}[\cdot, s] * \mathbf{K}\{g\}[\cdot, s] \ . \qquad (4.24)$$

The complete algorithm for computing a channel-coded feature map together with its derivatives with respect to similarity transformation of the underlying image is given in Alg. 4.2. The function $\texttt{convert}(\mathbf{c}_{\mathrm{mp}}, \mathbf{K}_{\mathrm{u}}, \mathbf{K}_{\mathrm{v}}, \mathbf{K}_{\mathrm{f}})$ mentioned in the pseudocode means running Alg. 4.1 for converting $\mathbf{c}_{\mathrm{mp}}$ using $\mathbf{K}_{\mathrm{u}}$ and $\mathbf{K}_{\mathrm{v}}$ in the spatial dimensions and $\mathbf{K}_{\mathrm{f}}$ in each feature dimension. For convenience, some standard coefficient matrices are given in Table 4.1, useful for creating first- and second-order B-spline CCFMs.

4.1.7 Required Monopieces

For a basis function of order k, all combinations of monopieces from order 0 to k are needed in each dimension, giving in total $(k+1)^D$ monopieces. Furthermore, because of the terms from uB'_v and vB'_u, some monopieces of order $k+1$ in one spatial dimension are needed – but only together with monopieces of order $k-1$ in the other spatial dimension. This gives in total $\beta = (k+1)^D + 2k(k+1)^{(D-2)}$ monopieces. Some common cases are presented in Table 4.2.

4.2 Complexity Analysis

In this section, the computational complexity of the method is compared to a more direct method. In this analysis, we consider an image of size $M \times M$ from which $D-2$ features are computed, such that the spatio-featural space has D dimensions like before. This feature map is encoded using N channels in each dimension, resulting in N^D channels in total. Let the basis function contain S pieces, such

	First order B-spline:	Second order B-spline:
$\mathbf{K}\{B\}$	$\begin{bmatrix} 1/2 & 1/2 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1/8 & 3/4 & 1/8 \\ -1/2 & 0 & 1/2 \\ 1/2 & -1 & 1/2 \end{bmatrix}$
$\mathbf{K}\{B'\}$	$\begin{bmatrix} -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1/2 & 0 & 1/2 \\ 1 & -2 & 1 \end{bmatrix}$
$\mathbf{K}\{xB\}$	$\begin{bmatrix} 1/4 & -1/4 \\ 0 & 0 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1/8 & 0 & -1/8 \\ -3/8 & 3/4 & -3/8 \\ 0 & 0 & 0 \\ 1/2 & -1 & 1/2 \end{bmatrix}$
$\mathbf{K}\{xB'\}$	$\begin{bmatrix} -1/2 & -1/2 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1/2 & 0 & -1/2 \\ 1/2 & 0 & -1/2 \\ 1 & -2 & 1 \end{bmatrix}$

Table 4.1: Some useful **K**-matrices. The topmost row in each matrix corresponds to polynomial order 0.

First-order B-spline, one feature f (12 monomials): $1 \quad u \quad u^2 \quad v \quad uv \quad v^2 \quad f \quad uf \quad u^2f \quad vf \quad uvf \quad v^2$
Second-order B-spline, one feature f (39 monomials): $1 \quad u \quad u^2 \quad u^3 \quad v \quad uv \quad u^2v \quad u^3v$ $v^2 \quad uv^2 \quad u^2v^2 \quad v^3 \quad uv^3$ $f \quad uf \quad u^2f \quad u^3f \quad vf \quad uvf \quad u^2vf \quad u^3vf$ $v^2f \quad uv^2f \quad u^2v^2f \quad v^3f \quad uv^3f$ $f^2 \quad uf^2 \quad u^2f^2 \quad u^3f^2 \quad vf^2 \quad uvf^2 \quad u^2vf^2 \quad u^3vf^2$ $v^2f^2 \quad uv^2f^2 \quad u^2v^2f^2 \quad v^3f^2 \quad uv^3f^2$

Table 4.2: Some examples of which monomials are required for different CCFMs.

that exactly S channels are active at the same time in each dimension. This gives in total S^D active channels for each input pixel.

- **A) Direct Approach** The simplest thing to do is to loop through the image and accumulate the value of each of the S^D bins that are active for each pixel position. We first need to create a zero array of size N^D and then compute the original encoding and the four derivatives separately, giving a total of $N^D + 5S^D M^2$ operations.

- **B) Piecewise Approach** As proposed in this chapter, we start with computing a monopiece encoding from the image data using β monopieces, where β can be derived according to Sect. 4.1.7. Since each pixel is sent to β bins, this requires βM^2 operations and gives us a monopiece encoding of size βN^D (step B1). From this representation, we create the final B-spline encoding and its derivatives by the technique described in Sect. 4.1.4.

  Computing $\mathbf{c}\{B\}$ and each of the other 6 intermediate encodings in Alg. 4.2 requires βD one-dimensional convolutions with a convolution kernel of size S, each requiring SN^D operations (step B2). To combine the intermediate encodings into final derivatives requires only a single extra pass through the data, using $11N^D$ operations (step B3), where the number 11 refers to the total number of terms of the right-hand sides of (4.16), (4.17), (4.23), (4.22). In total, this gives $\beta M^2 + (7\beta DS + 11)N^D$ operations. The constant 11 can be neglected in all practical cases.

If the image is large compared to the number of channels, the complexity of the piecewise approach will be dominated by the first pass through the image, requiring βM^2 operations. In the limit of infinitely large images, the monopieces method is a factor $5S^D/\beta$ faster than the direct method. As an example, consider the case of first-order B-spline basis functions and $D = 3$. Then $5S^D = 40$ while $\beta = 12$ according to Table 4.2.

On the other hand, if the image is small, the piecewise approach requires at least $7\beta DSN^D$ operations, while the direct approach requires only at least N^D. This gives a factor $7\beta DS$ in favor of the direct method, which is a clear advantage for small images and many channels.

Furthermore, increasing the size S of the basis function while keeping everything else constant gives a dramatic advantage to the piecewise method, since the number of operations grows like S^D for the direct approach but at worst only linearly in S for the piecewise method (if the complexity is dominated by the second term).

Some theoretical examples are shown in Fig. 4.1 using B-splines of first and second order. The exact time consumption and break-even point for a real system of course depend a lot on the exact implementation and hardware used. In Fig. 4.2, two competing C++ implementations were compared in a number of situations - one direct algorithm specialized for $D = 3$ and one general piecewise algorithm. The experiments were run on a Pentium M running at 1.8 GHz. The trends are similar to the theoretical curves but with different constant factors and break-even points. I conclude that the piecewise approach is comparatively better for large

images with few channels, while the direct approach is better for small images with many channels, with an advantage to the piecewise method for the cases considered in the later experiments.

4.3 Discussion

In this chapter, I have described one way of implementing channel-coded feature maps using separable piecewise polynomial basis functions. This approach shows favorable computational complexity compared to a direct encoding for large images with few channels. The main advantage comes from the fact that much intermediate results can be reused in computing the derivatives. However, the amount of computation needed still grows rapidly when the basis functions become larger or the spatio-featural space higher-dimensional, and the performance gain turned out to be lower than was originally hoped for.

This motivates trying to reduce the number of monopieces used. One motivation for channel-coded feature maps in the first place was to have a representation that responds smoothly to geometrical changes of the underlying image, with a coarse spatial resolution, but much information at each spatial position. Maybe these goals can be fulfilled with simpler basis functions, composed only of a subset of the monopieces needed for higher-order B-splines. This is related to the P-channel representation [28], where the number of monopieces used grows linearly in the number of feature dimensions. However, that representation is less smooth and less suited for taking derivatives. Finding a good trade-off between computational complexity and performance in any given application is subject to future research.

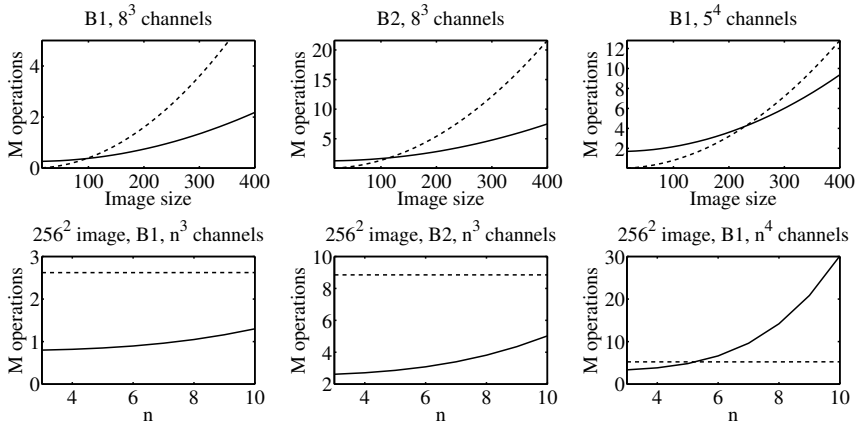

Figure 4.1: Theoretical time consumption for the two approaches in different situations measured in million operations. Dashed: direct method. Solid: piecewise method.

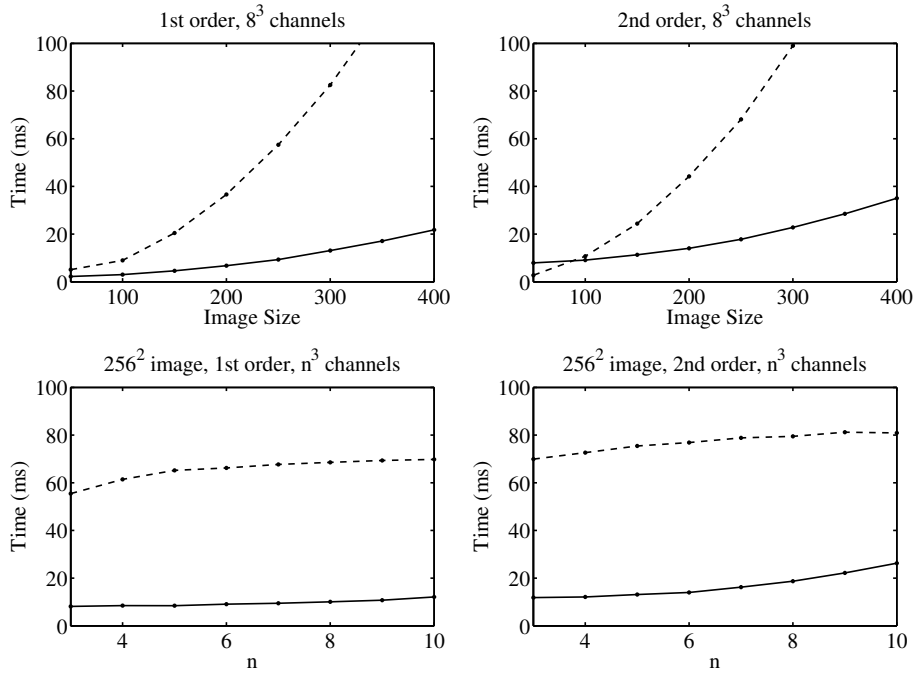

Figure 4.2: Empirical time consumption for two competing implementations. Dashed: direct method. Solid: piecewise method.

Chapter 5

Distance Measures on Channel Vectors

...where the Channel Vector gets compared to other channel vectors in different ways, and where the different ways of comparing channel vectors are compared. At the end of the chapter, we will have constructed our first simple view-based object recognizer.

In order to use channel-coded feature maps for view-based object recognition and similar tasks, we need to measure the distance between a query CCFM and a memorized prototype. One simple choice is to use the Euclidean distance directly on the channel values, but there are many other distance measures available. In this chapter, I will review some common distance measures on continuous density functions and histograms, and discuss which assumptions are violated when channel-coding is used instead of hard histograms. In Sect. 5.2.2, all presented distance measures will be evaluated experimentally on a view-based object recognition task.

There are many more ways of comparing histograms than are mentioned here. Some examples can be found in [80], also including some experimental comparison.

5.1 Distance Measures

5.1.1 Euclidean Distance

The most common way to define a distance measure between two vector spaces is the Euclidean distance. For continuous functions, the L^2 norm can be used:

$$d(f,g) = \|f-g\| = \left(\int |f(x)-g(x)|^2 \, \mathrm{d}x \right)^{1/2} . \tag{5.1}$$

This is well-defined on functions in L^2 (square-integrable functions). One problem of using this measure on PDFs is that point-masses (Diracs) are not allowed. This

is not just a mathematical problem but really leads to undesirable behavior in practice when dealing with highly peaked distributions. Consider two densities

$$p_1(x) = \Pi(x - 0.5) \tag{5.2}$$

$$p_2(x) = a\Pi\big(a(x - 0.5)\big) \tag{5.3}$$

where Π is the box function (see Sect. 1.4). As $a \to \infty$, p_2 approaches a Dirac distribution. The Euclidean distance between p_1 and p_2 is $\sqrt{a-1}$, which approaches infinity as $a \to \infty$. So, even if we stay within L_2 and never let p_2 become a Dirac, the distance measure becomes very sensitive for changes in a. Often when dealing with highly peaked distributions, the actual width of the peak depends only on the amount of noise – consider for example the distribution of graylevels in a noisy homogeneous image region.

When measuring the distance between channel vectors, the situation is less severe. As $a \to \infty$, the channel encoding of $p_2(x)$ converges to a finite, well-defined vector, namely the encoding of a single scalar. This means that there is no fundamental problem of using this distance measure on channel vectors, but it should not be viewed as a discrete approximation of some measure on continuous densities. In Sect. 5.2.1, an example is given that further highlights the problem with the Euclidean distance.

5.1.2 Chi-Squared Statistic

Another problem with the Euclidean distance is that the statistical properties of frequency-coded data are not taken into account. For example, consider two bins i and j in two different histograms $\mathbf{a}$ and $\mathbf{b}$. If $\mathbf{a}[i] = 10, \mathbf{b}[i] = 20$ and $\mathbf{a}[j] = 1000, \mathbf{b}[j] = 1010$, bin i indicates a more significant difference than bin j even though the squared distance between the bins is the same.

To make this more precise, the values in a histogram are often assumed to be independent Poisson-distributed random variables. The mean of a Poisson variable equals the variance, and if we have two measured histograms $\mathbf{h}_1$ and $\mathbf{h}_2$ of the same phenomenon, an estimate of the mean and variance is $(\mathbf{h}_1[n]+\mathbf{h}_2[n])/2$. The Euclidean distance can be improved by weighting each squared bin difference with the estimated variance, leading to the chi-square statistic

$$\chi^2(\mathbf{h}_1, \mathbf{h}_2) = \frac{1}{2} \sum_{n=1}^{N} \frac{(\mathbf{h}_1[n] - \mathbf{h}_2[n])^2}{\mathbf{h}_1[n] + \mathbf{h}_2[n]} \ . \tag{5.4}$$

This statistic is commonly used to compare frequency coded data, and is monotonically related to the probability that the two histograms originate from the same PDF. Some applications in computer vision can be found in [3, 64].

When moving to the case of overlapping channel basis functions, we can no longer assume that neighboring bins are independent. To compensate for this, we should instead use the Mahalanobis distance

$$d(\mathbf{h}_1, \mathbf{h}_2) = (\mathbf{h}_1 - \mathbf{h}_2)^{\mathrm{T}} \mathbf{C}^{-1} (\mathbf{h}_1 - \mathbf{h}_2) \ , \tag{5.5}$$

where $\mathbf{C}$ is the covariance matrix of the histogram. This distance measure is more time-consuming to compute than the one in (5.4), and requires knowledge about the covariance matrix $\mathbf{C}$. How to find this covariance matrix is far from obvious and requires a careful analysis with a well-defined statistical model over the stochastic process producing the histogram. This direction is not followed further.

5.1.3 Bhattacharyya Coefficient

Let $p_1(x)$ and $p_2(x)$ be two continuous density functions. According to [89], the Bhattacharyya coefficient is defined as

$$\rho(p_1, p_2) = \int_{-\infty}^{\infty} \sqrt{p_1(x)p_2(x)}\,\mathrm{d}x \tag{5.6}$$

and the Bhattacharyya distance as $B(p_1, p_2) = -\ln \rho(p_1, p_2)$. This is not a distance measure in a strict sense, since the triangle inequality is not satisfied [89]. In [16] it is suggested to instead use

$$d(p_1, p_2) = \sqrt{1 - \rho(p_1, p_2)}\ , \tag{5.7}$$

which can be identified as a measure proportional to the Euclidean distance between the point-wise square-roots of the densities:

$$\begin{aligned} \|\sqrt{p_1} - \sqrt{p_2}\| = &\left(\int_{-\infty}^{\infty} \left(\sqrt{p_1(x)} - \sqrt{p_2(x)}\right)^2 \mathrm{d}x\right)^{1/2} = \\ &\left(\int_{-\infty}^{\infty} p_1(x) - 2\sqrt{p_1(x)}\sqrt{p_2(x)} + p_2(x)\ \mathrm{d}x\right)^{1/2} = \\ &(2 - 2\rho(p_1, p_2))^{1/2} = \sqrt{2}\, d(p_1, p_2)\ . \end{aligned} \tag{5.8}$$

This distance will be referred to as the *square root distance*. One important property of square root densities is that they have a unit L_2 norm, since the integral of a PDF is 1. This means that all square root densities are square-integrable, which overcomes the Dirac problem from the Euclidean distance.

In principle, there is no problem to apply the square-root distance to histogram representations. The square-root distance on histograms can be seen as a discrete approximation of the continuous distance measure. Using overlapping channels does not cause any immediate problems either. Furthermore, from a practical point of view, a system that works with Euclidean distances can easily be reused to work on square root distances by simply taking the square root of the inputs before further processing. In [2], the Bhattacharyya distance is treated in more detail, including a discussion of how the Bhattacharyya distance can be seen to approximate the chi-square distance.

5.1.4 Kullback-Leibler Divergence

The final distance measure that will be considered is the *Kullback-Leibler (KL) Divergence*, also known as *directed divergence* or *relative information*, defined as

$$D_{\mathrm{KL}}(p_1, p_2) = \int_{-\infty}^{\infty} p_1(x) \ln\left(\frac{p_1(x)}{p_2(x)}\right) \mathrm{d}x \tag{5.9}$$

for continuous distributions and

$$D_{\mathrm{KL}}(p_1, p_2) = \sum_x p_1(x) \ln\left(\frac{p_1(x)}{p_2(x)}\right) \tag{5.10}$$

for discrete distributions. Note that this is not a strict distance measure – it is not even symmetric.

There is a number of information-theoretic interpretations of the KL-divergence. The perhaps most well-known one is as a measure of the extra message length required to transmit a symbol drawn from p_1 using a code-book optimized for p_2. A different interpretation related to the log-likelihood ratio is given in [89]. Let w_1 be the hypothesis that x is drawn from p_1 and w_2 the hypothesis that x is drawn from p_2. Let $\Lambda(x) = \ln(p_1(x)/p_2(x))$. We then have that

$$D_{\mathrm{KL}}(p_1, p_2) + D_{\mathrm{KL}}(p_2, p_1) = \mathrm{E}\left\{\Lambda(x)|w_1\right\} - \mathrm{E}\left\{\Lambda(x)|w_2\right\} \quad . \tag{5.11}$$

In words, the symmetric *divergence* equals the difference in means of the log-likelihood ratio used for testing the hypotheses w_1 and w_2 against each other, and is a measures of separation between w_1 and w_2.

When comparing a noisy, cluttered query view to a training view, the KL-divergence is not theoretically well-motivated. For example, if some bins are zero in the training CCFM but non-zero in the query CCFM, the KL-divergence is infinite. This may very well happen due to occlusion and clutter. I do not recommend using the KL-divergence, but include it here mainly because it was used in the related publication [28].

5.2 Examples and Experiments

5.2.1 The Problem with Euclidean Distance

I will now study an example that further illustrates the problem with the Euclidean distance that the other distance measures manage to overcome. Consider a case where a number of one-dimensional densities are encoded using rectangular histograms. Let $\theta(x)$ be the Heaviside step function, $x \in [0, 1]$ and

$$p_1(x) = 2\,\theta(x - 0.5) \tag{5.12}$$

$$p_2(x) = 2\left(1 - \theta(x - 0.5)\right) \tag{5.13}$$

(see Fig. 5.1). Using an even number of bins N, the histograms of these densities are

$$\mathbf{h}_1 = [0, \ldots, 0, 2/N, \ldots, 2/N] \tag{5.14}$$

$$\mathbf{h}_2 = [2/N, \ldots, 2/N, 0, \ldots, 0] \ , \tag{5.15}$$

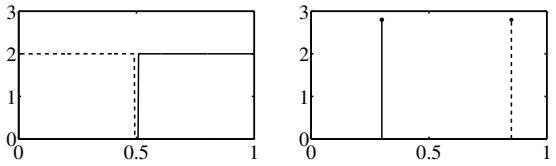

Figure 5.1: The density functions considered in Sect. 5.2.1 in order to illustrate the problem with the Euclidean distance.

and the Euclidean distance between them is

$$d_{12} = d(\mathbf{h}_1, \mathbf{h}_2) = \sqrt{N\left(\frac{2}{N}\right)^2} = \frac{2}{\sqrt{N}} \ . \tag{5.16}$$

Now consider a different situation, illustrated in the right part of Fig. 5.1, where

$$p_3(x) = \delta(x - x_1) \tag{5.17}$$
$$p_4(x) = \delta(x - x_2) \tag{5.18}$$

and $x_1 \neq x_2$. The histograms of these densities are

$$\mathbf{h}_3 = [0, \ \ldots \ , 0, 1, 0, \ldots 0] \tag{5.19}$$
$$\mathbf{h}_4 = [0, \ldots, 0, 1, 0, \ \ldots \ 0] \ . \tag{5.20}$$

If the number of bins is large enough, the non-zero bin is at different positions in both histograms, and the Euclidean distance is

$$d_{34} = d(\mathbf{h}_1, \mathbf{h}_2) = \sqrt{2} \ . \tag{5.21}$$

As $N \to \infty$, $d_{12} \to 0$ while d_{34} stays constant. The Euclidean distance fails to converge to a fixed value when the number of bins increases. Note that this problem cannot be fixed by some normalization of the distance with respect to the number of bins, since the two distances d_{12} and d_{34} show different behavior.

On the other hand, using the chi-square distance gives $d_{12} = d_{34} = 1$, and using the square root distance gives $d_{12} = d_{34} = \sqrt{2}$, which further illustrates their advantages over the Euclidean distance. The Kullback-Leibler divergence is infinite in both cases regardless of the direction, since $\mathbf{h}_1$ is zero when $\mathbf{h}_2$ is non-zero and vice versa.

The Euclidean distance is still be useful in practice and will be used in experiments later on in the thesis, but one should be aware of its shortcomings.

5.2.2 COIL Experiment

To evaluate the different distance measures, a view-based object recognition experiment was conducted using the COIL-100 database [76] (see Fig. 5.2). Each of the 100 objects contains 72 views for different rotations in the database. From

Figure 5.2: Some examples of objects in the COIL database.

#Chans	Weighting	KLD	Euclid	Sqrt-Euclid	chi-square
5^5	grad	99.1	99.0	99.3	99.4
5^5	ST-grad	99.2	98.9	99.4	99.4
8^3	ST-grad	92.4	93.2	93.9	94.1
5^3	ST-grad	92.9	93.4	93.8	94.0
8^3	none	89.2	88.8	90.4	90.3
5^3	none	89.6	89.8	89.7	89.8
8^3	grad	92.8	92.4	94.2	94.3
5^3	grad	93.6	94.7	95.1	95.2

Table 5.1: Recognition results on the COIL database.

each object, every 8th view was selected for training and the rest for evaluation, producing 9 training views spaced 40° apart and 63 evaluation views for each object. Each of the evaluation views was then classified by a simple nearest neighbor classifier using the given distance measure. This means that a view is classified correctly if the closest training view belongs to the same object class as the query view - the rotation angle is not estimated.

The results for different channel encoding settings are shown in Table 5.1. Each pixel was either weighted by its gradient magnitude (grad), by the soft-thresholded gradient magnitude (ST-grad) or unweighted. The CCFMs were normalized to unit sum for the KL-divergence and were left unnormalized for the other distance measures. When 3 channel dimensions were used (8^3 and 5^3 channels), only spatial position and orientation was encoded. When 5 channel dimensions were used, also the hue and saturation part of the HSV color space were included in the encoding.

Unsurprisingly, the results are in general much better when color is included. However, the focus here is on comparing distance measures, and it is interesting to see also how the distance measures perform in the more challenging case where color is disregarded. We note that both the square-root distance and the chi-square distance constantly beat the Euclidean distance regardless of the encoding settings. The chi-square distance is often marginally better than the sqrt-distance.

5.3 Discussion

This chapter has reviewed some common distance measures on histogram data and compared them experimentally on a view-based object recognition scenario. It was found that the chi-square distance seems to be the best choice for our data,

closely followed by the square-root distance.

However, the square root distance has an advantage in that once we have taken the square root of our CCFM, we can proceed by using the regular Euclidean distance. This makes it possible to use the distance measure in algorithms based on non-linear least squares optimization like the tracking from Sect. 3.3 or the pose estimation procedure to be presented in Chapter 11.

Chapter 6

Channels and Markov Models

...where the Maximum Entropy principle from Chapter 2 turns out to be the key in connecting the Channel Vector to Bayesian networks. Unfortunately, the principles proposed show to be hard to combine into a complete, competitive inference algorithm. This chapter goes in a direction different from the remainder of the thesis, but is still considered an essential piece of the global channel picture.

Probabilistic Graphical Models are a powerful modeling tool for many problems in artificial vision and other areas. Two important special cases of graphical models are Hidden Markov Models (HMM) and Markov Random Fields (MRF). The HMM is the underlying model for most tracking algorithms, and the Kalman filter can be derived from the special case of HMMs where the probabilities are all Gaussian. Markov Random Fields have been successfully used for image restoration [11], segmentation [62] and interpretation [22].

The Kalman filtering algorithm uses an analytical solution to the estimation problem, but for more complicated graphical structures or non-Gaussian models no such simple solutions are available. Most work on general graphical models is therefore performed in a discrete setting, where each node in the probabilistic structure can take one out of several discrete labels. However, in many application areas including image processing, the underlying signals are continuous in nature, and the amount of discretization introduced is rather arbitrary. Consider for example depth information in stereo vision or image intensities. There has been several attempts at representing real-valued labels and continuous density functions in belief propagation methods using ideas from particle filtering [50] or mixtures of Gaussians [86].

This chapter examines what happens if the hard discretizations are replaced with soft histograms using channel coding techniques. Recall that using soft histograms, peaks of the PDF can be represented with an accuracy higher than the bin spacing. Since the complexity of discrete message passing is quadratic in the number of labels (or bins in the histogram), it would be of great advantage if the

state space resolution could be reduced without impairing the accuracy. As an example, if the resolution can be reduced with a factor 4 in each dimension of a 3D state space, the total number of bins required is reduced by a factor $4^3 = 64$ and the time consumption by a factor $64^2 = 4096$.

First, I present a brief review of graphical models and the belief propagation algorithm. In later sections, I examine what happens when using channel vectors to represent messages and discuss how belief propagation using channel vectors can be implemented. Finally, in Sect. 6.4 the method is evaluated experimentally.

Note that this chapter should not be seen as presenting a complete method, but rather as exploring the idea of combining channels and Bayesian networks, identifying possibilities and key problems. There is a vast literature on the subject of graphical models, dealing with exponential families of distributions in general. Viewed in this context, the use of channel basis functions is only a special case of a more general theory. Thus, the contribution of this chapter is rather limited, and consists mainly in drawing a link between channel vectors and probabilistic graphical models and analyzing the problems and possibilities of our special basis functions.

6.1 Models and Algorithms

6.1.1 Graphical Models

A very elegant and general formulation of probabilistic graphical models and algorithms is available in the context of *factor graphs* [63]. Here, I present a briefer and less general version of the theory, suitable for the scope of this chapter.

A Probabilistic Graphical Model (PGM) is a graph G where each node is a random variable that is statistically dependent of other nodes in the graph only through its neighbors (the Markov property). Usually, the values of some nodes are known, and the rest are to be found. Let $\mathcal{N}$ be the set of node connections, such that $(i, j) \in \mathcal{N}$ if there is an edge between node i and j. Let $\mathcal{N}(i)$ denote the set of neighbors to node i. If the maximal *clique*[1] size is 2, the probability of a certain node labeling $\mathbf{x}$ can be factorized to

$$p(\mathbf{x}) = \prod_{(i,j)\in\mathcal{N}} \psi_{i,j}(x_i, x_j) \ , \tag{6.1}$$

where $\psi_{i,j}$ is a pairwise *compatibility function* of neighboring nodes i, j. These compatibility functions can be defined either from some real probabilistic knowledge of the problem or in a heuristic manner.

One typical model is the Markov Random Field (MRF) illustrated in Fig. 6.1. This model is useful e.g. for image denoising. Each of the shaded nodes represents an observed pixel, and each of the white nodes an unknown true pixel. These unknown nodes are often referred to as hidden nodes. If i, j are two true pixel nodes, the function $\psi(i, j)$ represents our prior knowledge about graylevels of adjacent pixels – for example, similar graylevels are more common than discontinuities. On

[1] A clique is a set of nodes which are all connected directly to each other.

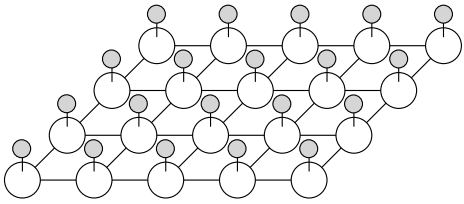

Figure 6.1: Illustration of a Markov Random Field for modeling an image. The white circles represent true (unknown) pixel values, and the shaded circles are observed pixel values.

the other hand, if i is a true pixel and j is the related observation, $\psi(i,j)$ represents the statistical relation between an observed pixel and the true value. This relationship depends on what type of noise we expect (Gaussian, salt & pepper etc). In an image denoising application, we would be given the value of all observation nodes, and the problem is to find the values of the *hidden* nodes that maximize some quality measure based on the joint probability $p(\mathbf{x})$. This quality measure determines which algorithm to choose.

