Linköping Studies in Science and Technology
Dissertation No. 858

Low and Medium Level Vision using Channel Representations

Per-Erik Forssén

Dissertation No. 858

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

Linköping March 2004

Low and Medium Level Vision using Channel Representations

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

ISBN 91-7373-876-X ISSN 0345-7524

Don't confuse the moon
with the finger that points at it.

Zen proverb

Abstract

This thesis introduces and explores a new type of representation for low and medium level vision operations called *channel representation*. The channel representation is a more general way to represent information than e.g. as numerical values, since it allows incorporation of uncertainty, and simultaneous representation of several hypotheses. More importantly it also allows the representation of "no information" when no statement can be given. A channel representation of a scalar value is a vector of *channel values*, which are generated by passing the original scalar value through a set of *kernel functions*. The resultant representation is *sparse* and *monopolar*. The word sparse signifies that information is not necessarily present in all channels. On the contrary, most channel values will be zero. The word monopolar signifies that all channel values have the same sign, e.g. they are either positive or zero. A zero channel value denotes "no information", and for non-zero values, the magnitude signifies the relevance.

In the thesis, a framework for channel encoding and local decoding of scalar values is presented. Averaging in the channel representation is identified as a regularised sampling of a *probability density function*. A subsequent decoding is thus a mode estimation technique.

The mode estimation property of channel averaging is exploited in the *channel smoothing* technique for image noise removal. We introduce an improvement to channel smoothing, called *alpha synthesis*, which deals with the problem of jagged edges present in the original method. Channel smoothing with alpha synthesis is compared to mean-shift filtering, bilateral filtering, median filtering, and normalized averaging with favourable results.

A fast and robust blob-feature extraction method for vector fields is developed. The method is also extended to cluster constant slopes instead of constant regions. The method is intended for view-based object recognition and wide baseline matching. It is demonstrated on a wide baseline matching problem.

A sparse scale-space representation of lines and edges is implemented and described. The representation keeps line and edge statements separate, and ensures that they are localised by inhibition from coarser scales. The result is however still locally continuous, in contrast to non-max-suppression approaches, which introduce a binary threshold.

The channel representation is well suited to learning, which is demonstrated by applying it in an associative network. An analysis of representational properties of associative networks using the channel representation is made.

Finally, a reactive system design using the channel representation is proposed. The system is similar in idea to recursive Bayesian techniques using particle filters, but the present formulation allows learning using the associative networks.

Acknowledgements

This thesis could never have been written without the support from a large number of people. I am especially grateful to the following persons:

My fiancée Linda, for love and encouragement, and for constantly reminding me that there are other important things in life.

All the people at the Computer Vision Laboratory, for providing a stimulating research environment, for sharing ideas and implementations with me, and for being good friends.

Professor Gösta Granlund, for giving me the opportunity to work at the Computer Vision Laboratory, for introducing me to an interesting area of research, and for relating theories of mind and vision to our every-day experience of being.

Anders Moe and Björn Johansson for their constructive criticism on this manuscript.

Dr Hagen Spies, for giving an inspiring PhD course, which opened my eyes to robust statistics and camera geometry.

Dr Michael Felsberg, for all the discussions on channel smoothing, B-splines, calculus in general, and Depeche Mode.

Johan Wiklund, for keeping the computers happy, and for always knowing all there is to know about new technologies and gadgets.

The Knut and Alice Wallenberg foundation, for funding research within the WITAS project.

And last but not least my fellow musicians and friends in the band Pastell, for helping me to kill my spare time.

About the cover

The front cover page is a collection of figures from the thesis, arranged to constitute a face, in the spirit of painter Salvador Dali. The back cover page is a photograph of Swedish autumn leaves, processed with the SOR method in section 7.2.1, using intensities in the range $[0, 1]$, and the parameters $d_{\max} = 0.05$, binomial filter of order 11, and 5 IRLS iterations.

Contents

Chapter 1

Introduction

1.1 Motivation

The work presented in this thesis has been performed within the WITAS[1] project [24, 52, 105]. The goal of the WITAS project has been to build an autonomous[2] Unmanned Aerial Vehicle (UAV) that is able to deal with visual input, and to develop tools and techniques needed in an autonomous systems context. Extensive work on adaptation of more conventional computer vision techniques to the WITAS platform has previously been carried out by the author, and is documented in [32, 35, 81]. This thesis will however deal with basic research aspects of the WITAS project. We will introduce new techniques and information representations well suited for computer vision in autonomous systems.

Computer vision is usually described using a three level model:

- The first level, *low-level vision* is concerned with obtaining descriptions of image properties in local regions. This usually means description of colour, lines and edges, motion, as well as methods for noise attenuation.

- The next level, *medium-level vision* makes use of the features computed at the low level. Medium-level vision has traditionally involved techniques such as joining line segments into object boundaries, clustering, and computation of depth from stereo image pairs. Processing at this level also includes more complex tasks, such as the estimation of *ego motion*, i.e. the apparent motion of a camera as estimated from a sequence of camera images.

- Finally, *high-level vision* involves using the information from the lower levels to perform abstract reasoning about scenes, planning etc.

The WITAS project involves all three levels, but as the title of this thesis suggests, we will only deal with the first two levels. The unifying theme of the thesis

[1]WITAS stands for *the Wallenberg laboratory for research on Information Technology and Autonomous Systems.*

[2]An autonomous system is self guided, or without direct control of an operator.

is a new information representation called *channel representation.* All methods developed in the thesis either make explicit use of channel representations, or can be related to the channel representation.

1.2 Overview

We start the thesis in **chapter 2** with a short overview of system design principles in biological and artificial vision systems. We also give an overview of different information representations.

Chapter 3 introduces the *channel representation*, and discusses its representational properties. We also describe how a compact representation may be converted into a channel representation using a *channel encoding*, and how the compact representation may be retrieved using a *local decoding.*

Chapter 4 relates averaging in the channel representation to estimation methods from robust statistics. We re-introduce the channel representation in a statistical formulation, and show that channel averaging followed by a local decoding is a *mode estimation* technique.

Chapter 5 introduces channel representations using other kernels than the $\cos^2$ kernel. The different kernels are compared in a series of experiments. In this chapter we also explore the interference during local decoding between multiple values stored in a channel vector. We also introduce the notion of *stochastic kernels*, and extend the channel representation to higher dimensions.

Chapter 6 describes an image denoising technique called *channel smoothing.* We identify a number of problems with the original channel smoothing technique, and give solutions to them, one of them being the *alpha synthesis* technique. Channel smoothing is also compared to a number of popular image denoising techniques, such as mean-shift, bilateral filtering, median filtering, and normalized averaging.

Chapter 7 contains a method to obtain a sparse scale-space representation of homogeneous regions. The homogeneous regions are represented as sparse *blob features.* The blob feature extraction method can be applied to both grey-scale and colour images. We also extend the method to cluster constant slopes instead of locally constant regions.

Chapter 8 contains a method to obtain a sparse scale-space representation of lines and edges. In contrast to non-max-suppression techniques, the method generates a locally continuous response, which should make it well suited e.g. as input to a learning machinery.

Chapter 9 introduces an *associative network architecture* that makes use of the channel representation. In a series of experiments the descriptive powers and the noise sensitivity of the associative networks are analysed. In the experiments we also compare the associative networks with conventional function approximation using local models. We also discuss the similarities and differences between the associative networks and Radial Basis Function (RBF) networks, Support Vector Machines (SVM), and Fuzzy control.

Chapter 10 incorporates the associative networks in a feedback loop, which allows *successive recognition* in an environment with *perceptual aliasing.* A system design is proposed, and is demonstrated by solving the localisation problem in a labyrinth. In this chapter we also use reinforcement learning to learn an exploratory behaviour.

1.3 Contributions

We will now list what is believed to be the novel contributions of this thesis.

- A framework for channel encoding and local decoding of scalar values is presented in chapter 3. This material originates from the author's licentiate thesis [34], and is also contained in the article "HiperLearn: A High Performance Channel Learning Architecture" [51].
- Averaging in the channel representation is identified as a regularised sampling of a *probability density function*. A subsequent decoding is thus a mode estimation technique. This idea was originally mentioned in the paper "Image Analysis using Soft Histograms" [33], and is thoroughly explained in chapter 4.
- The local decoding for 1D and 2D Gaussian kernels in chapter 5. This material is also published in the paper "Two-Dimensional Channel Representation for Multiple Velocities" [93].
- The channel smoothing technique for image noise removal, has been investigated by several people, for earlier work by the author, see the technical report [42] and the papers "Noise Adaptive Channel Smoothing of Low Dose Images" [87], and "Channel Smoothing using Integer Arithmetic" [38]. The *alpha synthesis* approach described in chapter 6 is however a novel contribution, not published elsewhere.
- The blob-feature extraction method developed in chapter 7. This is an improved version of the algorithm published in the paper "Robust Multi-Scale Extraction of Blob Features" [41].
- A scale-space representation of lines and edges is implemented and described in chapter 8. This chapter is basically an extended version of the conference paper "Sparse feature maps in a scale hierarchy" [39].
- The analysis of representational properties of an associative network in chapter 9. This material is derived from the article "HiperLearn: A High Performance Channel Learning Architecture" [51].
- The reactive system design using channel representation in chapter 10 is similar in idea to recursive Bayesian techniques using particle filters. The use of the channel representation to define transition and narrowing, is however believed to be novel. This material was also presented in the paper "Successive Recognition using Local State Models" [37], and the technical report [36].

1.4 Notations

The mathematical notations used in this thesis should resemble those most commonly in use in the engineering community. There are however cases where there are several common styles, and thus this section has been added to avoid confusion.

The following notations are used for mathematical entities:

s	Scalars (lowercase letters in italics)
$\mathbf{u}$	Vectors (lowercase letters in boldface)
$\boldsymbol{z}$	Complex numbers (lowercase letters in italics bold)
$\mathbf{C}$	Matrices (uppercase letters in boldface)
$\mathrm{s}(\mathbf{x})$	Functions (lowercase letters)

The following notations are used for mathematical operations:

$\mathbf{A}^T$	Matrix and vector transpose
$\lfloor x \rfloor$	The floor operation
$\langle\, \mathbf{x} \,\vert\, \mathbf{y} \,\rangle$	The scalar product
$\arg \boldsymbol{z}$	Argument of a complex number
$\mathrm{conj}\, \boldsymbol{z}$	Complex conjugate
$\lvert \boldsymbol{z} \rvert$	Absolute value of real or complex numbers
$\lVert \mathbf{z} \rVert$	Matrix or vector norm
$(s * f_k)(\mathbf{x})$	Convolution
$\mathrm{adist}(\varphi_1 - \varphi_2)$	Angular distance of cyclic variables
$\mathrm{vec}(\mathbf{A})$	Conversion of a matrix to a vector by stacking the columns
$\mathrm{diag}(\mathbf{x})$	Extension of a vector to a diagonal matrix.
$\mathrm{supp}\{f\}$	The support (definition domain, or non-zero domain) of function f.

Additional notations are introduced when needed.

Chapter 2

Representation of Visual Information

This chapter gives a short overview of some aspects of image interpretation in biological and artificial vision systems. We will put special emphasis on system principles, and on which information representations to choose.

2.1 System principles

When we view vision as a sense for robots and other real-time perception systems, the parallels with biological vision at the system level become obvious. Since an autonomous robot is in direct interaction with the environment, it is faced with many of the problems that biological vision systems have dealt with successfully for millions of years. This is the reason why biological systems have been an important source of inspiration to the computer vision community, since the early days of the field, see e.g. [74]. Since biological and mechanical systems use different kinds of "hardware", there are of course several important differences. Therefore the parallel should not be taken too far.

2.1.1 The world as an outside memory

Traditionally much effort in machine vision has been devoted to methods for finding detailed reconstructions of the external world [9]. As pointed out by e.g. O'Regan [83] there is really no need for a system that interacts with the external world to perform such a reconstruction, since the world is continually "out there". He uses the neat metaphor "the world as an outside memory" to explain why. By focusing your eyes at something in the external world, instead of examining your internal model, you will probably get more accurate and up-to-date information as well.

2.1.2 Active vision

If we do not need a detailed reconstruction, then what should the goal of machine vision be? The answer to this question in the paradigm of *active vision* [3, 4, 1] is that the goal should be generation of actions. In that way the goal depends on the situation, and on the problem we are faced with.

Consider the following situation: A helicopter is situated above a road and equipped with a camera. From the helicopter we want to find out information about a car on the road below. When looking at the car through our sensor, we obtain a blurred image at low resolution. If the image is not good enough we could simply move closer, or change the zoom of the camera. The distance to the car can be obtained if we have several images of the car from different views. If we want several views, we do not actually need several cameras, we could simply move the helicopter and obtain shots from other locations.

The key idea behind active vision is that an agent in the external world has the ability to *actively* extract information from the external world by means of its actions. This ability to act can, if properly used, simplify many of the problems in vision, for instance the *correspondence problem* [9].

2.1.3 View centred and object centred representations

Biological vision systems interpret visual stimuli by generation of image features in several retinotopic maps [5]. These maps encode highly specific information such as colour, structure (lines and edges), motion, and several high-level features not yet fully understood. An object in the field of view is represented by connections between the simultaneously active features in all of the feature maps. This is called a *view centred* representation [46], and is an object representation which is *distributed* across all the feature maps, or views. Perceptual experiments are consistent with the notion that biological vision systems use multiple such view representations to represent three-dimensional objects [12]. In chapters 7 and 8 we will generate sparse feature maps of structural information, that can be used to form a view centred object representation.

In sharp contrast, many machine vision applications synthesise image features into compact object representations that are independent of the views from which they are viewed. This approach is called an *object centred* representation [46]. This kind of representation also exists in the human mind, and is used e.g. in abstract reasoning, and in spoken language.