6.1.2 Belief Propagation

Let all observed nodes collectively be called $\mathbf{y}$. One way of finding a good solution to the labeling problem is to start by deriving the *marginal posterior probability* $p_{x_i|\mathbf{y}}(x)$ of each hidden node x_i, i.e. the conditional probability of a certain node given the values of the known nodes. We treat this function as a function of x only, since the observed values $\mathbf{y}$ are fixed.

In principle, the marginal posteriors can be obtained by inserting the known node labels into (6.1) and integrating out all other labels $x_j, j \neq i$ from $p(\mathbf{x})$. In practice, there are very many nodes, and this integration must be performed in some smart way. When G has no cycles, the marginal posteriors can be computed using the well-known belief propagation algorithm. Here, a message $m_{i\to j}$ is a function of x and represents the belief from node i about the label of a neighboring node j, with consideration also of all nodes behind node i. Each message is a function of the set of labels – in our case, a function of a real variable. Messages are recursively evaluated as

$$m_{j\to k}(x_k) = \int \psi_{j,k}(x_j, x_k)\tilde{p}_j(x_j)\,\mathrm{d}x_j \quad , \tag{6.2}$$

where $\tilde{p}_j$ is the aggregated incoming message to node j:

$$\tilde{p}_j = \prod_{i\in\mathcal{N}(j), i\neq k} m_{i\to j} \tag{6.3}$$

This is illustrated in Fig. 6.2. Messages are propagated in all directions in the tree, starting at the leaves, and the marginal posterior of a given node j is the product

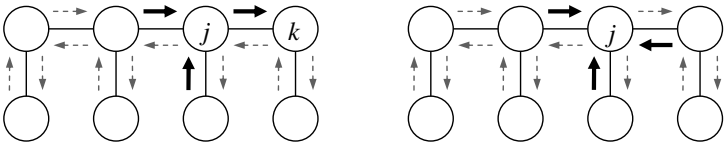

Figure 6.2: A Bayesian network. Left: One step of the belief propagation algorithm, with incoming and outgoing messages illustrated with a solid arrow. After the execution of the entire algorithm, messages between all connected nodes have been evaluated in both directions (dashed arrows). Right: The final marginal posterior is computed by multiplying all incoming messages (solid) to each node.

of all incoming messages from all directions:

$$p_{x_j|\mathbf{y}} = \prod_{i \in \mathcal{N}(j)} m_{i \to j} \ . \tag{6.4}$$

The belief propagation algorithm is often used also on graphs with cycles, even though it is not trivial to give a statistical interpretation of this. In this case, it is referred to as *loopy belief propagation*. An analysis of this method is outside the scope of this thesis, but for the interested reader I recommend the chapter on variational inference in [12]. Further sources of related methods are [71, 61, 93]. Once the marginals are obtained, we can select the values of our nodes to minimize some cost function. The *minimum mean-square* estimate (MMS) is obtained by minimizing a quadratic loss function

$$\mathcal{E}(x) = \int p_{x_j|\mathbf{y}}(\xi)(x-\xi)^2 \, \mathrm{d}\xi \ . \tag{6.5}$$

This means that we minimize the expected mean square error of our estimate. The *marginal maximum a-posteriori* estimate (MMAP) is obtained by simply choosing the x_j which maximizes $p_{x_j|\mathbf{y}}$, independently for each node. In principle, once the marginal posteriors are available, we can select our estimated node values using any quality measure which treats each node independently.

6.1.3 MAP Estimation

Another commonly used estimate is the maximum a-posteriori estimate (MAP), not to be confused with MMAP. The MAP estimate is the estimate that maximizes the joint posterior probability $p_{\mathbf{z}|\mathbf{y}}$, where $\mathbf{z}$ contains all hidden nodes. Maximizing this joint probability is not necessarily the same thing as maximizing each marginal separately. According to [63], this can be computed by a version of the message passing algorithm where the integration step (6.2) is replaced by a maximum operation:

$$m_{j \to k}(x_k) = \max_j \psi_{j,k}(x_j, x_k) \tilde{p}_j(x_j) \ . \tag{6.6}$$

When all messages have been passed, the incoming messages to each nodes are integrated using (6.4) as before, but this no longer produces marginal posteriors, but just some intermediate representation

$$q_j = \prod_{i \in \mathcal{N}(j)} m_{i \to j} \ . \tag{6.7}$$

The MAP estimate is then given by maximizing each q separately. Versions of this algorithm are called the *Viterbi* algorithm [91], (dynamic programming), and the max-sum algorithm [63].

6.2 PDF Representations and Maximum Entropy

During the execution of the belief propagation algorithm, messages need to be multiplied in (6.3) and transformed through ψ in (6.2). In the classical, discrete case, messages are vectors which can be multiplied elementwise, and (6.2) is replaced by a matrix product. In the continuous case, things get more tricky. Since we cannot represent arbitrary continuous functions, we must restrict ourselves to some subset of continuous PDFs which is representable by a finite number of parameters. Recall from Sect. 2.5 that given a number of constraints $\langle f_n, p \rangle = d_n$, the maximum entropy choice of p is

$$p(x) = \exp\left(\sum_n \lambda_n f_n(x) \right) \ . \tag{6.8}$$

In this section, I use this result to put the Gaussian, the hard histogram and the channel vectors in a common framework, aiming at understanding what (6.2) and (6.3) should become when using channel coding.

6.2.1 Gaussians

Assume that all we know about $p(x)$ is the first two moments $c_1 = \int x p(x)\,\mathrm{d}x$ and $c_2 = \int x^2 p(x)\,\mathrm{d}x$. Using the maximum entropy principle, we get $p(x) \propto \exp(\lambda_1 x + \lambda_2 x^2)$, which is a Gaussian distribution. This famous result gives a theoretical motivation for approximating a PDF with a Gaussian when only the mean and variance is known. In this case, it is possible to express the relationship between $\mathbf{c}$ and $\boldsymbol{\lambda}$ analytically, and it is easy to verify that $\lambda_1 = c_1/(c_2 - c_1^2)$ and $\lambda_2 = -1/(2(c_2 - c_1^2))$. Two Gaussians represented with a mean and a variance can now be multiplied by computing the λ-coefficients, adding them and switching back to c_1 and c_2.

6.2.2 Hard Histograms

Assume that we have measured a hard histogram of $p(x)$ such that our $\mathbf{c}$ vector consists of

$$\mathbf{c}[n] = \int \Pi(w^{-1}(x - \tilde{x}_n)) p(x)\,\mathrm{d}x \ = \int_{\tilde{x}_n - w/2}^{\tilde{x}_n + w/2} p(x)\,\mathrm{d}x \,, \quad \forall n \ , \tag{6.9}$$

where Π is the box function (see Sect. 1.4), $\tilde{x}_n$ are the bin centers and w is the bin width. The MEM choice of $p(x)$ is now of the form

$$p(x) = \exp\left(\sum_n \lambda_n \Pi\big(w^{-1}(x - \tilde{x}_n)\big)\right) \quad , \tag{6.10}$$

meaning that we have a piecewise constant expression in the exponent. This makes the entire $p(x)$ piecewise constant. But then

$$\mathbf{c}[n] = \int_{\tilde{x}_n - w/2}^{\tilde{x}_n + w/2} \exp(\lambda_n)\,\mathrm{d}x \quad , \tag{6.11}$$

so we must have $\lambda_n = \ln \mathbf{c}[n]/w$. In this case, $p(x)$ can truly be treated as a discrete distribution, and two PDFs can be added and multiplied just by adding and multiplying the histograms.

6.2.3 Soft Histograms

Now consider what happens if we create a histogram where the bins are soft and overlapping. Let

$$\mathbf{c}[n] = \frac{1}{I}\sum_{i=1}^{I} B_n(x_i) \approx \int B_n(x)p(x)\,\mathrm{d}x \quad , \tag{6.12}$$

where $B_n(x) = B(x - \tilde{x}_n)$ is the channel basis function like in previous chapters. I use the second-order B-spline kernel (see Sect. 2.3.1 and Appendix B) as the main example. From a channel vector $\mathbf{c}$, the MEM choice of $p(x)$ is

$$p(x) \propto \exp\left(\sum_n \lambda_n B(x - \tilde{x}_n)\right) \quad . \tag{6.13}$$

Unfortunately, there is no simple closed-form solution for finding the vector $\boldsymbol{\lambda}$ from a channel vector $\mathbf{c}$. In Sect. 2.5, this transformation was done iteratively using a Newton method. In contrast to the hard histogram, this interpretation does not let us perform multiplications directly on the histogram $\mathbf{c}$. In order to multiply two PDFs represented by channel vectors, we need to compute and sum the exponential parameters, and then move back to the channel domain.

Reconsider (6.13). Since $B(x)$ is a piecewise quadratic function, so is the entire exponent of (6.13). This means that $p(x)$ is actually a *piecewise Gaussian* function. This is an interesting and perhaps surprising observation. But since we know that global quadratic measurements result in a Gaussian PDF, and that the B-splines make local quadratic measurements of the PDF, it is natural that we end up with a locally Gaussian-looking function. In contrast, the popular Gaussian Mixture Models (GMMs) are *not* piecewise Gaussian, since the sum of two Gaussians is not again a Gaussian.

6.3 Message Propagation using Channels

As a step in the belief propagation algorithm, we need to propagate the aggregated messages at a current node through ψ to obtain the new outgoing message:

$$m(x_{\text{out}}) = \int \psi(x_{\text{in}}, x_{\text{out}}) \tilde{p}_{\text{in}}(x_{\text{in}}) \, \mathrm{d}x_{\text{in}} \quad . \tag{6.14}$$

This is essentially (6.2), but with some indices dropped to simplify the notation. In general, the resulting $m(x_{\text{out}})$ is not in the exponential family of $\tilde{p}_{\text{in}}(x_{\text{in}})$. In order to stay within this family, we choose instead an output message $\tilde{m}(x_{\text{out}})$ within our exponential family, and choose this $\tilde{m}$ such that $\langle m, B_i \rangle = \langle \tilde{m}, B_i \rangle, \forall i$, meaning that m and $\tilde{m}$ have the same soft histogram representation. This choice of $\tilde{m}$ minimizes the KL-divergence $D(m||\tilde{m})$ subject to $\tilde{m}$ being in the desired exponential family [71].

Each coefficient $\mathbf{c}_{\text{out}}[n]$ of the soft histogram $\mathbf{c}_{\text{out}}$ representing the outgoing message $\tilde{m}(x_{\text{out}})$ is then

$$\begin{aligned} \mathbf{c}_{\text{out}}[n] = &\langle B_n, m \rangle = \int B_n(x_{\text{out}}) \left(\int \psi(x_{\text{in}}, x_{\text{out}}) \tilde{p}_{\text{in}}(x_{\text{in}}) \, \mathrm{d}x_{\text{in}} \right) \mathrm{d}x_{\text{out}} \; = \\ &= \int \left(\int B_n(x_{\text{out}}) \psi(x_{\text{in}}, x_{\text{out}}) \, \mathrm{d}x_{\text{out}} \right) \tilde{p}_{\text{in}}(x_{\text{in}}) \, \mathrm{d}x_{\text{in}} \quad . \end{aligned} \tag{6.15}$$

By defining

$$q_n(x_{\text{in}}) = \int B_n(x_{\text{out}}) \psi(x_{\text{in}}, x_{\text{out}}) \, \mathrm{d}x_{\text{out}} \quad , \tag{6.16}$$

we get the simple relationship

$$\mathbf{c}_{\text{out}}[n] = \langle q_n, \tilde{p}_{\text{in}} \rangle \quad . \tag{6.17}$$

We see that it is enough to consider a finite number of functions q_n that measure the contribution to each output channel from the input density. In order to represent ψ efficiently, we can restrict ourselves to functions $q_n(x_{\text{in}})$ that can be expressed using a finite number of parameters. The goal is to find the equivalent of the matrix representing $\psi(x_{\text{in}}, x_{\text{out}})$ in the discrete case. The following two choices fit well with the soft histogram representation:

6.3.1 Linear q Representation

By letting

$$q_n(x_{\text{in}}) = \sum_{\nu} a_{n,\nu} B_\nu(x_{\text{in}}) \quad , \tag{6.18}$$

(6.17) becomes

$$\mathbf{c}_{\text{out}}[n] = \sum_{\nu} a_{n,\nu} \langle B_\nu, \tilde{p}_{\text{in}} \rangle = \sum_{\nu} a_{n,\nu} \mathbf{c}_{\text{in}}[\nu] \quad , \tag{6.19}$$

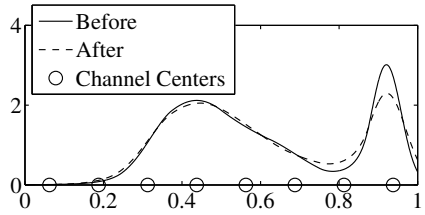

Figure 6.3: An example of a PDF represented as a soft histogram transformed through a smoothing integration kernel.

and the entire operation can be expressed as a linear operation directly on the channel vectors according to

$$\mathbf{c}_{\text{out}} = \mathbf{A}\mathbf{c}_{\text{in}} \quad . \tag{6.20}$$

This is analogous to the hard histogram case, and is intuitively appealing since the operation remains a linear mapping. An optimal $\mathbf{A}$ could be found by creating training samples of input vectors $\mathbf{c}_{\text{in}}$ and $\mathbf{c}_{\text{out}}$ and using the least squares techniques described in Chapter 7.

6.3.2 Exponential q Representation

The second approach is to consider functions $q_n(x_{\text{out}})$ that can be expressed in the same exponential form as $\tilde{p}_{\text{in}}$:

$$q_n(x_{\text{in}}) = \exp\left(\sum_\nu a_{n,\nu} B_\nu(x_{\text{in}})\right) \quad . \tag{6.21}$$

Now, the scalar product (6.17) can be computed by adding the exponential parameters of $\tilde{p}_{\text{in}}$ and q_n and integrating the corresponding continuous PDF, e.g. using some erf lookup-table operation. Each such integral operation gives one element of the output channel vector, and by repeating the process for each q_n, we get the entire soft histogram of $m(x_{\text{out}})$. The coefficients $a_{n,\nu}$ can be organized in a matrix $\mathbf{A}$ which summarizes the entire ψ.

In a preprocessing step, the functions q_n can be computed from ψ according to (6.16). The exponential parameters of each q_n can then be computed by finding the soft histogram of each q_n and do the "$\mathbf{c}$ to $\boldsymbol{\lambda}$ conversion".

In Fig. 6.3 an example is shown where a PDF is transformed through a smoothing kernel using this method. Note the accuracy compared to the channel centers.

6.3.3 Closedness of the Integral Operator

For a general linear mapping $\mathbf{A}$ operating on a channel vector $\mathbf{c}$, there is no guarantee that the output $\mathbf{c}_{\text{out}} = \mathbf{A}\mathbf{c}$ is a valid channel vector (a vector containing the mean parameters of some PDF). One example of an invalid vector is $[0, 0, 1, 0, 0]$. Since the channels are overlapping, even the most peaked PDF would produce

non-zero entries in several bins. For the exponential representation above, there is also no theoretical guarantee of closedness, even though this has not been a problem in practice so far. Finding compact, expressive representations of $\psi(x_{\text{in}}, x_{\text{out}})$ free of this problem is currently an unsolved issue and requires more attention.

6.3.4 The Max Operation for MAP

In this section, I will mention briefly what happens when trying to run the MAP algorithm instead of finding the posterior marginals. In this case, we must apply the maximum operation (6.6) instead of the integral operator. By taking the logarithm of both sides, we get

$$\log m_{j\to k}(x_k) = \max_j \log \psi_{j,k}(x_j, x_k) + \log \tilde{p}_j(x_j) \ . \tag{6.22}$$

Since there is no summation involved in the probability domain, we can work directly in the logarithic domain without a need to ever consider the channel vector $\mathbf{c}$. Most implementations of the traditional MAP algorithm for discrete labels use log-probabilities to replace the multiplications with sums, and we can basically do the same. The difference is that our $\boldsymbol{\lambda}$ coefficients should be interpreted as coefficients of a basis function expansion rather than simply discrete samples.

The tricky thing here is how to represent the outgoing log-message. Even if the involved quantities on the right hand side of (6.22) are made up from separable B-splines, the max operation is non-linear, and we cannot expect $\log m_{j\to k}$ to be a linear combination of B-splines unless some approximation is made.

This line of research has not been continued. I was looking for a compact and elegant "channelized" version of the MAP algorithm, and since there was no single obvious way to do it, the subject was dropped in favor of all other more promising issues that could be examined.

6.3.5 Relation to Variational Inference

Variational inference [12] is a rather general way of performing inference in probabilistic models by approximating the posterior distribution with some simpler form. A common choice is to assume a certain factorization, a certain exponential form or both. The factorization is useful for graphs with loops, leading to *variational message passing* [93]. A related method is *Expectation Propagation* (EP) [71, 72], also covered by [12]. The methods in this chapter could be seen as an instance of *assumed density filtering* or *moment matching* (ADF) [71, 14], which can be seen as a simple special case of EP.

Variational inference and EP talk about exponential families of probability densities in general, and by using our particular exponential form with soft bin functions, we can create channel versions of all related methods. However, the main problem persists – that there is no easy way to switch between the exponential and mean parameters. Due to the lack of an efficient method, the subject will not be explored more deeply in this thesis.

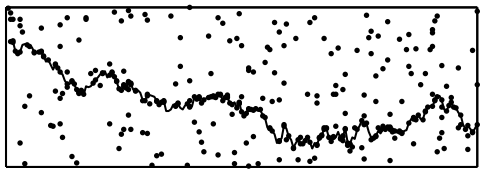

Figure 6.4: Example of a true process together with the observations.

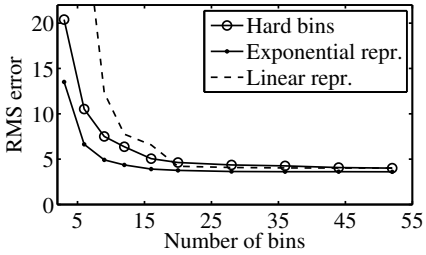

Figure 6.5: The RMS error for different number of bins, averaged over 10 runs.

6.4 Experiments

6.4.1 Hidden Markov Model

As a first experiment, a simple Hidden Markov Model was considered. A sequence of hidden states $\{x_t\}$ following a Brownian motion was constructed, from which a sequence of observations $\{\mathbf{y}_t\}$ was obtained. Formally, we have

$$x_t = x_{t-1} + v_t \tag{6.23}$$

$$\mathbf{y}_t = g(x_t) \tag{6.24}$$

where v_t is an independent Gaussian noise term. The observation function g gives a random-length vector of measurements, where each element $y_t[i]$ is distributed according to

$$p(y_t[i]) = k_1 + k_2 \exp\left[-(y_t[i] - x_t)^2/\sigma^2\right] \quad , \tag{6.25}$$

meaning that each measurement is either a normally distributed inlier or an outlier. The goal is to find the MSE estimate of the hidden state sequence by first finding the marginal distribution of the hidden state at each time step. The MSE estimate is then the expectation of each x_t from these marginals. The marginals are found using belief propagation in a forward and a backward step.

To have an "ideal" method to compare with, the state space was quantized to a fine grid of 200 levels, such that the basic discrete belief propagation algorithm on this fine grid gives the optimal solution. This is referred to as the "high resolution" method in the experiments. In addition to this, the state space was

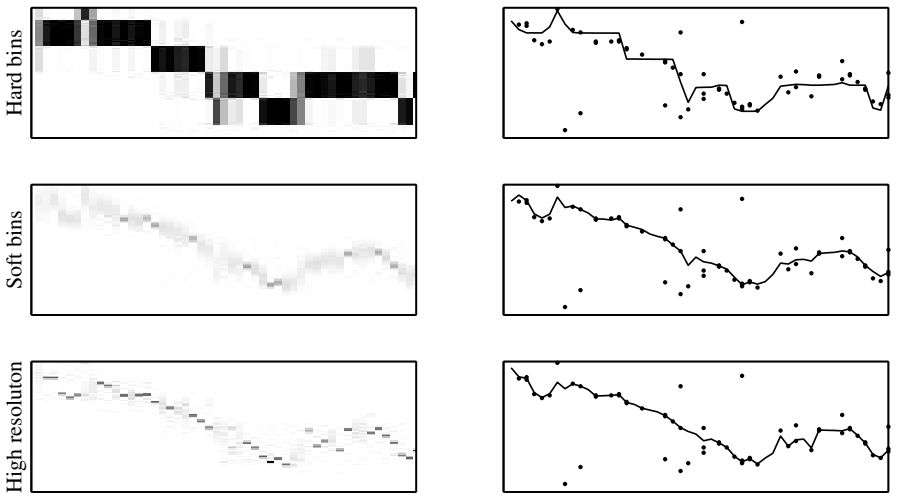

Figure 6.6: Illustration of the posterior marginals (left) and resulting MMAP trajectories (right) for the three methods in a small region of the state space.

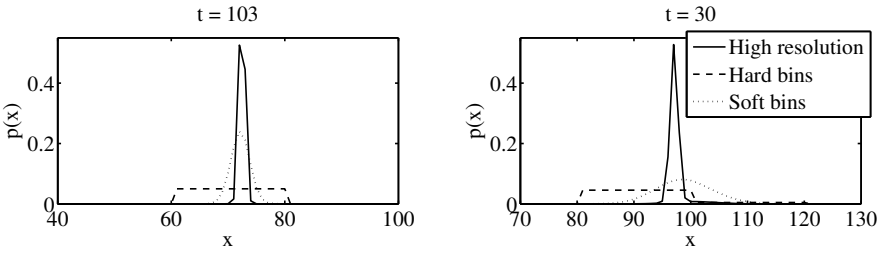

Figure 6.7: Zoomed 1D plot of the marginals at two distinct time steps.

further quantized down to just 10 bins, using both hard and soft histograms. The true state sequence and observations of a typical run is shown in Fig. 6.4. The RMS errors for different number of bins for the hard and soft histogram are shown in Fig. 6.5 using both the linear and exponential representation of ψ. The result was averaged over 10 runs for each bin configuration. Since the exponential representation produced the best results, this method was selected for a more detailed examination.

Figure 6.6 shows a small part of the posterior densities and the resulting MSE state sequence estimates of the example in Fig. 6.4. We clearly see the quantization caused by the hard bins. On the other hand, the soft histogram method is faithful to the measurements and gives a result more similar to the high-resolution method.

Figure 6.7 shows the posterior at two distinct time steps as 1D plots, to further visualize the representative power of soft histograms. In the left plot, the high-resolution marginal is a peak much narrower than the bin spacing, which is

represented well in the soft histogram. This plot is taken from a time t when the trajectory passes exactly through a measurement. In the right plot, there is more uncertainty about the exact position of the trajectory, visible as a small tail at the right side of the peak of the high-resolution posterior. The soft histogram is not able to represent this, and gives a more blurred PDF. However, the position of the maximum is still quite accurate compared to the hard histogram.

6.4.2 Boundary Detection

As a more realistic demonstration, the method was applied to boundary detection in ultrasonic imagery[2]. The image is of a blood vessel, and the objective is to locate the boundary between two different types of tissue. In [65], this problem was solved using dynamic programming, which can be viewed as a special case of the Viterbi algorithm for MAP estimation in a hidden Markov model. Each column of the image is viewed as an observation, and the position of the boundary is the sought hidden variable.

To test the soft histogram method, I look for the MMAP estimate of the boundary position. In absence of a true statistical model and in order to keep things simple, I constructed a heuristic model from the Sobel filter response, such that the probability of a boundary position y_x at column x given the observed image column obs_x is

$$p(y_x|\text{obs}_x) = k_1 \exp\left[-k_2 I_y(x, y_x)\right] \quad , \tag{6.26}$$

where I_y is the vertical component of the image gradient. Two adjacent boundary positions are related by

$$\psi(y_x, y_{x+1}) = k_3 \exp\left[-k_4 (y_x - y_{x+1})^2\right] \quad . \tag{6.27}$$

The constants k_i were selected manually for good results.

The space of boundary positions was quantized to 16 levels, and the marginals were computed using both hard and using soft histograms with the exponential representation of ψ. Each marginal was then maximized to produce a boundary position for each image column. The qualitative results are shown in Fig. 6.8 with the bin centers displayed to the left. The superior resolution produced by the soft histograms can clearly be seen, even though the result is not perfect.

6.5 Discussion

Using soft histograms instead of hard discretizations is intuitively appealing. The maximum entropy principle provides a natural framework in which to link the histograms to continuous density functions. Within this framework, hard discretizations, Gaussian representations and soft histograms are just three different instances of the same underlying principle. The equivalents of multiplying two PDFs and transforming a PDF through an integration kernel have been examined,

[2]The image was provided by Prof. Tomas Gustavsson, Chalmers University of Technology.

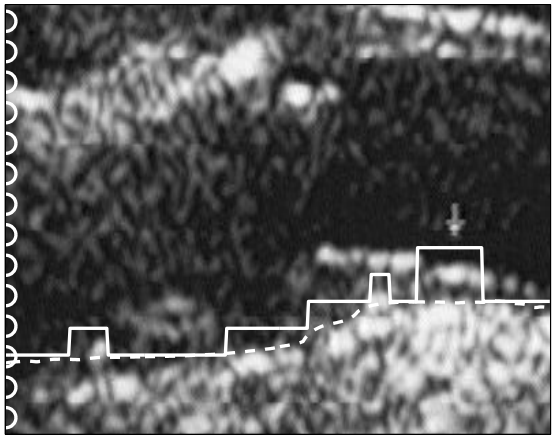

Figure 6.8: Detection results in an ultrasonic image using both hard quantization and soft histograms.

which are the two key operations of belief propagation methods. Furthermore, the neurological basis of the channel representation makes this combination a plausible candidate for how information is represented, transported and integrated in the human visual system.

There is however still some open questions regarding the representation of the integral operator, as discussed in Sect. 6.3, and the computational complexity of the conversion between $\mathbf{c}$ and $\boldsymbol{\lambda}$ is prohibitive. In the current implementation, the conversion is performed using an iterative method, where each step involves computing integrals over Gaussians. For low state-space dimensionalities, this can be done using lookup-table techniques, but for higher dimensionalities, there is currently no satisfying solution. For some applications, like gray-scale image reconstruction and stereo, one-dimensional state spaces are sufficient.

As often in computer vision, there is a competition between a straight-forward simple approach and a more sophisticated but slower approach. A comparison of which method offers the best accuracy using the same amount of computation should be made before drawing any conclusions on the usefulness of this approach. This has not been done within my PhD work.

Part II

Learning

Chapter 7

Associative Networks

...where the Channel Vector gets involved in machine learning for the first time, except from the simple nearest-neighbor approach in Chapter 5. We find that incremental updating is more fun than batch mode processing, and briefly discuss the relationship to the Hough transform and similar methods.

In [40, 52], a learning method using channel coding was presented. The basic idea is to encode both the input and the output samples into channel vectors, and then find a linear mapping using standard least-squares techniques, but with a non-negativity constraint as a regularizer. This gives the method some interesting properties like being able to represent discontinuities and being robust against outliers in the training data. In this chapter, the basic learning method is reviewed.

The original method was presented as a batch-mode method, where all training data were assumed to be available at once. In this chapter, I also present an incremental version of the method, and compare some versions of the method experimentally. For a cognitive system operating in a continuous environment, the number of training samples is expected to be huge, and incremental processing is believed to be much more efficient than storing all samples for later batch processing. Channel coding is suitable for such incremental processing [39]. Finally, in Sect. 7.3, the relation between associative networks and *voting methods* like the Hough transform is discussed briefly.

7.1 Associative Networks

This section reviews the associative networks, as presented in [52]. This is a general method to associate input and output channel vectors using a linear mapping. Let $\mathbf{a}$ and $\mathbf{u}$ be the channel-coded input and output respectively. The objective is to find a linear mapping $\mathbf{C}$ such that $\mathbf{u} = \mathbf{C}\mathbf{a}$. The matrix $\mathbf{C}$ is referred to as a *linkage matrix*. Assume that we are given T training samples $\mathbf{a}_t, \mathbf{u}_t$. These can be

organized in matrices $\mathbf{A}, \mathbf{U}$:

$$\begin{aligned}\mathbf{A} &= [\mathbf{a}_1 \ \ldots \ \mathbf{a}_T] \\ \mathbf{U} &= [\mathbf{u}_1 \ \ldots \ \mathbf{u}_T] \ .\end{aligned} \tag{7.1}$$

The matrix $\mathbf{C}$ is now found by solving the constrained least-squares problem

$$\min_{\mathbf{C}\geq 0} \sum_t \|\mathbf{C}\mathbf{a}_t - \mathbf{u}_t\|_{\mathrm{F}}^2 = \min_{\mathbf{C}\geq 0} \|\mathbf{C}\mathbf{A} - \mathbf{U}\|_{\mathrm{F}}^2 \ , \tag{7.2}$$

where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius matrix norm. The positivity (monopolarity) constraint introduced on $\mathbf{C}$ ensures that the output $\mathbf{u}$ will always be positive, and leads to a sparse matrix $\mathbf{C}$. This sparseness is useful both from a computational perspective and as a regularization to avoid over-fitting. The minimization problem is solved using a *projected Landweber* method, leading to the iterative solution strategy

$$\mathbf{C} \leftarrow \max(0, \mathbf{C} - (\mathbf{C}\mathbf{A} - \mathbf{U})\mathbf{A}^{\mathrm{T}}\mathbf{D}) \ , \tag{7.3}$$

where $\mathbf{D}$ is a preconditioner and is taken as $\mathbf{D} = \mathrm{diag}^{-1}\{\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{1}\}$. This is the basic method, but alternatives using different normalizations are also presented in [52].

To use channel-coded inputs is nothing new and can be seen as a special case of RBF networks with uniformly spaced, fixed kernels. The interesting part begins when also the outputs are channel coded. One key property of this is that outputs which are sufficiently different will never be mixed together. As an example, consider a situation where the inverse kinematics of a robotic arm is to be learned. The input to the network is the desired 3D position and orientation of the end-effector (configuration), and the output is the angles of each joint (control command). Often, a single configuration can be reached by several different control commands. Taking the mean value of these commands produces something which is not related to the desired configuration. This kind of non-uniqueness may occur any time when the inverse of some physical phenomenon is to be learned. By channel-coding the outputs, we can make sure that different cases are kept separate.

This is illustrated in Fig. 7.1, where a non-unique function is approximated with channel-coding in the output space. In the overlapping region, the channel-coded output consists of two peaks, and the decoding selects one of the peaks. Most traditional methods would average the two peaks and produce something which is rather different from the training data.

7.2 Incremental Learning

7.2.1 Basic Method

Taking the transpose in (7.2) reveals the similarity to a traditional least-squares problem:

$$\min_{\mathbf{C}\geq 0} \|\mathbf{A}^{\mathrm{T}}\mathbf{C}^{\mathrm{T}} - \mathbf{U}^{\mathrm{T}}\|_{\mathrm{F}}^2 \ . \tag{7.4}$$

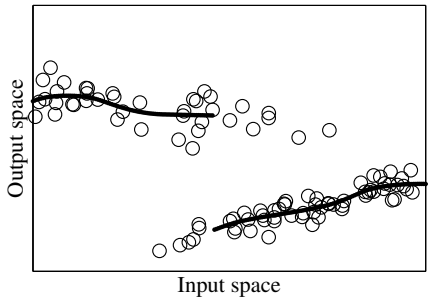

Figure 7.1: Illustration of the behavior of associative networks. The circles represent training samples, and the piecewise continuous curves are the estimated output of all possible inputs.

For a moment, I will ignore the positivity constraint on $\mathbf{C}$. These will be reintroduced in Sect. 7.2.2.

Each column $\mathbf{c}_n$ of $\mathbf{C}^{\mathrm{T}}$ corresponds to one column of $\mathbf{U}^{\mathrm{T}}$ and a particular output channel. These channels can be processed independently, and the traditional normal equations can be formulated separately for each $\mathbf{c}_n$, but it will be more convenient to treat all channels at once, giving the normal equations in matrix form:

$$\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{C}^{\mathrm{T}} = \mathbf{A}\mathbf{U}^{\mathrm{T}} \ . \tag{7.5}$$

Now define

$$\mathbf{G} = \mathbf{A}\mathbf{A}^{\mathrm{T}} = \sum_{t=1}^{T} \mathbf{a}_t {\mathbf{a}_t}^{\mathrm{T}} \tag{7.6}$$

$$\mathbf{H} = \mathbf{U}\mathbf{A}^{\mathrm{T}} = \sum_{t=1}^{T} \mathbf{u}_t {\mathbf{a}_t}^{\mathrm{T}} \ . \tag{7.7}$$

The unconstrained problem is now $\mathbf{C}\mathbf{G} = \mathbf{H}$. All information from the training data required for finding $\mathbf{C}$ is available in the matrices $\mathbf{G}$ and $\mathbf{H}$. The sizes of $\mathbf{G}$ and $\mathbf{H}$ only depend on the number of channels on the input and output side. Thus, these matrices do not grow as the number of samples increases, and are hence suited to be updated incrementally. The update rule is

$$\mathbf{G}^{(t)} \leftarrow \gamma \mathbf{G}^{(t-1)} + \mathbf{a}_t {\mathbf{a}_t}^{\mathrm{T}} \tag{7.8}$$

$$\mathbf{H}^{(t)} \leftarrow \gamma \mathbf{H}^{(t-1)} + \mathbf{u}_t {\mathbf{a}_t}^{\mathrm{T}} \ . \tag{7.9}$$

We see that the elements of $\mathbf{G}$ and $\mathbf{H}$ are accumulators, summing the products of distinct channels. Here, a *forgetting factor* γ has been introduced. If $\gamma = 1$, all training samples will be weighted equally, and if $\gamma < 1$, old training samples will

be down-weighted exponentially. The elements of $\mathbf{G}$ and $\mathbf{H}$ are

$$\mathbf{G}[i,j] = \sum_{t=1}^{T} \gamma^{T-t} \mathbf{a}_t[i]\mathbf{a}_t[j] \tag{7.10}$$

$$\mathbf{H}[i,j] = \sum_{t=1}^{T} \gamma^{T-t} \mathbf{u}_t[i]\mathbf{a}_t[j] \ , \tag{7.11}$$

where $\mathbf{a}_t[i], \mathbf{u}_t[i]$ is the value of channel i in training sample t for the input and output respectively. Note that $\mathbf{G}$ is symmetric. Also consider the case where all input samples $\mathbf{a}_t$ are channel encodings of single values. Since the kernel function of the channel coding has compact support, we have that $\mathbf{a}_t[i]\mathbf{a}_t[j] = 0$ for i and j that are sufficiently far apart. This holds for all t, and consequently $\mathbf{G}[i,j] = 0$ for such i, j. This means that $\mathbf{G}$ is a symmetric band matrix, and hence relatively cheap to invert. Since $\mathbf{G}$ is sparse, only the non-zero elements need to be represented.

The derivation of this method is similar to ideas from the *recursive least squares* (RLS) method commonly used in adaptive filtering [46]. Using the full RLS method, the solution to the unconstrained problem could be found recursively without the need to invert the matrix $\mathbf{G}$ in each time step. However, since we are primarily interested in positive solutions of $\mathbf{C}$, this method is of limited use. Another alternative to the projected Landweber method would be to use coordinate descent. This has been examined in [32]. A detailed examination of different algorithms for solving the minimization problem is outside the scope of this thesis and has not been performed.

7.2.2 Handling the Positivity Constraint

The positivity constraint is handled by noting that the iterative solution in (7.3) can be written as

$$\mathbf{C} \leftarrow \max\left[0, \mathbf{C} - (\mathbf{C}\mathbf{G} - \mathbf{H})\ \mathrm{diag}^{-1}\{\mathbf{G}\mathbf{1}\}\right] \ . \tag{7.12}$$

During the incremental learning, we can keep track of a $\mathbf{C}$ computed using the training set we have so far. When new samples arrive, $\mathbf{G}$ and $\mathbf{H}$ are updated, and (7.12) can be run for a few iterations to obtain a new $\mathbf{C}$, using the old $\mathbf{C}$ as an initial guess. Since $\mathbf{G}$ and $\mathbf{H}$ have only been changed slightly, one might suspect that the expected number of iterations required in each step is very low. One option is to perform exactly one iteration when each new sample arrives. In this case, the entire incremental update can be written as

$$\mathbf{G}^{(t)} \leftarrow \gamma\mathbf{G}^{(t-1)} + \mathbf{a}_t\mathbf{a}_t{}^{\mathrm{T}} \tag{7.13}$$

$$\mathbf{H}^{(t)} \leftarrow \gamma\mathbf{H}^{(t-1)} + \mathbf{u}_t\mathbf{a}_t{}^{\mathrm{T}} \tag{7.14}$$

$$\mathbf{C}^{(t)} \leftarrow \max\left[0, \mathbf{C}^{(t-1)} - (\mathbf{C}^{(t-1)}\mathbf{G}^{(t)} - \mathbf{H}^{(t)})\ \mathrm{diag}^{-1}\{\mathbf{G}^{(t)}\mathbf{1}\}\right] \ . \tag{7.15}$$

To understand how a new sample affects the linkage matrix in this case, we can insert (7.13) and (7.14) into the update term from (7.15) and get

$$\begin{aligned}
&\mathbf{C}^{(t-1)}\mathbf{G}^{(t)} - \mathbf{H}^{(t)} = \\
&\mathbf{C}^{(t-1)}\big(\gamma\mathbf{G}^{(t-1)} + \mathbf{a}_t\mathbf{a}_t{}^{\mathrm{T}}\big) - \big(\gamma\mathbf{H}^{(t-1)} + \mathbf{u}_t\mathbf{a}_t{}^{\mathrm{T}}\big) \; = \\
&\gamma\big(\mathbf{C}^{(t-1)}\mathbf{G}^{(t-1)} - \mathbf{H}^{(t-1)}\big) + \big(\mathbf{C}^{(t-1)}\mathbf{a}_t - \mathbf{u}_t\big)\mathbf{a}_t{}^{\mathrm{T}} = \\
&\gamma\big(\mathbf{C}^{(t-1)}\mathbf{G}^{(t-1)} - \mathbf{H}^{(t-1)}\big) + \big(\hat{\mathbf{u}}_t^{(t-1)} - \mathbf{u}_t\big)\mathbf{a}_t{}^{\mathrm{T}} \;\; , \qquad (7.16)
\end{aligned}$$

where $\hat{\mathbf{u}}_t^{(t-1)}$ is the output of the network in step $(t-1)$ from input $\mathbf{a}_t$. Here, the first term is an update related to the error on previous training samples, accumulated in $\mathbf{G}$ and $\mathbf{H}$. The second term is an update related to the new sample $(\mathbf{a}_t, \mathbf{u}_t)$. Assume now that at some point we have the optimal linkage matrix $\mathbf{C}$, and that a new sample arrives affecting some new part of the linkage matrix. If $\mathbf{C}^{(t-1)}$ is optimal, the first term (old errors term) in (7.16) is zero. The update is only related to the new sample, and is made only in the part of the linkage matrix where there is an error in the output and where active input features are present. The preconditioner $\mathrm{diag}^{-1}\{\mathbf{G}^{(t)}\mathbf{1}\}$ is also involved in weighting the updates, but does not change the important point – that the updating is relatively local in $\mathbf{C}$. Because of this locality, the updating can be done very efficiently.

7.2.3 Maintaining Two Datasets for Early Stopping

It is well-known that linear least-squares problems of this kind suffer from semi-convergence due to over-fitting. This is a problem especially if the number of training samples is small compared to the number of parameters in the model. In [52], the iterative solution from (7.3) was run and the performance was monitored on a separate validation set. Even though the error on the training set constantly decreased, after a certain number of iterations, the error on the validation set started to increase. It was suggested to use *early stopping* as a regularization method. The network can be trained on a training set, and the error evaluated on a separate validation set. The iterative optimization (7.12) is stopped once the error on the validation starts to increase. In this section, it is shown how this behavior can be achieved in an incremental way, without keeping neither the entire training nor validation set.

Let $\mathbf{A}, \mathbf{U}$ be the channel-coded inputs and outputs of the validation set according to (7.1). The sum-of-squares error on this dataset using an arbitrary linkage matrix $\mathbf{C}$ is

$$\begin{aligned}
\mathcal{E}^2 &= \sum_{t=1}^{T} \|\mathbf{C}\mathbf{a}_t - \mathbf{u}_t\|^2 = \sum_{t=1}^{T} (\mathbf{C}\mathbf{a}_t - \mathbf{u}_t)^{\mathrm{T}}(\mathbf{C}\mathbf{a}_t - \mathbf{u}_t) = \\
&= \sum_{t=1}^{T} \mathbf{a}_t{}^{\mathrm{T}}\mathbf{C}^{\mathrm{T}}\mathbf{C}\mathbf{a}_t - 2\sum_{t=1}^{T} \mathbf{u}_t{}^{\mathrm{T}}\mathbf{C}\mathbf{a}_t + \sum_{t=1}^{T} \|\mathbf{u}_t\|^2 \;\; . \qquad (7.17)
\end{aligned}$$

This error will be used only to compare different linkage matrices, and since the last term is independent of $\mathbf{C}$, it can be dropped. With a slight abuse of notation,

the same symbol $\mathcal{E}$ will still be used. The terms in the first two sums can be rewritten using the Frobenius product, producing

$$\mathcal{E}^2 = \sum_{t=1}^{T} \left\langle \mathbf{C}^{\mathrm{T}}\mathbf{C}, \mathbf{a}_t\mathbf{a}_t{}^{\mathrm{T}} \right\rangle_{\mathrm{F}} - 2\sum_{t=1}^{T} \left\langle \mathbf{C}, \mathbf{u}_t\mathbf{a}_t{}^{\mathrm{T}} \right\rangle_{\mathrm{F}} \ . \tag{7.18}$$

By moving the summation inside the products and recognizing $\mathbf{G}$ and $\mathbf{H}$ from (7.6)-(7.7), we get the compact expression

$$\mathcal{E}^2 = \left\langle \mathbf{C}^{\mathrm{T}}\mathbf{C}, \mathbf{G} \right\rangle_{\mathrm{F}} - 2\left\langle \mathbf{C}, \mathbf{H} \right\rangle_{\mathrm{F}} \ . \tag{7.19}$$

The strategy is now to maintain matrices $\mathbf{G}_{\mathrm{tr}}, \mathbf{H}_{\mathrm{tr}}$ for the training set and $\mathbf{G}_{\mathrm{val}}, \mathbf{H}_{\mathrm{val}}$ for the validation set by assigning different proportions of the training samples to each set. When the linkage matrix $\mathbf{C}$ is needed, the iterative solution procedure (7.12) is performed using $\mathbf{G}_{\mathrm{tr}}, \mathbf{H}_{\mathrm{tr}}$. During the iterations, the error is monitored using (7.19) with $\mathbf{G}_{\mathrm{val}}, \mathbf{H}_{\mathrm{val}}$, and the $\mathbf{C}$ currently under calculation. The procedure is stopped when $\mathcal{E}^2$ starts to increase.

7.2.4 Stochastic Gradient Method

There may be cases where the number of input features is much larger than the number of output features, and where the channels overlap in such a way that the number of non-zero elements in $\mathbf{G}$ is prohibitively large. In such cases, a stochastic gradient [47] method is an alternative. The idea of stochastic gradients is to update the parameters using gradient descent, but to compute the gradient from the last training sample only. In this context, we would also like to keep the monopolarity constraint. This can be achieved by dropping the "old errors" term from (7.16), leading to

$$\mathbf{C}^{(t)} \leftarrow \max\left[0,\ \mathbf{C}^{(t-1)} - (\hat{\mathbf{u}}_t^{(t-1)} - \mathbf{u}_t)\mathbf{a}_t{}^{\mathrm{T}}\mathrm{diag}^{-1}\{\mathbf{G}^{(t)}\mathbf{1}\}\right] \ . \tag{7.20}$$

This may perhaps be more accurately described as a *stochastic projected Landweber method*, but will be referred to as a stochastic gradient method throughout this text. Assuming a normalized input vector such that $\sum_i a_t[i] = 1$, we get

$$\begin{aligned}
\mathbf{G}^{(t)}\mathbf{1} &= \left[\sum_j \mathbf{G}^{(t)}[i,j]\right]_i = \\
&= \left[\sum_j \sum_{\tau=1}^{t} \gamma^{t-\tau}\mathbf{a}_\tau[i]\mathbf{a}_\tau[j]\right]_i = \\
&= \sum_{\tau=1}^{t} \gamma^{t-\tau}\mathbf{a}_\tau = \mathbf{s}^{(t)} \ .
\end{aligned} \tag{7.21}$$

Instead of keeping track of $\mathbf{G}$, we can keep track of the discounted sum of input channel vectors $\mathbf{a}$, here denoted $\mathbf{s}^{(t)}$. This sum can easily be updated incrementally

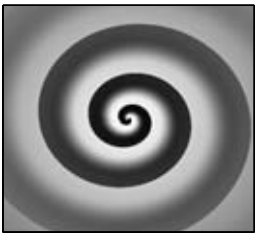

Figure 7.2: Target function. The horizontal and vertical axis are the input space, and the graylevel represent the output values.

using

$$\mathbf{s}^{(t)} \leftarrow \gamma \mathbf{s}^{(t-1)} + \mathbf{a}_t \ . \tag{7.22}$$

The update to $\mathbf{C}$ can then be written as

$$\mathbf{C}^{(t)} \leftarrow \max \left[0,\ \mathbf{C}^{(t-1)} - (\hat{\mathbf{u}}_t^{(t-1)} - \mathbf{u}_t)(\mathbf{a}_t ./ \mathbf{s}^{(t)})^{\mathrm{T}} \right] \ , \tag{7.23}$$

where ./ denotes element-wise division. To avoid division by zero, $\mathbf{s}$ can be initialized to some small value. Note that the first time channel n is non-zero, we have $\mathbf{a}_t[n]/\mathbf{s}^{(t)}[n] = 1$. This means that the update will be very fast when new input channels are activated. As channel n appears in more and more training samples, $\mathbf{s}[n]$ will start to increase, and the update of $\mathbf{C}$ will be made in smaller and smaller steps.

7.2.5 Experiments

The incremental learning methods were tested on the double spiral dataset used and described in [30]. The target function is shown in Fig. 7.2. The input coordinates were encoded using 25×25 evenly distributed separable $\cos^2$-channels, and the output using 8 $\cos^2$-channels. A maximum of 1000 random samples were used.

Several schemes were compared. The iterations were made either until $\|\mathbf{C}^{(t)} - \mathbf{C}^{(t-1)}\|_{\mathrm{F}} < \delta$, with $\delta = 0.02$ (*full*), or until the error started to increase on a separately maintained validation set (*early-stop*). In the latter case, every fourth training samples was incrementally assigned to this validation set. The linkage matrix $\mathbf{C}$ was optimized either using the previous $\mathbf{C}$ as an initial guess (*previous C*), or by doing a complete re-optimization starting from an all-zero matrix (*zero-init*). The last case may not be considered as an incremental procedure, but was included for comparison.

As another option, exactly one iteration of $\mathbf{C}$ was performed each time a new sample was added (*single*). Finally, also the *stochastic gradient* method from Sect. 7.2.4 was examined. If we assume a constant flow of training samples, we may

not have time to stop the system and perform an unknown number of iterations, so these two are perhaps the most realistic options.

The linkage matrix $\mathbf{C}$ was re-optimized after every fourth training sample, except for the *single iteration* and *stochastic gradient* methods, where it was updated after each new sample (due to the non-iterative nature of these methods). The reconstruction errors of all methods are shown in Fig. 7.3. The error was computed on a validation set consisting of a grid covering the entire data range. This validation set is of course not the same as the incrementally maintained validation set in the early-stopping methods, which explains why the error is not necessarily monotonically decreasing even for early-stopping methods.

Also note that the approximation errors of the full optimizations are quite different depending on the initial $\mathbf{C}$ used in the optimization. When the number of training samples is low compared to the degrees of freedom in $\mathbf{C}$, the learning problem is likely to be underdetermined in parts of $\mathbf{C}$, and different solutions could be selected depending on the initialization of the iterations. Even if several solutions give an equally small error on the training set, the difference in results on the entire grid may be large.

Figure 7.4 shows the number of iterations required at each re-optimization. For the full optimizations, we see that the maximum number of iterations was frequently reached. The slow convergence of the projected Landweber method has also been noted previously [32]. However, as these results show, it is not desired to iterate until convergence anyway, since the early-stopping methods clearly perform better than the full optimizations.

7.3 Relation to Voting Methods

With a *voting method*, I refer to any method which extracts some kind of primitives from an image, described with feature vectors f in some feature space. For each f, a *vote function* produces a *vote density* $d(v), v \in \mathcal{V}$, where $\mathcal{V}$ is the *vote space*. The vote densities from all individual primitives are summed together, and the output of the method is given by modes of the vote density.

As an example, consider the Hough transform. The primitives here are edge pixels characterized by image position such that $f = (x, y)$. The vote space is the space of line parameters (θ, ρ) and is represented using a histogram. The vote function is the function mapping an edge pixel in the original image to a vote density contribution, which in the Hough case maps a pixel coordinate (x, y) to a sinusoid in (θ, ρ)-space.

The Hough transform represents the vote density as a discretized histogram, but other methods may represent the density in different ways. In some methods, each primitive may vote on just one or a few number of discrete hypotheses. This is the case in the popular local features approach to object recognition. Here, the primitives can be e.g. affine-invariant descriptors extracted around homogeneous regions [78] or SIFT keypoints [67]. In that case, the vote space consists of object parameters like position, rotation and scale, and the vote density is a number of Diracs, represented by a list of points. Mode seeking is usually done by some

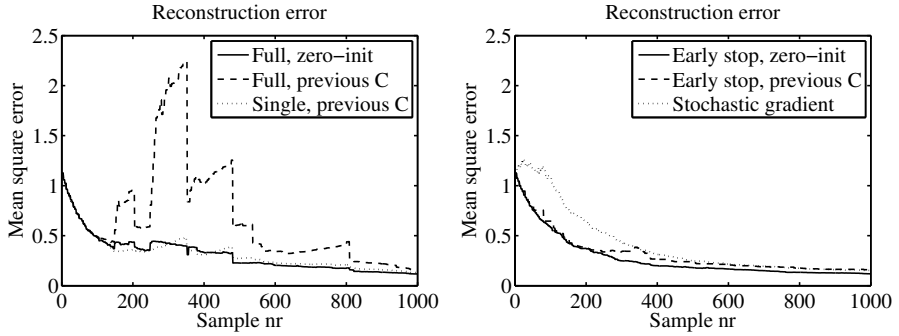

Figure 7.3: Reconstruction error for all methods on a separate validation set.

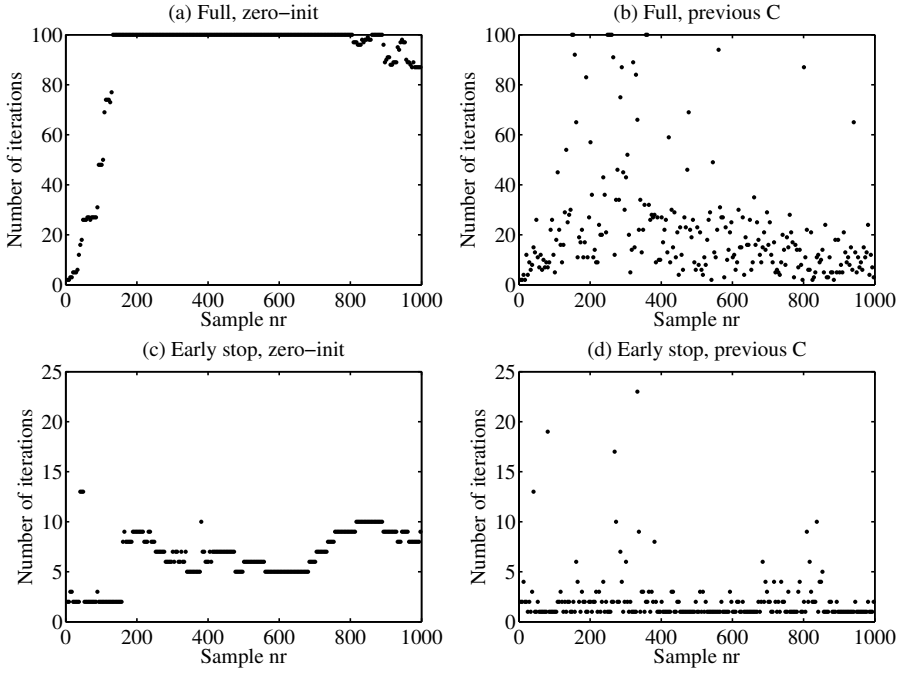

Figure 7.4: Number of iterations required at each optimization for the iterating methods.

clustering algorithm like mean shift clustering [15].

There are several properties that make the associative networks especially well-suited for voting methods. Instead of mapping every input to just a single output, as in many traditional machine learning approaches, these networks give a channel vector as output, which can represent several hypotheses with different weights. The decoding of a channel vector has been shown to correspond to robust estimation of an underlying continuous density function – in this case the vote density. Modes in vote space can be found by summing the responses $\mathbf{u}$ from each feature and decoding the resulting channel vector. This has the potential of giving a much higher accuracy than in a hard-binned histogram approach. Finally, the positivity constraint comes in naturally, since there is usually no such thing as a negative vote.

Under certain circumstances, it is possible to process all feature points in a given image at once, due to the linearity of the network. Assume that we have a set of channel coded feature vectors $\mathbf{a}_i$ generating responses $\hat{\mathbf{u}}_i$. Let $\bar{\mathbf{u}}$ be the final channel-coded vote density. We then have

$$\bar{\mathbf{u}} = \sum_{i=1}^{I} \hat{\mathbf{u}}_i = \sum_{i=1}^{I} \mathbf{C}\mathbf{a}_i = \mathbf{C} \sum_{i=1}^{I} \mathbf{a}_i = \mathbf{C}\bar{\mathbf{a}} \ . \tag{7.24}$$

We see that all encoded feature vectors can be combined into a single vector $\bar{\mathbf{a}}$, which is fed through the network once. If the number of primitives is large, this is significantly more efficient than processing each one individually. The resulting $\bar{\mathbf{u}}$ is the channel-coded vote density, which can be decoded by any available decoding algorithm.

Using associative networks, the behavior of Hough transform methods can be learned from examples. This is in the spirit of the COSPAL project (see Sect. 1.3), where one goal was to replace engineering with learning in artificial cognitive systems. Some initial exploration of this idea was made in the technical report [60], but these experiments are not reproduced in the thesis since this direction of research was not followed much further. Instead, I shifted focus to the correspondence-free learning setting, which is treated in Chapter 8.

7.4 Discussion

This chapter has shown how to update the associative networks incrementally. The full power of the recursive least squares method was not used because of the positivity constraint on the linkage matrix. In the RBF literature, recursive orthogonal methods can also be found, which show improved numerical properties, see e.g. [36]. In the future, the application of such methods to the associative networks should be examined. We have also touched the application of associative networks in Hough-transform methods. What remains to be done is a detailed study of the accuracy of the line detection compared to the bin spacing.

In the next chapter, we continue to analyze the behavior of associative networks using similar techniques but in a different setting.

Chapter 8

Correspondence-Free Learning

...where we try to relax some of the assumptions of supervised learning in order to create something new and exciting. It is found that in order to learn something, it is not necessary to know what belongs together. Correspondence is a luxury we can do without.

Traditional supervised learning approaches [47] have mostly aimed at solving a *classification* or *regression* problem. In both cases, the starting point is almost always a number of corresponding examples of input and output data. In this chapter, I consider the harder problem of learning relations between two continuous spaces without actually having access to matching training samples, but only to internally unordered sets of input/output examples (Fig. 8.1). The goal is to learn the mapping by looking at several such examples. I call this problem setting *correspondence-free learning*.

A related but rather different problem is finding a parameterized mapping (e.g. a homography) between two spaces, given only a single set of unordered points. For this problem, robust methods like RANSAC [29, 45], have been highly successful. Other related approaches are [4, 18], all looking for parameterized mappings. In [20], a minimum-work transformation between two point sets is sought instead of a parameterized mapping. All these approaches have in common that they start out from just a single set of points in each domain. As a result, the types of transformations that can be obtained are very limited. In this chapter, we seek a virtually non-parametric transformation, but assume having access to a large number of sets of unordered points as training data. Despite an extensive literature search, I am not aware of any other work directed at this problem formulation.

The actual algorithm is in fact identical to the associate networks from Chapter 7, the only difference being that the inputs and outputs within each group are summed before sending them as training examples to the system. The contribution of this chapter is in analyzing the correspondence-free behavior theoretically and experimentally.

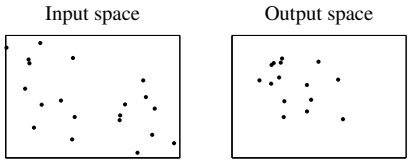

Figure 8.1: One training example of a 2D problem: 15 points in each space, with unknown correspondence structure + 5 outliers in the input space.

The correspondence-free learning problem is expected to be encountered frequently by self-organizing cognitive systems. A discrete example is language acquisition, where a child hears a lot of different words while observing a lot of different events in the world, having no idea which word should map to which particular aspect of its experiences. A more continuous example is in learning the *perception-action map*. In this context, a percept can be understood as a feature of a visual input, e.g. an edge segment or an image region. A cognitive system is confronted with a large number of percepts observed simultaneously. Each of these percepts transforms as a result of an action. Given a percept list $\mathbf{p}_t$ at time t and another list $\mathbf{p}_{t+1}$ at time $t+1$ as a result of some action $\mathbf{a}$, it is desired to learn the mapping $(\mathbf{p}_t, \mathbf{a}) \rightarrow \mathbf{p}_{t+1}$ without necessarily knowing a priori which percepts from the two time instances are corresponding. As a final example, consider the temporal credit assignment problem in reinforcement learning [88], which is the problem of attributing a reward to some previous action. Assume that an agent generates actions that produce a randomly delayed reward. By taking the set of all actions performed and all rewards given in the previous T time steps, we know that some of these actions correspond to some reward, but the exact correspondence structure is unknown. Using a number of such sets as training samples, the action-reward mapping could be learned.

These ideas are related to the COSPAL project (see Sect. 1.3). In COSPAL, one key issue was to minimize the amount of hard-coding and modeling and use as much learning as possible. The correspondence-free learning is one attempt in this direction.

8.1 Problem and Solution

8.1.1 Notation and Problem Formulation

Consider an input space $\mathcal{X}$ and an output space $\mathcal{Y}$. We want to find a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ given a set of training samples $\{S_t \mid t = 1 \dots T\}$. Each sample S_t is a tuple (X_t, Y_t), where X_t and Y_t are sets of input and output points:

$$X_t = \{x_{t,i} \mid i = 1 \dots I_{\text{in}}\} \subset \mathcal{X} \tag{8.1}$$

$$Y_t = \{y_{t,i} \mid i = 1 \dots I_{\text{out}}\} \subset \mathcal{Y} \ . \tag{8.2}$$

Furthermore, X_t and Y_t are divided into *inliers* and *outliers*. For each inlier $x_{t,i}$, there is an inlier $y_{t,j}$ such that $y_{t,j} = f(x_{t,i})$, where f is an unknown function that

we want to approximate. The outliers are random and independently distributed. The sets X_t and Y_t are unordered, such that we have no information about which $x_{t,i}$'s and $y_{t,j}$'s are corresponding or which are outliers. One example of a single training sample S_t for an $\mathbb{R}^2 \to \mathbb{R}^2$ problem is shown in Fig. 8.1.