2.1.4 Robust perception

In the book “The Blind Watchmaker” [23] Dawkins gives an account of the echolocation sense of bats. The bats described in the book are almost completely blind, and instead they emit ultrasound cries and use the echoes of the cries to perceive the world. The following is a quote from [23]:

> *It seems that bats may be using something that we could call a 'strangeness filter'. Each successive echo from a bat's own cries produces a picture of the world that makes sense in terms of the previous picture of the world built up with earlier echoes. If the bat's brain hears an echo from another bat's cry, and attempts to incorporate this into the picture of the world that it has previously built up, it will make no sense. It will appear as though objects in the world have suddenly jumped in various random directions. Objects in the real world do not behave in such a crazy way, so the brain can safely filter out the apparent echo as background noise.*

A crude equivalent to this strangeness filter has been developed in the field of *robust statistics* [56]. Here samples which do not fit the used model at all are allowed to be rejected as *outliers*. In this thesis we will develop another robust technique, using the channel information representation.

2.1.5 Vision and learning

As machine vision systems become increasingly complex, the need to specify their behaviour without explicit programming becomes increasingly apparent.

If a system is supposed to act in an un-restricted environment, it needs to be able to behave in accordance with the current surroundings. The system thus has to be flexible, and needs to be able to generate context dependent responses. This leads to a very large number of possible behaviours that are difficult or impossible to specify explicitly. Such context dependent responses are preferably learned by subjecting the system to the situations, and applying percept-response association [49].

By using learning, we are able to define *what* our system should do, not *how* it should do it. And finally, a system that is able to learn, is able to adapt to changes, and to act in novel situations that the programmer did not foresee.

2.2 Information representation

We will now discuss a number of different approaches to representation of information, which are used in biological and artificial vision systems. This is by no means an exhaustive presentation, it should rather be seen as background, and motivation for the representations chosen in the following chapters of this thesis.

2.2.1 Monopolar signals

Information processing cells in the brain exhibit either *bipolar* or *monopolar* responses. One rare example of bipolar detectors is the hair cells in semicircular canals of the vestibular system[1]. These cells hyperpolarise when the head rotates one way, and depolarise when it is rotated the other way [61].

[1]The vestibular system coordinates the orientation of the head.

Bipolar signals are typically represented numerically as values in a range centred around zero, e.g. $[-1.0, 1.0]$. Consequently monopolar signals are represented as non-negative numbers in a range from zero upwards, e.g. $[0, 1.0]$.

Interestingly there seem to be no truly bipolar detectors at any stage of the visual system. Even the bipolar cells of the retina are monopolar in their responses despite their name. The disadvantage with a monopolar detector compared to a bipolar one is that it can only respond to one aspect of an event. For instance do the retinal bipolar cells respond to either bright, or dark regions. Thus there are twice as many retinal bipolar cells, as there could have been if they had had bipolar responses. However, a bipolar detector has to produce a maintained discharge at the equilibrium. (For the bipolar cells this would have meant maintaining a level in-between the bright, and dark levels.) This results in bipolar detectors being much more sensitive to disturbances [61]. Monopolar, or non-negative representations will be used frequently throughout this thesis.

Although the use of monopolar signals is widespread in biological vision systems, it is rarely found in machine vision. It has however been suggested in [45].

2.2.2 Local and distributed coding

Three different strategies for representation of a system state using a number of signals is given by Thorpe in [97]. Thorpe uses the following simple example to illustrate their differences: We have a stimulus that can consist of a horizontal or a vertical bar. The bar can be either white, black, or absent (see figure 2.1). For simplicity we assume that the signals are binary, i.e. either active or inactive.

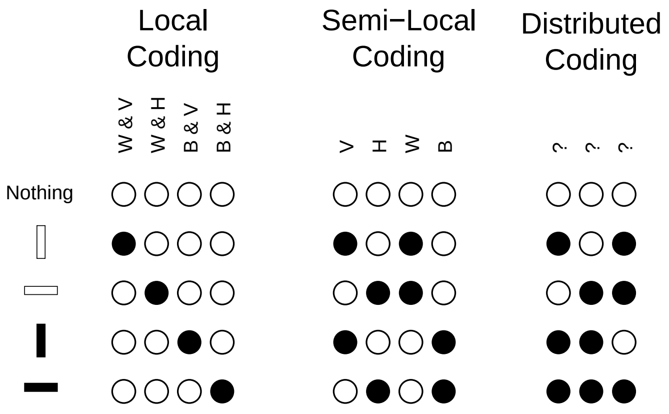


Figure 2.1: Local, semi-local, and distributed coding. Figure adapted from [97].

One way to represent the *state* of the bar is to assign one signal to each of the possible system states. This is called a *local coding* in figure 2.1, and the result is a *local representation*. One big advantage with a local representation is that the system can deal with several state hypotheses at once. In the example in figure 2.1, two active signals would mean that there was two bars present in the scene. Another way is to assign one output for each state of the two properties: orientation and colour. This is called *semi-local coding* in figure 2.1. As we move away from a completely local representation, the ability to deal with several hypotheses gradually disappears. For instance, if we have one vertical and one horizontal bar, we can deal with them separately using a semi-local representation only if they have the same colour.

The third variant in figure 2.1 is to assign one stimulus pattern to each system state. In this representation the number of output signals is minimised. This results in a representation of a given system state being distributed across the whole range of signals, hence the name *distributed representation*. Since this variant also succeeds at minimising the number of output signals, it is also a *compact coding* scheme.

These three representation schemes are also different in terms of *metric*. A *similarity metric* is a measure of how similar two states are. The coding schemes in figure 2.1 can for instance be compared by counting how many active (i.e. non-zero) signals they have in common. For the local representation, no states have common signals, and thus, in a local representation we can only tell whether two states are the same or not. For the distributed representation, the similarity metric is completely random, and thus not useful.

For the semi-local representation however, we get a useful metric. For example, bars with the same orientation, but different colour will have one active signal in common, and are thus halfway between being the same state, and being different states.

2.2.3 Coarse coding

We will now describe a coding scheme called *coarse coding*, see e.g. [96]. Coarse coding is a technique that can represent continuous state spaces. In figure 2.2 the plane represents a continuous two dimensional state space. This space is coded using a number of feature signals with circular receptive fields, illustrated by the circles in the figure.

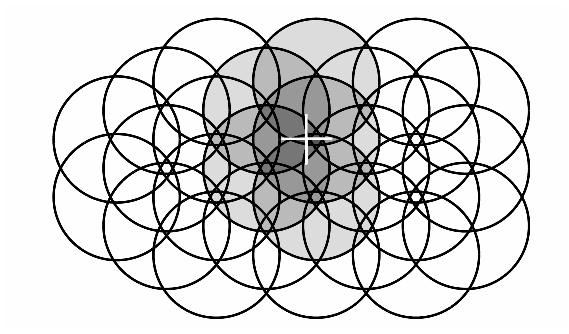


Figure 2.2: Coarse coding. Figure adapted from [96].

Each feature signal is binary, i.e. either active or inactive, and is said to coarsely represent the location in state space. Since we have several features which are partially overlapping, we can get a rough estimate of where in state-space we are, by considering all the active features. The white cross in the figure symbolises a particular state, and each feature activated by this state has its receptive field coloured grey. As can be seen, we get an increasingly darker shade of grey where several features are active, and the region where the colour is the darkest contains the actual state. Evidently, a small change in location in state space will result in a small change in the activated feature set. Thus coarse coding results in a useful similarity metric, and can be identified as a semi-local coding scheme according to the taxonomy in section 2.2.2. As we add more features in a coarse coding scheme, we obtain an increasingly better better resolution of the state space.

2.2.4 Channel coding

The *multiple channel hypothesis* is discussed by Levine and and Shefner [71] as a model for human analysis of periodic patterns. According to [71], the multiple channel hypothesis was first made by Campbell and Robson, 1968 in [13]. The multiple channel hypothesis constitutes a natural extension of coarse coding to smoothly varying features called *channels*, see figure 2.3. It is natural to consider smoothly varying and overlapping features for representation of continuous phenomena, but there is also evidence for channel representations of discrete state spaces such as representation of quantity in primates [79].

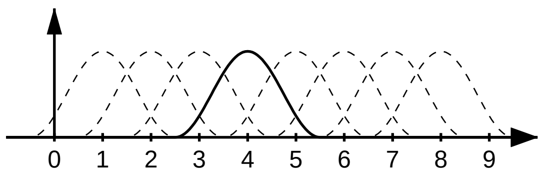


Figure 2.3: Linear channel arrangement. One channel function is shown in solid, the others are dashed.

The process of converting a state variable into channels is known in signal processing as *channel coding*, see [89] and [90], and the resultant information representation is called a *channel representation* [46, 10, 80]. Representations using channels allow a state space resolution much better than indicated by the number of channels, a phenomenon known as *hyperacuity* [89].

As is common in science, different fields of research have different names for almost the same thing. In neuroscience and computational neurobiology the concept *population coding* [108] is sometimes used as a synonym for channel representation. In neural networks the concept of *radial basis functions* (RBF) [7, 58] is used to describe responses that depend on the distance to a specific position. In control theory, the *fuzzy membership functions* also have similar shape and application [84]. The relationship between channel representation, RBF networks and Fuzzy control will be explored in section 9.7.

2.2.5 Sparse coding

A common coding scheme is the *compact coding* scheme used in data compression algorithms. Compact coding is the solution to an optimisation where the information content in each output signal is maximised. But we could also envision a different optimisation goal: maximisation of the information content in the *active* signals only (see figure 2.4). Something similar to this seems to happen at the lower levels of visual processing in mammals [31]. The result of this kind of optimisation on visual input is a representation that is *sparse*, i.e. most signals are inactive. The result of a sparse coding is typically either a local, or a semi-local representation, see section 2.2.2.

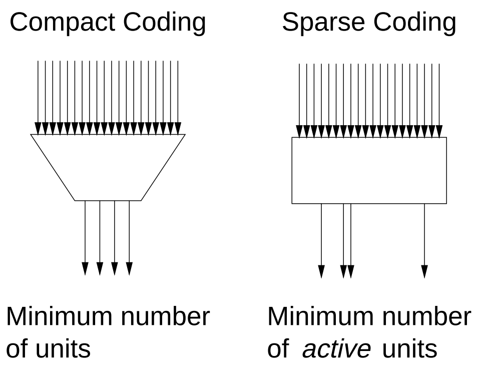


Figure 2.4: Compact and sparse coding. Figure adapted from [31].

As we move upwards in the interpretation hierarchy in biological vision systems, from cone cells, via centre-surround cells to the simple and complex cells in the visual cortex, the feature maps tend to employ increasingly sparse representations [31].

There are several good reasons why biological systems employ sparse representations, many of which could also apply to machine vision systems. For biological vision, one advantage is that the amount of signalling is kept at a low rate, and this is a good thing, since signalling wastes energy. Sparse coding also leads to representations in which pattern recognition, template storage, and matching are made easier [31, 75, 35]. Compared to compact representations, sparse features convey more information when they are active, and contrary to how it might appear, the amount of computation will not be increased significantly, since only the *active* features need to be considered.

Both coarse coding and channel coding approximate the sparse coding goal. They both produce representations where most signals are inactive. Additionally, an active signal conveys more information than an inactive one, since an active signal tell us roughly where in state space we are.

Chapter 3

Channel Representation

In this chapter we introduce the channel representation, and discuss its representational properties. We also derive expressions for channel encoding and local decoding using $\cos^2$ kernel functions.

3.1 Compact and local representations

3.1.1 Compact representations

Compact representations (see chapter 2) such as numbers, and generic object names (`house`, `door`, `Linda`) are useful for communicating precise pieces of information. One example of this is the human use of language. However, compact representations are not well suited to use if we want to learn a complex and unknown relationship between two sets of data (as in function approximation, or regression), or if we want to find patterns in one data set (as in clustering, or unsupervised learning).

Inputs in compact representations tend to describe temporally and/or spatially distant events as one thing, and thus the actual meaning of an input cannot be established until we have seen the entire training set. Another motivation for localised representations is that most functions can be sufficiently well approximated as locally linear, and linear relationships are easy to learn (see chapter 9 for more on local learning).

3.1.2 Channel encoding of a compact representation

The advantages with localised representations mentioned above motivate the introduction of the *channel representation* [46, 10, 80]. The channel representation is an *encoding* of a signal value x, and an associated confidence $r \geq 0$. This is done by passing x through a set of localised *kernel functions* $\{\mathrm{B}^k(x)\}_1^K$, and weighting the result with the confidence r. Each output signal is called a *channel*, and the vector consisting of a set of channel values

$$\mathbf{u} = r \begin{pmatrix} \mathrm{B}^1(x) & \mathrm{B}^2(x) & \dots & \mathrm{B}^K(x) \end{pmatrix}^T \tag{3.1}$$

is said to be the *channel representation* of the signal–confidence pair (x, r), provided that the channel encoding is *injective* for $r \neq 0$, i.e. there should exist a corresponding decoding that reconstructs x, and r from the channel values.

The confidence r can be viewed as a measure of reliability of the value x. It can also be used as a means of introducing a *prior*, if we want to do *Bayesian inference* (see chapter 10). When no confidence is available, it is simply taken to be $r = 1$.

Examples of suitable kernels for channel representations include Gaussians [89, 36, 93], B-splines [29, 87], and windowed $\cos^2$ functions [80]. In practise, any kernel with a shape similar to the one in figure 3.1 will do.

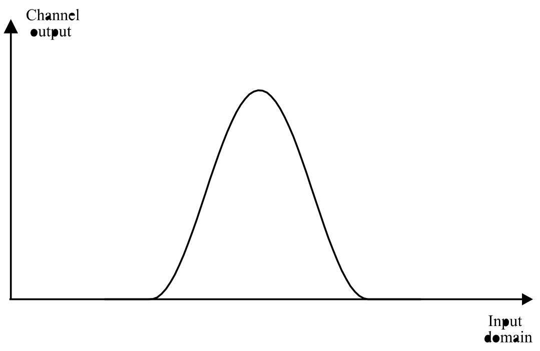


Figure 3.1: A kernel function that generates a channel from a signal.