It will be convenient for the later discussion to assume that I_{in} and I_{out} are the same for all t and that each S_t contains I_c corresponding (x, y)-pairs and O_x, O_y outliers in $\mathcal{X}$ and $\mathcal{Y}$ respectively, such that $I_{\text{in}} = I_c + O_x$ and $I_{\text{out}} = I_c + O_y$. The proposed method will work even if each training sample contains a different ratio of inliers and outliers, but the theoretical analysis will be more clear this way.

The goal is now to learn the function f. It is not obvious how to define a cost criterion to minimize for this problem. Instead, I adopt a bottom-up approach, first describing the proposed method and later analyzing its theoretical properties.

8.1.2 Solution using Associative Networks

To solve the correspondence-free problem, we encode all inputs and outputs in each X_t and Y_t together, and seek a direct linear transformation in the channel domain. From each training sample S_t, we define

$$\bar{\mathbf{a}}_t = \sum_{i=1}^{I_{\text{in}}} \mathbf{a}_{t,i} = \sum_{i=1}^{I_{\text{in}}} \text{enc}(x_{t,i}) \tag{8.3}$$

$$\bar{\mathbf{u}}_t = \sum_{i=1}^{I_{\text{out}}} \mathbf{u}_{t,i} = \sum_{i=1}^{I_{\text{out}}} \text{enc}(y_{t,i}) \ . \tag{8.4}$$

We now want to find a $\mathbf{C}$ according to

$$\min_{\mathbf{C}} \frac{1}{T} \sum_{t=1}^{T} \|\mathbf{C}\bar{\mathbf{a}}_t - \bar{\mathbf{u}}_t\|^2 \tag{8.5}$$

This is the same kind of problem formulation considered in Chapter 7. Analogous to Sect. 7.2, we construct the normal equations $\mathbf{CG} = \mathbf{H}$, with

$$\mathbf{G} = \frac{1}{T} \sum_{t=1}^{T} \bar{\mathbf{a}}_t \bar{\mathbf{a}}_t^{\mathrm{T}} \ , \quad \mathbf{H} = \frac{1}{T} \sum_{t=1}^{T} \bar{\mathbf{u}}_t \bar{\mathbf{a}}_t^{\mathrm{T}} \ . \tag{8.6}$$

Ideally, we would like to get a matrix $\mathbf{C}$ similar to what would have been obtained from the basic associative network method with known correspondences. To motivate (8.5) intuitively, assume that there exists a perfect $\mathbf{C}$ implementing the sought mapping f exactly, such that $\text{enc}(y) = \mathbf{C}\ \text{enc}(x)$ if $y = f(x)$. If there are no outliers in X_t and Y_t, we would also have $\bar{\mathbf{u}}_t = \mathbf{C}\bar{\mathbf{a}}_t$ because of the linearity. However, when no such exact $\mathbf{C}$ exists or when there are outliers, it is not as obvious how the solution to this problem relates to the solution to the ordered problem.

8.2 Asymptotical Properties

In this section, we examine the asymptotical properties of the method as the number of samples drawn goes to infinity. Will the chosen $\mathbf{C}$ approach that of the corresponding ordered, outlier-free problem, or will it be biased in some way?

Each training sample S_t is now viewed as a realization of a random process, where $\mathbf{a}_{t,i}, \mathbf{u}_{t,j}$ are realizations of the random variables $\mathbf{a}_i, \mathbf{u}_j$. We assume that the inliers and outliers follow the same distribution and are drawn independently. This means that all $\mathbf{a}_i$'s are i.i.d and all $\mathbf{u}_j$'s are i.i.d (but the $\mathbf{a}_i$'s and $\mathbf{u}_j$'s of course are linked together by f). In a similar way, we can view $\bar{\mathbf{a}}_t$ and $\bar{\mathbf{u}}_t$ as different realizations of the random vectors $\bar{\mathbf{a}}$ and $\bar{\mathbf{u}}$, where

$$\bar{\mathbf{a}} = \sum_{i=1}^{I_{\text{in}}} \mathbf{a}_i, \quad \bar{\mathbf{u}} = \sum_{i=1}^{I_{\text{out}}} \mathbf{u}_i \ . \tag{8.7}$$

To summarize, a "bar" always means "sum over i", and dropping the index t means "view as a stochastic variable".

8.2.1 The Ideal Ordered Problem

We would like to compare the behavior of the method to a hypothetical ideal situation. For simplicity of presentation, assume that the first I_c x's and y's in each S_t are mutually corresponding inliers, such that $y_{t,i} = f(x_{t,i})$ for $1 \leq i \leq I_c$, and that the rest are outliers. An associative network with access to the ordering structure would try to find the following $\mathbf{C}$:

$$\min_{\mathbf{C}} \frac{1}{TI_c} \sum_{t=1}^{T} \sum_{i=1}^{I_c} \|\mathbf{C}\mathbf{a}_{t,i} - \mathbf{u}_{t,i}\|^2 \ . \tag{8.8}$$

As $T \to \infty$, this expression tends towards

$$\min_{\mathbf{C}} \mathrm{E_c}\left\{\|\mathbf{C}\mathbf{a}_i - \mathbf{u}_i\|^2\right\} \ , \tag{8.9}$$

where $\mathrm{E_c}$ means the expectation over the inlier set, i.e. for $i \leq I_c$. The normal equations of this problem are $\mathbf{C}\mathbf{G}_c = \mathbf{H}_c$, with

$$\mathbf{G}_\mathrm{c} = \mathrm{E_c}\left\{\mathbf{a}_i \mathbf{a}_i{}^\mathrm{T}\right\}, \quad \mathbf{H}_\mathrm{c} = \mathrm{E_c}\left\{\mathbf{u}_i \mathbf{a}_i{}^\mathrm{T}\right\} \tag{8.10}$$

analogous to (8.6).

8.2.2 The Correspondence-Free Problem

We now return to the correspondence-free problem. The main result of this section can be summarized in the following theorem:

Theorem 8.1 *Let* $\mathbf{a}_\mu = \mathrm{E}\{\mathbf{a}_i\}, \mathbf{u}_\mu = \mathrm{E}\{\mathbf{u}_i\}$ *(which is independent of* i*). Then, as* $T \to \infty$*, the unordered problem from* (8.5) *is equivalent to*

$$\min_{\mathbf{C}} \mathcal{E}(\mathbf{C}) \tag{8.11}$$

where

$$\mathcal{E}(\mathbf{C}) = \mathrm{E_c}\left\{\|\mathbf{C}\mathbf{a}_i - \mathbf{u}_i\|^2\right\} + (I_{in} - 1)\|\mathbf{C}\mathbf{a}_\mu - k\mathbf{u}_\mu\|^2, \quad \textit{and} \tag{8.12}$$

$$k = (I_{in}I_c - I_c)^{-1}(I_{in}I_{out} - I_c) \ . \tag{8.13}$$

This holds both with and without a positivity constraint on $\mathbf{C}$.

Before jumping to the proof (given in Sect. 8.2.3), we analyze the consequences of this theorem. First assume that $O_y = 0$. Then $I_{\text{out}} = I_c$, $k = 1$, and the unordered problem becomes equivalent to the ordered problem but with the additional term $\|\mathbf{C}\mathbf{a}_\mu - \mathbf{u}_\mu\|^2$ included in the minimization. The larger I_{in} is, the more weight this term gets. As $I_{\text{in}} \to \infty$, the problem approaches a constrained minimization problem, where the first term is minimized subject to the last term being exactly zero. But $\mathbf{C}\mathbf{a}_\mu = \mathbf{u}_\mu$ is a very natural constraint, just saying that the mean output of the method should equal the true mean of the channel encoded output training samples. Furthermore, this constraint only uses up one degree of freedom of each row of $\mathbf{C}$ and is not expected to degrade the performance much.

Still assuming $O_y = 0$, we note that the number of x-outliers O_x and the number of correspondences I_c enters (8.11) only through the coefficient $(I_{\text{in}} - 1)$, which shows that increasing O_x has the same effect as increasing I_c. However, this only holds for the asymptotical solution – the speed of convergence may be degraded. Also, keep in mind that we assumed the outliers to follow the same distribution as the inliers.

Unfortunately, when $O_y > 0$ the story is different. Then $k \approx I_{\text{out}}/I_c > 1$, and suddenly the second term of (8.11) forces $\mathbf{C}\mathbf{a}_\mu$ to be larger than is motivated by the corresponding data only. There is an imbalance between the two terms of (8.11), which leads to undesired results.

8.2.3 Proof of Theorem 8.1

As $T \to \infty$, $\mathbf{G}$ and $\mathbf{H}$ from (8.6) tend towards

$$\mathbf{G} = \mathrm{E}\left\{\bar{\mathbf{a}}\bar{\mathbf{a}}^{\mathrm{T}}\right\}, \quad \mathbf{H} = \mathrm{E}\left\{\bar{\mathbf{u}}\bar{\mathbf{a}}^{\mathrm{T}}\right\} \ . \tag{8.14}$$

By combining (8.7) and (8.14) we get

$$\mathbf{G} = \mathrm{E}\left\{\left(\sum_{i=1}^{I_{\text{in}}}\mathbf{a}_i\right)\left(\sum_{i=1}^{I_{\text{in}}}\mathbf{a}_i\right)^{\mathrm{T}}\right\} \tag{8.15}$$

$$\mathbf{H} = \mathrm{E}\left\{\left(\sum_{i=1}^{I_{\text{out}}}\mathbf{u}_i\right)\left(\sum_{i=1}^{I_{\text{in}}}\mathbf{a}_i\right)^{\mathrm{T}}\right\} \ . \tag{8.16}$$

Expanding the products and swapping the sum and expectation gives

$$\mathbf{G} = \sum_{i=1}^{I_{\text{in}}}\sum_{j=1}^{I_{\text{in}}}\mathrm{E}\left\{\mathbf{a}_i{\mathbf{a}_j}^{\mathrm{T}}\right\}, \quad \mathbf{H} = \sum_{i=1}^{I_{\text{in}}}\sum_{j=1}^{I_{\text{out}}}\mathrm{E}\left\{\mathbf{u}_j{\mathbf{a}_i}^{\mathrm{T}}\right\} \ . \tag{8.17}$$

The expectation $\mathrm{E}\left\{\mathbf{a}_i\mathbf{a}_j^T\right\}$ is independent of the actual indicies i, j – what matters is only if $i = j$ or not (note that $\mathrm{E_c}\left\{\mathbf{a}_i{\mathbf{a}_i}^{\mathrm{T}}\right\} = \mathrm{E}\left\{\mathbf{a}_i{\mathbf{a}_i}^{\mathrm{T}}\right\}$, since the inliers and outliers are assumed to follow the same distribution). Thus, we can split the sum into two parts:

$$\mathbf{G} = I_{\mathrm{in}}\mathrm{E_c}\left\{\mathbf{a}_i{\mathbf{a}_i}^{\mathrm{T}}\right\} + ({I_{\mathrm{in}}}^2 - I_{\mathrm{in}})\mathrm{E}_{i\neq j}\left(\mathbf{a}_i{\mathbf{a}_j}^{\mathrm{T}}\right) \quad . \tag{8.18}$$

$\mathbf{H}$ can be treated in a similar way. $\mathrm{E}\left\{\mathbf{u}_j{\mathbf{a}_i}^{\mathrm{T}}\right\}$ is only dependent on whether $\mathbf{a}_i$ and $\mathbf{u}_j$ correspond or not. Since each training sample contains I_c correspondences, we can split the sum into

$$\mathbf{H} = I_c\mathrm{E_c}\left\{\mathbf{u}_i{\mathbf{a}_i}^{\mathrm{T}}\right\} + (I_{\mathrm{in}}I_{\mathrm{out}} - I_c)\mathrm{E_{nc}}\left\{\mathbf{u}_j{\mathbf{a}_i}^{\mathrm{T}}\right\} \quad , \tag{8.19}$$

where $\mathrm{E_c}$ takes the expectation over corresponding inliers $\mathbf{a}_i, \mathbf{u}_i$, and $\mathrm{E_{nc}}$ takes the expectation over non-corresponding pairs – inliers as well as outliers.

Note that the first expectation terms in (8.18) and (8.19) are exactly $\mathbf{G}_{\mathrm{c}}$ and $\mathbf{H}_{\mathrm{c}}$ from (8.10) of the ordered problem. Furthermore, the factors within the expectation of the last terms are independent, since non-corresponding (x, y)-pairs are assumed to be drawn independently. We can exchange the order of the expectation and the product, which gives

$$\mathbf{G} = I_{\mathrm{in}}\mathbf{G}_{\mathrm{c}} + ({I_{\mathrm{in}}}^2 - I_{\mathrm{in}})\mathbf{a}_\mu{\mathbf{a}_\mu}^{\mathrm{T}} \tag{8.20}$$

$$\mathbf{H} = I_c\mathbf{H}_{\mathrm{c}} + (I_{\mathrm{in}}I_{\mathrm{out}} - I_c)\mathbf{u}_\mu{\mathbf{a}_\mu}^{\mathrm{T}} \quad . \tag{8.21}$$

$\mathbf{C}$ only needs to be determined up to a multiplicative constant since the channel decoding is invariant to a constant scaling of the channel vectors. This means that we can normalize $\mathbf{G}$ and $\mathbf{H}$ by dividing by I_{in} and I_c respectively. We reuse the symbols $\mathbf{G}$ and $\mathbf{H}$ and write

$$\mathbf{G} = \mathbf{G}_{\mathrm{c}} + (I_{\mathrm{in}} - 1)\mathbf{a}_\mu{\mathbf{a}_\mu}^{\mathrm{T}} \tag{8.22}$$

$$\mathbf{H} = \mathbf{H}_{\mathrm{c}} + I_c^{-1}(I_{\mathrm{in}}I_{\mathrm{out}} - I_c)\mathbf{u}_\mu{\mathbf{a}_\mu}^{\mathrm{T}} \quad . \tag{8.23}$$

On the other hand, consider the normal equations of (8.11). Appendix A motivates my notation on derivatives and shows some useful results. Differentiating $\mathcal{E}(\mathbf{C})$ following (A.22) and (A.23) gives

$$\begin{aligned} \mathrm{d}\mathcal{E} \; &= 2\left\langle \mathbf{C}\,\mathrm{E_c}\left\{\mathbf{a}_i{\mathbf{a}_i}^{\mathrm{T}}\right\} - \mathrm{E_c}\left\{\mathbf{u}_i{\mathbf{a}_i}^{\mathrm{T}}\right\},\, \mathrm{d}\mathbf{C} \right\rangle_{\mathrm{F}} + \\ &\quad + 2(I_{\mathrm{in}} - 1)\left\langle \mathbf{C}\mathbf{a}_\mu{\mathbf{a}_\mu}^{\mathrm{T}} - k\mathbf{u}_\mu{\mathbf{a}_\mu}^{\mathrm{T}},\, \mathrm{d}\mathbf{C} \right\rangle_{\mathrm{F}} \quad . \end{aligned} \tag{8.24}$$

Recognizing $\mathbf{G}_{\mathrm{c}}$ and $\mathbf{H}_{\mathrm{c}}$ from (8.10) and combining the products into a single one gives

$$\begin{aligned} \mathrm{d}\mathcal{E} \; &= 2\left\langle \mathbf{C}(\mathbf{G}_{\mathrm{c}} + 2(I_{\mathrm{in}} - 1)\mathbf{a}_\mu{\mathbf{a}_\mu}^{\mathrm{T}}) - (\mathbf{H}_{\mathrm{c}} + (I_{\mathrm{in}} - 1)k\mathbf{u}_\mu{\mathbf{a}_\mu}^{\mathrm{T}}),\, \mathrm{d}\mathbf{C} \right\rangle_{\mathrm{F}} = \\ &= 2\left\langle \mathbf{C}\mathbf{G} - \mathbf{H},\, \mathrm{d}\mathbf{C} \right\rangle_{\mathrm{F}} \end{aligned} \tag{8.25}$$

with $\mathbf{G}$ and $\mathbf{H}$ from (8.22) - (8.23). From this expression, the normal equations are simply $\mathbf{C}\mathbf{G} - \mathbf{H} = 0$, which are the same normal equations as obtained in

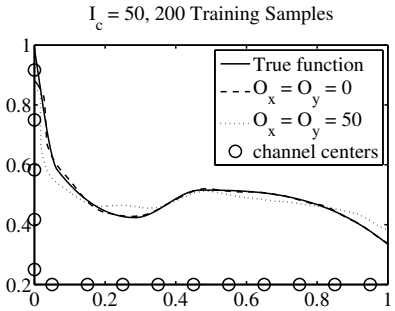

Figure 8.2: True 1D function and two examples of resulting approximations.

the correspondence-free setting. This shows that the correspondence-free setting effectively minimizes (8.11) in the unconstrained case.

For the constrained case, the projected Landweber as used in Chapter 7 is simply a gradient descent on $\mathcal{E}(\mathbf{C})$ together with a projection into the feasible set in each iteration. Since the gradient of $\mathcal{E}(\mathbf{C})$ is $\mathbf{CG} - \mathbf{H}$, we see that running the batch mode updating scheme

$$\mathbf{C} \leftarrow \max\left[0, \mathbf{C} - (\mathbf{CG} - \mathbf{H})\ \mathrm{diag}^{-1}\{\mathbf{G1}\}\right] \tag{8.26}$$

from (7.12) using $\mathbf{G}$ and $\mathbf{H}$ obtained from the correspondence-free setting is equivalent to minimizing (8.11) with a projected Landweber method. This completes the proof of Theorem 8.1.

8.3 Experiments

8.3.1 1D Example

In the first experiment, a one-dimensional function was learned using various numbers of inliers and outliers. Figure 8.2 shows the function together with the learned approximation in two different settings, with the channel centers indicated. Note that the accuracy is much higher than the channel spacing.

A number of experiments on learning speed have been performed. In all cases, the results were averaged over 30 runs. In Fig. 8.3, the RMS approximation error is shown as a function of the number of training samples. Note that the method converges faster when the number of simultaneously presented pairs increases, which is explained by the fact that the total number of (x, y)-pairs grows faster in this case. Also note that O_x does not affect the asymptotical solution, as expected, but does affect the rate of convergence. When O_y is large the method breaks down and leaves a residual error.

Figure 8.4 shows the approximation after 50 training samples plotted against the number of correspondences I_c in each sample (no outliers). We see that the benefit of using more (x, y)-pairs in each training sample saturates rather quickly. Furthermore, if I_c is very high, all vectors $\bar{\mathbf{a}}$ and $\bar{\mathbf{u}}$ have a large DC offset, and the

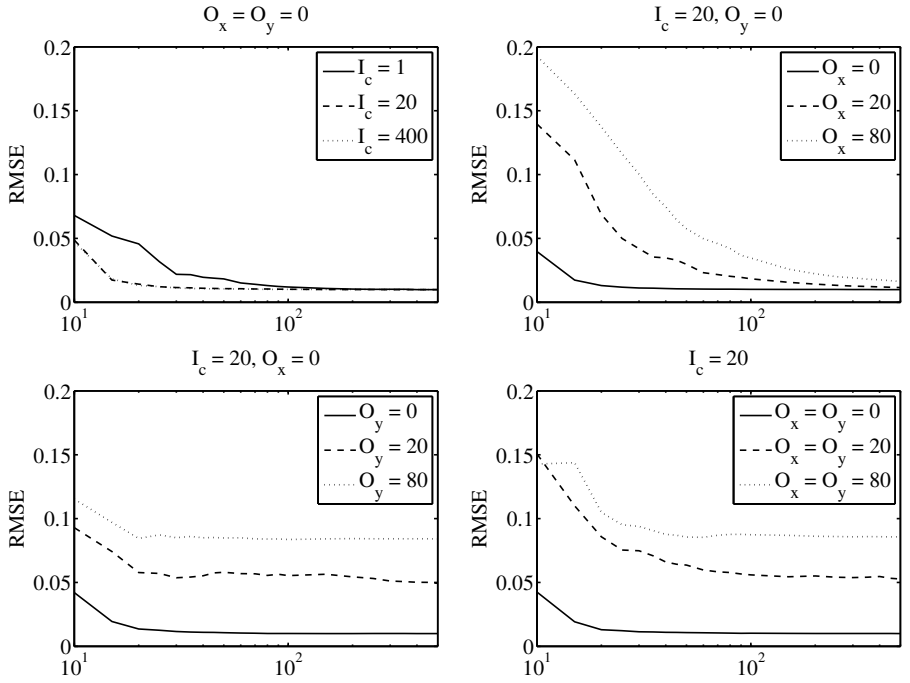

Figure 8.3: Approximation error as a function of number of training examples under various configurations (averaged over 30 runs).

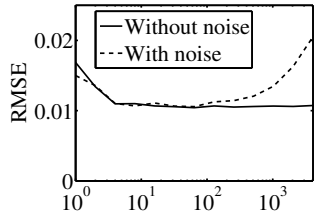

Figure 8.4: Approximation error as a function of the number of correspondences within each training example (no outliers).

significant part will be small in comparison, which can lead to numerical problems. The dashed curve of Fig. 8.4 shows the final approximation error when white Gaussian noise with a standard deviation of 1% of the mean channel magnitude has been added to the channel vectors. When I_c is large, this relatively small noise term starts to destroy the significant part of the channel vectors, which leads to an increased error.

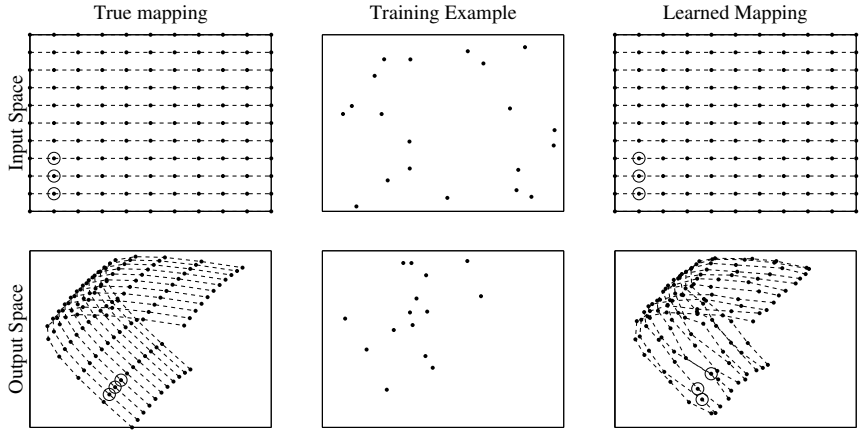

Figure 8.5: 2D example of correspondence-free learning. Some points have been marked with circles to visualize the orientation of the mapping. See the text for details.

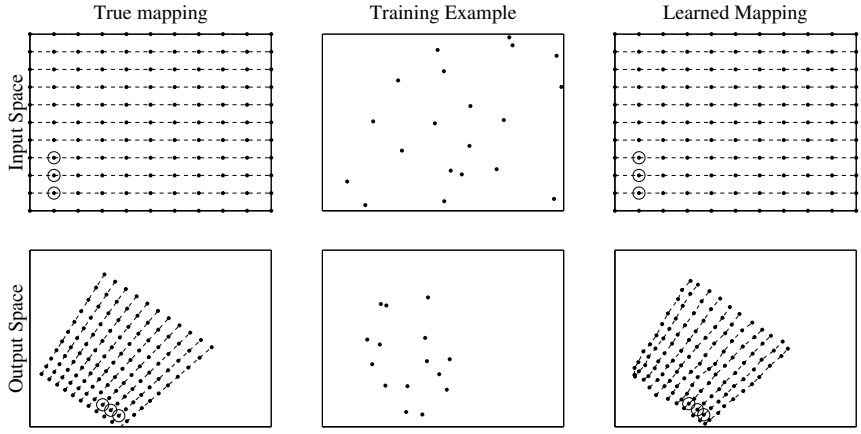

Figure 8.6: One additional 2D example of correspondence-free learning, this time the mapping to learn was a homography.

8.3.2 2D Examples

The second experiment shows the qualitative behavior of an $\mathbb{R}^2 \to \mathbb{R}^2$ mapping. Each training example consisted of 15 unordered points in both spaces, plus 5 additional outliers in the input space. Both spaces were encoded using 12×12 channels. The behavior of the system after being presented with 200 such training examples is shown in Fig. 8.5.

A second example is shown in Fig. 8.6, where the mapping to learn was a homography. Of course, this was unknown to the system, and exactly the same setup is used to learn this mapping as in the first example.

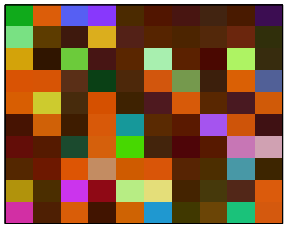

Figure 8.7: Example of training input to the color learning experiment. In this example, the user input was “brown, orange”.

Figure 8.8: Results of the color learning experiment. A random input color is shown together with the generated responses.

8.3.3 Discrete Output Space

In the final experiment, the input space was the 3D space of RGB colors and the output space was a discrete set of color labels. A training set was constructed by letting a user manually judge the dominant color(s) of a number of pixels. The user had the opportunity to select zero, one or more labels from a predefined list. The presented image often included several dominant colors. The correspondence between individual pixels and color labels is unknown. The motivation for this experiment is to show how symbolic and continuous entities can be used seamlessly within the channel-coding framework.

The images presented to the observer were constructed by creating two random dominant colors in RGB space, around which normally distributed pixel values were drawn. Within each training sample, 20% of the pixels were from one color and 50% from the other. The rest of the pixels are outliers, uniformly distributed in RGB space. Figure 8.7 shows an example of one such randomly generated training input. After being presented with 100 images of this kind together with the user input, the system was tested on some random colors. Some typical examples are shown in Fig. 8.8.

8.4 Discussion

In this chapter I have studied the rather uncommon problem of learning mappings through training data with unknown correspondence structure. A rather simple method has been presented which gives surprisingly good results. In the outlier- and noise-free case, the mapping is learned at least as quickly as in the known-correspondences case, regardless of the size of each group. Outliers in the input space following the same distribution as the inliers are suppressed, but arbitrary outliers in any domain lead to remaining errors.

The second 2D example ties to the discussion about perception-action learning in the introduction to this chapter. The input and output points could be points from the ground plane extracted from two cameras. As the system moves around, it encounters different situations, with interest points located on different positions on the ground plane, but there is never an explicit information about which point in the left image is corresponding to a certain point in the right image. There are of course several sophisticated stereo matching algorithms available that do the job perfectly, but all of them rely on a lot of engineering. The typical algorithm would find tentative correspondences, select subsets using RANSAC, find homography candidates by solving equation systems, and verify and refine promising candidates. On the other hand, in the correspondence-free learning context, everything just boils down to a linear mapping between two channel-coded spaces. This certainly follows the COSPAL philosophy of minimum application-specific engineering.

So far, the method has only been evaluated on rather artificial problems, but the kinds of situations in which the method would be applicable are indicated. Since all approaches to artificial cognitive systems have so far failed to produce anything even remotely similar to human beings in terms of learning and self-organizing capabilities, we must try to attack the problem from new angles by looking at alternative methods, structures and problem formulations. This chapter has been an attempt in that direction.

Chapter 9

Locally Weighted Regression

...where the Channel Vector will have to wait patiently while we explore another machine learning approach. However, the wait will not be too long, since in the next chapter, the locally weighted regression will be applied to channel-coded feature maps, and the two will live happily ever after.

This chapter reviews the method of *locally weighted regression* as described by Atkeson and Schaal [5]. I propose using a version of this method with an adaptive kernel width, making the method invariant to a rescaling of the input space. I also present the analytical derivative of the approximation procedure. This derivative will be needed in later chapters, when the method will be used in an iterative optimization procedure.

9.1 Basic Method

Locally weighted regression is a general method for learning in medium-dimensional spaces. Assume that we during training are given a set of T input-output pairs $(\mathbf{x}_t, \mathbf{y}_t)$. In operation mode, we are given a query $\mathbf{q}$ and want to produce an estimated output $\hat{\mathbf{y}}$. The actual training consists only of storing the training examples. At runtime, we select a number of training samples which are close to $\mathbf{q}$ in the input space and fit a local linear model to these samples. Each sample is weighted by a radially symmetric basis function centered at the query point. The principle is illustrated in Fig. 9.1.

Let B be a monotonically decreasing weighting kernel profile which is non-zero on $[0, 1]$ only. The training samples are weighted according to

$$w_t = B(s^{-1}\|\mathbf{q} - \mathbf{x}_t\|) \tag{9.1}$$

where s will be called the *kernel width*. Most w_t's will be zero, and the corresponding training samples can be disregarded. Since we are only looking at the function in a small neighborhood around $\mathbf{q}$, it is reasonable to assume that the function is

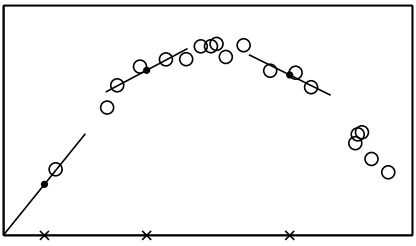

Figure 9.1: Illustration of the locally weighted regression principle. At a given query input (marked with a cross), a local linear model is fitted to the training samples, and the output is computed from this local model.

approximately locally linear. We seek a local approximation $\mathbf{f}$ of the form

$$\mathbf{f}(\mathbf{x}) = \mathbf{y}_0 + \mathbf{C}(\mathbf{x} - \mathbf{x}_0) \ . \tag{9.2}$$

The vector $\mathbf{x}_0$ can be chosen arbitrarily. It will be convenient to use $\mathbf{x}_0 = \mathbf{q}$, since then $\mathbf{f}(\mathbf{q}) = \mathbf{y}_0$. We find the $\mathbf{y}_0$ and $\mathbf{C}$ that fit the training data as well as possible by solving the weighted least-squares problem

$$\min_{\mathbf{C},\mathbf{y}_0} \sum_t w_t \|\mathbf{y}_0 + \mathbf{C}(\mathbf{x}_t - \mathbf{x}_0) - \mathbf{y}_t\| \ . \tag{9.3}$$

By constructing

$$\tilde{\mathbf{C}} = [\mathbf{C}\ \mathbf{y_0}] \tag{9.4}$$

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}_1 - \mathbf{x}_0) & \dots & (\mathbf{x}_T - \mathbf{x}_0) \\ 1 & \dots & 1 \end{bmatrix} \tag{9.5}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1 & \dots & \mathbf{y}_T \end{bmatrix} \tag{9.6}$$

$$\mathbf{W} = \mathrm{diag}(\mathbf{w}), \quad \mathbf{w} = \begin{bmatrix} w_1 & \dots & w_T \end{bmatrix}^{\mathrm{T}} \tag{9.7}$$

the problem can be rewritten compactly as

$$\min_{\tilde{\mathbf{C}}} \|(\tilde{\mathbf{C}}\mathbf{X} - \mathbf{Y})\mathbf{W}^{1/2}\|_{\mathrm{F}} \ . \tag{9.8}$$

I consider only the case where the number of training samples with non-zero weight is larger than the degrees of freedom of the local model. Unless the configuration of sample points is degenerate, $\mathbf{X}$ is full-rank, and the solution is given by

$$\tilde{\mathbf{C}} = \mathbf{Y}\mathbf{W}\mathbf{X}^{\mathrm{T}}(\mathbf{X}\mathbf{W}\mathbf{X}^{\mathrm{T}})^{-1} \ . \tag{9.9}$$

9.2 Weighting Strategies

One problem with this approach is that we do not know how many training samples that will be included within the support of B. If the kernel width is small, there

may be too few points included to produce a robust estimate of the linear model, or in the extreme case no points at all will be included. In contrast, if the weighting kernel is too large, the linear approximation may be poor. This section describes how this problem can be solved by introducing an adaptive kernel width.

9.2.1 Adaptive Kernel Width

An alternative to using a weighting kernel could be to always select the K nearest neighbors without any weighting in the spirit of the K nearest neighbor classifier [12]. The number K can be selected in a way that suits the linear model fitting. As an example, a linear model mapping $\mathbb{R}^3$ to $\mathbb{R}$ has 4 free parameters, so at least 4 points are needed to fully determine the mapping. Using more points increases the robustness against noise but may render the local linearity assumption invalid.

However, when the query point is shifted such that the set of K nearest neighbors changes, the resulting linear model could change abruptly, which causes a discontinuity in the mapping. Such discontinuities can cause problems when using the mapping together with iterative optimization algorithms, as will be the case in Chapter 11.

The proposed method places itself somewhere in between these two extremes by using a weighting kernel that is scaled according to the distance to the nearest training samples. Let the support of $B(r)$ be $[0, 1]$ and sort the training samples by their distance to $\mathbf{q}$ such that $\|\mathbf{q} - \mathbf{x}_t\| \leq \|\mathbf{q} - \mathbf{x}_{t+1}\|$ for all t. We now select the kernel width s in (9.1) as

$$s = \beta \|\mathbf{q} - \mathbf{x}_K\| \ . \tag{9.10}$$

If $\beta = 1$, this gives zero weight to the K'th sample and non-zero weight to all samples strictly closer to $\mathbf{q}$. Most of the time, this will produce $K - 1$ samples with non-zero weights, as at the rightmost query point of Fig. 9.2. However, if there are several samples with the same distance to $\mathbf{q}$ as $\mathbf{x}_K$, all these samples will get zero weight as well. This situation is illustrated in the leftmost query point of Fig. 9.2. Having to few active samples may cause the model fitting to be very unreliable. In the extreme case where all the K closest neighbors are equally far away from the query point, no single sample will have non-zero weight. Choosing β slightly larger than 1 ensures that there is *at least* K samples with non-zero weight.

Given a fixed set of training samples, each weight w_t is a continuous function of $\mathbf{q}$, while $\mathbf{C}, \mathbf{y}_0$ depend continuously on the w_t's. This ensures that the approximated function is continuous in $\mathbf{q}$, at least as long as $\mathbf{X}$ is full-rank such that the inverse in (9.9) is well-defined.

9.2.2 Reducing Noise Sensitivity

Having an adaptive kernel weight like in the previous section also has its drawbacks. As more and more training samples get included close to a given query point, the kernel width becomes smaller and smaller. If the training samples are noisy, the local behavior will be dominated by noise once the sampling density is

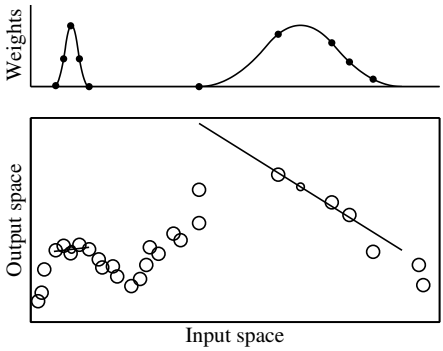

Figure 9.2: Illustration of adaptive kernel width with $K = 5, \beta = 1$. The upper plot shows the weights of the active samples at two different query points.

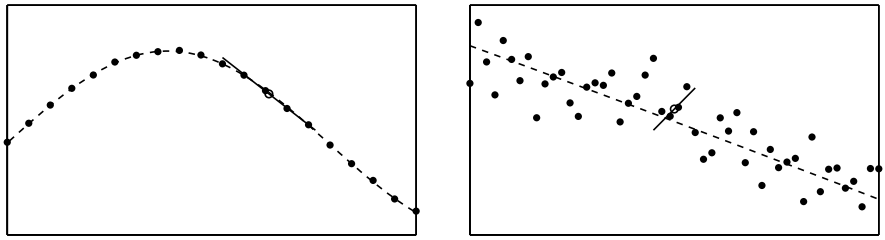

Figure 9.3: Illustration of the noise sensitivity problem when there are many training samples.

high enough, as illustrated in Fig. 9.3. This means that when the sampling density increases, it will be less and less meaningful to do a local linear approximation based on K samples only. Especially the slope of the local model may be very different from the slope of the approximated function.

In this case we would actually benefit from having a fixed kernel, which would increase the accuracy by including more and more points in the approximation. If we have some prior knowledge about the noise levels, we could prescribe a *minimum kernel width* and choose s as

$$s = \max(\beta \|\mathbf{q} - \mathbf{x}_K\|, s_{\min}) \ . \tag{9.11}$$

This is the recommended strategy. If no prior knowledge about $s_{\min}$ is available and an ad-hoc choice is not good enough, we must think about another approach. One option might be to select the scale where the sample points best appear to follow a linear model. Doing so in a computationally cheap way may be difficult, and a detailed exploration of different scale selection criteria is outside the scope of this thesis.

9.3 Analytical Jacobian

9.3.1 Motivation

At run-time, a local linear model is fitted to the nearest training points, and the approximated function value is $\mathbf{y}_0$ from this approximation. Let $\mathbf{g}$ be the function which takes a query $\mathbf{q}$ as input, runs the local model fitting, and returns the computed $\mathbf{y}_0$. At a first glance, one may believe that the derivative of $\mathbf{g}$ is $\mathbf{C}$ from the local model fitting, but this is in fact incorrect. This section derives the true analytical expression for the Jacobian of $\mathbf{g}$.

Intuitively, if we shift our query $\mathbf{q}$ a little and rerun the entire procedure, the weights of the training samples will be different, and different $\mathbf{y}_0$ and $\mathbf{C}$ might be selected. There is no theoretical guarantee that $\mathbf{y}_0$ actually moved in the direction indicated by $\mathbf{C}$. This is illustrated in Fig. 9.4, where a one-dimensional example was chosen for illustrative purposes. The unfilled circles represent training samples, and the lines are examples of linear models constructed at some query points. In the left plot, the training samples are completely random, while in the right plot they follow a smooth sinusoidal curve without noise. The solid curves show the function $\mathbf{g}$ and are constructed by computing the $\mathbf{y}_0$'s from input points on a fine grid.

We clearly see that the derivative of the local model $\mathbf{f}$ is different from the derivative of the approximation $\mathbf{g}$, even in the noise-free case. In fact, the two derivatives may even have different signs. If this model is used as part of a gradient-based optimization (which will be the case in Chapter 10 and 11), a line search on $\mathbf{g}$ in the direction specified by $\mathbf{C}$ would fail to find a descent step if the true gradient actually points in the opposite direction.

In the remainder of this section I derive an analytical expression for the Jacobian of $\mathbf{g}$ in the general, multi-dimensional case. The derivation is rather involved and may be skipped without loss of continuity. The final results are summarized in Section 9.3.5. Many general relations for derivatives in linear algebra are given in Appendix A. These rules will be frequently used here, so unless you are fluent in linear algebra differentiation, you are advised to browse through this appendix first.

9.3.2 Derivative of g

First, recall that $\mathbf{x}_0$ can be selected arbitrarily. I argued that $\mathbf{x}_0 = \mathbf{q}$ is a good choice in runtime, such that the output is given simply by $\mathbf{y}_0$. When deriving the Jacobian, we are better off keeping $\mathbf{x}_0$ fixed, since that makes the entire matrix $\mathbf{X}$ fixed (independent on $\mathbf{q}$) and greatly simplifies the derivation. The output of the method is then

$$\mathbf{g}(\mathbf{q}) = \mathbf{y}_0 + \mathbf{C}(\mathbf{q} - \mathbf{x}_0) \ . \tag{9.12}$$

Note that both $\mathbf{C}$ and $\mathbf{y}_0$ depend on $\mathbf{q}$ through the weights $\mathbf{w}$, although this dependence is not made explicit in (9.12) for compactness. Differentiating $\mathbf{g}$ with

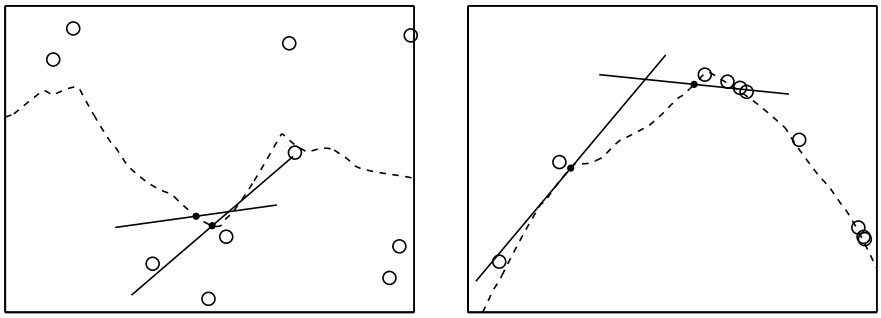

Figure 9.4: Illustration of some examples where the approximated derivative has the wrong sign. See the text for details.

respect to $\mathbf{q}$ gives

$$\mathrm{d}\mathbf{g}(\mathbf{q}) \;=\; \mathrm{d}\mathbf{y}_0 \;+\; \mathrm{d}\mathbf{C}\,(\mathbf{q}-\mathbf{x}_0) + \mathbf{C}\,\mathrm{d}\mathbf{q} \quad . \tag{9.13}$$

We evaluate the derivative at $\mathbf{q} = \mathbf{x}_0$, which makes the second term above vanish. The complete Jacobian is then

$$\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{q}}(\mathbf{x}_0) = \frac{\mathrm{d}\mathbf{y}_0}{\mathrm{d}\mathbf{q}} + \mathbf{C} \quad . \tag{9.14}$$

We see that the derivative of $\mathbf{g}$ is the derivative $\mathbf{C}$ of the local model completed with an extra term $\frac{\mathrm{d}\mathbf{y}_0}{\mathrm{d}\mathbf{q}}$, which can be seen as a compensation for the fact that the weights of the local model fitting are changed as the query point is moved. Intuitively, if the local model fits the closest points very well, the exact values of the weights should be less important. For a linear model that fits the training data perfectly, this compensating term is expected to be zero.

9.3.3 Derivative of $\mathbf{y}_0$

Recall that $\mathbf{y}_0$ is the rightmost column of $\tilde{\mathbf{C}}$ and that $\tilde{\mathbf{C}}$ is given by (9.9), repeated here for convenience:

$$\tilde{\mathbf{C}} = \mathbf{Y}\mathbf{W}\mathbf{X}^{\mathrm{T}}(\mathbf{X}\mathbf{W}\mathbf{X}^{\mathrm{T}})^{-1} \quad . \tag{9.15}$$

To simplify the notation, define

$$\mathbf{M} = \mathbf{X}\mathbf{W}\mathbf{X}^{\mathrm{T}} \quad . \tag{9.16}$$

Since we kept $\mathbf{x}_0$ fixed, $\mathbf{W}$ is the the only object in (9.15) that depends on $\mathbf{q}$. Differentiating $\tilde{\mathbf{C}}$ by using the product rule gives

$$\mathrm{d}\tilde{\mathbf{C}} \;=\; \mathbf{Y}\,\mathrm{d}\mathbf{W}\,\mathbf{X}^{\mathrm{T}}\mathbf{M}^{-1} + \mathbf{Y}\mathbf{W}\mathbf{X}^{\mathrm{T}}\,\mathrm{d}(\mathbf{M}^{-1}) \quad . \tag{9.17}$$

Differentiating matrix inverses is handled in Appendix A, and we see that $\mathrm{d}(\mathbf{M}^{-1}) = -\mathbf{M}^{-1}\mathbf{X}\,\mathrm{d}\mathbf{W}\,\mathbf{X}^{\mathrm{T}}\mathbf{M}^{-1}$. This produces

$$\begin{aligned}\mathrm{d}\tilde{\mathbf{C}} &= \mathbf{Y}\,\mathrm{d}\mathbf{W}\,\mathbf{X}^{\mathrm{T}}\mathbf{M}^{-1} - \underbrace{\mathbf{Y}\mathbf{W}\mathbf{X}^{\mathrm{T}}\mathbf{M}^{-1}}_{\tilde{\mathbf{C}}}\mathbf{X}\,\mathrm{d}\mathbf{W}\,\mathbf{X}^{\mathrm{T}}\mathbf{M}^{-1} = \\ &= (\mathbf{Y} - \tilde{\mathbf{C}}\mathbf{X})\,\mathrm{d}\mathbf{W}\,\mathbf{X}^{\mathrm{T}}\mathbf{M}^{-1} \ . \end{aligned} \tag{9.18}$$

Define $\mathbf{R} = \mathbf{Y} - \tilde{\mathbf{C}}\mathbf{X}$. This matrix is recognized as the (unweighted) residuals from the minimization problem (9.8). Since we are only interested in $\mathrm{d}\mathbf{y}_0$, we define $\mathbf{v}$ as the rightmost column of $\mathbf{X}^{\mathrm{T}}\mathbf{M}^{-1}$, which together with (9.18) lets us write

$$\mathrm{d}\mathbf{y}_0 = \mathbf{R}\,\mathrm{d}\mathbf{W}\,\mathbf{v} \ . \tag{9.19}$$

Since $\mathbf{W}$ is a diagonal matrix, we can write $\mathrm{d}\mathbf{W}\,\mathbf{v}$ as $\mathrm{diag}(\mathbf{v})\,\mathrm{d}\mathbf{w}$, which gives

$$\mathrm{d}\mathbf{y}_0 = \mathbf{R}\,\mathrm{diag}(\mathbf{v})\,\mathrm{d}\mathbf{w} \ , \tag{9.20}$$

and the complete derivative of $\mathbf{y}_0$ becomes

$$\frac{\mathrm{d}\mathbf{y}_0}{\mathrm{d}\mathbf{q}} = \mathbf{R}\,\mathrm{diag}(\mathbf{v})\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\mathbf{q}} \ . \tag{9.21}$$

As expected, if the local model fits the training samples perfectly, $\mathbf{R}$ becomes a zero matrix, and the entire term $\frac{\mathrm{d}\mathbf{y}_0}{\mathrm{d}\mathbf{q}}$ vanishes.

9.3.4 Derivatives of w

What remains now is to find the derivative of the weight vector $\mathbf{w}$ with respect to the query $\mathbf{q}$. This derivative depends on our weighting strategy. Consider first the simplest case of fixed kernel width. A certain weight w_t is here defined as

$$w_t = B(s^{-1}\|\mathbf{q} - \mathbf{x_t}\|) \ , \tag{9.22}$$

where s is the kernel width. From Appendix A, we have that $\mathrm{d}\|\mathbf{x}\| = \|\mathbf{x}\|^{-1}\mathbf{x}^{\mathrm{T}}\,\mathrm{d}\mathbf{x}$. The chain rule gives us

$$\begin{aligned}\mathrm{d}w_t &= B'(s^{-1}\|\mathbf{q} - \mathbf{x_t}\|)s^{-1}\,\mathrm{d}\|\mathbf{q} - \mathbf{x_t}\| = \\ &= B'(s^{-1}\|\mathbf{q} - \mathbf{x_t}\|)s^{-1}\|\mathbf{q} - \mathbf{x_t}\|^{-1}(\mathbf{q} - \mathbf{x_t})^{\mathrm{T}}\,\mathrm{d}\mathbf{q} \ . \end{aligned} \tag{9.23}$$

To make the notation more compact, define $\mathbf{p}_t = \mathbf{q} - \mathbf{x}_t$ and let $\tilde{}$ denote vector normalization. The entire Jacobian of $\mathbf{w}$ is then

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\mathbf{q}} = s^{-1}\begin{pmatrix} B'(s^{-1}\|\mathbf{p}_1\|)\,\tilde{\mathbf{p}}_1{}^{\mathrm{T}} \\ \cdots \\ B'(s^{-1}\|\mathbf{p}_T\|)\,\tilde{\mathbf{p}}_T{}^{\mathrm{T}} \end{pmatrix} \ . \tag{9.24}$$

If an adaptive kernel width is used according to Sect. 9.2.1, the situation is slightly more tricky. Consider the definition

$$w_t = B\left(\frac{\|\mathbf{q} - \mathbf{x}_t\|}{\beta\|\mathbf{q} - \mathbf{x}_*\|}\right) \ , \tag{9.25}$$

where $\mathbf{x}_*$ is the training sample that determines the kernel width. For compactness, define

$$\begin{aligned}\mathbf{p}_* &= \mathbf{q} - \mathbf{x}_* \\ z_t &= \frac{\|\mathbf{p}_t\|}{\|\mathbf{p}_*\|} \ .\end{aligned} \tag{9.26}$$

By use of (A.11) and (A.12) from Appendix A and the chain rule, we find that

$$\begin{aligned}\mathrm{d}w_t &= B'(\beta^{-1}z_t)\beta^{-1}\,\mathrm{d}z_t \\ &= B'(\beta^{-1}z_t)\beta^{-1}\|\mathbf{p}_*\|^{-1}\big(\,\mathrm{d}\|\mathbf{p}_t\| - z_t\,\mathrm{d}\|\mathbf{p}_*\|\,\big) = \\ &= B'(\beta^{-1}z_t)\beta^{-1}\|\mathbf{p}_*\|^{-1}\big(\tilde{\mathbf{p}}_t^{\mathrm{T}}\,\mathrm{d}\mathbf{p}_t - z_t\tilde{\mathbf{p}}_*^{\mathrm{T}}\,\mathrm{d}\mathbf{p}_*\,\big) = \\ &= B'(\beta^{-1}z_t)\beta^{-1}\|\mathbf{p}_*\|^{-1}\big(\tilde{\mathbf{p}}_t^{\mathrm{T}} - z_t\tilde{\mathbf{p}}_*^{\mathrm{T}}\big)\,\mathrm{d}\mathbf{q}\end{aligned} \tag{9.27}$$

and the complete Jacobian becomes

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\mathbf{q}} = \beta^{-1}\|\mathbf{p}_*\|^{-1}\begin{pmatrix} B'(\beta^{-1}z_1)\big(\tilde{\mathbf{p}}_1 - z_1\,\tilde{\mathbf{p}}_*\big)^{\mathrm{T}} \\ \cdots \\ B'(\beta^{-1}z_T)\big(\tilde{\mathbf{p}}_T - z_T\,\tilde{\mathbf{p}}_*\big)^{\mathrm{T}}\end{pmatrix} \ . \tag{9.28}$$

9.3.5 Summary

The complete Jacobian is given by combining (9.14) and (9.21), with $\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\mathbf{q}}$ selected by (9.24) or (9.28). In pseudo-Matlab notation, this can be written as

$$\mathbf{v} = (\mathbf{X}^{\mathrm{T}}\mathbf{M}^{-1})[:, \mathrm{end}] \tag{9.29}$$

$$\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{q}}(\mathbf{x}_0) = \mathbf{C} + (\mathbf{Y} - \tilde{\mathbf{C}}\mathbf{X})\mathrm{diag}(\mathbf{v})\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}\mathbf{q}} \ . \tag{9.30}$$

The compensated Jacobian is not much more expensive to compute than the approximative. The major workload in the main method is in solving the minimization problem (9.8). This is typically done using the pseudo-inverse $(\mathbf{X}\mathbf{W}^{1/2})^{\dagger} = \mathbf{W}^{1/2}\mathbf{X}^{\mathrm{T}}\mathbf{M}^{-1}$, which can be computed using the SVD of $(\mathbf{X}\mathbf{W}^{1/2})$. Once this pseudo-inverse is available, $\mathbf{v}$ is easy to obtain, and the true Jacobian is available after a few matrix multiplications.

9.4 Discussion

This chapter has reviewed the basic method of locally weighted regression and discussed different ways of selecting an adaptive kernel width. This subject is in no way exhausted, and it could be possible to find more advanced methods. However, as the kernel width selection becomes more sophisticated, it becomes increasingly difficult to find an analytical derivative of the entire procedure.

On the other hand, it can be argued whether or not this analytical derivative is needed. The matrix $\mathbf{C}$ is a good approximation to the derivative as long as the residuals of the local linear model fitting are small.

One large part of the workload of the algorithm consists in finding the K nearest neighbors of the query sample. Much work on fast neighbor lookup structures has been done by using various forms of tree structures [7, 73]. This topic is outside the scope of this thesis.

Part III

Pose Estimation

Chapter 10

Linear Interpolation on CCFMs

...where we will try a different linear approach, and find that locality is king. We also find that it is sometimes preferable to turn a problem on its head. At the end of the chapter, we will have constructed our first view-based pose estimation method.

This chapter deals with interpolation on channel-coded feature maps and linear equation systems for the purpose of classification or estimation of continuous pose parameters of an object. In this chapter, I do not deal with issues such as localizing an object in a larger image or giving continuous values for position, rotation and scale in the image plane. In the later chapters 11 and 12, all pieces will be put together in a complete object recognition system, where object detection in a cluttered scene is also handled.

The examples within this chapter are taken from the COIL-100 database [76], where 100 different object have been photographed for different orientations around the vertical axis and placed well-centered in 128×128-pixel images. This dataset is often criticized for being too simple since there is no background clutter or occlusions but is still useful for illustrative purposes.

Previous chapters have covered the associative networks, which transform a channel-coded scalar or vector to another channel-coded scalar or vector using a linear mapping. In this chapter, the input is instead an entire channel-coded feature map. This makes the problem rather different, and I argue that a direct linear mapping from the CCFM to the response is no longer sufficient.

This chapter begins from a slightly different angle – from the idea of view interpolation – and ends up with connecting the locally weighted regression from Chapter 9 to channel-coded feature maps. The Euclidean distance is used in the presentation, but recall from Chapter 5 that we could as easily use the square-root distance instead by simply taking the square root of all channel values before further processing.

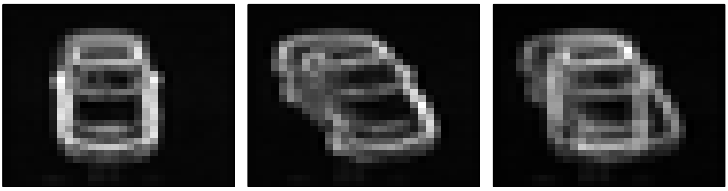

Figure 10.1: Illustration of why view interpolation on raw images does not work.

10.1 View Interpolation

As a motivating example, consider Fig. 10.1. Here, two images of a toy car viewed from different poses are represented by low-resolution gradient magnitude images which we refer to as $\mathbf{a}_1$ and $\mathbf{a}_2$. If we interpolate linearly between these two views and construct a new view $\mathbf{a} = 0.5\mathbf{a}_1 + 0.5\mathbf{a}_2$, we see that this view does not match the expected appearance of the car at an intermediate pose, showing that this space is not well-suited for linear view interpolation. In order for this kind of view interpolation to work at all, the resolution must be small enough for the same object structure to appear within the same pixel of the low-resolution image. For this car, we would have to settle with perhaps 4×4 pixels, which is a rather non-informative representation.

In contrast, consider representing the the car with a channel-coded feature map of spatial resolution 4×4, but using at the same time 4 orientation channels and 4 color channels. This would give 16 different 4×4 images, each one containing information about a certain kind of image structure, namely edges of a specific orientation and color. Due to the low spatial resolution, it is more likely that interpolating between the CCFMs from two views of an object produces something similar to the expected CCFM of an intermediate view. At the same time, the specificity of the representation is improved compared to a simple downsampling of the graylevels.

This motivates trying to represent an observed view $\mathbf{q}$ as a linear combination of a number of training views which are located close enough in feature space for view interpolation to be meaningful. In general, given a set of such views $\mathbf{a}_t$ organized in a matrix $\mathbf{A}$, we look for a set of linear coefficients $\mathbf{r}$ such that $\mathcal{E} = \|\mathbf{A}\mathbf{r} - \mathbf{q}\|$ is minimized. If we want a strict convex combination of our training views, we can enforce that each $r_k \geq 0$ and $\mathbf{r}^{\mathrm{T}}\mathbf{1} = 1$. If we are satisfied with an unconstrained solution of $\mathbf{r}$, it is given as $\mathbf{r} = \mathbf{A}^{\dagger}\mathbf{q}$.

If each CCFM included in $\mathbf{A}$ has some continuous parameters (e.g. pose angles) associated to it, we can use the same interpolation coefficients to interpolate between these parameters in order to get an estimated parameter vector for the query view. This will be discussed in more detail in Sect. 10.2.2. But first, we will move to a different problem formulation, which turns out to be more or less the same thing as view interpolation.

10.2 Least-Squares Formulations

This section gives an overview of some linear least-squares formulations that can be constructed using channel-coded feature maps as input. The purpose of the presentation is to identify the limitations of these methods and motivate using local linear models. The view interpolation idea from the previous section will be used as a tool for interpreting the behavior of these linear methods.

10.2.1 Basic Problem Formulation

Assume that we are given a number of training examples $(\mathbf{a}_t, \mathbf{u}_t)$, where the $\mathbf{a}_t$'s are CCFMs and the $\mathbf{u}_t$'s some associated responses. These $\mathbf{u}_t$'s can in general represent any information associated with the input, and some concrete examples will be considered in the next few subsections. Let $\mathbf{q}$ be a query CCFM. We now look for a linear mapping $\hat{\mathbf{u}} = \mathbf{C}\mathbf{q}$ that gives an estimated response from the query. By organizing the training samples in matrices $\mathbf{A} = [\mathbf{a}_1, \dots \mathbf{a}_T]$ and $\mathbf{U} = [\mathbf{u}_1, \dots \mathbf{u}_T]$, $\mathbf{C}$ can be found by solving the minimization problem

$$\mathbf{C} = \arg\min_{\mathbf{C}} \|\mathbf{C}\mathbf{A} - \mathbf{U}\|_{\mathrm{F}} \ . \tag{10.1}$$

Often, the number of training examples will be less than the dimensionality of the CCFMs. In this case, the problem is under-determined, and the traditional least-squares selects the solution with minimum norm. In any case, the solution can be written as

$$\mathbf{C} = \mathbf{U}\mathbf{A}^{\dagger} \ . \tag{10.2}$$

This gives the entire mapping from a query input to response as

$$\hat{\mathbf{u}} = \mathbf{U}\mathbf{A}^{\dagger}\mathbf{q} \ . \tag{10.3}$$

An important interpretation of this expression is obtained by recalling that $\mathbf{r} = \mathbf{A}^{\dagger}\mathbf{q}$ is the solution to the view interpolation problem from Sect. 10.1. This shows that the solution of (10.1) can be obtained by first writing the query view as a linear combination of the training views, and then using the same set of coefficients to interpolate between the training outputs in $\mathbf{U}$ by

$$\hat{\mathbf{u}} = \mathbf{U}\mathbf{r} = \sum_t \mathbf{r}[t]\mathbf{u}_t \ . \tag{10.4}$$

An alternative here is to replace $\mathbf{r} = \mathbf{A}^{\dagger}\mathbf{q}$ with the constrained $\mathbf{r}$ from Sect. 10.1. The effects of doing so will be studied in more detail later on.

This interpretation will be the tool for analyzing the behavior of different linear mappings in this chapter. In order to understand the behavior of the least-squares method for different representations of the output space, it is thus sufficient to understand how the representations behave under linear interpolation. In the next few subsections, the consequences of some different representations of $\mathbf{u}$ are discussed.

[0.01 −0.11 0.61 0.40 −0.22 0.22 −0.07 0.14 0.03]
[0.00 0.00 0.51 0.31 0.00 0.00 0.09 0.09 0.00]

Figure 10.2: A query view with interpolation coefficients (in CCFM space) for all training views of the same object class. Top: Unconstrained coefficients. Bottom: Positive, unit-sum coefficients.

10.2.2 Continuous Pose Parameters

The first case is where $\mathbf{u}$ is a vector of real-valued parameters governing the appearance of an object. The typical example is where $\mathbf{u} = [\theta, \phi]^{\mathrm{T}}$ is a pair of rotation angles around two axes distinct from the optical axis. However, $\mathbf{u}$ could also include quantities like the opening angle of a pair of scissors or the rotation angles of the joints of a robotic arm.

The parameters in $\mathbf{u}$ do not necessarily have to refer to intrinsic properties of the object. Instead, a set of *purposeful* parameters can be used, for example a set of real-valued parameters representing how a robotic arm should be moved in order to grip the object. Having such a direct mapping between an observation and an action without using any engineered intermediate representation like object pose is along the philosophy upon which the COSPAL project was based (see Sect. 1.3).