In the following sections, we will exemplify the properties of channel representations with the $\cos^2$ kernel. Later on we will introduce the Gaussian, and the B-spline kernels. We also make a summary where the advantages and disadvantages of each kernel are compiled. Finally we put the channel representation into perspective by comparing it with other local model techniques.

3.2 Channel representation using the $\cos^2$ kernel

We will now exemplify channel representation with the $\cos^2$ kernel

$$\mathrm{B}^k(x) = \begin{cases} \cos^2(\omega d(x,k)) & \text{if} \quad \omega d(x,k) \leq \frac{\pi}{2} \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

Here the parameter k is the *kernel centre*, ω is the *channel width*, and $d(x, k)$ is a distance function. For variables in linear domains (i.e. subsets of $\mathbb{R}$) the Euclidean distance is used,

$$d(x, k) = |x - k| \,, \tag{3.3}$$

and for periodic domains (i.e. domains isomorphic with $\mathbb{S}$) with period K a modular[1] distance is used,

$$d_K(x, k) = \min(\text{mod}(x - k, K), \text{mod}(k - x, K)). \tag{3.4}$$

The measure of an angle is a typical example of a variable in a periodic domain. The total domain of a signal x can be seen as cut up into a number of local but partially overlapping intervals, $d(x, k) \leq \frac{\pi}{2\omega}$, see figure 3.2.

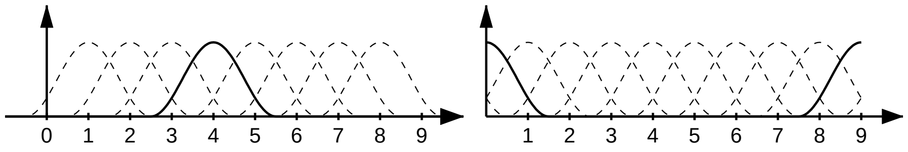


Figure 3.2: Linear and modular arrangements of $\cos^2$ kernels. One kernel is shown in solid, the others are dashed. Channel width is $\omega = \pi/3$.

For example, the channel representation of the value $x = 5.23$, with confidence $r = 1$, using the kernels in figure 3.2 (left), becomes

$$\mathbf{u} = \begin{pmatrix} 0 & 0 & 0 & 0.0778 & 0.9431 & 0.4791 & 0 & 0 \end{pmatrix}^T .$$

As can be seen, many of the channel values become zero. This is often the case, and is an important aspect of channel representation, since it allows more compact storage of the channel values. A channel with value zero is said to be *inactive*, and a non-zero channel is said to be *active*.

As is also evident in this example, the channel encoding is only able to represent signal values in a bounded domain. The exact size of the represented domain depends on the method we use to decode the channel vector, thus we will first derive a decoding scheme (in section 3.2.3) and then find out the size of the represented domain (in section 3.3).

In order to simplify the notation in (3.2), the channel positions were defined as consecutive integers, directly corresponding to the indices of consecutive kernel functions. We are obviously free to scale and translate the actual signal value in any desired way, before we apply the set of kernel functions. For instance, a signal value ξ can be scaled and translated using

$$x = \mathit{scale} \cdot (\xi - \mathit{translation}), \tag{3.5}$$

to fit the domain represented by the set of kernel functions $\{\mathrm{B}^k(x)\}_1^K$. Non-linear mappings $x = \mathrm{f}(\xi)$ are of course also possible, but they should be monotonous for the representation to be non-ambiguous.

[1]using the modulo operation $\text{mod}(x, K) = x - \lfloor x/K \rfloor K$

3.2.1 Representation of multiple values

Since each signal value will only activate a small subset of channels, most of the values in a channel vector will usually be zero. This means that for a large channel vector, there is room for more than one scalar. This is an important aspect of the channel representation, that gives it an advantage compared to compact representations. For instance, we can simultaneously represent the value 7 with confidence 0.3 *and* the value 3 with confidence 0.7 in the same channel vector

$$\mathbf{u} = \begin{pmatrix} 0 & 0.175 & 0.7 & 0.175 & 0 & 0.075 & 0.3 & 0.075 \end{pmatrix}^T .$$

This is useful to describe ambiguities. Using the channel representation we can also represent the statement “no information”, which simply becomes an all zero channel vector

$$\mathbf{u} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T .$$

There is an interesting parallel to multiple responses in biological sensory systems. If someone pokes two fingers in your back, you can feel where they are situated if they are a certain distance apart. If they are too close however, you will instead perceive one poking finger in-between the two. A representation where this phenomenon can occur is called *metameric* in psychology[2], and the states (one poking finger, or two close poking fingers) that cannot be distinguished in the given representation are called *metamers*. The metamery aspect of a channel representation (using Gaussian kernels) was studied by Snippe and Koenderink in [89, 90] from a perceptual modelling perspective.

We will refer to the smallest distance between sensations that a channel representation can handle as the *metameric distance*. Later on (section 5.4) we will have a look at how small this distance actually is for different channel representations. The typical behaviour is that for large distances between encoded values we have no interference, for intermediate distances we do have interference, and for small distances the encoded values will be averaged [34, 87].

3.2.2 Properties of the $\cos^2$ kernel

The $\cos^2$ kernel was the first one used in a practical experiment. In [80] Nordberg et al. applied it to a simple pose estimation problem. A network with channel inputs was trained to estimate channel representations of distance, horizontal position, and orientation of a wire-frame cube. The rationale for introducing the $\cos^2$ kernel was a constant norm property, and constant norm of the derivative.

Our motivations for using the $\cos^2$ kernel (3.2) is that it has a localised support, which ensures sparsity. Another motivation is that for values of $\omega = \pi/N$ where $N \in \{3, 4, ...\}$ we have

$$\sum_k \mathrm{B}^k(x) = \frac{\pi}{2\omega} \qquad \text{and} \qquad \sum_k \mathrm{B}^k(x)^2 = \frac{3\pi}{8\omega}. \tag{3.6}$$

[2]Another example of a metameric representation is colour, which basically is a three channel representation of wavelength.

This implies that the sum, and the vector norm of a channel value vector generated from a single signal–confidence pair is *invariant* to the value of the signal x, as long as x is within the represented domain of the channel set (for proofs, see theorems A.3 and A.4 in the appendix). The constant sum implies that the encoded value, and the encoded confidence can be decoded independently. The constant norm implies that the kernels locally constitute a *tight frame* [22], a property that ensures uniform distribution of signal energy in the channel space, and makes a decoding operation easy to find.

3.2.3 Decoding a $\cos^2$ channel representation

An important property of the channel representation is the possibility to retrieve the signal–confidence pairs stored in a channel vector. The problem of decoding signal and confidence values from a set of channel function values, superficially resembles the reconstruction of a continuous function from a set of frame coefficients. There is however a significant difference: *we are not interested in reconstructing the exact shape of a function, we merely want to find all peak locations and their heights.*

In order to decode several signal values from a channel vector, we have to make a *local decoding*, i.e. a decoding that assumes that the signal value lies in a specific limited interval (see figure 3.3).

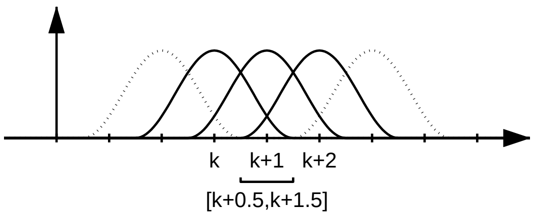


Figure 3.3: Interval for local decoding ($\omega = \pi/3$).

For the $\cos^2$ kernel, and the local tight frame situation (3.6), it is suitable to use decoding intervals of the form $[k - 1 + N/2, k + N/2]$ (see theorem A.1 in the appendix). The reason for this is that a signal value in such an interval will only activate the N nearest channels, see figure 3.3. Decoding a channel vector thus involves examining all such intervals for signal–confidence pairs, by computing estimates using only those channels which should have been activated.

The local decoding is computed using a method illustrated in figure 3.4. The channel values, u^k, are now seen as samples from a kernel function translated to have its peak at the represented signal value $\hat{x}$.

We denote the index of the first channel in the decoding interval by l (in the figure we have $l = 4$), and use groups of consecutive channel values $\{u^l, u^{l+1}, \ldots, u^{l+N-1}\}$.

If we assume that the channel values of the N active channels constitute an encoding of a single signal–confidence pair (x, r), we obtain N equations

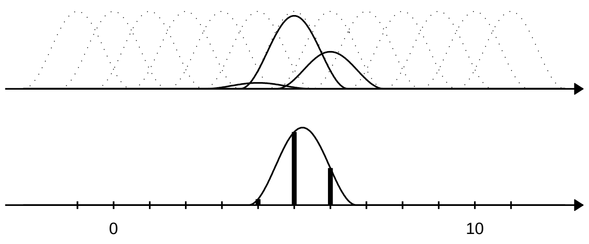


Figure 3.4: Example of channel values ($\omega = \pi/3$, and $\hat{x} = 5.23$).

$$\begin{pmatrix} u^l \\ u^{l+1} \\ \vdots \\ u^{l+N-1} \end{pmatrix} = \begin{pmatrix} r\mathrm{B}^l(x) \\ r\mathrm{B}^{l+1}(x) \\ \vdots \\ r\mathrm{B}^{l+N-1}(x) \end{pmatrix} . \tag{3.7}$$

We will now transform an arbitrary row of this system in a number of steps

$$u^{l+d} = r\mathrm{B}^{l+d}(x) = r\cos^2(\omega(x-l-d)) \tag{3.8}$$
$$u^{l+d} = r/2(1+\cos(2\omega(x-l-d)) \tag{3.9}$$
$$u^{l+d} = r/2(1+\cos(2\omega(x-l))\cos(2\omega d)+\sin(2\omega(x-l))\sin(2\omega d)) \tag{3.10}$$
$$u^{l+d} = \begin{pmatrix} \frac{1}{2}\cos(2\omega d) & \frac{1}{2}\sin(2\omega d) & \frac{1}{2} \end{pmatrix} \begin{pmatrix} r\cos(2\omega(x-l)) \\ r\sin(2\omega(x-l)) \\ r \end{pmatrix} . \tag{3.11}$$

We can now rewrite (3.7) as

$$\underbrace{\begin{pmatrix} u^l \\ u^{l+1} \\ \vdots \\ u^{l+N-1} \end{pmatrix}}_{\mathbf{u}} = \underbrace{\frac{1}{2}\begin{pmatrix} \cos(2\omega 0) & \sin(2\omega 0) & 1 \\ \cos(2\omega 1) & \sin(2\omega 1) & 1 \\ \vdots & \vdots & \vdots \\ \cos(2\omega(N-1)) & \sin(2\omega(N-1)) & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} r\cos(2\omega(x-l)) \\ r\sin(2\omega(x-l)) \\ r \end{pmatrix}}_{\mathbf{p}} . \tag{3.12}$$

For $N \geq 3$, this system can be solved using a least-squares fit

$$\mathbf{p} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{u} = \mathbf{W}\mathbf{u} . \tag{3.13}$$

Here $\mathbf{W}$ is a constant matrix, which can be computed in advance and be used to decode all local intervals. The final estimate of the signal value becomes

$$\hat{x} = l + \frac{1}{2\omega}\arg\left[p_1 + \boldsymbol{i}p_2\right] . \tag{3.14}$$

For the confidence estimate, we have two solutions

$$\hat{r}_1 = |p_1 + \boldsymbol{i} p_2| \quad \text{and} \quad \hat{r}_2 = p_3 \,. \tag{3.15}$$

The case of $\omega = \pi/2$ requires a different approach to find $\hat{x}$, $\hat{r}_1$, and $\hat{r}_2$ since $\mathbf{u} = \mathbf{Ap}$ is under-determined when $N = 2$. Since the channel width $\omega = \pi/2$ has proven to be not very useful in practise, this decoding approach has been moved to observation A.5 in the appendix.

When the two confidence measures are equal, we have a group of consecutive channel values $\{u^l,\ u^{l+1},\ \ldots,\ u^{l+N-1}\}$ that originate from a single signal value x. The fraction $\hat{r}_1/\hat{r}_2$ is independent of scalings of the channel vector, and could be used as a measure of the validity of the model assumption (3.7). The model assumption will quite often be violated when we use the channel representation. For instance, response channels estimated using a linear network will not in general fulfill (3.7) even though we may have supplied such responses during training. We will study the robustness of the decoding (3.14), as well as the behaviour in case of interfering signal–confidence pairs in chapter 5. See also [36].

The solution in (3.14) is said to be a *local decoding*, since it has been defined using the assumption that the signal value lies in a specific interval (illustrated in figure 3.3). If the decoded value lies outside the interval, the local peak is probably better described by another group of channel values. For this reason, decodings falling outside their decoding intervals are typically neglected.

We can also note that for the local tight frame situation (3.6), the matrix $\mathbf{A}^T\mathbf{A}$ becomes diagonal, and we can compute the local decoding as a local weighted summation of complex exponentials

$$\hat{x} = l + \frac{1}{2\omega} \arg \left[\sum_{k=l}^{l+N-1} u^k e^{\boldsymbol{i} 2\omega(k-l)} \right] . \tag{3.16}$$

For this situation the relation between neighbouring channel values tells us the signal value, and the channel magnitudes tell us the confidence of this statement. In signal processing it is often argued that it is important to attach a measure of confidence to signal values [48]. The channel representation can be seen as a unified representation of signal *and* confidence.