If the constrained $\mathbf{r}$ is used, the resulting output of the algorithm in (10.4) becomes a weighted mean of all $\mathbf{u}_t$'s. This has the same problem with outliers as most methods based on mean values have. Consider the example in Fig. 10.2. Here, even if there is a clear peak at the correct views, the other coefficients still have non-zero values and will contribute to the weighted mean value. The situation is better but still not perfect when the constrained coefficients are used. This motivates using a strategy that removes the effect of outliers by including only views close to the final estimate.

In [27], this method was used for ego-localization from a fish-eye camera. The input was a P-channel-coded feature map (a representation related to CCFMs) and the output was 6 continuous parameters representing the position and orientation of the camera. The experiments showed nice results despite the lack of an outlier-suppressing mechanism, which shows that this simple method can still work well if the data is well-behaved.

10.2.3 Classification

Instead of finding continuous parameters, we can address the problem of classification. In this case, the output is not a continuous parameter but a discrete class label. Assume that we are given a training set consisting of T tuples $(\mathbf{a}_t, l_t)$, where $\mathbf{a}_t$ is a CCFM and $l_t \in [1, L]$ is the true class label of this input. We then want to estimate the class of a query CCFM $\mathbf{q}$. In order to use the linear framework, we

represent the true classes using vectors $\mathbf{u}$ according to

$$\mathbf{u}_t[l] = \begin{cases} 1 & l = l_t \\ 0 & \text{otherwise} \end{cases} . \tag{10.5}$$

This is referred to as a "1-of-K binary coding scheme" in [12], and the vector $\mathbf{u}$ is often called a *discrimination vector*. This scheme is frequently used for encoding class labels in multi-class problems (see also [47]). Since the CCFMs are already quite high-dimensional, it may be argued that the problem is likely to be linearly separable according to Cover's theorem on the separability of patterns [47].

We now look for a linear mapping $\mathbf{C}$ that gives a membership score for each class using $\mathbf{u} = \mathbf{Cq}$. This $\mathbf{C}$ can be found according to (10.1)-(10.2), giving the complete relationship from query to output as $\mathbf{u} = \mathbf{U}\mathbf{A}^\dagger\mathbf{q}$ like before. To analyze this method, we first interpret $\mathbf{r} = \mathbf{A}^\dagger\mathbf{q}$ as a set of interpolation coefficients as usual and note that

$$\mathbf{u}[l] = \sum_t \mathbf{U}[l,t]\mathbf{r}[t] = \sum_{t\,:\,l_t = l} \mathbf{r}[t] \ . \tag{10.6}$$

Because of the special binary structure of $\mathbf{U}$, each $\mathbf{u}[l]$ is simply the sum of the interpolation coefficients in $\mathbf{r}$ corresponding to all views from class l. However, this may not be a good certainty measure. Consider again Fig. 10.2. The CCFM from a cup with two handles – one at each side – may very well be possible to express as a linear combination of the CCFMs from the first and fifth view of Fig. 10.2. This classification algorithm would not distinguish between this case and a case where only neighboring views participate with large coefficients.

In the extreme case where each class has exactly one training view, $\mathbf{U}$ is diagonal and $\mathbf{u} = \mathbf{r}$. This approach was used on the COIL database in the related publication [28]. P-channels were used as view representation. The view with the largest $\mathbf{r}$ coefficient determined the final estimated object class. This technique has a different drawback. In Fig. 10.2, the largest coefficient is 0.61 (or 0.51 for constrained coefficients) because the query view happens to be in between two training views. For a query view that shows the same pose as the training views, the largest score would be close to 1. This means that the score varies in an undesired way depending on the exact view of the object.

From these examples, we can draw the conclusion that it should be beneficial to combine neighboring views only and use the approximation error as a quality measure rather than the interpolation coefficients.

10.2.4 Channel-Coded Output

Consider an image set like the COIL database (Fig. 10.3). Assume for a moment that we only look at a single object and want to find the best matching view. There are two options. We can treat each view as a separate class and use the classification procedure in Sect. 10.2.3 to find the view in memory that best matches the query. Alternatively, we can view it as a continuous pose estimation problem and use a continuous parameter as output, according to Sect. 10.2.2.

Figure 10.3: Some examples of objects in the COIL database.

A third option would be to encode the continuous pose parameters into a channel vector $\mathbf{u}$. This produces a representation which is somewhere in between the discrete and continuous case. We know from Sect. 2.4 that averaging a set of channel vectors and decoding the result is equivalent to computing a robust mean of the encoded values. Thus, if the $\mathbf{u}_t$'s are channel-coded scalars instead of regular scalars, (10.4) will produce a weighted robust mean of the encoded values. This is significantly more well-suited for situations like the one in Fig. 10.2, since only the views close to the peak would actually contribute to the resulting pose estimate. However, the coefficients $\mathbf{r}$ are still produced using an interpolation between all views, even views which are eventually not needed. It is a bit disturbing that views considered as outliers still are allowed to contribute to our interpolation coefficients. It would make sense to run the process twice; once to find the peak, and then once again using views close to the peak only. But in this second run, channel-coding the output is no longer necessary, since only related views are included in the first place.

Using channel-coding both on the input and output side of a linear mapping may seem very similar to the methods in 7 and 8, but there is a major difference in the representation on the input side. Using associative networks, the input is a channel-coded single value. In this chapter, the input is an entire channel-coded feature map, i.e. the sum of a number of encoded single points. In the correspondence-free learning case, we also summed up the channel vectors from several encoded points, but we were looking for a mapping from individual points in the input space to individual points in the output space. A superposition of points on the input side should simply map to the same superposition of points in the output space. In this chapter however, we are not looking for a mapping from single points in the spatio-feature space. Rather, it is the entire joint configuration of points that is significant. The same analysis can not be applied in these two cases.

10.3 Local Weighting and Inverse Modeling

As has been seen in the previous section, in order for view interpolation to perform well, we should select only those views which can be expected to be connected by a smooth manifold in feature space. This can be done using the locally weighted regression idea from Chapter 9. If we select the K training views that closest to the query view, weight them according some strategy from Chapter 9 and perform the view interpolation from Sect. 10.2.2, we get something which is very related

to locally weighted regression.

A problem with this approach is that the least-squares problem minimization problem is under-determined, especially now when only a few neighboring views are included. The standard least-squares framework chooses the solution with minimum norm, which may not be the best choice. In the next section, we will obtain a more robust method by instead approximating the forward model.

10.3.1 Approximating the Forward Model

Consider a function $\mathbf{f}$ that maps a continuous pose vector $\mathbf{u}$ (e.g. a set of pose angles ϕ, θ) to a feature vector $\mathbf{a}$:

$$\mathbf{a} = \mathbf{f}(\mathbf{u}) \ . \tag{10.7}$$

What our system is supposed to actually solve is the inverse of $\mathbf{f}$ – given a channel-coded feature map, find the pose angles of the object. However, there are several reasons for trying to model $\mathbf{f}$ instead of its inverse. First, it is usually easier to find a mapping from a low-dimensional space to a high-dimensional space than the other way around, and the space of pose angles has much lower dimensionality than the feature vectors. Secondly, in approximating the forward model, we can enforce the correct intrinsic dimensionality of the manifold which increases the robustness against noise.

In approximating $\mathbf{f}$, the method of locally weighted regression (LWR) can also be applied. Here, the input space is relatively low-dimensional, so we can easily choose a K that makes the model fitting over-determined unless we are really unlucky in how the training views are distributed, such that the problem is degenerated.

10.3.2 Solving the Inverse Problem

The LWR method produces a linear approximation of the forward model given a point of linearization $\mathbf{u}$. This section describes how to solve the inverse problem by formulating it as a minimization problem

$$\min_{\mathbf{u}} \|\mathbf{f}(\mathbf{u}) - \mathbf{q}\| \ , \tag{10.8}$$

meaning that we look for the $\mathbf{u}$ that brings the expected appearance of the object as close as possible to the observed $\mathbf{q}$. One solution strategy is to start by finding the prototype $\mathbf{a}_t$ closest to our query vector $\mathbf{q}$ and linearize $\mathbf{f}$ around the associated $\mathbf{u}_t$ according to the locally weighted regression technique. The linearized version of (10.8) is then

$$\min_{\mathbf{u}} \|\mathbf{f}_0 + \mathbf{C}\mathbf{u} - \mathbf{q}\| \ , \tag{10.9}$$

from which $\mathbf{u}$ can be obtained as $\mathbf{u} = \mathbf{C}^{\dagger}(\mathbf{q} - \mathbf{f}_0)$. Note that $\mathbf{C}^{\dagger}$ and $\mathbf{f}_0$ depend on how the neighbors are weighted, which in turn depends on the point of linearization. This motivates using an iterative procedure, where $\mathbf{f}$ is linearized

around successively better and better estimates of $\mathbf{u}$. This essentially becomes a Gauss-Newton method for solving (10.8) and is related to the patch tracking from Sect. 3.3. In Chapter 11, these two problems will be fused into a single iterative optimization.

To increase the robustness of the method, the whole optimization process could be run several times for different initial guesses of $\mathbf{u}$, e.g. the $\mathbf{u}_t$'s associated to the K best matching $\mathbf{a}_t$'s.

10.4 Summary

The techniques described in this chapter can be combined into several distinct methods. I suggest to avoid including all available training views in the same least-squares problem but to restrict the interpolation to a subset of views which are believed to be related to the query view. I suggest taking the response $\mathbf{u}$ from the best matching training view and fine-tune $\mathbf{u}$ by running an iterative optimization based on an LWR model of the forward function according to Sect. 10.3.2.

The forward model requires that each training view is equipped with a true pose parameter. If the training views are associated with a class label only, this model can not be applied. It may still motivated to run the view interpolation from Sect. 10.1 anyhow, since the fact that the query view can be written as a convex combination of two training views from the same object class is a strong indication that the view in fact belongs to this class. In this case I suggest to run the view interpolation between the K best matching views within each class, and use the residual error between the interpolated CCFM and the query CCFM as a quality measure.

10.5 Experiments

10.5.1 Pose Estimation in Clutter

In the first experiment, the goal is to determine the pose of a toy car given its known position in the image. The motivation for assuming a known image position is to test the performance of the view interpolation method in isolation. In order to provide a realistic setting, the method was trained on images of the car with black background and evaluated on images with cluttered background (see Fig. 10.4 and 10.5)[1]. Views were available for every 5° in both the ϕ and θ domain. The method was trained on views 20° apart in each domain and evaluated for all intermediate angles, such that the exact same angle is never present in both the training and evaluation set. Table 10.1 shows the RMS error of the estimated angles (both individually and together) for parameter variations around some manually selected good values. In the best case, the pose error was around 3°, which is 15% of the training view spacing.

The most striking observation is that the results are dramatically degraded when L_1 normalization is used. This is because of the background clutter. Since

[1] The images were produced by Viksten, Johansson, Moe in connection with [33]

Figure 10.4: All training views. Y-axis: θ, X-axis: ϕ.

Figure 10.5: Examples of evaluation views with background clutter.

Options	$\mathcal{E}_\theta$	$\mathcal{E}_\phi$	$\mathcal{E}_{\theta,\phi}$
Default	3.1	3.0	3.0
Distance: sqrt	5.6	4.8	5.2
Weights: soft threshold	3.4	3.1	3.3
Preprocess-σ: 2	3.7	3.6	3.7
Normalization: L1	11.0	16.6	14.1
Channels: $8 \times 8 \times 6$	3.5	3.4	3.4
Channels: $6 \times 6 \times 6$	4.4	4.9	4.7
Neighbors: 5	3.5	3.0	3.3
Neighbors: 6	3.6	3.1	3.4

Table 10.1: Pose estimation results on cluttered background for some parameter variations around a manually selected good choice. Default choice is (Channels: $8 \times 8 \times 8$, Normalization: None, Weights: Grad-magn, Preprocess-σ: 1, Distance: Euclidean, Neighbors: 4).

$n_x \times n_y$	$\mathcal{E}_{\theta,\phi}$
6x6	1.2
8x8	1.2
10x10	1.3
12x12	1.4
14x14	1.4

n_f	$\mathcal{E}_{\theta,\phi}$
4	1.2
6	1.2
8	1.3
10	1.3

neighbors	$\mathcal{E}_{\theta,\phi}$
3	1.3
4	1.2
5	1.2
6	1.2
7	1.3
8	1.6
9	1.9

Table 10.2: Reproduction of results from [59]. RMS error in degrees for pose angles around the manually selected option $8 \times 8 \times 6$ channels, 4 neighbors, euclidean distance, raw gradient magnitude weighting. Left: Varying spatial resolution. Middle: Varying number of orientation channels. Right: Varying number of neighbors.

there are much more edges present in the cluttered images, normalizing the representation decreases the values of the channels containing features from the object.

In [59], a similar dataset was used, but there the evaluation views also had black background. The result was an RMS pose error of between 1° and 2° depending on the parameters. For completeness, these results are reproduced in Table 10.2.

These datasets will be revisited in Chapter 11, where the position of the object in the image plane will also be adjusted simultaneously.

10.5.2 COIL Classification

In the second experiment, I examine whether it is possible to improve the classification results on the COIL database by introducing a view interpolation. In this case, I am not interested in determining the actual pose of the object, but just to find its class label. First, the query view was compared to each training

Features	Eval/train ratio	r_{NN}	r_{I}	Improvement
orient	8	93.30%	95.35%	30%
	6	95.78%	97.65%	44%
	4	97.94%	99.26%	64%
orient+color	8	99.35%	99.49%	22%
	6	99.67%	99.93%	80%
	4	99.89%	99.98%	83%

Table 10.3: Recognition results on the COIL database using Gauss-Newton on LWR.

view. The best four matching object classes were selected for a refinement step. In this refinement, the Gauss-Newton method was run on the forward LWR model according to Sect. 10.3.2, initialized on the best matching view within the given object class. The cost of a certain class assignment was then taken as the residual error of the view interpolation process.

The best parameters from the experiments in Chapter 5 were used (square-root distance, 5 channels in each dimension, non-thresholded gradient magnitude as weights). The experiments were run both with and without color included in the channel coding.

The resulting classification rates r_{NN} for the nearest neighbor method and r_{I} for the interpolated version are shown in Table 10.3. The column "Improvement" is computed as $(r_{\mathrm{I}} - r_{\mathrm{NN}})/(1 - r_{\mathrm{NN}})$ and shows the percentage of the classification errors that were removed by switching to the interpolated method. The results clearly indicate that the view interpolation helps improving the classification, especially if the training views are located more closely in pose space. This is natural, since the local linear approximation of the training view manifold is more accurate when the training views are close together. The best classification rate of 99.98% obtained using ratio 4 means that only a single one out of the 5400 evaluation views was erroneously classified.

10.6 Discussion

This chapter has discussed some variations on linear systems and interpolation between CCFMs. The recommended approach is to perform a local interpolation between neighboring views in pose space. This method gives improved classification results compared to a raw nearest neighbor method and produces rather accurate estimates of continuous pose parameters even in the presence of background clutter.

In Sect. 10.4, two distinct situations were considered – either the true pose parameters of the training views are known or unknown. A situation somewhere in between is where an object is observed continuously while changing pose, but where the actual pose parameters are unknown. This is along the ideas of the COSPAL project (see Sect. 1.3), where one goal is to keep the amount of engineering and user intervention in artificial vision systems to a minimum. Instead of

supplying the system with manually created training views, a more flexible option is to let the system gather training views autonomously. One way to achieve this is to move the robot semi-randomly, trying to push objects around in order to create variations in pose. This was done within the project, see e.g. [31]. The system could get some information about which views are close together in pose space by simply assuming that closeness in time implies closeness in pose space [39]. By connecting views believed to be close to some graph structure, a number of neighbors in this graph could be selected without knowing their actual pose parameters. Alternatively, some unsupervised learning method could be applied to structure the training views into a continuous manifold. This kind of analysis was never made in COSPAL or within my PhD work but is an interesting direction of future research.

Chapter 11

Simultaneous View Interpolation and Tracking

...where the tracking and view interpolation using channel-coded feature maps decide to merge themselves into a single optimization problem. This results in a more complete pose estimation method and makes it possible to track an object through changes in pose.

Object recognition and pose estimation can be done in several ways. One popular approach is based on local features. Local coordinate frames are constructed around points or regions of interest [66, 78], and features from each local frame vote for a certain object and pose hypothesis. In the model-based approach [17, 87], a geometrical model is fitted to the observed image. This approach is often very accurate, but requires a good initial guess and a manually constructed 3D model. Global appearance-based methods extract features from the appearance of the entire object and match these to training views in memory. Ever since [79, 74], the most common approach seems to be using PCA.

This chapter describes a view-based pose estimation method using channel-coded feature maps that combine the patch tracking from Chapter 3 and the view interpolation using locally weighted regression from Chapter 9 and 10. The method simultaneously optimizes the position of the object in the image and a set of continuous pose parameters as defined by the training set.

The motivation for using full object views is two-fold. The first reason is that once we have formed an initial object hypothesis, it makes sense to use as much image information as possible in order to get an accurate estimate. The second reason is that using full views, we can focus on the interpolation and view representation, and ignore other aspects like how to choose interest points and construct local frames. This makes it easier to compare different view representations. Similar interpolation techniques as proposed here should however be possible to integrate also in a local features approach.

In contrast to model-based methods, our approach requires no knowledge of 3D geometry in the system, and is in no way specific to 3D pose estimation. The

training set could consist of any parameterized image set, e.g. a robotic arm in different configurations. This is along the philosophy of the COSPAL project (see Sect. 1.3), in which as little engineered knowledge of the world as possible should be included in the system.

11.1 Algorithm

11.1.1 Image and Pose Parameters

Let us consider how an object may look in an image and ignore illumination for a moment. The set of parameters governing the appearance of the object can be factored into one set that causes transformations in the image plane only (translation, rotation, scale change) which will be called *image parameters* and one set of remaining parameters, e.g. rotation around the vertical and horizontal axis which will be called *pose parameters*. For exactness, it should be noted that when an object is moved closer to the camera, the change of perspective causes other changes in appearance than just a scale change. This parameter could be included as a pose parameter, but when the size of the object is small relative to the distance between the object and the camera, this perspective effect is negligible.

Consider now a different problem, namely that of estimating the position and orientation of a camera. If the image is resampled into cylindrical coordinates, rotating the camera around its optical center causes translations and rotations in the image plane only. On the other hand, moving the camera causes the image to change in a way that is dependent on the unknown depth information at each pixel. Occluded areas may become non-occluded and vice versa. In this case, the image parameters would be the rotation of the camera, while the pose parameters would be the camera translation.

In the view-based object recognition framework presented in this chapter, the pose parameters are handled by a learning approach while the image parameters are handled analytically.

11.1.2 Appearance Model

Assume that we are given a training set consisting of views of the object for different pose parameters $\boldsymbol{\theta}_t$, but for the same canonical image parameters. Each training view is encoded into a channel-coded feature map $\mathbf{c}_t$. We now introduce a function $\mathbf{c} = \mathbf{f}(\boldsymbol{\theta})$ that gives a channel-coded feature map from a pose parameter, such that the training views can be seen as input-output examples of this function. To approximate $\mathbf{f}$ for intermediate pose parameters, the view interpolation using locally weighted regression from the previous two chapters is used. This method is briefly summarized here, to make this chapter more self-consistent.

Assume that we want to find the CCFM corresponding to a pose parameter $\boldsymbol{\theta}_0$. First, weight all training samples according to

$$w_t = K(s^{-1}\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_0\|) \ , \tag{11.1}$$

where K is a smooth Gaussian-looking weighting kernel, but with compact support on $[0, 1]$, and s is a scale parameter. This scale parameter could be selected adaptively according to Sect. 9.2.1 or as a fixed value based on some prior knowledge about the density of training views in pose space. We then find a local linear model by solving the weighted least-squares problem

$$\min_{\mathbf{A},\mathbf{b}} \sum_t w_t \|(\mathbf{A}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_0) + \mathbf{b} - \mathbf{c}_t\|^2 \ . \tag{11.2}$$

Due to the compact support of K, this produces an interpolation using neighboring points only. From the Taylor expansion of $\mathbf{f}$, we can identify $\mathbf{b}$ and $\mathbf{A}$ as the approximated function value and derivative respectively:

$$\mathbf{f}(\boldsymbol{\theta}) \approx \mathbf{f}(\boldsymbol{\theta}_0) + \mathbf{f}'(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0) \approx \mathbf{b} + \mathbf{A}(\boldsymbol{\theta} - \boldsymbol{\theta}_0) \ . \tag{11.3}$$

However, the derivative of the local model is not the same as the derivative of the entire approximation procedure. In Chapter 9, an analytical expression for this derivative was presented. This analytical derivative is only marginally more expensive to compute and should be used if one runs into numerical problems using $\mathbf{A}$ as derivative.

11.1.3 Feature Extraction

In runtime, we want to find a patch in the query image that contains the object. This patch is defined by parameters (s, α, x_0, y_0), where (x_0, y_0) is the patch center, s the radius and α the orientation. From such a patch, a channel-coded feature map is extracted, and this CCFM is compared to the feature maps extracted from the training views. This whole process can be modeled by a function $\mathbf{c} = \mathbf{g}(\boldsymbol{\psi})$, giving a CCFM from the query image for a given set of image parameters $\boldsymbol{\psi}$. This formulation is used also in Chapter 3 for patch tracking, and in that chapter I also presented the derivatives of the function $\mathbf{g}$.

As before, the theory is general in that any features can be used. Color and local orientation will be the primary example, but other interesting features include curvature, corner energy, etc.

11.1.4 Optimization

Both the image and pose parameters can be optimized simultaneously by the minimization problem

$$\min_{\boldsymbol{\theta},\boldsymbol{\psi}} \|\mathbf{r}(\boldsymbol{\theta}, \boldsymbol{\psi})\| = \min_{\boldsymbol{\theta},\boldsymbol{\psi}} \|\mathbf{f}(\boldsymbol{\theta}) - \mathbf{g}(\boldsymbol{\psi})\| \ . \tag{11.4}$$

In words, this will find a position in the image that looks as similar as possible to something which can be interpolated from the training views.

Equation (11.4) can be solved using your favorite optimization method. I used Gauss-Newton method with a simple backtracking line search [77]. The update step direction $\mathbf{s}$ is given by

$$\mathbf{J}\mathbf{s} = -\mathbf{r} \ , \tag{11.5}$$

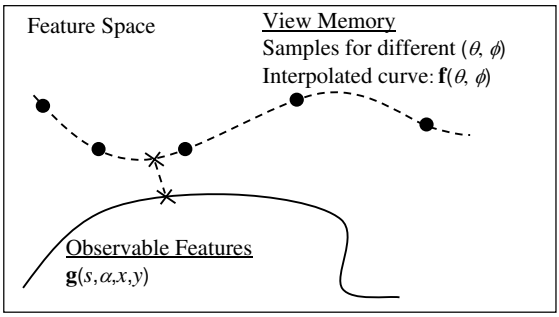

Figure 11.1: Illustration of the simultaneous optimization of image and pose parameters.

where $\mathbf{J}$ is the Jacobian of $\mathbf{r}$, composed from the Jacobians of $\mathbf{f}$ and $\mathbf{g}$:

$$\mathbf{J} = \left[\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\boldsymbol{\theta}}, \; -\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\boldsymbol{\psi}}\right] \quad . \tag{11.6}$$

In each step of the iterations, we measure $\mathbf{g}(\boldsymbol{\psi})$ directly in the query image by cutting out a new patch using the current $\boldsymbol{\psi}$ and extracting a new feature vector from this patch. A faster but less accurate option would be to keep the original feature vector and Jacobian, and use them as a linear approximation of $\mathbf{g}$ throughout the entire solution procedure.

To fully understand the method, it is useful to have a geometrical image in mind. The ranges of output of the functions $\mathbf{f}$ and $\mathbf{g}$ define two manifolds in feature space. The first manifold contains all expected appearances of the object, learned from training examples, and the second one contains all observable feature vectors at different positions in the query image. The objective is to find one point on each manifold such that the distance between the two points is minimal. This is illustrated in Fig. 11.1.

What the Gauss-Newton method does is to approximate each manifold with its tangent plane and find the points of minimal distance on these hyperplanes. Let $\mathbf{f}(\boldsymbol{\theta}+\mathbf{s}_{\boldsymbol{\theta}}) \approx \mathbf{f}(\boldsymbol{\theta})+\mathbf{f}'(\boldsymbol{\theta})\mathbf{s}_{\boldsymbol{\theta}}$ and $\mathbf{g}(\boldsymbol{\psi}+\mathbf{s}_{\boldsymbol{\psi}}) \approx \mathbf{g}(\boldsymbol{\psi})+\mathbf{g}'(\boldsymbol{\psi})\mathbf{s}_{\boldsymbol{\psi}}$. The minimum-distance points are given by the over-determined equation system

$$\mathbf{f}(\boldsymbol{\theta}) + \mathbf{f}'(\boldsymbol{\theta})\mathbf{s}_{\boldsymbol{\theta}} = \mathbf{g}(\boldsymbol{\psi}) + \mathbf{g}'(\boldsymbol{\psi})\mathbf{s}_{\boldsymbol{\psi}} \quad , \tag{11.7}$$

which is solved by (11.5) with $\mathbf{s} = [\mathbf{s}_{\boldsymbol{\theta}} \; \mathbf{s}_{\boldsymbol{\psi}}]^{\mathrm{T}}$. If the linear approximation is good enough, we can expect good results even after a single iteration.

11.2 Experiments

11.2.1 Quantitative Evaluation

The method was evaluated on the same dataset used in Chapter 10 with a toy car scanned on a turn-table. Views were available for every 5° in both the θ and

$n_x \times n_y$	$\mathcal{E}_\mathrm{s}$	$\mathcal{E}_\mathrm{xy}$	$\mathcal{E}_\mathrm{q}$
6x6	2.4%	1.3%	0.016
8x8	2.2%	1.3%	0.016
10x10	2.4%	1.5%	0.017
12x12	2.5%	1.5%	0.017
14x14	2.7%	1.6%	0.018

n_f	$\mathcal{E}_\mathrm{s}$	$\mathcal{E}_\mathrm{xy}$	$\mathcal{E}_\mathrm{q}$
4	2.4%	1.3%	0.015
6	2.4%	1.3%	0.016
8	2.5%	1.3%	0.015
10	2.7%	1.3%	0.016

neighbors	$\mathcal{E}_\mathrm{s}$	$\mathcal{E}_\mathrm{xy}$	$\mathcal{E}_\mathrm{q}$
3	2.2%	1.3%	0.016
4	2.2%	1.3%	0.016
5	2.1%	1.2%	0.016
6	2.1%	1.1%	0.015
7	2.3%	1.3%	0.017
8	2.4%	1.3%	0.020

Table 11.1: Reproduction of experiments from [59], evaluated on black background. RMS error for some parameter variations around the around the manually selected option $8 \times 8 \times 6$ channels, 4 neighbors.

ϕ domain. The method was trained on views 20° apart, and evaluated on all intermediate views. This gives 50 training views and 629 evaluation views. The optimization was initialized with the correct image parameters and at the closest pose parameters present in the training set.

One problem in measuring the performance here is that the set of angles $[\alpha, \theta, \phi]$ is ambiguous such that two distinct set of angles can represent the same 3D orientation of the object. This makes it impossible to measure the error in each angle separately. To avoid this problem, the pose angles and image rotation were combined into a rotation quaternion, and the error was measured in the quaternion domain. As a rule of thumb, an RMS quaternion error of 0.015 corresponds to around 2° error in each angle. The scale parameter s is the radius of the local frame containing the object. The error in scale and position is measured relative to the true s.

In [59], the method was both trained and evaluated on views with black background. For completeness, the results from this paper are reproduced in Table 11.1. In a more recent experiment, the method was instead evaluated on views with cluttered background, as shown in Fig. 10.5 of Sect. 10.5. The same parameter combinations as in Sect. 10.5.1 were used, and the results are available in Table 11.2.

The current implementation runs at a few frames per second on an AMD Athlon 3800+.

11.2.2 Tracking through Pose Changes

The method was also applied on video sequences of the toy car viewed by a moving camera. The image and pose parameters of the car were optimized for each frame

Options	$\mathcal{E}_{\mathrm{s}}$	$\mathcal{E}_{\mathrm{xy}}$	$\mathcal{E}_{\mathrm{q}}$
Default	6.0%	4.3%	0.044
Distance: sqrt	7.1%	4.8%	0.049
Weights: soft threshold	5.6%	4.1%	0.040
Preprocess-σ: 2	6.8%	5.3%	0.039
Normalization: L1	6.6%	5.3%	0.062
Channels: $8 \times 8 \times 6$	5.1%	3.5%	0.044
Channels: $6 \times 6 \times 6$	8.0%	9.6%	0.064
Neighbors: 5	6.0%	4.3%	0.051
Neighbors: 6	6.0%	4.4%	0.050

Table 11.2: Evaluation on cluttered background for some parameter variations around a manually selected good choice. Default choice is (Channels: $8 \times 8 \times 8$, Normalization: None, Weights: Grad-magn, Preprocess-σ: 1, Distance: Euclidean, Neighbors: 4).

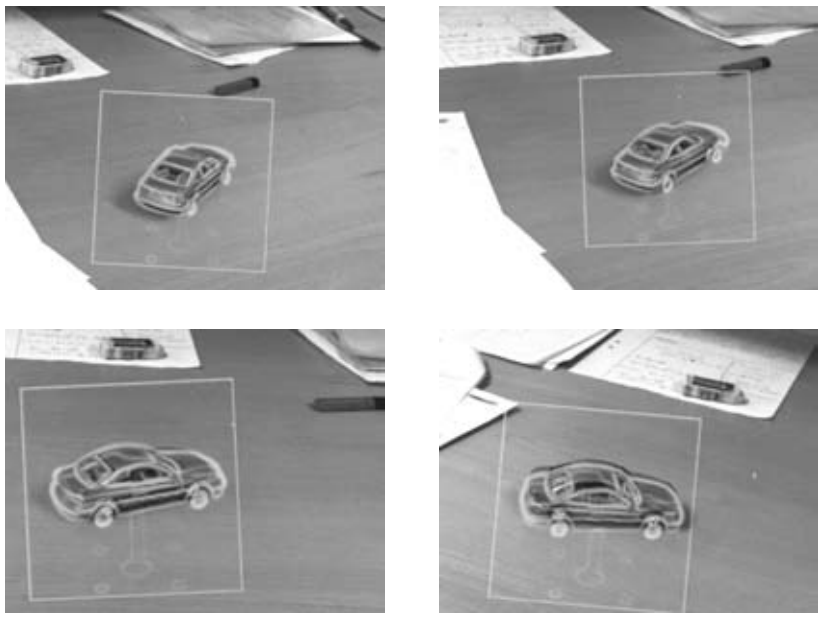

Figure 11.2: Frames 1, 40, 80 and 120 of a sequence where the toy car was tracked through pose changes.

of the sequence, using the estimated parameters from previous frame as an initial guess. The parameters of the first frame were selected manually since object detection has not been handled in this chapter. Some frames of one such video clip are shown in Fig. 11.2. For the graphical overlay, a set of images of the car spaced 5° apart were used, such that the estimated pose angles are quantized to 5° in the illustration. No video sequence with ground truth was available, so no quantitative evaluation was made in this case. Fig. 11.2 still shows that the method manages to track the car through significant changes in pose.

11.3 Discussion

In this chapter, I have described an accurate interpolation method for view-based pose estimation using local linear models and Gauss-Newton optimization. Some variations in the view representation have been compared in terms of fitness to this framework. However, the evaluation is in no way complete. There are several more parameters to play around with, e.g. the amount of overlap between basis functions, different soft thresholdings of the orientation image etc. It would also be interesting to perform a more detailed study on the performance of this representation compared to other approaches like PCA, wavelet representations etc. The experiments should also be extended to include occlusion and changes in illumination. Furthermore, the tracking should be evaluated on a dataset with known ground truth.

One problem with the dataset is that the spatial size and rough orientation of the toy car is rather different for different poses. Still, each training view used the same object bounding box. This has the effect that rather few channels are activated by the front and back views of the car, leading to a poor resolution. On the other hand, if we increase the resolution too much, the linearity assumption will no longer be valid. I expect that the results can be improved by prenormalizing the local frame of each training view such that the training views become more similar. This could be done automatically by trying to align each new training view to the previous ones using image space transformations according to Sect. 3.3.

Chapter 12

A Complete Object Recognition Framework

Congratulations, you reached the final main chapter! Here we will deal with topics that are not central to the thesis but of great practical importance when it comes to creating a working system. This chapter is based much more on experience and practical considerations than earlier chapters, even though it is my intention to support all statements with convincing arguments.

Now all mathematical and algorithmical building blocks are in place. At this point we are experts in how to compute channel-coded feature maps and its derivatives, and using them in various learning scenarios. What remains are some more practical issues, like which actual features to use. There is a vast number of options available. Should we include color? How should the orientation be computed and weighted? Should we normalize the channel-coded feature maps, and how?

This final chapter gives a discussion of such issues based on my implemented system. I will also discuss how choosing the parameterization of the system can help making it easier to tune. Finally, I touch the subject of object-oriented design and describe how virtual interfaces were used to make the system more flexible.

12.1 Object Detection

So far I only addressed the issue of classifying and estimating the pose and position of an object where the rough position of the object in the image is already given. In order to build a complete object recognition system, the issue of object detection must also be solved. It would be infeasible to scan a large query image by exhaustively varying both the position, rotation and scale of the region of interest, at least using a straight-forward implementation. Instead, I used a hypothesize-verify approach based on matching image primitives.

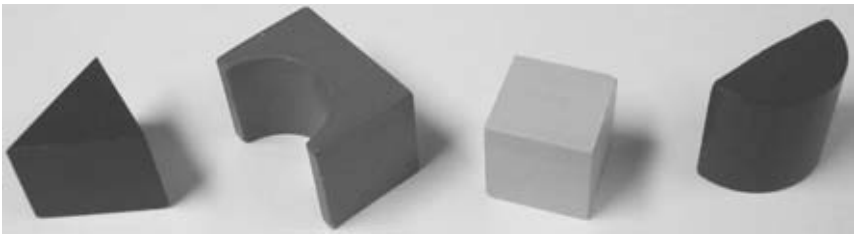

Figure 12.1: The type of objects used in the COSPAL project.

12.1.1 Primitives

From the query image and each training view, a number of primitives are extracted, for example interest points, line segments or regions of some kind. In order to be used in the system, these primitives must be equipped with an orientation and a scale parameter. One option is the patch-duplets in [53], where a pair of interest points is used, from which the rotation and scale are simply the angle of the line connecting the points and the distance between the points respectively. Another option is the SIFT keypoints from [66], which are extreme points of difference-of-Gaussian filters in scale space. In [78], homogeneous image regions are extracted, from which not just scale and orientation but an entire local affine frame (LAF) is constructed.

In the COSPAL project [1], the objects of interest were rather simple geometrical shapes (see Fig. 12.1), which motivated using line segments extracted from edges. The line segments were constructed by growing lines from seed points of large gradient magnitude. The orientation and length of the line was adjusted to minimize a cost function based on the gradient magnitude and color homogeneity on one side of the line. This is related to e.g. the line detector from [75]. Using these line primitives is not recommended for general objects, and the object recognizer is implemented such that it is easy to switch to a different feature extractor. The system design which makes this possible will be described in Sect. 12.4.

After the primitives are extracted, the primitives from the query view are matched to primitives in the training views. This matching can be made more selective by equipping each primitive with some descriptor, e.g. a channel-coded feature map. One tentative match defines a similarity transform between the training view and the query image. This similarity transform then defines the local frame in the query image in which the entire object is expected to be seen, giving us an object hypothesis (see Fig. 12.2). This hypothesis is then either verified or discarded. Two methods of verifying hypotheses have been tried and are described in the following subsections.

12.1.2 Verification using Full Object View

Each hypothesis gives us a local similarity frame (a position, orientation and scale) in the query image where the object is expected to be seen, as illustrated in Fig. 12.2. In full-view verification, the position of this frame is then adjusted to-

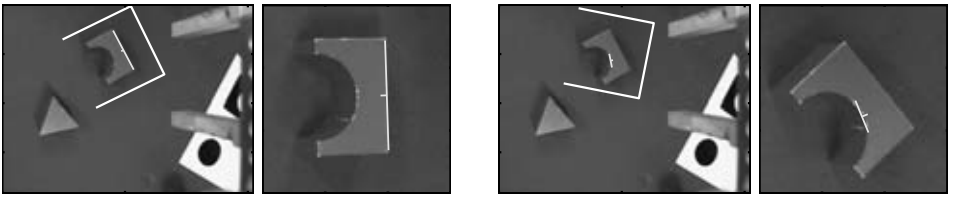

Figure 12.2: Illustration of the similarity relationship between the query view and training views, from an actual run of the system. In the verification step, the left hypothesis got a lower cost than the right one.

gether with the pose parameters of the object according to the technique described in Chapter 11.

This method works well, but is relatively slow when the system has been trained for many objects, and several objects are present in the query image. The most time-consuming step in the current implementation is the extraction of CCFMs from the query image. First of all, one CCFM must be extracted in each step of the iterative refinement. But even if this refinement is just run on the best few hypotheses, at least one CCFM must be extracted from the query image for each hypothesis. These CCFMs can not be reused, since each hypothesis defines its own local frame.

12.1.3 Verification using Local Patches

Another option is to extract one CCFMs around each primitive only. This can be done once and for all for each query image. The verification is then done using these CCFMs only. This is more related to the common local features approach for object recognition, see e.g. [67, 78, 81]. For the hypothesis construction, only one primitive from the given training view is used. The similarity transform obtained from the hypothesis predicts where the rest of the primitives from the training view are expected to be found in the query image. These predicted primitives are then matched to actual primitives in the query view, and a complete score of the hypothesis is computed based on a combination of the geometrical error of the primitive and of the difference between the CCFMs of the query and training primitive.

Using this approach, the number of CCFMs extracted from the query image is greatly reduced, but it becomes more difficult to apply the iterative refinement. In principle, the image position of each CCFM could be adjusted to minimize some cost, but this becomes more difficult than in the full-view case since there are several patches to optimize at once, and some sort of geometrical consistency must be maintained between the patches. Adjusting the pose parameters related to the training views becomes even more difficult since there is no clear correspondence between primitives of different training views.

If this verification scheme is used, we must also decide how the cost of the different patches should be combined in creating a total cost for each object hypothesis. This can be tricky especially if the same detection threshold is to be used

for all objects. The simplest approach is to sum the cost of all matching patches and give a constant penalty for missing patches. In order not to punish objects containing many patches, the cost should then be divided with the total number of patches contained in the given training view. However, not all patches are equally informative. A match between patches containing much structure should somehow provide more evidence of the object hypothesis. A heuristic approach to partially overcome this problem was proposed in [67]. Each primitive of the query view was matched to primitives from the training views. The cost was then taken as the ratio of distances associated with the best match and the next second best match, which gives an advantage to highly specific patches. A detailed study on such techniques are outside the scope of this thesis.

12.2 Preprocessing

Features such as local orientation and color can be computed in several ways, and there are different strategies for weighting the pixels based on the certainty of the feature extraction. In this section, I will discuss different choices for feature extraction and preprocessing.

12.2.1 Orientation

The orientation of local image structures can be estimated in several ways. The most straight-forward way is to take the angle of the gradient produced by a simple operator like the Sobel filter. One major problem with this approach is that the edges separating an object from the background are often among the most prominent edges of an object. If the color of the background is changed from dark to bright, the direction of the gradient can change 180°. In order to be invariant to background color, we can use double angle representation [43] or, equivalently, take the gradient orientation modulo π.

For color images, we could simply convert the image to grayscale before the gradient filtering, but this approach fails to find edges between isoluminant areas. Simply filtering each channel separately and superposing the result gives the same effect. Instead, I recommend using the color edge detector suggested by [95], also described in [51] and used in the related publication [31].

Now assume that we want to include local orientation in a channel-coded feature map. Any orientation estimate will be very noise in areas where no clear image structure is present. The simplest approach is to nevertheless weight each pixel equally, hoping that these more or less random orientations will be uniformly distributed, contributing roughly equally much to each channel. While this approach was used in [27] with good results, I personally find it much too unreliable. A region with a weak oriented texture will get as much (if not more) attention as a clear and sharp edge.

The second idea is to weight each pixel according to the gradient magnitude by using the weights w_i in the encoding (3.2), such that strong edges contribute more to the CCFM than weak edges. The exact value of the gradient magnitude is however often not a reliable feature, since it depends on factors such as camera

Figure 12.3: Thresholded gradient magnitude with the proposed normalization, using constant threshold and increasing σ to the right.

blur. This leads us to using thresholded gradient magnitude, such that the exact magnitude does not matter as long as it is large enough. However, the thresholded gradient magnitude has the form of a strip around the edge, and the width of this strip depends on the fuzziness of the edge (see Fig. 12.3). If the image is preprocessed by a smoothing filter, the width of different edges becomes more similar. This is the approach used in the implemented system. Another option could be to run a ridge-non-max-suppression on the edge image (e.g. Canny-style) prior to creating the CCFMs.

It should be noted that we should never use a threshold relative to the maximum gradient magnitude in the image or patch. In theory, this would make the algorithm more invariant to illumination, since multiplying the entire image with a constant does not change the results. However, if the entire image or patch contains only a relatively homogeneous background, this would cause lots of edges to be detected everywhere, and some positions in the background may accidentally be recognized as objects. Furthermore, the maximal gradient magnitude value in the image is typically a very unstable feature. It would be interesting to test different edge detectors with more stable certainty measures, e.g. [23].

12.2.2 Color

In order to be invariant to illumination effects such as shadows, it is advantageous to use a color space that separates the color into intensity and two illumination-independent color components like in the HSV (or HSI) color space [37]. By using only the hue and saturation component of the HSV color, we get a representation of color which is theoretically independent of lighting intensity and shadows. This strategy was used in the P-channel matching in e.g. [27]. However, this representation has a singularity near pure black and pure white, making the hue of these colors very unstable and in practice often determined by image noise only.

A simpler representation that avoids this problem it to use intensity (mean value of red, green and blue) and two color components $R - G$ and $Y - B$ (red minus green, yellow minus blue). These three components constitute an orthogonal basis in RGB space. This is a relatively common color representation in computer vision and also has some similarity to the receptive fields of certain color sensitive cells in the brain [49].

If color is used together with weighted local orientation, each pixel is weighted according to the gradient magnitude, which causes only pixels close to edges to

contribute to the CCFM. Often, the color within a uniform image region is a more stable feature than the color at actual edge pixels. Consider the case of a non-maximum-suppressed edge representing the termination of an occluding surface. The actual color on the edge may be the color of the occluded or occluding surface or a blend of both depending on the precision of the edge detection and the characteristics of the camera. Instead of picking the color from the actual edge, we should pick the colors on each side of the edge, located d pixels away orthogonal to the edge in each direction.

If thresholded edges are used without non-max-suppression, each edge will give rise to a strip of non-zero weights around the edge (see Fig. 12.3). If the color values are extracted from the original unfiltered image, the colors from both sides of all edges will be included as well as the blended colors from the actual edge. The proportion of blended an unblended colors is dependent on the fuzziness of the edge, which is in turn often dependent on the image capture conditions. However, this is still one of the simplest approaches and works well in practice. This is the method used in the experiments.

12.3 Parameter Decoupling

When building a large system of any kind, the number of parameters to tune tends to increase rapidly in the size of the system. It is important to carefully consider how these parameters are expressed in order minimize the burden of manual parameter tuning. This is best illustrated with an example. Assume that we have a simple system that finds edges by first blurring the image with a Gaussian with standard deviation σ, and then detects edges using a threshold T on the Sobel gradient magnitude. We now have two parameters to tune: σ and T. When σ is increased, all edges get more blurred, and the gradient magnitude will be lower everywhere, forcing us to select a new value for T as well. With just two parameters this may be a small problem, but imagine a tightly coupled system with tens or hundreds of parameters, where changing one means that many (if not all) other parameters need to be adjusted accordingly.

A system that does not have this tight relationship between parameters, where one can be changed more or less independently from the others, is said to have an *orthogonal parameterization*[1], and the process of making the parameters more independent is referred to as *parameter decoupling*. This section describes how this is achieved in my system. This is a practical aspect that may not be of great theoretical interest, and most advice given here may seem rather obvious. Nevertheless, overlooking this aspect can easily cost you many hours of tedious parameter tuning.

12.3.1 Prefiltering Sigma and Gradient Magnitude

The first issue relates to the introductory example of gradient magnitude thresholding. It is very common to normalize a Gaussian filter to unit sum. However, this

[1] Note that orthogonal in this context is not to be understood in a strictly mathematical sense.

is not the best approach if the blurring is to be followed by a gradient operation and thresholding.

Consider a discrete image $\mathbf{I}$ and any Gaussian-looking discrete smoothing filter kernel $\mathbf{h}$ of size $N \times N$. For simplicity of notation, $\mathbf{h}$ is indexed with the origin at the center such that the filtering result at pixel (x, y) is

$$(\mathbf{I} * \mathbf{h})[x, y] = \sum_{i,j} \mathbf{h}[i, j]\ \mathbf{I}[x + i, y + j] \ , \tag{12.1}$$

The discrete derivative of this filtering result with respect to x is

$$\frac{\mathrm{d}(\mathbf{I} * \mathbf{h})}{\mathrm{d}x}[x, y] = \sum_{i,j} \mathbf{h}[i, j]\ \mathbf{I}'_x[x + i, y + j] \ . \tag{12.2}$$

If $\mathbf{I}$ contains a vertical edge passing through (x, y), this gives the gradient magnitude of the filtered edge. If the edge is a step edge of unit height, $\mathbf{I}'_x$ is a "Dirac-line", and the derivative of the filtered image becomes

$$k = \frac{\mathrm{d}(\mathbf{I} * \mathbf{h})}{\mathrm{d}x}[x, y] = \sum_{j} \mathbf{h}[0, j] \ . \tag{12.3}$$

If the edge is not an ideal step edge, $\mathbf{I}'_x$ will contain a ridge, but the total sum of $\mathbf{I}'_x$ in the neighborhood will only depend on the height of the edge (the intensity difference between both sides of the edge). This means that (12.3) will still be a good approximation if the filter kernel varies slowly in the area of the ridge, that is if the standard deviation is large compared to the edge width.

If the filter kernel is isotropic, the same holds for edges in all directions. This shows that in order to keep the gradient magnitude of edges independent of the preprocessing filter size, we should normalize the filter kernel such that $\sum_j \mathbf{h}[0, j] = 1$, in contrast to the traditional unit normalization of the entire filter kernel. Some thresholded edges obtained using this normalization are shown in Fig. 12.3.

12.3.2 Patch Resolution and CCFMs

Consider an image patch of size (X, Y) from which a channel-coded feature map is to be constructed. If we resample the patch to a different resolution (kX, kY), the CCFM should be approximately the same in order to decouple the patch resolution from other parameters of the system. This is achieved in different ways depending on how the CCFM is constructed. The simplest case is where the encoding weight $w(x, y) = 1$ everywhere. In this case, simply use

$$c_n = \frac{1}{I} \sum_i B(\mathbf{x}_i - \tilde{\mathbf{x}}_n) \ , \tag{12.4}$$

where I is the number of encoded pixels. Things are less obvious when the weights $w(x, y)$ also depend on the input image. Consider the case where the weights are thresholded, non-max-suppressed gradient magnitudes. In this case, the number

of pixels with non-zero weights does not scale linearly in the number of pixels of the entire patch. Instead, since the non-max-suppressed edges make up a one-dimensional structure, the total edge length scales linearly in the *sides* of the patch. The normalization to use is rather

$$c_n = \frac{1}{\sqrt{I}} \sum_i w(x_i, y_i) B(\mathbf{x}_i - \tilde{\mathbf{x}}_n) \ . \tag{12.5}$$

If raw gradient magnitudes *without* thresholding and non-max-suppression are used as weights, the lateral extension of the edge must also be considered. Consider the gradient magnitude of a step edge filtered with a Gaussian with two different σ. If the normalization from 12.3.1 is used, the maximal value of the magnitude is roughly independent of σ, and all that differs is the width of the gradient magnitude profile (see Fig. 12.3). This means that

$$\sum_{x,y} w(x, y) \propto \sigma l \ , \tag{12.6}$$

where l is the total length of the edge. The length of the edge is proportional to the lengths of the sides of the image patch, whereas σ is a design parameter that can be set independent from the patch resolution. This means that a properly normalized CCFM in this case would be computed as

$$c_n = \frac{1}{\sigma\sqrt{I}} \sum_i w(x_i, y_i) B(\mathbf{x}_i - \tilde{x}_n) \ . \tag{12.7}$$

If the gradient magnitude is thresholded without non-max suppression, the width of the thresholded gradient is still proportional to σ, so (12.7) is still valid. If the gradient magnitude is subjected to some non-linear operation, e.g. a smooth thresholding, the analysis could be made much more complicated. I will stop here and use (12.7) as a heuristic normalization whenever the gradient magnitude is somehow used for computing the weights.

At this point, one may come to believe that just normalizing the CCFMs with the total sum of the input weights at runtime is a better solution. However, this has its own drawbacks, illustrated for example in the experiments in Sect. 10.5.1, where the pose estimation results were severely impaired by using this kind of normalization. The recommended approach is to normalize in a data-independent way, but still take all expected systematic effects of resolution and prefiltering into account as is described in this section.

12.3.3 CCFM Distance

Ideally, we would like to make our distance measure between the CCFMs independent of the number of channels. As was seen in Sect. 5.2.1, this is not possible using the Euclidean distance. In order to obtain this, I suggest using the square-root distance according to

$$\sum_n \left(\sqrt{\mathbf{c}_1[n]} - \sqrt{\mathbf{c}_2[n]}\right)^2 \ . \tag{12.8}$$

Note especially that the distance is not normalized by dividing with the total number of channels. Consider the spatial resolution of channels and an image region where the features are relatively homogeneous. If the channel resolution is doubled in the horizontal dimension, the number of pixels contributing to each channel is halved, which makes the new channel values approximately half the old ones. At the same time, the number of channels is doubled, so the final distance is approximately the same, without the need for further normalization.

12.4 Object-Oriented Design

Also in software engineering, people talk about orthogonal systems. Here, orthogonality means that different parts of a system can be varied independently. The implementation of one class should be able to change without affecting other parts of the system, and different classes should only communicate through well-defined interfaces. This section describes parts of my object recognition system from an object-oriented design perspective. I do not aim at a complete description of the system, but highlight some examples where the orthogonality principle has been used. All code listings contain C++ code extracted from my actual implemented system, but the code has been simplified in order to illustrate the key concepts. In the pseudocode, variables with the prefix `m_` are class member variables, and class names starting with capital `I` denote virtual interfaces.

A more detailed description about my implementation is available as a technical report [54]. A classical textbook on object-oriented design is [34], and for more C++ specific concepts, I strongly recommend [70].

12.4.1 Hypothesize-Verify Framework

Many methods in computer vision use the *hypothesize-verify*-approach. In this approach, there is some mechanism for creating hypotheses, often at random, and another mechanism to verify these hypotheses by some more extensive computation. One example of this approach is finding transformations using RANSAC [45]. Here, the hypothesize step consists of selecting a minimal subset of correspondences. In the verification step, the transformation is computed using these correspondences, and the number of inliers (points that fit to this transformation) is used as a quality measure.

The object recognition approach described in Sect. 12.1 is another instance of hypothesize-verify. In a flexible view-based object recognition framework, we would like to be able to select the verification strategy independently from how the hypotheses are created. This is possible using virtual interfaces. Listing 12.1 contains the interface specifications and a simple implementation of an abstract hypothesize-and-verify algorithm that returns all hypotheses with a cost lower than a threshold.

In another situation we might want to look for the single best hypothesis, to stop once we find a hypothesis that is better than a certain threshold, to log all examined hypotheses, and so on. The proposed design separates the control structure from the actual methods used for selecting and verifying hypotheses. This

Listing 12.1: Definition of an abstract hypothesize and verify framework

```
class IHypothesizer{
public:
  virtual void start(const Image& img) = 0;

  virtual ObjectHyp* getNextHyp() = 0;
};



class IVerifier{
public:
  virtual double verify(const ObjectHyp& hyp) = 0;
};



HypList* ObjectRecognizer::hypAndVerify(
    const Image& img, IHypothesizer* hypothesizer, IVerifier* verifier)
{
  HypList* hypList = new HypList();

  hypothesizer->start(img);
  while(...){
    ObjectHyp* hyp = hypothesizer->getNextHyp();

    double cost = verifier->verify(hyp);

    if(cost < m_threshold){
      hyp->cost = cost;
      hyps->add(hyp);
    }
  }

  return hypList;
}
```

Listing 12.2: Interface for a primitives detector

```
struct SimilarityFrame{
  double x;
  double y;
  double scale;
  double rot;

  vector<double> extra;
};


class ISimilarityDetector{
public:
  virtual vector<SimilarityFrame> run(const Image& img) = 0;

  virtual double match(const SimilarityFrame& sf1,
                       const SimilarityFrame& sf2,
                       bool useExtraOnly) = 0;
};
```

creates a more orthogonal system, where any hypothesizer can be used together with any verifier in any control structure.

12.4.2 Image Primitives

The interface of a general image primitives detector that fits within the implemented system is shown in Listing 12.2. Any class deriving from this abstract base class can be used. The class `SimilarityFrame` defines a general image primitive equipped with a position, rotation, scale and possibly some extra information, like the color of a region or line segment.

At first, one may believe that a primitives detector should simply detect primitives, and that it is sufficient to specify the `run` function in the interface. However, when matching prototypes between the query view and training view we should take into account any extra information equipped to the primitive (like color). Only the detector itself knows what this extra information means, so the matching must be done by the detector. Furthermore, when verifying a hypothesis using the local patch approach from Sect. 12.1.3, each primitive from the training view is transformed to the query view based on the current hypothesis and matched with the primitives in the query view. This matching is done using not only the similarity-invariant information, but also including the actual similarity parameters in order to enforce geometrical consistency between all primitives of the object. The best strategy for weighting the error in position, rotation and scale can be different for different primitives. For example, the endpoints of a line segment are often much less reliable than its orientation. This means that an error in orientation should be more costly than an error in scale.

The proposed solution is to include the matching function within the detector as a function `match`. A boolean flag determines whether the actual similarity parameters should be included in the matching. This design encapsulates all parts of the system that require knowledge about the primitive extraction into a single

Listing 12.3: Interface for a CCFM Encoder

```
class ICcfmEncoder
{
public:

  virtual void encode(
    const Image& image, const Image& weights, Ccfm& result) const = 0;

  virtual void encodeWithSimilDerivs(
    const Image& image, const Image& weights, CcfmWithSimilDerivs& ccfm) const = 0;
};
```

class, which makes it easy to switch from one detector to another, even at run-time.

12.4.3 View Representations

A related situation is the extraction of a channel-coded feature map including its derivatives from a similarity frame. The rest of the system should not have to know about details of this feature extraction. For example, the iterative pose optimization procedure sees only a feature vector and its derivatives. By encapsulating all information about how these features are computed into a single class, it is easy to switch between different encoding algorithms, or to a completely different view description. All that is required by the view encoder is that it derives from the abstract base class `ICcfmEncoder`, specified in Listing 12.3. The `Ccfm` and `CcfmWithSimilDerivs` are simple classes that contain the actual CCFM as a vector, without or with the derivatives respectively.

Chapter 13

Concluding Remarks

Research is like a tree search. Every question answered raises new questions, and different strategies can be adopted to traverse this tree. In a depth-first search, you select an approach and try to exhaust it, following up on every new question that appears. In a breadth-first search, you first try a multitude of approaches, keeping many questions unanswered until you decide which approach to dig deeper into. My work has been a breadth-first search restricted to the subject of channel-coding within computer vision and machine learning. During my thesis work I addressed several subjects like Bayesian statistics, correspondence-free learning, tracking and object recognition in a quest for the application in which channel-coding finds its best use. The issue that finally caught my largest interest, partly due to the requirements of the COSPAL project, was object recognition and pose estimation.

The objective of the work behind this thesis was to explore the properties of and find good uses for channel-coding in artificial learning vision systems. From a cognitive systems point-of-view, one important characteristic is its position in between the continuous and discrete domain. This property has been exploited e.g. in the correspondence-free learning, where discrete and continuous quantities can be used more or less seamlessly. Another key property is the possibility of representing the presence of multiple values. This makes channel-coded feature maps more informative and robust against clutter than a simple low-resolution image patch.

The use of B-spline piecewiseness should be considered as an engineering tool for speeding things up on today's serial computers. The exact shape of the basis function is not expected to have any practical significance, and the computational advantage may not hold on a different platform, e.g. a highly parallel analog system more similar to the human brain.

The disadvantage of the breadth-first approach is that it leaves less time for making exhaustive experiments and comparisons, and leaves many questions unanswered. Some remaining open issues are:

- Do we need all monopieces used in Chapter 4, or can we get as good results in practice using only a small subset?

- How much better is tracking using CCFMs compared to e.g. a KLT tracker? Is it worth the additional effort?
- How can the ideas about soft message passing in Chapter 6 be used to construct a computationally feasible algorithm?
- In what applications will the correspondence-free learning from Chapter 8 be really useful?
- What other features except color and orientation are useful?
- How well does the pose estimation method from Chapter 10 and 11 perform compared to any other published method?
- How much does the methods presented in this thesis have to do with what is actually going on in the human brain?

I simply have to realize that only so much can be done within a given amount of time. Some chapters of this thesis could probably be expanded into an entire thesis of its own by following-up on all loose ends, making experiments for different application areas and detailed comparisons to other published methods. There is always more to do. At some point one needs to draw a line and summarize what has been done so far in order to be able to move forward. I hereby draw my line and conclude my thesis.

Appendices

A Derivatives in Linear Algebra

A.1 Introduction

Taking derivatives of expressions involving vectors and matrices is often considered complicated and confusing. This appendix gives a practical crash course in linear algebra differentiation, without rigorous mathematical overhead. The goal of this section is being able to compute complicated derivatives like $\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\mathbf{w}}$ with

$$\mathbf{y} = (\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{W}\mathbf{p} \tag{A.1}$$

$$\mathbf{W} = \mathrm{diag}(\mathbf{w}) \tag{A.2}$$

related to what is required in Chapter 9. If you know how to find this derivative, you can safely skip this appendix.

First of all, when dealing with linear algebra, we should think about the derivative as a linear mapping transforming an increment of the input variable to an increment of the output variable. In the one-dimensional case, this mapping is conveniently represented by a single number. In the multi-dimensional case, it will be more convenient to express this linear mapping implicitly using differential notation, like e.g. $\mathrm{d}y = 2\mathbf{x}^{\mathrm{T}}\,\mathrm{d}\mathbf{x}$. This should be interpreted as "the derivative of y with respect to $\mathbf{x}$ is the linear mapping $\Delta\mathbf{x} \mapsto 2\mathbf{x}^{\mathrm{T}}\,\Delta\mathbf{x}$ ".

There are several ways to define differentials more precisely, see e.g. [19, 21]. In practice however, we can just think of $\mathrm{d}\mathbf{X}$ as a small increment of $\mathbf{X}$, such that if $\mathbf{X}$ is an $m \times n$ matrix, so is $\mathrm{d}\mathbf{X}$. These differentials can be manipulated and moved around just like regular vectors and matrices. If we ever need to express the derivative explicitly, it will be denoted as $\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\mathbf{x}}$ and defined from the indirect relationship

$$\mathrm{d}\mathbf{y} = \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\mathbf{x}}\,\mathrm{d}\mathbf{x} \quad . \tag{A.3}$$

Note that $\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\mathbf{x}}$ in this case is treated as a symbol of its own, and should not be interpreted as a true division of $\mathrm{d}\mathbf{y}$ and $\mathrm{d}\mathbf{x}$.