3.3 Size of the represented domain

As mentioned in section 3.2, a channel representation is only able to represent values in a bounded domain, which has to be known beforehand. We will now derive an expression for the size of this domain. We start by introducing a notation for the *active domain* (non-zero domain, or support) of a channel

$$S_k = \{x : \mathrm{B}^k(x) > 0\} =]l_k, u_k[\tag{3.17}$$

where l_k and u_k are the lower and upper bounds of the active domain. Since the kernels should go smoothly to zero (see section 3.2.2), this is always an open interval, as indicated by the brackets. For the $\cos^2$ kernel (3.2), and the constant sum situation (3.6), the *common support* of N channels, S_k^N becomes

$$S_k^N = S_k \cap S_{k+1} \cap \ldots \cap S_{k+N-1} =]k-1+N/2, k+N/2[\,. \tag{3.18}$$

This is proven in theorem A.1 in the appendix. See also figure 3.5 for an illustration.

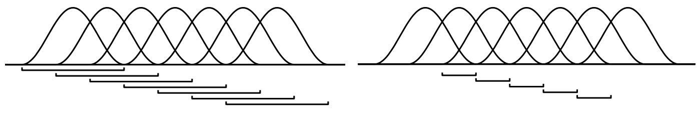


Figure 3.5: Common support regions for $\omega = \pi/3$. Left: supports S_k for individual channels. Right: common supports S_k^3.

If we perform the local decoding using groups of N channels with $\omega = \pi/N$, $N \in \mathbb{N}/\{1\}$, we will have decoding intervals of type (3.18). These intervals are all of length 1, and thus they do not overlap (see figure 3.5, right). We now modify the upper end of the intervals

$$S_k^N =]k-1+N/2, k+N/2] \tag{3.19}$$

in order to be able to join them. This makes no practical difference, since all that happens at the boundary is that one channel becomes inactive. For a channel representation using K channels (with $K \geq N$) we get a represented interval of type

$$R_K^N = S_1^N \cup S_2^N \cup \ldots \cup S_{K-N+1}^N =]N/2, K+1-N/2] \tag{3.20}$$

This expression is derived in theorem A.2 in the appendix.

For instance $K = 8$, and $\omega = \pi/3$ (and thus $N = 3$), as in figure 3.2, left, will give us

$$R_8^3 =]3/2, 8+1-3/2] =]1.5, 7.5]\,.$$

3.3.1 A linear mapping

Normally we will need to scale and translate our measurements to fit the represented domain for a given channel set. We will now describe how this linear mapping is found.

If we have a variable $\xi \in [r_l, r_u]$ that we wish to map to the domain $R_K^N =]R_L, R_U]$ using $x = t_1\xi + t_0$, we get the system

$$\begin{pmatrix} R_L \\ R_U \end{pmatrix} = \begin{pmatrix} 1 & r_l \\ 1 & r_u \end{pmatrix} \begin{pmatrix} t_0 \\ t_1 \end{pmatrix} \tag{3.21}$$

with the solution

$$t_1 = \frac{R_U - R_L}{r_u - r_l} \quad \text{and} \quad t_0 = R_L - t_1 r_l\,. \tag{3.22}$$

Inserting the boundaries of the represented domain R_K^N, see (3.20) gives us

$$t_1 = \frac{K+1-N}{r_u - r_l} \quad \text{and} \quad t_0 = \frac{N}{2} - t_1 r_l \,. \tag{3.23}$$

This expression will be used in the experiment sections to scale data to a given set of kernels.

3.4 Summary

In this chapter we have introduced the channel representation concept. Important properties of channel representations are that we can represent ambiguous statements, such as "either the value 3 or the value 7". We can also represent the confidence we have in each hypothesis, i.e. statements like "the value 3 with confidence 0.6 or the value 7 with confidence 0.4" are possible. We are also able to represent the statement "no information", using an all zero channel vector.

The signal–confidence pairs stored in a channel vector can be retrieved using a *local decoding*. The local decoding problem superficially resembles the reconstruction of a continuous signal from a set of samples, but it is actually different, since we are only interested in finding the peaks of a function. We also note that the decoding has to be local in order to decode multiple values.

An important limitation in channel representation is that we can only represent signals with bounded values. I.e. we must know a largest possible value, and a smallest possible value of the signal to represent. For a bounded signal, we can derive an optimal linear mapping that maps the signal to the interval a given channel set can represent.

Chapter 4

Mode Seeking and Clustering

In this chapter we will relate averaging in the channel representation to estimation methods from robust statistics. We do this by re-introducing the channel representation in a slightly different formulation.

4.1 Density estimation

Assume that we have a set of vectors $\mathbf{x}_n$, that are measurements from the same source. Given this set of measurements, can we make any prediction regarding a new measurement? If the process that generates the measurements does not change over time it is said to be a *stationary stochastic process*, and for a stationary process, an important property is the relative frequencies of the measurements. Estimation of relative frequencies is exactly what is done in probability density estimation.

4.1.1 Kernel density estimation

If the data $\mathbf{x}_n \in \mathbb{R}^d$ come from a discrete distribution, we could simply count the number of occurrences of each value of $\mathbf{x}_n$, and use the relative frequencies of the values as measures of probability. An example of this is a histogram computation. However, if the data has a continuous distribution, we instead need to estimate a *probability density function* (PDF) $f : \mathbb{R}^d \to \mathbb{R}_+ \cup \{0\}$. Each value $f(\mathbf{x})$ is non-negative, and is called a *probability density* for the value $\mathbf{x}$. This should not be confused with the *probability* of obtaining a given value, which is normally zero for a signal with a continuous distribution. The integral of $f(\mathbf{x})$ over a domain tell us the probability of $\mathbf{x}$ occurring in this domain. In all practical situations we have a finite amount of samples, and we will thus somehow have to limit the degrees of freedom of the PDF, in order to avoid over-fitting to the sample set. Usually a smoothness constraint is applied, as in the *kernel density estimation* methods, see e.g. [7, 43]. A *kernel density estimator* estimates the value of the PDF in point $\mathbf{x}$ as

$$\hat{f}(\mathbf{x}) = \frac{1}{Nh^d}\sum_{n=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_n}{h}\right) \tag{4.1}$$

where $K(\mathbf{x})$ is the *kernel function*, and h is a scaling parameter that is usually called the *kernel width*, and d is the dimensionality of the vector space. If we require that

$$H(\mathbf{x}) \geq 0 \quad \text{and} \quad \int H(\mathbf{x})d\mathbf{x} = 1 \quad \text{for} \quad H(\mathbf{x}) = \frac{1}{h^d}K\left(\frac{\mathbf{x}}{h}\right) \tag{4.2}$$

we know that $\hat{f}(\mathbf{x}) \geq 0$ and $\int \hat{f}(\mathbf{x})d\mathbf{x} = 1$ as is required of a PDF.

Using the scaled kernel $H(\mathbf{x})$ above, we can rewrite (4.1) as

$$\hat{f}(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} H(\mathbf{x}-\mathbf{x}_n)\,. \tag{4.3}$$

In other words (4.1) is a sample average of $H(\mathbf{x}-\mathbf{x}_n)$. As the number of samples tends to infinity, we obtain

$$\begin{aligned}\lim_{N\to\infty}\frac{1}{Nh^d}\sum_{n=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_n}{h}\right) &= E\left\{H\left(\mathbf{x}-\mathbf{x}_n\right)\right\} = \int f(\mathbf{x}_n)H(\mathbf{x}-\mathbf{x}_n)d\mathbf{x}_n \\ &= (f * H)(\mathbf{x})\,. \end{aligned} \tag{4.4}$$

This means that in an expectation sense, the kernel $H(\mathbf{x})$ can be interpreted as a low-pass filter acting on the PDF $f(\mathbf{x})$. This is also pointed out in [43]. Thus $H(\mathbf{x})$ is the smoothness constraint, or regularisation, that makes the estimate more stable. This is illustrated in figure 4.1. The figure shows three kernel density estimates from the same sample set, using a Gaussian kernel

$$K(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}}e^{-0.5\mathbf{x}^T\mathbf{x}} \tag{4.5}$$

with three different kernel widths.

4.2 Mode seeking

If the data come from a number of different sources, it would be a useful aid in prediction of new measurements to have estimates of the means and covariances of the individual sources, or *modes* of the distribution. See figure 4.1 for an example of a distribution with four distinct modes (the peaks). Averaging of samples in *channel representation* [47, 80, 34] (see also chapter 3), followed by a *local decoding* is one way to estimate the modes of a distribution.

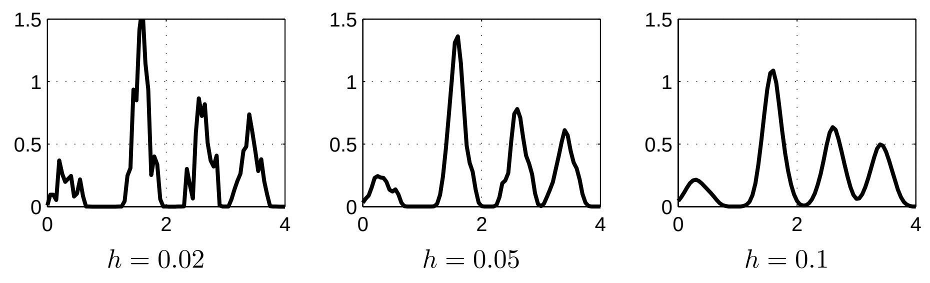


Figure 4.1: Kernel density estimates for three different kernel widths.

4.2.1 Channel averaging

With the interpretation of the convolution in (4.4) as a low-pass filter, it is easy to make the association to signal processing with sampled signals, and suggest regular sampling as a representation of $\hat{f}(\mathbf{x})$. If the sample space $\mathbb{R}^d$ is low dimensional, and samples only occur in a *bounded domain*[1] $\mathcal{A}$, (i.e. $f(\mathbf{x}) = 0 \ \forall \mathbf{x} \notin \mathcal{A}$) it is feasible to represent $\hat{f}(\mathbf{x})$ by estimates of its values at regular positions. If the sample set $\mathcal{S} = \{\mathbf{x}_n\}_1^N$ is large this would also reduce memory requirements compared to storing all samples.

Note that the analogy with signal processing and sampled signals should not be taken too literally. We are not at present interested in the exact shape of the PDF, we merely want to find the modes, and this does not require the kernel $H(\mathbf{x})$ to constitute a band-limitation, as would have been the case if reconstruction of (the band-limited version of) the continuous signal $\hat{f}(\mathbf{x})$, from its samples was our goal.

For simplicity of notation, we only consider the case of a one dimensional PDF $f(x)$ in the rest of this section. Higher dimensional channel representations will be introduced in section 5.6.

In the channel representation, a set of non-negative kernel functions $\{\mathrm{H}^k(x)\}_1^K$ is applied to each of the samples x_n, and the result is optionally weighted with a confidence $r_n \geq 0$,

$$\mathbf{u}_n = r_n \begin{pmatrix} \mathrm{H}^1(x_n) & \mathrm{H}^2(x_n) & \dots & \mathrm{H}^K(x_n) \end{pmatrix}^T . \tag{4.6}$$

This operation defines the *channel encoding* of the signal–confidence pair (x_n, r_n), and the resultant vector $\mathbf{u}_n$ constitutes a channel representation of the signal–confidence, provided that the channel encoding is *injective* for $r \neq 0$, i.e. there exists a corresponding decoding that reconstructs the signal, and its confidence from the channels.

We additionally require that the consecutive, integer displaced kernels $H^k(x)$, are shifted versions of an even function $H(x)$, i.e.

$$H^k(x) = H(x - k) = H(k - x) . \tag{4.7}$$

[1]Bounded in the sense that $\mathcal{A} \subset \{\mathbf{x} : (\mathbf{x} - \mathbf{m})^T \mathbf{M} (\mathbf{x} - \mathbf{m}) \leq 1\}$ for some choice of $\mathbf{m}$ and $\mathbf{M}$.

We now consider an average of channel vectors

$$\mathbf{u} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_n \quad \text{with elements} \quad u^k = \frac{1}{N}\sum_{n=1}^{N} u_n^k\,. \tag{4.8}$$

If we neglect the confidence r_n, we have

$$u_n^k = H(x_n - k) = H(k - x_n)\,. \tag{4.9}$$

By inserting (4.9) into (4.8) we see that $u^k = \hat{f}(k)$ according to (4.3). In other words, *averaging of samples in the channel representation is equivalent to a regular sampling of a kernel density estimator.* Consequently, the expectation value of a channel vector $\mathbf{u}$ is a sampling of the PDF $f(x)$ filtered with the kernel $H(x)$. I.e. for each channel value we have

$$E\{u_n^k\} = E\{H^k(x_n)\} = \int H^k(x)f(x)dx = (f * H)(k)\,. \tag{4.10}$$

We now generalise the interpretation of the local decoding in section 3.2.3. The local decoding of a channel vector is a procedure that takes a subset of the channel values (e.g. $\{u^k \dots u^{k+N-1}\}$), and computes the mode location x, the confidence/probability density r, and if possible the standard deviation σ of the mode

$$\{x, r, \sigma\} = \text{dec}(u^k, u^{k+1}, \dots, u^{k+N-1})\,. \tag{4.11}$$

The actual expressions to compute the mode parameters depend on the used kernel. A local decoding for the $\cos^2$ kernel was derived in section 3.2.3. This decoding did not give an estimate of the standard deviation, but in chapter 5 we will derive local decodings for Gaussian and B-spline kernels as well (in sections 5.1.1 and 5.2.2), and this motivates the general formulation above.

4.2.2 Expectation value of the local decoding

We have identified the local decoding as a mode estimation procedure. Naturally we would like our mode estimation to be as accurate as possible, and we also want it to be *unbiased*. This can be investigated using the expectation value of the local decoding. Recall the expectation value of a channel (4.10). For the $\cos^2$ kernel this becomes

$$E\{u_n^k\} = \int_{S_k} \cos^2(\omega(x-k))f(x)dx \tag{4.12}$$

where S_k is the support of kernel k (see section 3.3). We will now require that the PDF is restricted to the common support S_l^N used in the local decoding. This allows us to write the expectation value of one of the channel values used in the decoding as

$$E\{u_n^{l+d}\} = \int_{S_l^N} \cos^2(\omega(x-l-d))f(x)dx$$
$$= \frac{1}{2}\begin{pmatrix}\cos(2\omega d) & \sin(2\omega d) & 1\end{pmatrix} \underbrace{\begin{pmatrix}\int_{S_l^N} \cos(2\omega(x-l))f(x)dx \\ \int_{S_l^N} \sin(2\omega(x-l))f(x)dx \\ \int_{S_l^N} f(x)dx\end{pmatrix}}_{E\{\mathbf{p}\}} \tag{4.13}$$

using the same method as in (3.8)-(3.11). We can now stack such equations for all involved channel values, and solve for $E\{\mathbf{p}\}$. This is exactly what we did in the derivation of the local decoding. If we assume a Dirac PDF, i.e. $f(x) = r\delta(x-\mu)$, we obtain

$$E\{\mathbf{p}\} = \begin{pmatrix} r\cos(2\omega(\mu-l)) \\ r\sin(2\omega(\mu-l)) \\ r \end{pmatrix} . \tag{4.14}$$

Plugging this into the final decoding step (3.14) gives us the mode estimate $\hat{x} = \mu$. In general however, (3.14) will not give us the exact mode location. In appendix A, theorem A.6 we prove that, if a mode f is restricted to the support of the decoding S_l^N, and is even about the mean μ (i.e. $f(\mu+x) = f(\mu-x)$), (3.14) is an *unbiased* estimate of the mean

$$E\{\hat{x}\} = l + \frac{1}{2\omega}\arg\left[E\{p_1\} + \boldsymbol{i}E\{p_2\}\right] = \mu = E\{x_n\} . \tag{4.15}$$

When f has an odd component, the local decoding tends to overshoot the mean slightly, *seemingly always in the direction of the mode of the density*[2]. In general however, these conditions are not fulfilled. It is for instance impossible to have a shift invariant estimate for non-Dirac densities, when the decoding intervals S_l^N are non-overlapping. For an experimental evaluation of the behaviour under more general conditions, see [36].

4.2.3 Mean-shift filtering

An alternative way to find the modes of a distribution is through gradient ascent on (4.1), as is done in *mean-shift filtering* [43, 16]. Mean-shift filtering is a way to cluster a sample set, by moving each sample toward the closest mode, and this is done through gradient ascent on the kernel density estimate.

Assuming that the kernel $K(\mathbf{x})$ is differentiable, the gradient of $f(\mathbf{x})$ can be estimated as the gradient of (4.1). I.e

$$\hat{\nabla} f(\mathbf{x}) = \frac{1}{Nh^{d+1}} \sum_{n=1}^{N} \nabla K\left(\frac{\mathbf{x}-\mathbf{x}_n}{h}\right) . \tag{4.16}$$

This expression becomes particularly simple if we use the *Epanechnikov kernel* [43]

[2]Note that this is an empirical observation, no proof is given.

$$K(\mathbf{x}) = \begin{cases} c(1-\mathbf{x}^T\mathbf{x}) & \text{if } \mathbf{x}^T\mathbf{x} \leq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4.17}$$

Here c is a normalising constant that ensures $\int K(\mathbf{x})d\mathbf{x} = 1$. For the kernel (4.17) we define[3] the gradient as

$$\nabla K(\mathbf{x}) = \begin{cases} -2c\mathbf{x} & \text{if } \mathbf{x}^T\mathbf{x} \leq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4.18}$$

Inserting this into (4.16) gives us

$$\hat{\nabla} f(\mathbf{x}) = \frac{2c}{Nh^{d+2}} \sum_{\mathbf{x}_n \in S_h(\mathbf{x})} (\mathbf{x}_n - \mathbf{x}) \,, \tag{4.19}$$

where $S_h(\mathbf{x}) = \{\mathbf{x}_n \in \mathbb{R}^d : ||\mathbf{x}_n - \mathbf{x}|| \leq h\}$ is the support of the kernel (4.17).

Instead of doing gradient ascent using (4.19) directly, Fukunaga and Hostetler [43] use an approximation of the *normalised gradient*

$$\hat{\nabla} \ln f(\mathbf{x}) \approx \frac{\hat{\nabla} f(\mathbf{x})}{\hat{f}(\mathbf{x})} = \frac{d+2}{h^2}\mathbf{m}(\mathbf{x}) \tag{4.20}$$

where $\mathbf{m}(\mathbf{x})$ is the *mean shift vector*

$$\mathbf{m}(\mathbf{x}) = \frac{1}{k(\mathbf{x})} \sum_{\mathbf{x}_n \in S_h(\mathbf{x})} (\mathbf{x}_n - \mathbf{x}) \,. \tag{4.21}$$

Here $k(\mathbf{x})$ is the number of samples inside the support $S_h(\mathbf{x})$.

Using the normalised gradient, [43] proceeds to define the following clustering algorithm

$$\text{For all } n \quad \begin{cases} \bar{\mathbf{x}}^0 & = \mathbf{x}_n \\ \bar{\mathbf{x}}^{i+1} & = \bar{\mathbf{x}}^i + a\hat{\nabla} \ln f(\bar{\mathbf{x}}^i) \end{cases} \tag{4.22}$$

where a should be chosen to guarantee convergence. In [43] the choice
$a = h^2/(d+2)$ is made. This results in the iteration rule

$$\bar{\mathbf{x}}^{i+1} = \frac{1}{k(\mathbf{x})} \sum_{\bar{\mathbf{x}}_n^i \in S_h(\bar{\mathbf{x}}^i)} \bar{\mathbf{x}}_n^i \,. \tag{4.23}$$

That is, in each iteration we replace $\bar{\mathbf{x}}^i$ by a mean inside a window centred around it. Cheng [16] calls this mean of already shifted points a *blurring process*, and contrasts this with the *non-blurring process*

$$\bar{\mathbf{x}}^{i+1} = \frac{1}{k(\mathbf{x})} \sum_{\mathbf{x}_n \in S_h(\bar{\mathbf{x}}^i)} \mathbf{x}_n \,. \tag{4.24}$$

[3]The gradient is actually undefined when $\mathbf{x}^T\mathbf{x} = 1$.

The behaviour of the two approaches for clustering are similar, but only the non-blurring mean-shift is mode seeking, and is the one that is typically used. Cheng also generalises the concept of mean shift, by considering (4.24) to be a mean shift using the uniform kernel,

$$K(\mathbf{x}) = \begin{cases} 1 & \mathbf{x}^T\mathbf{x} \leq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{4.25}$$

and suggests other weightings inside the window $S_h(\mathbf{x})$. The generalised mean-shift iteration now becomes

$$\bar{\mathbf{x}}^{i+1} = \frac{\sum K(\mathbf{x}_n - \bar{\mathbf{x}}^i)\mathbf{x}_n}{\sum K(\mathbf{x}_n - \bar{\mathbf{x}}^i)} \,. \tag{4.26}$$

Since mean shift according to (4.26) using the kernel (4.25) finds the modes of (4.1) using the Epanechnikov kernel (4.17), the Epanechnikov kernel is said to be the *shadow* of the uniform kernel. Similarly mean shift using other kernels finds peaks of (4.1) computed from their shadow kernels.

Figure 4.2 shows an illustration of mean shift clustering. A mean shift iteration has been started in each of the data points, and the trajectories $\bar{\mathbf{x}}^0$, $\bar{\mathbf{x}}^1$, $\ldots \bar{\mathbf{x}}^*$ have been plotted. As can be seen in the centre plot, all trajectories end in either of five points, indicating successful clustering of the data set.

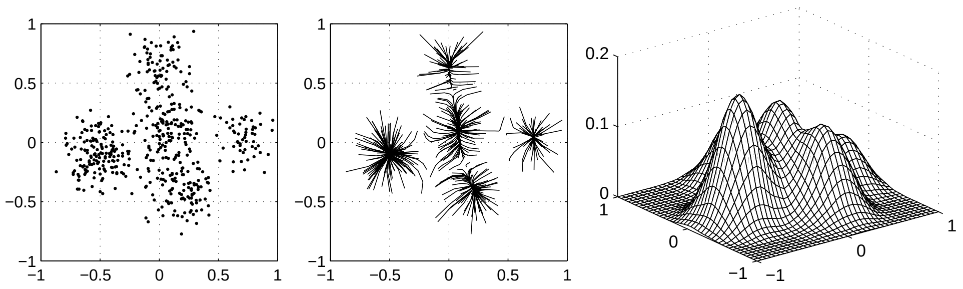


Figure 4.2: Illustration of mean shift filtering. Left: data set. Centre: trajectories of the mean shift iterations using a uniform kernel with $h = 0.3$. Right: Kernel density estimate using Epanechnikov kernel with $h = 0.3$.

Gradient ascent with fixed step length often suffers from slow convergence, but (4.26) has the advantage that it moves quickly near points of low density (this can be inferred by observing that the denominator of (4.26) is a kernel density estimator without the normalisation factor.).

Mean shift filtering was recently re-introduced in the signal processing community by the Cheng paper [16], and has been developed by Comaniciu and Meer in [19, 20] into algorithms for edge-preserving filtering, and segmentation.

4.2.4 M-estimators

M-estimation[4] (first described by Huber in [62] according to Hampel et al. [56]), or *the approach based on influence functions* [56] is a technique that attempts to remove the sensitivity to outliers in parameter estimation. Assume $\{\mathbf{x}_n\}_1^N$ are samples in some parameter space, and we want to estimate the parameter choice $\bar{\mathbf{x}}$ that best fits the data. This estimation problem is defined by the the following objective function

$$\bar{\mathbf{x}} = \arg\min_{\mathbf{x}} J(\mathbf{x}) = \arg\min_{\mathbf{x}} \sum_{n=1}^{N} \rho(r_n/h) \quad \text{where} \quad r_n = ||\mathbf{x} - \mathbf{x}_n|| \,. \tag{4.27}$$

Here $\rho(r)$ is an *error norm*[5], r_n are residuals, and h is a scale parameter. The error norm $\rho(r)$ should be nondecreasing with increasing r [95].

Both the least-squares problem, and median computation are special cases of M-estimation with the error norms $\rho(r) = r^2$, and $\rho(r) = |r|$ respectively. See figure 4.3 for some common examples of error norms.

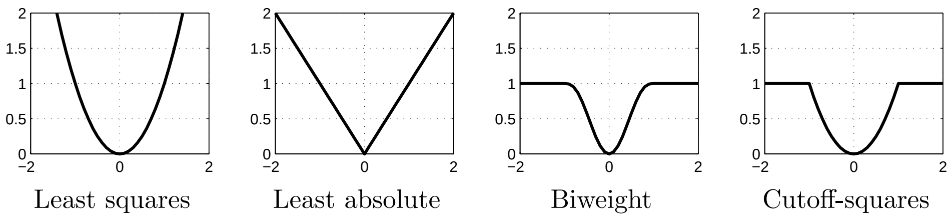


Figure 4.3: Some common error norms.

Note that, in general, (4.27) is a non-convex problem which may have several local minima.

Solutions to (4.27) are also solutions to

$$\sum_{n=1}^{N} \varphi(r_n/h) = 0 \quad \text{where} \quad \varphi(r) = \frac{\partial \rho(r)}{\partial r} \,. \tag{4.28}$$

The function $\varphi(r)$ is called the *influence function.* The solution to (4.27) or (4.28) is typically found using *iterated reweighted least-squares*[6] (IRLS), see e.g. [109, 95]. As the name suggests IRLS is an iterative algorithm. In general it requires an initial guess close to the optimum of (4.27).

To derive IRLS we start by assuming that the expression to be minimised in (4.27) can be written as a weighted least-squares expression

[4]The name comes from "generalized *m*aximum likelihood" according to [56].
[5]The error norm is sometimes called a *loss function.*
[6]Iterated reweighted least squares is also known as a W-estimator [56].

$$J(\mathbf{x}) = \sum_{n=1}^{N} \rho(r_n/h) = \sum_{n=1}^{N} r_n^2 w(r_n/h)\,. \tag{4.29}$$

We now compute the gradient with respect to $\{r_n\}_1^N$ of both sides

$$\frac{1}{h}\begin{pmatrix}\varphi(r_1/h) & \dots & \varphi(r_N/h)\end{pmatrix}^T = \frac{2}{h}\begin{pmatrix}r_n w(r_n/h) & \dots & r_N w(r_N/h)\end{pmatrix}^T\,. \tag{4.30}$$

This system of equations is fulfilled for the weight function $w(r) = \varphi(r)/r/2$. This can be simplified to $w(r) = \varphi(r)/r$, while still giving a solution to (4.27). This gives us the weights for one step in the IRLS process:

$$\bar{\mathbf{x}}^i = \arg\min_{\mathbf{x}} \sum_{n=1}^{N} (r_n^i)^2 w(r_n^{i-1}/h) \quad \text{where} \quad r_n^i = ||\bar{\mathbf{x}}^i - \mathbf{x}_n||\,. \tag{4.31}$$

Each iteration of (4.31) is a convex problem, which can be solved by standard techniques for least-squares problems, such as Gaussian elimination or SVD. By computing the derivative with respect to $\bar{\mathbf{x}}^i$ (treating w as constant weights), we get

$$\sum_{n=1}^{N} 2(\bar{\mathbf{x}}^i - \mathbf{x}_n) w(r_n^{i-1}/h) = \mathbf{0} \tag{4.32}$$

with the solution

$$\bar{\mathbf{x}}^i = \frac{\sum \mathbf{x}_n w(r_n^{i-1}/h)}{\sum w(r_n^{i-1}/h)} = \frac{\sum \mathbf{x}_n w(||\bar{\mathbf{x}}^{i-1} - \mathbf{x}_n||/h)}{\sum w(||\bar{\mathbf{x}}^{i-1} - \mathbf{x}_n||/h)}\,. \tag{4.33}$$

By comparing this with (4.26), we see that the iterations in IRLS are equivalent to those in mean-shift filtering if we set the kernel in mean-shift filtering equal to the scaled weight function in IRLS i.e.

$$K(\bar{\mathbf{x}} - \mathbf{x}_n) = w(||\bar{\mathbf{x}} - \mathbf{x}_n||/h)\,. \tag{4.34}$$

From this we can also infer that the error norm corresponds to the kernel of the corresponding kernel density estimator, up to a sign (mean-shift is a maximisation, and M-estimation is a minimisation) and an offset.