One common source of confusion is how the derivative should be transposed. This becomes clear from (A.3) since the product must be well-defined. For example, the derivative of a column vector with respect to a scalar is a column vector,

whereas the derivative of a scalar with respect to a column vector is a row vector. Furthermore, the derivative of a vector with respect to another vector is (in general) a matrix, and the derivative of a matrix with respect to a scalar is a matrix.

However, when taking the derivative of a matrix with respect to another matrix, we have no guarantee that the derivative can be expressed in the form prescribed by (A.3). For example, assume that we have $\mathrm{d}\mathbf{A} = \mathbf{X}\,\mathrm{d}\mathbf{B}\,\mathbf{X}^{\mathrm{T}}$. There is no general way in which this expression can be manipulated to put $\mathrm{d}\mathbf{B}$ at the end unless we introduce higher order linear objects. In this case, we are much better off using the differential notation.

The only basic rules of differentiation we will need are the following:

$$\mathrm{d}(\mathbf{A}+\mathbf{B}) = \mathrm{d}\mathbf{A} + \mathrm{d}\mathbf{B} \tag{A.4}$$

$$\mathrm{d}(\mathbf{A}\mathbf{B}) = \mathrm{d}\mathbf{A}\,\mathbf{B} + \mathbf{A}\,\mathrm{d}\mathbf{B} \tag{A.5}$$

$$\mathrm{d}(\mathbf{A}^{\mathrm{T}}) = (\,\mathrm{d}\mathbf{A}\,)^{\mathrm{T}} \tag{A.6}$$

$$\mathrm{d}\mathbf{C} = 0 \quad \text{for constant } \mathbf{C}\ . \tag{A.7}$$

The frequently used linearity $\mathrm{d}(\mathbf{C}\mathbf{A}) = \mathbf{C}\,\mathrm{d}\mathbf{A}$ (for a constant $\mathbf{C}$) follows from (A.5) and (A.7). By applying these simple rules systematically, we can tackle most linear algebra expressions.

A.2 Derivatives with respect to Vectors

As the first example of how to differentiate with respect to a vector variable, consider

$$\begin{aligned}\mathrm{d}(\|\mathbf{v}\|^2) &= \mathrm{d}(\mathbf{v}^{\mathrm{T}}\mathbf{v}) = \\ &= \mathrm{d}(\mathbf{v}^{\mathrm{T}})\,\mathbf{v} + \mathbf{v}^{\mathrm{T}}\,\mathrm{d}\mathbf{v} = \\ &= (\,\mathrm{d}\mathbf{v}\,)^{\mathrm{T}}\mathbf{v} + \mathbf{v}^{\mathrm{T}}\,\mathrm{d}\mathbf{v} = \\ &= 2\mathbf{v}^{\mathrm{T}}\,\mathrm{d}\mathbf{v}\ ,\end{aligned} \tag{A.8}$$

giving the explicit derivative as

$$\frac{\mathrm{d}(\|\mathbf{v}\|^2)}{\mathrm{d}\mathbf{v}} = 2\mathbf{v}^{\mathrm{T}}\ . \tag{A.9}$$

Another example is (considering $\mathbf{C}$ constant)

$$\begin{aligned}\mathrm{d}(\mathbf{x}^{\mathrm{T}}\,\mathbf{C}\mathbf{x}) &= \mathrm{d}(\mathbf{x}^{\mathrm{T}})\,\mathbf{C}\mathbf{x} + \mathbf{x}^{\mathrm{T}}\,\mathrm{d}(\mathbf{C}\mathbf{x}) = \\ &= (\,\mathrm{d}\mathbf{x}\,)^{\mathrm{T}}\mathbf{C}\mathbf{x} + \mathbf{x}^{\mathrm{T}}\mathbf{C}\,\mathrm{d}\mathbf{x} = \\ &= \mathbf{x}^{\mathrm{T}}(\mathbf{C}^{\mathrm{T}} + \mathbf{C})\,\mathrm{d}\mathbf{x}\ .\end{aligned} \tag{A.10}$$

The Euclidean norm of a vector can be handled using

$$\begin{aligned}\mathrm{d}\|\mathbf{v}\| &= \mathrm{d}(\|\mathbf{v}\|^2)^{1/2} = \frac{1}{2}(\|\mathbf{v}\|^2)^{-1/2}\,\mathrm{d}\big(\|\mathbf{v}\|^2\big) = \\ &= \|\mathbf{v}\|^{-1}\mathbf{v}^{\mathrm{T}}\,\mathrm{d}\mathbf{v}\ .\end{aligned} \tag{A.11}$$

A useful variety of the quotient rule can be derived as

$$\begin{aligned} \mathrm{d}(x^{-1}\mathbf{v}) \;&=\; \mathrm{d}(x^{-1})\,\mathbf{v} + x^{-1}\,\mathrm{d}\mathbf{v} \;=\; -x^{-2}\,\mathrm{d}x\,\mathbf{v} + x^{-1}\,\mathrm{d}\mathbf{v} \;= \\ &= x^{-1}(\,\mathrm{d}\mathbf{v} \;- x^{-1}\mathbf{v}\,\mathrm{d}x\,) \;. \end{aligned} \tag{A.12}$$

This can be used to differentiate a normalized vector. Let $\tilde{\mathbf{v}} = \|\mathbf{v}\|^{-1}\mathbf{v}$. We then have

$$\begin{aligned} \mathrm{d}\tilde{\mathbf{v}} \;&= \|\mathbf{v}\|^{-1}(\,\mathrm{d}\mathbf{v} \;- \tilde{\mathbf{v}}\,\mathrm{d}\|\mathbf{v}\|\,) = \\ &= \|\mathbf{v}\|^{-1}(\,\mathrm{d}\mathbf{v} \;- \tilde{\mathbf{v}}\|\mathbf{v}\|^{-1}\mathbf{v}^{\mathrm{T}}\,\mathrm{d}\mathbf{v}\,) = \\ &= \|\mathbf{v}\|^{-1}(\mathbf{I} - \tilde{\mathbf{v}}\tilde{\mathbf{v}}^{\mathrm{T}})\,\mathrm{d}\mathbf{v} \;. \end{aligned} \tag{A.13}$$

If the vector instead is normalized to unit sum by $\tilde{\mathbf{v}} = (\mathbf{1}^{\mathrm{T}}\mathbf{v})^{-1}\mathbf{v}$, we get

$$\begin{aligned} \mathrm{d}\tilde{\mathbf{v}} \;&= (\mathbf{1}^{\mathrm{T}}\mathbf{v})^{-1}(\,\mathrm{d}\mathbf{v} \;- \tilde{\mathbf{v}}\,\mathrm{d}(\mathbf{1}^{\mathrm{T}}\mathbf{v})\,) = \\ &= (\mathbf{1}^{\mathrm{T}}\mathbf{v})^{-1}(\,\mathrm{d}\mathbf{v} \;- \tilde{\mathbf{v}}\mathbf{1}^{\mathrm{T}}\,\mathrm{d}\mathbf{v}\,) = \\ &= (\mathbf{1}^{\mathrm{T}}\mathbf{v})^{-1}(\mathbf{I} - \tilde{\mathbf{v}}\mathbf{1}^{\mathrm{T}})\,\mathrm{d}\mathbf{v} \;. \end{aligned} \tag{A.14}$$

A.3 Derivatives with respect to Matrices

The inverse of a matrix can be handled by a product rule trick. Note that $\mathrm{d}\mathbf{I} = 0$, since $\mathbf{I}$ is constant. We can then write

$$\mathrm{d}\mathbf{I} \;=\; \mathrm{d}(\mathbf{A}^{-1}\mathbf{A}) \;=\; \mathrm{d}(\mathbf{A}^{-1})\,\mathbf{A} + \mathbf{A}^{-1}\,\mathrm{d}\mathbf{A} \;= 0 \;. \tag{A.15}$$

By solving for $\mathrm{d}(\mathbf{A}^{-1})$, we get

$$\mathrm{d}(\mathbf{A}^{-1}) \;= -\mathbf{A}^{-1}\,\mathrm{d}\mathbf{A}\,\mathbf{A}^{-1} \;. \tag{A.16}$$

This is an example where we really get an advantage from working with differentials instead of derivatives, since the explicit derivative $\frac{\mathrm{d}\mathbf{A}^{-1}}{\mathrm{d}\mathbf{A}}$ can not be expressed without higher-order linear objects.

When taking the derivative of a scalar with respect to a matrix $\mathbf{M}$, it may seem natural to represent the derivative with a matrix of the same dimensions as $\mathbf{M}$, but this does not follow (A.3) and is contradictory to the special case where $\mathbf{M}$ has only one column (and looks like a vector). Some textbooks, e.g. [10], actually use the opposite transpose. My advice is to avoid expressing derivatives explicitly unless (A.3) is followed and stick to the implicit differential form instead, since this is less ambiguous.

When differentiating scalar-valued expressions with respect to matrices, it is often convenient to use the Frobenius matrix product:

$$\big\langle \mathbf{A}, \mathbf{B} \big\rangle_{\mathrm{F}} = \sum_{i,j} \mathbf{A}[i,j]\mathbf{B}[i,j] \;. \tag{A.17}$$

Some useful relations involving this product are

$$\big\langle \mathbf{AB}, \mathbf{C} \big\rangle_{\mathrm{F}} = \big\langle \mathbf{B}, \mathbf{A}^{\mathrm{T}}\mathbf{C} \big\rangle_{\mathrm{F}} = \big\langle \mathbf{A}, \mathbf{CB}^{\mathrm{T}} \big\rangle_{\mathrm{F}} \tag{A.18}$$

$$\big\langle \mathbf{A}, \mathbf{B} \big\rangle_{\mathrm{F}} = \mathrm{tr}(\mathbf{A}^{\mathrm{T}}\mathbf{B}) = \mathrm{tr}(\mathbf{AB}^{\mathrm{T}}) = \mathrm{tr}(\mathbf{B}^{\mathrm{T}}\mathbf{A}) = \mathrm{tr}(\mathbf{BA}^{\mathrm{T}}) \;. \tag{A.19}$$

The product rule of differentiation also hold for the Frobenius product, in which case it looks like

$$\mathrm{d}\big(\langle \mathbf{A}, \mathbf{B} \rangle_{\mathrm{F}}\big) = \langle\, \mathrm{d}\mathbf{A}\,, \mathbf{B} \rangle_{\mathrm{F}} + \langle \mathbf{A},\, \mathrm{d}\mathbf{B}\,\rangle_{\mathrm{F}} \ . \tag{A.20}$$

Related to Chapter 7 and 8, we consider differentiating the least-squares expression

$$\mathcal{E} = \|\mathbf{C}\mathbf{A} - \mathbf{U}\|_{\mathrm{F}}^2 \ , \tag{A.21}$$

where $\mathbf{A}$ and $\mathbf{U}$ are constant. Using the Frobenius product, we get

$$\begin{aligned} \mathrm{d}\mathcal{E} \ &= \ \mathrm{d}\langle \mathbf{C}\mathbf{A} - \mathbf{U}, \mathbf{C}\mathbf{A} - \mathbf{U} \rangle_{\mathrm{F}} \ = \\ &= \langle\, \mathrm{d}\mathbf{C}\,\mathbf{A}, \mathbf{C}\mathbf{A} - \mathbf{U} \rangle_{\mathrm{F}} + \langle \mathbf{C}\mathbf{A} - \mathbf{U},\, \mathrm{d}\mathbf{C}\,\mathbf{A} \rangle_{\mathrm{F}} \\ &= 2\langle (\mathbf{C}\mathbf{A} - \mathbf{U})\mathbf{A}^{\mathrm{T}},\, \mathrm{d}\mathbf{C}\,\rangle_{\mathrm{F}} \ . \end{aligned} \tag{A.22}$$

Here, we could identify $(\mathbf{C}\mathbf{A} - \mathbf{U})\mathbf{A}^{\mathrm{T}}$ as the explicit derivative, but in order to avoid confusion about the transpose, I recommend using the implicit form from (A.22).

Least-squares problems based on expectation values can be handled in a similar way. If $\mathbf{a}$ and $\mathbf{u}$ are random vectors, we have

$$\begin{aligned} \mathcal{E} &= \mathrm{E}\left\{ \|\mathbf{C}\mathbf{a} - \mathbf{u}\|^2 \right\} \quad \Rightarrow \\ \mathrm{d}\mathcal{E} \ &= 2\langle \mathbf{C}\,\mathrm{E}\left\{\mathbf{a}\mathbf{a}^{\mathrm{T}}\right\} - \mathrm{E}\left\{\mathbf{u}\mathbf{a}^{\mathrm{T}}\right\},\, \mathrm{d}\mathbf{C}\,\rangle_{\mathrm{F}} \ . \end{aligned} \tag{A.23}$$

This result is often used in statistical signal processing, see e.g. [46].

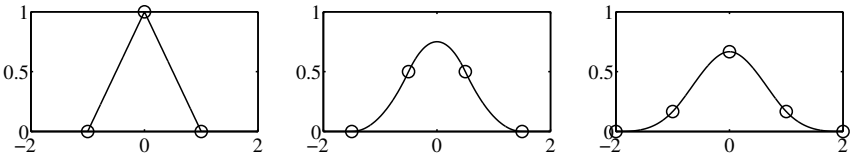

Figure B.1: First, second and third-order B-splines with knots marked as circles.

B Splines

B.1 B-Splines

In this thesis, the B-splines will play a central role as channel basis functions. A *spline* is a piecewise polynomial with the pieces tied together in regularly spaced positions called *knots*. It will be convenient to assume that the knots are unit-spaced, which can be done without loss of generality. A k'th order spline is $k-1$ times differentiable function. One central result in [90] is that all k'th order splines can be written as a linear combination of regularly spaced special basis functions called k'th order B-splines, often denoted by B_k. The zero'th order B-spline is the box function

$$B_0(x) = \begin{cases} 1 & -0.5 < x \leq 0.5 \\ 0 & \text{otherwise} \end{cases} . \tag{B.1}$$

The B-spline of order $k+1$ is obtained by convolving B_k with B_0. This produces a piecewise polynomial of order $k+1$ with unit-spaced knots. Some examples are shown in Fig. B.1. Written out explicitly, the first and second-order B-spline are

$$B_1(x) = \begin{cases} 1-|x| & 0 \leq |x| \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{B.2}$$

$$B_2(x) = \begin{cases} \frac{3}{4} - x^2 & 0 \leq |x| \leq \frac{1}{2} \\ \frac{1}{2}(\frac{3}{2} - |x|)^2 & \frac{1}{2} < |x| \leq \frac{3}{2} \\ 0 & \text{otherwise} \end{cases} . \tag{B.3}$$

In the rest of the thesis, I will avoid using a subscript k to denote B-spline order, since this subscript will be needed for other purposes in Chapter 4.

Apart from being the basis function for general k'th order splines, the B-splines have a number of interesting properties, described in [90].

B.2 Function Interpolation

Assume that we are given a discrete set of training examples $(\mathbf{x}_t, y_t)$ and look for a function $f(\mathbf{x})$ that connects to these points perfectly. One way of doing this is to choose a basis function $B(\mathbf{x})$, put one such basis function at each $\mathbf{x}_t$ and adjust the linear coefficients of each basis function. We look for a function of the form

$$f(\mathbf{x}) = \sum_t a_t B(\mathbf{x} - \mathbf{x}_t) \tag{B.4}$$

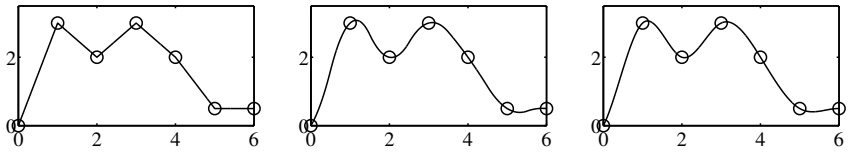

Figure B.2: The same points interpolated using first, second and third-order splines.

and find coefficients a_t such that $f(\mathbf{x}_t) = y_t$ for each t. This is just a linear equation system in the coefficients a_t which can be summarized as

$$\mathbf{Ba} = \mathbf{y} \ , \tag{B.5}$$

where $\mathbf{B}[i,j] = B(\mathbf{x}_i - \mathbf{x}_j)$, $\mathbf{y} = [y_1, \ldots, y_T]^{\mathrm{T}}$ (see e.g. [47]).

One key result in the theory on radial basis function networks is a connection between the basis function used and the properties of the resulting interpolated f. It can be shown that a function interpolation like above often finds the function f that minimizes some additional smoothness term [47]. In fact, for any linear differential operator L, there is a radially symmetric basis function B such that function interpolation like above using B as basis function finds the function f that minimizes

$$\int \|Lf(\mathbf{x})\|^2 \, \mathrm{d}\mathbf{x} \ . \tag{B.6}$$

If f is viewed as a thin metal plate that is forced to connect to the training points, L and B can be constructed to minimize the bending energy of this plate, leading to *thin plate interpolation*. B can also be selected for simplicity, e.g. as a Gaussian, and the corresponding L can be found theoretically, such that we know what is minimized.

B.3 B-Spline Interpolation

In the case of splines, the B-spline kernel acts as our basis function. In spline interpolation, we are given regularly spaced points to interpolate between, e.g. at the integers. We put a B-spline at each integer and find the interpolation coefficients by solving the equations (B.5).

Since the basis functions are all located at the integers, $\mathbf{B}[i,j] = B(i-j)$, and $\mathbf{B}$ will be a Toeplitz matrix, where only the main diagonals have non-zero elements. This allows solving (B.5) by a recursive filter which can be derived using the Z-transform. I will not go into details, but refer to [90]. The number of operations required to solve (B.5) using a recursive filter is $O(Tb)$ where T is the number of sample points and b is the *overlap* – the number of B-splines active at each position. In Fig. B.2, an example is shown where a number of points are interpolated using B-splines of order 0, 1 and 2.

Bibliography

[1] COSPAL project. `http://www.cospal.org`, Oct. 2007.

[2] F. J. Aherne, N. Thacker, and P. I. Rockett. The Bhattacharyya metric as an absolute similarity measure for frequency coded data. *Kybernetika*, 32(4):1 – 7, 1997.

[3] T. Ahonen, A. Hadid, and M. Pietikäinen. Face recognition with local binary patterns. In *Proc. European Conf. on Computer Vision*, pages 469–481, 2004.

[4] K. Arun, T. Huang, and S. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-9:698–700, 1987.

[5] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[6] D. Ballard. Generalizing the Hough transform to detect arbitrary patterns. *Pattern Recognition*, 2(13):111–122, 1981.

[7] J. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbor search in high-dimensional spaces. In *CVPR*, 1997.

[8] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(24):509–522, April 2002.

[9] A. Berger, S. D. Pietra, and V. D. Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):790–799, 1996.

[10] D. S. Bernstein. *Matrix Mathematics*. Princeton University Press, 2005.

[11] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society*, 48(3):259–302, 1986.

[12] C. N. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[13] M. J. Black and A. Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *Int. Journal of Computer Vision*, 19(1):57–91, 1996.

[14] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. 14th Annual Conf. on Uncertainty in Artificial Intelligence*, pages 33–42, 1998.

[15] Y. Cheng. Mean shift, mode seeking and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[16] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 25(5):564 – 577, May 2003.

[17] A. Comport, E. Marchand, and F. Chaumette. A real-time tracker for markerless augmented reality. In *Proc. The Second IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 36–45, 2003.

[18] P. David, D. DeMenthon, R. Duraiswami, and H. Samet. SoftPOSIT: Simultaneous pose and correspondence determination. *International Journal of Computer Vision*, 59:259–284, Sept. 2004.

[19] L. Debnath and P. Mikusiński. *Introduction to Hilbert Spaces with Applications*. Academic Press, 1999.

[20] M. Demirci, A. Shokoufandeh, S. Dickinson, Y. Keselman, and L. Bretzner. Many-to-many feature matching using spherical coding of directed graphs. In *Proc. 8th European Conf. on Computer Vision (ECCV)*, LNCS 3021, pages 322–335, May 2004.

[21] H. M. Edwards. *Advanced Calculus: A Differential Approach*. Birkhäuser, 1994.

[22] A. El-Yacoubi, M. Gilloux, R. Sabourin, and C. Suen. An HMM-based approach for off-line unconstrained handwritten word modeling and recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21(8):752–760, August 1999.

[23] M. Felsberg, R. Duits, and L. Florack. The monogenic scale space on a rectangular domain and its features. *International Journal of Computer Vision*, 64(2–3), 2005.

[24] M. Felsberg, P.-E. Forssén, and H. Scharr. Efficient robust smoothing of low-level signal features. Technical Report LiTH-ISY-R-2619, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, August 2004.

[25] M. Felsberg, P.-E. Forssén, and H. Scharr. Channel smoothing: Efficient robust smoothing of low-level signal features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(2):209–222, February 2006.

[26] M. Felsberg and G. Granlund. P-channels: Robust multivariate M-estimation of large datasets. In *International Conference on Pattern Recognition*, Hong Kong, August 2006.

[27] M. Felsberg and J. Hedborg. Real-time view-based pose recognition and interpolation for tracking initialization. *Journal of Real-Time Image Processing*, 2:103–115, 2007.

[28] M. Felsberg and J. Hedborg. Real-time visual recognition of objects and scenes using p-channel matching. In *Proc. 15th Scandinavian Conference on Image Analysis*, volume 4522 of *LNCS*, pages 908–917, 2007.

[29] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[30] P.-E. Forssén. *Low and Medium Level Vision using Channel Representations*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, March 2004. Dissertation No. 858, ISBN 91-7373-876-X.

[31] P.-E. Forssén and A. Moe. Autonomous learning of object appearances using colour contour frames. In *3rd Canadian Conference on Computer and Robot Vision*, Québec City, Québec, Canada, June 2006. IEEE Computer Society.

[32] V. Franc, M. Navara, and V. Hlavac. Sequential coordinate-wise algorithm for non-negative least squares problem. Technical Report CTU-CMP-2005-06, Faculty of Electrical Engineering, Czech Technical University, Technicka 2, 166 27 Prague 6, Czech Republic, February 2005.

[33] B. J. Fredrik Viksten and A. Moe. Comparison of local descriptors for pose estimation. Technical Report LiTH-ISY-R-2841, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, November 2007.

[34] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[35] M. S. Gazzaniga, R. B. Ivry, and G. R. Mangun. *Cognitive Neuroscience*. W. W. Norton & Company, second edition, 2002.

[36] J. B. Gomm and D. L. Yu. Selecting radial basis function network centers with recursive orthogonal least squares training. *IEEE Transactions on Neural Networks*, 11(2), March 2000.

[37] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice-Hall, second edition, 2002.

[38] G. Granlund. Organization of architectures for cognitive vision systems. In H. Christensen and H. Nagel, editors, *Cognitive Vision Systems*, LNCS. Springer, 2006.

[39] G. Granlund. Personal communication, 2007.

[40] G. Granlund, P.-E. Forssén, and B. Johansson. HiperLearn: A high performance learning architecture. Technical Report LiTH-ISY-R-2409, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, January 2002.

[41] G. H. Granlund. The complexity of vision. *Signal Processing*, 74(1):101–126, April 1999. Invited paper.

[42] G. H. Granlund. An associative perception-action structure using a localized space variant information representation. In *Proceedings of Algebraic Frames for the Perception-Action Cycle (AFPAC)*, Kiel, Germany, September 2000.

[43] G. H. Granlund and H. Knutsson. *Singal Processing for Computer Vision*. Kluwer Academic Publishers, 1995.

[44] G. H. Granlund and A. Moe. Unrestricted recognition of 3-D objects for robotics using multi-level triplet invariants. *Artificial Intelligence Magazine*, 25(2):51–67, 2004.

[45] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2001.

[46] M. H. Hayes. *Statistical digital signal processing*. John Wiley & Sons, 1996.

[47] S. Haykin. *Neural Networks, A Comprehensive Foundation*. Prentice Hall, second edition, 1999.

[48] M. T. Heath. *Scientific Computing, an Introductory Survey*. McGraw-Hill, second edition, 2002.

[49] D. Hubel. Eye, brain, and vision. http://hubel.med.harvard.edu/. Accessed February 2008.

[50] M. Isard. PAMPAS: Real-valued graphical models for computer vision. In *Proc. Conf. Computer Vision and Pattern Recognition*, 2003.

[51] B. Jähne, H. Haußecker, and P. Geißler, editors. *Handbook of Computer Vision and Applications*. Academic Press, 2000.

[52] B. Johansson. *Low Level Operations and Learning in Computer Vision*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, December 2004. Dissertation No. 912, ISBN 91-85295-93-0.

[53] B. Johansson and A. Moe. Patch-duplets for object recognition and pose estimation. In *2nd Canadian Conference on Computer and Robot Vision*, pages 9–16, Victoria, BC, Canada, May 2005. IEEE Computer Society.

[54] E. Jonsson. Object recognition using channel-coded feature maps: C++ implementation documentation. Technical Report LiTH-ISY-R-2838, Dept. EE, Linköping University, 2008.

[55] E. Jonsson and M. Felsberg. Efficient computation of channel-coded feature maps through piecewise polynomials. *Journal of Image and Vision Computing*. Submitted.

[56] E. Jonsson and M. Felsberg. Reconstruction of probability density functions from channel representations. In *Proc. 14th Scandinavian Conference on Image Analysis*, 2005.

[57] E. Jonsson and M. Felsberg. Correspondence-free associative learning. In *Proc. International Conference on Pattern Recognition (ICPR)*, 2006.

[58] E. Jonsson and M. Felsberg. Soft histograms for message passing. In *Int. workshop on the representation and use of prior knowledge in vision (WRUPKV)*, May 2006.

[59] E. Jonsson and M. Felsberg. Accurate interpolation in appearance-based pose estimation. In *Proc. 15th Scandinavian Conference on Image Analysis*, 2007.

[60] E. Jonsson, M. Felsberg, and G. Granlund. Incremental associative learning. Technical Report LiTH-ISY-R-2691, Dept. EE, Linköping University, Sept 2005.

[61] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37:183 – 233, 1999.

[62] S. Krishnamachari and R. Chellappa. Multiresolution Gauss-Markov random field models for texture segmentation. *IEEE Trans. Image Processing*, 6(2), February 1997.

[63] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2), February 2001.

[64] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *Int. Journal of Computer Vision*, 43(1):29–44, 2001.

[65] Q. Liang, I. Wendelhag, J. Wikstrand, and T. Gustavsson. A multiscale dynamic programming procedure for boundary detection in ultrasonic artery images. *IEEE Transactions on medical imaging*, 19(2):127–142, February 2000.

[66] D. G. Lowe. Object recognition from local scale-invariant features. In *IEEE Int. Conf. on Computer Vision*, Sept 1999.

[67] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2):91–110, 2004.

[68] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. 7th International Joint Conference on Artificial Intelligence*, pages 674 – 679, April 1981.

[69] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1998.

[70] S. Meyers. *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*. Addison-Wesley, 3rd edition, 2005.

[71] T. Minka. Expectation propagation for approximate bayesian inference. In *Proc. 17th Conf. in Uncertainty in Artificial Intelligence*, pages 362–369, 2001.

[72] T. Minka. *Expectation Propagation for Approximate Bayesian Inference*. PhD thesis, MIT, 2001.

[73] A. W. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. In *Proc. 14th International Conference on Machine Learning*, pages 236–244. Morgan Kaufmann, 1997.

[74] H. Murase and S. Nayar. Visual learning and recognition of 3-d objects from appearance. *International Journal of Computer Vision*, 14(1):5–24, 1995.

[75] R. C. Nelson. Finding line segments by stick growing. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 16(5), May 1994.

[76] S. Nene, S. K. Nayar, and H. Murase. Columbia object image library (coil-100). Technical Report CUCS-006-96, Dept. of Computer Science, Columbia University, 1996.

[77] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

[78] S. Obdrzalek and J. Matas. Object recognition using local affine frames on distinguished regions. In *British Machine Vision Conf.*, 2002.

[79] A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *CVPR*, 1994.

[80] J. Puzicha, Y. Rubner, C. Tomasi, and J. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. In *Proc. Int. Conf. on Computer Vision*, 1999.

[81] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce. 3D object modeling and recognition using affine-invariant patches and multi-view spatial constraints. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2003.

[82] S. Se, D. Lowe, and J. Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *Proc. Int. Conf. on Robotics and Automation*, 2001.

[83] J. Shi and C. Tomasi. Good features to track. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR94)*, June 1994.

[84] H. P. Snippe and J. J. Koenderink. Discrimination thresholds for channel-coded systems. *Biological Cybernetics*, 66:543–551, 1992.

[85] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. Brooks / Cole, 1999.

[86] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation. In *Proc. Conf. Computer Vision and Pattern Recognition*, 2003.

[87] R. C. T Drummond. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7), July 2002.

[88] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, May 1992.

[89] C. W. Therrien. *Decision, Estimation and Classification: an introduction into pattern recognition and related topics.* John Wiley & Sons, Inc., 1989.

[90] M. Unser. Splines: A perfect fit for signal and image processing. *IEEE Signal Processing Magazine*, 16(6):22–38, November 1999.

[91] S. V. Vaseghi. *Advanced Digital Signal Processing and Noise Reduction.* John Wiley & Sons, second edition, 2000.

[92] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. Int. Conf. on Computer Vision and Pattern Recognition*, 2001.

[93] J. Winn and C. M. Bishop. Variational message passing. *Journal of Machine Learning Research*, 6:661–694, 2005.

[94] C. Yang, R. Duraiswami, and L. Davis. Fast multiple object tracking via a hierarchical particle filter. In *Proc. Int. Conf. on Computer Vision*, volume 1, pages 212–219, October 2005.

[95] S. D. Zenzo. A note on the gradient of a multi-image. *Computer Vision, Graphics and Image Processing*, 33:116–125, Jan 1986.