4.2.5 Relation to clustering

Clustering is the problem of partitioning a non-labelled data set into a number of clusters, or classes, in such a way that similar samples are grouped into the same cluster. Clustering is also known as *data partitioning*, *segmentation* and *vector quantisation*. It is also one of the approaches to *unsupervised learning*. See e.g. [63] for an overview of different clustering approaches.

The mode seeking problem is related to clustering in the sense that each mode can be seen as a natural cluster prototype. For mean shift, this connection is especially direct, since each sample can be assigned a class label depending on which mode the mean-shift iteration ended in [19, 20]. For channel averaging we can use the distances to each of the decoded modes to decide which cluster a sample should belong to.

4.3 Summary and comparison

In this chapter we have explained three methods that find modes in a distribution. All three methods start from a set of samples, and have been shown to estimate the modes of the samples under a regularisation with a smoothing kernel (for M-estimation we only get one mode). We have shown that non-blurring meanshift is equivalent to a set of M-estimations using IRLS started in each sample. Channel averaging, on the other hand is a different method, that approaches the same problem from a different angle.

Which method is preferable depends on the problem to be solved. Both channel averaging and mean-shift filtering are intended for estimation of all modes of a distribution, or for clustering. If we have a large number of samples inside the kernel window, mean-shift filtering is at a disadvantage, since each iteration involves an evaluation of e.g. (4.16) for all samples, which would be cumbersome indeed. In contrast, for mode seeking using channel averaging, only the averaging step is affected by the number of samples. Another advantage with channel averaging is that it has a constant data-independent complexity. For mean-shift we can only have an *expected* computational complexity since it is an iterative method, where the convergence speed depends on the data. For high dimensional sample spaces, or if the domain of the sample space is large, mean-shift is at an advantage, since the number of required channels grows exponentially with the vector space dimension. When the used kernel is small, mean-shift also becomes favourable.

Chapter 5

Kernels for Channel Representation

In this chapter we will introduce two new kernel functions and derive encodings and local decodings for them. The kernels are then compared with the $\cos^2$ kernel (introduced in chapter 3), with respect to a number of properties. We also introduce the notion of stochastic kernels, and study the interference of multiple peaks when decoding a channel vector. Finally we extend the channel representation to higher dimensions.

5.1 The Gaussian kernel

Inspired by the similarities between channel averaging and kernel density estimation (see sections 4.1.1 and 4.2.1) we now introduce the Gaussian kernel

$$\mathrm{B}^k(x) = e^{-\dfrac{(x-k)^2}{2\sigma^2}} \,. \tag{5.1}$$

Here the k parameter is the channel centre, and σ is the channel width. The σ parameter can be related to the ω parameter of the $\cos^2$ kernel by requiring that the areas under the kernels should be the same. Since $A_{\text{gauss}} = \sqrt{2\pi}\sigma$, and $A_{\cos^2} = \pi/2\omega$, we get

$$\sigma = \sqrt{\pi/8}/\omega \quad \text{and} \quad \omega = \sqrt{\pi/8}/\sigma\,. \tag{5.2}$$

Just like before, channel encoding is done according to (3.1). Figure 5.1 shows an example of a Gaussian channel set. Compared to the $\cos^2$ kernel, the Gaussian kernel has the disadvantage of not having a compact support. This means that we will always have small non-zero values in each channel (unless we threshold the channel vector, or weight the channel values with a relevance $r = 0$). Additionally, the Gaussian kernels do not have either constant sum or norm as the $\cos^2$ kernels do, see (3.6).

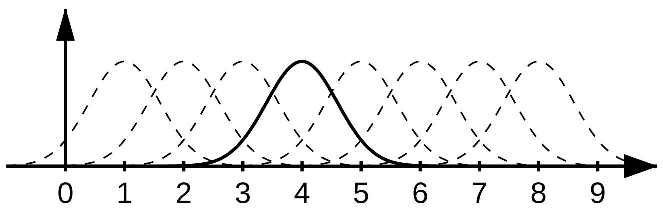


Figure 5.1: Example of a Gaussian channel set. The width $\sigma = 0.6$ corresponds roughly to $\omega = \pi/3$.

5.1.1 A local decoding for the Gaussian kernel

We will now devise a local decoding for this channel representation as well. If we look at three neighbouring channel values around $k = l$, we obtain three equations

$$\begin{pmatrix} u^{l-1} \\ u^{l} \\ u^{l+1} \end{pmatrix} = \begin{pmatrix} r\mathrm{B}^{l-1}(x) \\ r\mathrm{B}^{l}(x) \\ r\mathrm{B}^{l+1}(x) \end{pmatrix} . \tag{5.3}$$

The logarithm of an arbitrary row can be written as

$$\ln u^{l+d} = \ln r + \ln \mathrm{B}^{l+d}(x) = \ln r - \frac{(x-l-d)^2}{2\sigma^2} \tag{5.4}$$

$$= \begin{pmatrix} 1 & d & d^2 \end{pmatrix} \underbrace{\begin{pmatrix} \ln r - \frac{(x-l)^2}{2\sigma^2} & \frac{x-l}{\sigma^2} & -\frac{1}{2\sigma^2} \end{pmatrix}^T}_{\mathbf{p}} . \tag{5.5}$$

If we apply this to each row of (5.3), we obtain an equation system of the form

$$\ln \mathbf{u} = \mathbf{D}\mathbf{p} \tag{5.6}$$

with the solution

$$\mathbf{p} = \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & -1 & \frac{1}{2} \end{pmatrix}}_{\mathbf{D}^{-1}} \ln \mathbf{u} . \tag{5.7}$$

From the solution $\mathbf{p}$ we can find the estimates $\hat{x}$, $\hat{\sigma}$, and $\hat{r}$ as

$$\hat{x} = l - \frac{p_2}{2p_3} , \qquad \hat{\sigma} = \sqrt{-\frac{1}{2p_3}} , \quad \text{and} \quad \hat{r} = e^{p_1 - \frac{p_2^2}{4p_3}} .$$

This gives us one decoding per group of 3 channels. Just like in the $\cos^2$ decoding (see section 3.2.3) we remove those decodings that lie more than a distance 0.5 from the decoding interval centre, since the corresponding peak is probably better described by another group of channels.

The Gaussian kernel has potential for use in channel averaging, see section 4.2.1, since they provide a direct estimate of the standard deviation for an additive Gaussian noise component. Using the addition theorem for Gaussian variances, we can estimate the noise as $\hat{\sigma}_{\text{noise}} = \sqrt{\hat{\sigma}^2 - \sigma^2}$. Just like in the $\cos^2$ case, the $\hat{r}$-value of the decoding is the probability density at the mode. The new parameter $\hat{\sigma}_{\text{noise}}$ tells us the width of the mode, and can be seen as a measure of how accurate the localisation of the mode is. For a high confidence in a peak location $\hat{\sigma}_{\text{noise}}$ should be small, and $\hat{r}N$ (where N is the number of samples) should be large.

5.2 The B-spline kernel

B-splines are a family of functions normally used to define a basis for interpolating splines, see e.g. [21, 100]. The central[1] B-spline of degree zero is defined as a rectangular pulse

$$\mathrm{B}^0(x) = \begin{cases} 1 & -0.5 \leq x < 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

B-splines of higher degrees are defined recursively, and can be obtained through convolutions

$$\mathrm{B}^n(x) = \mathrm{B}^{n-1} * \mathrm{B}^0(x) = \underbrace{\mathrm{B}^0 * \mathrm{B}^0 \ldots * \mathrm{B}^0(x)}_{(n+1)\text{ times}} . \tag{5.9}$$

As the degree is increased, the basis functions tend toward a Gaussian shape (see figure 5.2). In fact, according to the central limit theorem, a Gaussian is obtained as n approaches infinity.

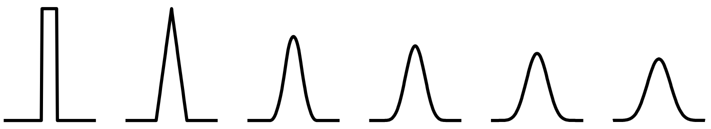


Figure 5.2: Central B-splines of degrees 0,1...5.

If we require explicit expressions for the piecewise polynomials which the B-spline consists of, the following *recurrence relation* [21] is useful

$$\mathrm{B}^n(x) = \frac{x + (n+1)/2}{n}\mathrm{B}^{n-1}_{-1/2}(x) + \frac{(n+1)/2 - x}{n}\mathrm{B}^{n-1}_{1/2}(x) . \tag{5.10}$$

Here, shifted versions of a B-spline are denoted $\mathrm{B}^n_k(x) = \mathrm{B}^n(x-k)$. Using (5.10), we obtain the following expression for B-splines of degree 1

[1]B-splines are often defined with $\mathrm{B}^0(x)$ having the support $x \in [0, 1]$ instead.

$$\begin{aligned}
\mathrm{B}^1(x) &= (x+1)\mathrm{B}^0_{-1/2}(x) + (1-x)\mathrm{B}^0_{1/2}(x) && (5.11)\\
&= \begin{cases} x+1 & -1 \le x < 0 \\ 1-x & 0 \le x < 1 \\ 0 & \text{otherwise.} \end{cases} && (5.12)
\end{aligned}$$

For degree 2 we get

$$\begin{aligned}
\mathrm{B}^2(x) &= \frac{x+3/2}{2}\mathrm{B}^1_{-1/2}(x) + \frac{3/2-x}{2}\mathrm{B}^1_{1/2}(x) && (5.13)\\
&= \frac{(x+3/2)^2}{2}\mathrm{B}^0_{-1}(x) + \left(\frac{3}{4} - x^2\right)\mathrm{B}^0(x) + \frac{(x-3/2)^2}{2}\mathrm{B}^0_1(x) && (5.14)\\
&= \begin{cases} (x+3/2)^2/2 & -1.5 \le x < -0.5 \\ 3/4 - x^2 & -0.5 \le x < 0.5 \\ (x-3/2)^2/2 & 0.5 \le x < 1.5 \\ 0 & \text{otherwise.} \end{cases} && (5.15)
\end{aligned}$$

By applying the binomial theorem on the Fourier transform of (5.9), and going back it is also possible to derive the following expression [100]

$$\mathrm{B}^n(x) = \frac{1}{n!}\sum_{k=0}^{n+1}\binom{n+1}{k}(-1)^k \max\left(0, x-k+\frac{n+1}{2}\right)^n . \tag{5.16}$$

5.2.1 Properties of B-splines

The B-spline family have a number of useful properties:

1. Positivity
$$\mathrm{B}^n(x) > 0,\ x \in \,]-(n+1)/2, (n+1)/2[\ .$$

2. Compact support
$$\mathrm{B}^n(x) = 0,\ x \notin [-(n+1)/2, (n+1)/2]\ .$$

3. Constant sum
$$\sum_k \mathrm{B}^n_k(x) = 1 \quad \text{regardless of } x. \tag{5.17}$$
For a proof, see theorem B.1 in the appendix.

4. For B-splines of degree $n \ge 1$, the original scalar value may be retrieved by the first moment
$$x = \sum_k k\mathrm{B}^n_k(x)\,. \tag{5.18}$$
For a proof, see theorem B.2 in the appendix.

These properties, make B-splines useful candidates for kernels in the *channel representation.*

5.2.2 B-spline channel encoding and local decoding

Using B-spline kernels of degree ≥ 1 we may define a B-spline channel representation, where a signal–confidence pair (x, r) can be encoded according to (3.1). I.e. we have

$$u^k = r\mathrm{B}_k(x)\,. \tag{5.19}$$

Due to the constant sum property of the B-spline kernel (5.17), the confidence may be extracted from a channel set as

$$r = \sum_k u^k\,. \tag{5.20}$$

We can further retrieve the signal value times the confidence using the first moment (see equation 5.18)

$$xr = \sum_k ku^k\,.$$

Thus it is convenient to first compute the confidence, and then to extract the signal value as

$$\hat{x} = \frac{\sum_k ku^k}{\sum_k u^k} = \frac{1}{r}\sum_k ku^k\,. \tag{5.21}$$

Figure 5.3 shows an example of a B-spline channel set. Just like the $\cos^2$ kernel, the B-splines have compact support, and constant sum. Their norm however is not constant. A value x encoded using a B-spline channel representation of degree n, will have at most $n+1$ active channel values. This makes a local decoding using a group of $n+1$ consecutive channel values reasonable. It also means that a B-spline channel representation of degree n is comparable to a $\cos^2$ channel representation with width

$$\omega = \pi/(n+1)\,. \tag{5.22}$$

Just like in the $\cos^2$ and Gaussian decodings, (see sections 3.2.3 and 5.1.1) we remove those decodings that lie more than a distance 0.5 from the decoding window centre.

Note that the local decoding described here assumes a Dirac PDF, see section 4.2.2. A more elaborate local decoding, which deals with non-Dirac PDFs has been developed by Felsberg, and a joint publication is under way [30].

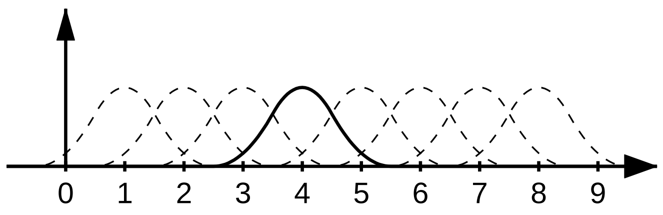


Figure 5.3: Example of a B-spline channel set. The degree $n = 2$ is comparable to $\omega = \pi/3$ for the $\cos^2$ kernels.

5.3 Comparison of kernel properties

In this section we will compare the three kernel families, $\cos^2$, Gaussians and B-splines. We will compare them according to a number of different criteria.

5.3.1 The constant sum property

The sum[2] of a channel vector $\mathbf{u} = r\begin{pmatrix}\mathrm{B}^1(x) & \mathrm{B}^2(x) & \dots & \mathrm{B}^K(x)\end{pmatrix}^T$ is given by

$$||\mathbf{u}(x)||_1 = \sum_{k=1}^{K} |r\mathrm{B}^k(x)| = r\sum_{k=1}^{K} \mathrm{B}^k(x)\,. \tag{5.23}$$

As proven in theorems A.3 and B.1, (5.23) is constant for $\cos^2$ and B-spline channels as long as x is inside the represented domain (i.e. $x \in R_N^K$). Note that since the Gaussian kernels do not have compact support, (5.23) will actually depend on the value of K. Figure 5.4 (left) shows numerical evaluations of (5.23) for $\sigma = 0.4$, 0.6, 0.8, and 1.0. A channel set of $K = 11$ channels has been used, with channel positions $-3, -2, \dots 7$.

As can be seen in the figure the sum peaks near channel positions and has distinct minima right in-between two channel positions. To measure the amount of deviation we use a normalised peak-to-peak distance

$$\varepsilon_{\mathrm{pp}} = (\max(||\mathbf{u}||_1) - \min(||\mathbf{u}||_1))/\mathrm{mean}(||\mathbf{u}||_1)\,. \tag{5.24}$$

This measure is plotted in figure 5.4 (right). As can be seen, the sum is practically constant for $\sigma > 0.6$. For a finite number of channels, the deviation from the result in figure 5.4 is very small[3].

[2]Since the channel representation is non-negative, the sum is actually the l_1 norm.
[3]When $K = 500$, the deviation is less than 2×10^{-4} (worst case is $\sigma = 1.0$).

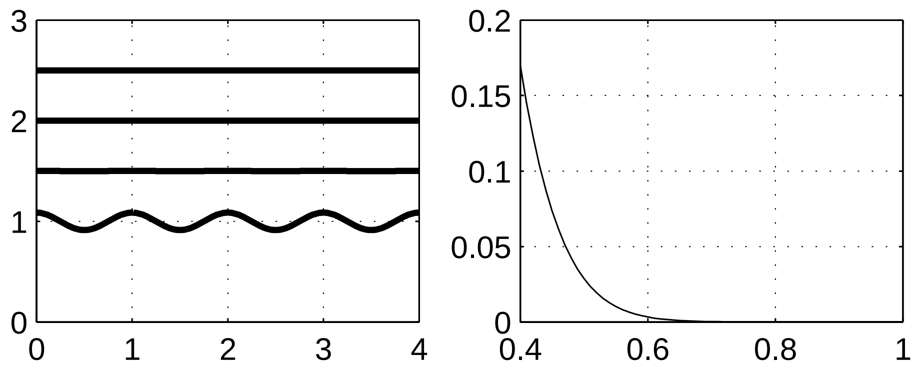


Figure 5.4: Sum as function of position. Left: Sums for $\sigma = 0.4, 0.6, 0.8, 1.0$ (bottom to top). Right: Normalised peak-to-peak distance $\varepsilon_{pp}(\sigma)$.

5.3.2 The constant norm property

The l_2 norm $||\mathbf{u}||$ of a channel vector $\mathbf{u}(x) = r\left(\mathrm{B}^1(x) \quad \mathrm{B}^2(x) \quad \ldots \quad \mathrm{B}^K(x)\right)^T$ is given by

$$||\mathbf{u}(x)||^2 = r^2 \sum_{k=1}^{K} \mathrm{B}^k(x)^2 \,. \tag{5.25}$$

As proven in theorem A.4, (5.25) is constant for $\cos^2$ channels as long as x is inside the represented domain (i.e. $x \in R_N^K$). Since the Gaussian kernels do not have compact support, (5.25) will depend on the value of K, so they are a bit problematic. Figure 5.5 shows a numerical comparison of Gaussian and B-spline kernels. In order to compare them, kernels with corresponding[4] widths have been used (i.e according to $\sigma = (n+1)/\sqrt{8\pi}$). The experiment used $K = 11$ channels positioned at $-3, -2, \ldots 7$. For a finite number of channels, the deviation from this experiment is very small[5] for the Gaussians.

As can be seen in the figure the norm peaks near channel positions and has distinct minima right in-between two channel positions. To compare the amount of deviation we use a normalised peak-to-peak distance

$$\varepsilon_{\mathrm{pp}} = (\max(||\mathbf{u}||) - \min(||\mathbf{u}||))/\mathrm{mean}(||\mathbf{u}||) \tag{5.26}$$

on the interval $[0, 4]$. Figure 5.5 (right) shows plots of this measure for the Gaussian and B-spline kernels. As can be seen, both kernels tend toward a constant norm as the channel width is increased. The Gaussians however have a significantly faster convergence. This is most likely due to their non compact support. For all widths except ($n = 1$, $\sigma = 0.4$) the deviation is smaller for the Gaussians.

[4]For a motivation to the actual correspondence criteria see discussion around (5.2) and (5.22).
[5]When $K = 500$, the deviation is less than 5×10^{-10} (worst case is $\sigma = 1.0$).

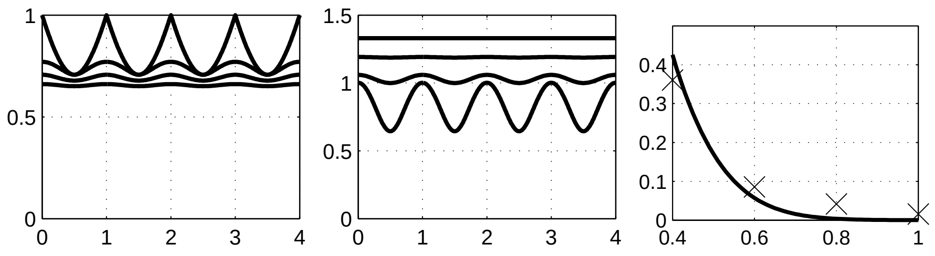


Figure 5.5: Norm as function of position. Left: Using B-spline kernels n = 1,2,3,4 (top to bottom curves). Centre: Using Gaussian kernels with σ = 0.40, 0.60, 0.80, 1.00 (bottom to top curves). Right: Solid $\varepsilon_{\mathrm{pp}}(\sigma)$ for Gaussian kernels. Crosses indicate the B-spline results for $n = 1, 2, 3, 4$.

5.3.3 The scalar product

We will now have a look at the scalar product of two channel vectors. Note that the constant norm property does not imply a position invariant scalar product, merely that the highest value of the scalar product stays constant. We will thus compare all three kernels. Figure 5.6 shows a graphical comparison of the scalar product functions for $\cos^2$, B-spline, and Gaussian kernels.

In this experiment we have encoded the value 0, and computed scalar products between this vector and encodings of varying positions. i.e.

$$s(x) = \mathbf{u}(0)^T\mathbf{u}(x) = \sum_{k=1}^{K} \mathrm{B}^k(0)\mathrm{B}^k(x)\,. \tag{5.27}$$

Each plot shows 10 superimposed curves $s(x)$, where the channel positions have displaced in steps of 0.1. I.e.

$$s(x) = \sum_{k=1}^{K} \mathrm{B}^k(0-d)\mathrm{B}^k(x-d) \tag{5.28}$$

for $d \in \{0, 0.1, 0.2, \dots 0.9\}$. As can be seen, the scalar product is position variant for all kernels, but the amount of variance with position decreases as the channel width increases. For the $\cos^2$ kernel, the position variance goes to zero as we approach the peak of scalar product function (5.27). As we increase the channel widths the position variance drops for the other kernels, and especially for the Gaussians with $\sigma = 1.0$, $s(x)$ looks very stable.

The problem with the position variance of the scalar product is two-fold. As figure 5.7 illustrates, the peak of the scalar product is even moved as the alignment changes.

Such a behaviour would make the scalar product unsuitable e.g. as a measure of similarity. To avoid the displacement of the peak, we could consider a normalised scalar product

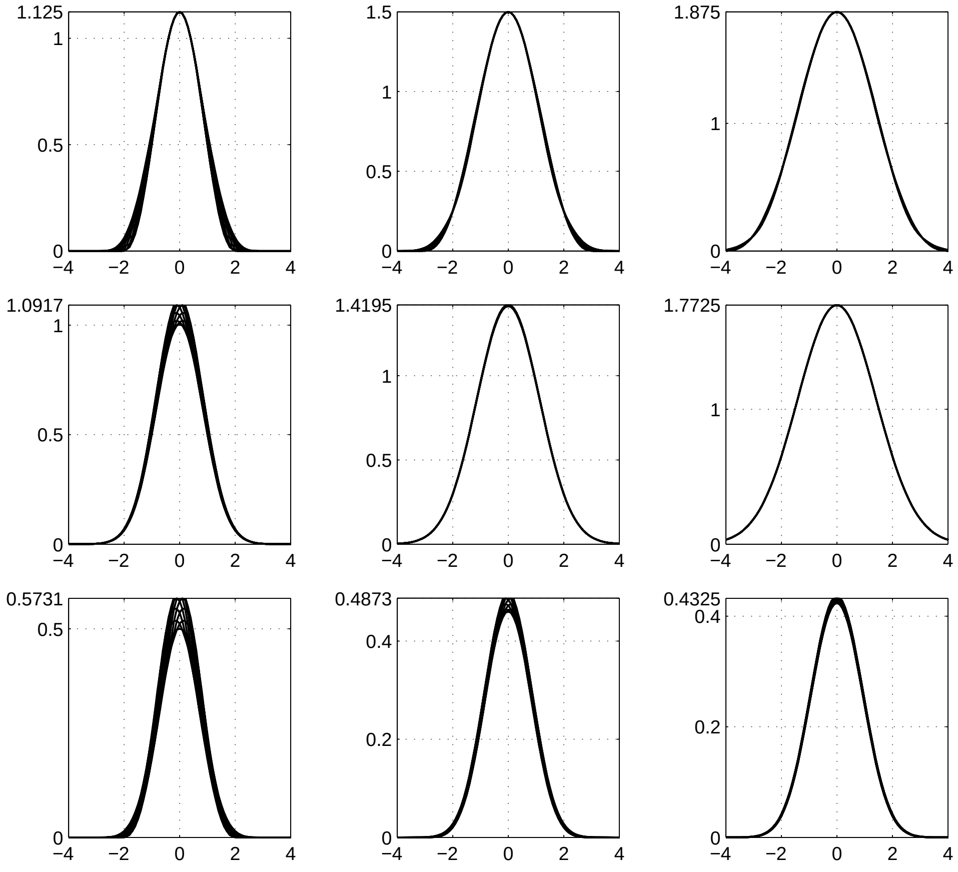


Figure 5.6: Superimposed scalar product functions for different alignments of the channel positions. Top to bottom: $\cos^2$, Gaussian, and B-spline kernels. Left to right: $\omega = \pi/3, \pi/4, \pi/5$, $\sigma = 0.6, 0.8, 1.0$, $n = 2, 3, 4$.

$$s(x) = \frac{\mathbf{u}(0)^T \mathbf{u}(x)}{||\mathbf{u}(0)||||\mathbf{u}(x)||} \,. \tag{5.29}$$

Figure 5.8 shows the same experiment as in figure 5.6, but using the normalised scalar product (5.29). As can be seen, the normalisation ensures the the peak is in the right place for all kernels, but it does not make all of the position dependency problems go away.

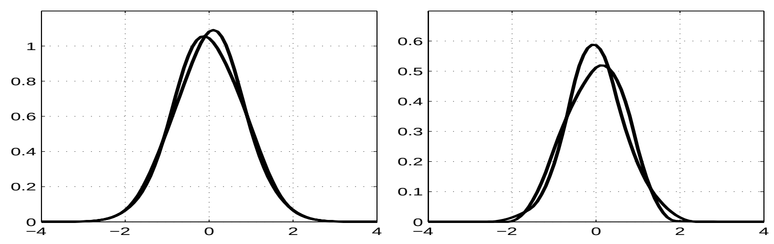


Figure 5.7: Different peak positions for different channel alignments. Left: Two Gaussian scalar product functions ($\sigma = 0.6$). Right: Two B-spline scalar product functions ($n = 2$).

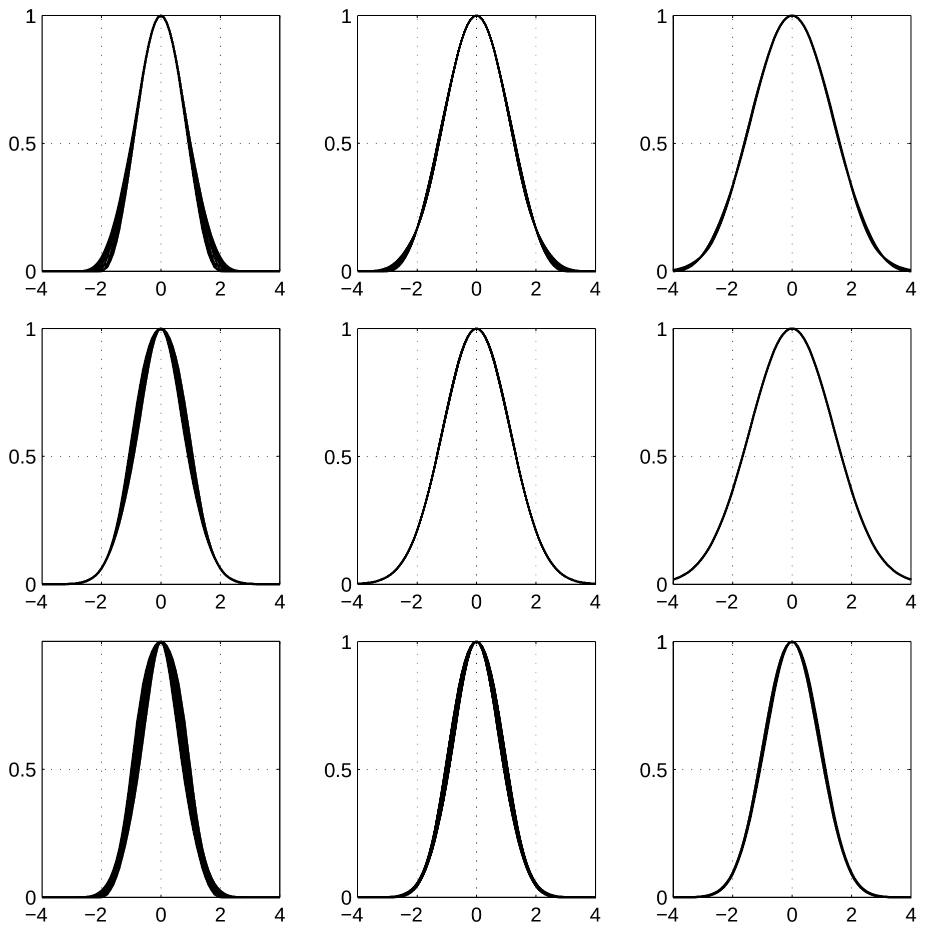


Figure 5.8: Superimposed normalised scalar product functions for different alignments of the channel positions. Top to bottom: $\cos^2$, Gaussian, and B-spline kernels. Left to right: $\omega = \pi/3, \pi/4, \pi/5$, $\sigma = 0.6, 0.8, 1.0$, $n = 2, 3, 4$.

5.4 Metameric distance

As mentioned in section 3.2.1, the channel representation is able to represent multiple values. In this section we will study interference between multiple values represented in a channel vector. A representation where the representation of two values is sometimes confused with their average is called a *metameric representation*, see section 3.2.1. To study this behaviour for the channel representation, we now channel encode two signals, sum their channel representations, and decode. The two signals we will use are

$$f_1(x) = x \quad \text{and} \quad f_2(x) = 6 - x \quad \text{for} \quad x \in [0, 6]\,. \tag{5.30}$$

These signals are shown in figure 5.9, left. As can be seen they are different near the edges of the plot, and more similar in value near the centre. If we channel encode these signals and decode their sum, we will obtain two distinct decodings near the edges of the plot, and near the centre of the plot we will obtain their average. Initially we will use a channel representation with integer channel positions, and a channel width of $\omega = \pi/3$. The centre plot of figure 5.9 shows the result of channel encoding, summing and decoding the two signals.

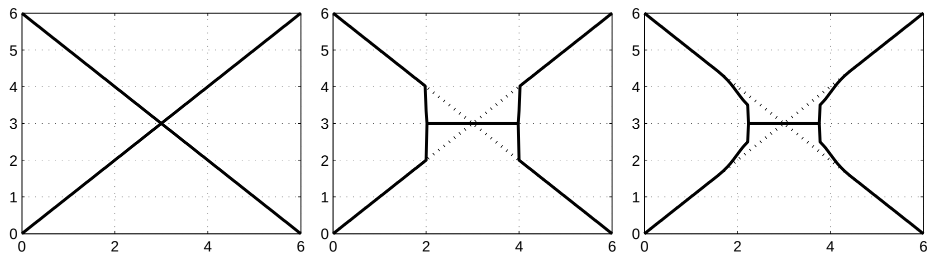


Figure 5.9: Illustration of metamerism. Left: Input signals. Centre: Decoded outputs. Using $\cos^2$ channels at integer positions, and $\omega = \pi/3$. Right: Result using channels at half integer positions instead. Dotted curves are the input signals.

This nice result is however merely a special case. If we move the channel positions such that they are not positioned at integer positions we will get different results. Figure 5.9, right shows the result when the channels are positioned at half-integers instead. As can be seen here, we now have an interval of interference, where the decoded values have moved closer to each other, before the interval where the two values are averaged. The two cases in figure 5.9 are the extreme cases in terms of allowable value distance. When the channel positions are aligned with the intersection point of the two signals, as in the centre plot, the smallest allowed distance, or *metameric distance* is the largest, $d_{\text{mm}} = 2.0$. When the signal intersection point falls right in-between two channel positions, the metameric distance is the smallest $d_{mm} = 1.5$.

The metameric distance also depends on the used channel width. The top row of figure 5.10 shows the metameric distances for three different channel widths of the $\cos^2$ kernel. Each plot shows 10 superimposed curves, generated using different alignments of the signal intersection point and the channel positions. From these plots we can see that the metameric distance increases with the channel width. Another effect of increasing the channel width is that the position dependency of the interference is reduced.

The experiment is repeated for the B-spline, and the Gaussian kernels in figure 5.10, middle and bottom rows. We have chosen kernel widths corresponding to the ones for the $\cos^2$ kernel according to (5.2), and (5.22). As can be seen, the $\cos^2$, and the B-spline kernel behave similarly, while the Gaussian kernel has a slightly slower increase in metameric distance with increasing kernel width. The reason for this is that we have used a constant decoding window size for the Gaussian kernel, whereas the window size increases with the channel width for both $\cos^2$ and B-spline kernels.

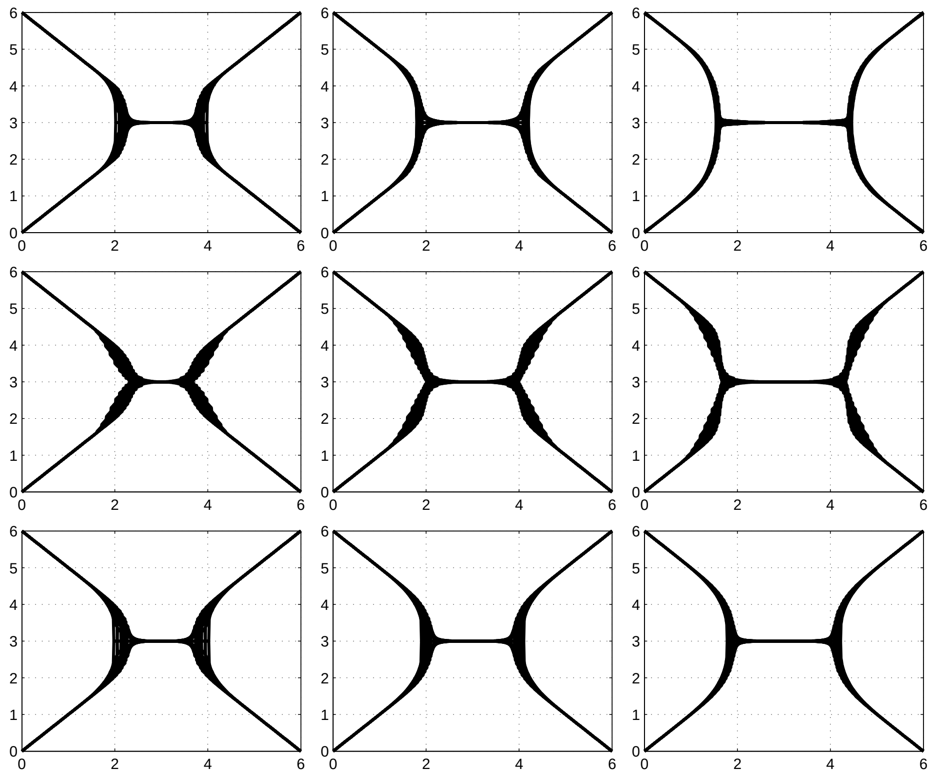


Figure 5.10: Metamerism for different channel widths. Top row: $\cos^2$ channels, width $\omega = \pi/3$, $\pi/4$, $\pi/5$. Middle row: B-spline channels, order 2, 3, 4. Bottom row: Gaussian channels, $\sigma = 0.6$, 0.8, 1.0. All plots show 10 superimposed curves with different alignments of the channel centres.

5.5 Stochastic kernels

In section 4.2.1 we identified averages of channel values as samples from a PDF convolved with the used kernel function. Now, we will instead assume that we have measurements x_n from some source $\mathcal{S}$, and that the measurements have been contaminated with additive noise from a source $\mathcal{D}$, i.e.

$$x_n = p_n + \eta_n \quad \text{with} \quad p_n \in \mathcal{S} \quad \text{and} \quad \eta_n \in \mathcal{D}\,. \tag{5.31}$$

For this situation a channel representation using a kernel $H(x)$ will have channels with expectation values

$$E\{u_k\} = (f * \eta * H)(x - k)\,. \tag{5.32}$$

Here $f(x)$, and $\eta(x)$ are the density functions of the source and the noise respectively. If the number of samples in an average is reasonably large, it makes sense to view $(\eta * H)(x - k)$ as a *stochastic kernel*.[6] In order to make the stochastic kernel as compact as possible, we will now consider the rectangular kernel

$$H^k(x) = \begin{cases} 1 & \text{when} \quad -0.5 < x - k \leq 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{5.33}$$

Figure 5.11 shows estimates of three stochastic kernels, together with the PDF of the added noise. The noise is of triangular[7], or TPDF type, and is what is typically used to de-correlate the errors in audio quantisation with dithering [60].

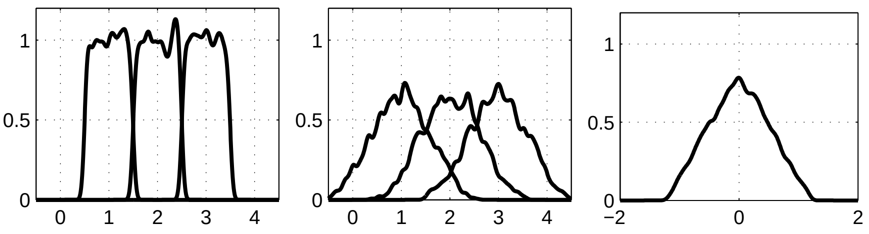


Figure 5.11: Stochastic bins. Left: Estimated PDFs of $H^k(x) = 1$ for $k \in [1, 2, 3]$. Centre: PDFs with addition of noise before the kernel function. Right: Estimated density of noise.

In general, the kernel (5.33) is not a good choice for density estimation. If $f(x)$ is discontinuous, or changes rapidly, it will cause aliasing-like effects on the estimated PDF. Such aliasing effects can be reduced by *dithering* see e.g. [54]. Dithering is the process of adding a small amount of noise (with certain characteristics) to a signal prior to a quantisation. Dithering is commonly used in image

[6]For earlier accounts of this idea, see [34, 36].

[7]A triangular noise sample is generated by averaging two samples from a rectangular distribution.

reproduction with a small number of available intensities or colours, as well as in perceptual quality improvement of digital audio [60].

5.5.1 Varied noise level

We will now have a look at how the precision of a local decoding is affected by addition of noise before applying the kernel functions. We will use three channels $k = 1, 2, 3$, with kernels $H^k(x)$ as defined in (5.33), and channel encode measurements x_n corresponding to source values $p \in [1.5, 2.5]$. To obtain the mode location with better accuracy than the channel distance, we use the local decoding derived for the Gaussian kernel (see section 5.1.1).

We will try averages of $N = 10$, 30, and 1 000 samples, and compute the absolute error $|\hat{x} - p|$, where $\hat{x}$ is the local decoding. To remove any bias introduced by the source value p (and also to make the curves less noisy) the errors are averaged over all tested source values, giving a mean absolute error (MAE). We will try source values $p \in [1.5, 2.5]$ in steps of 0.01.

The standard deviation of the noise η, see (5.31), is varied in the range $[0, 1]$ in steps of 0.01. Two noise distributions are tested, rectangular noise and triangular noise. The plots in figure 5.12 show the results.

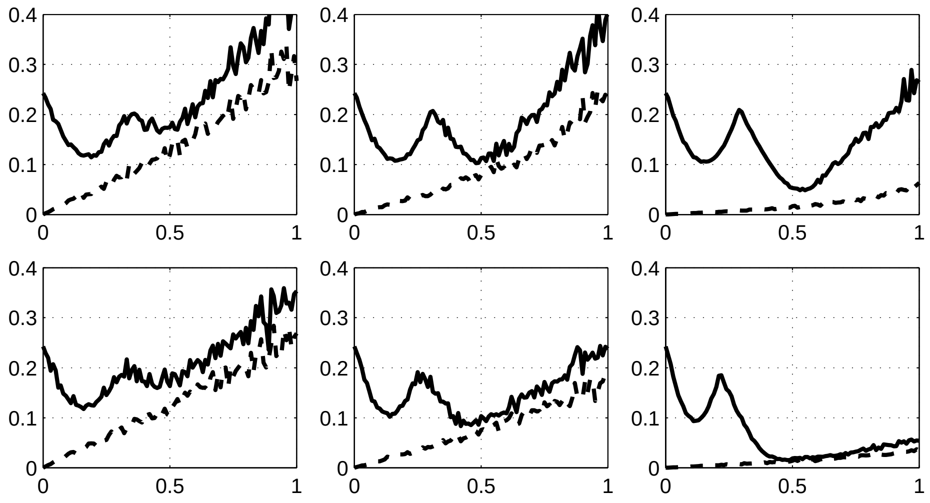


Figure 5.12: MAE of local decoding as function of noise level.
Top row: Results using rectangular noise. Bottom row: Results using triangular noise. Left to right: Number of samples $N = 10$, $N = 30$, $N = 1\,000$. *Solid curves show MAE for rectangular kernels, dashed curves show MAE for Gaussian kernels with* $\sigma = 0.8$. *Number of source values for which the curves are averaged is* 101.

As can be seen in the plot, the optimal noise level is actually above zero for the rectangular kernel. This is due to the expectation of the channel values being the convolution of the kernel and the noise PDF, see (5.32). The added noise thus results in a smoother PDF sampling. Finding the optimal noise given a kernel function will be called *the dithering problem.*

Which noise is optimal depends on the source density $f(x)$, and on the used decoding scheme. In this experiment it is thus reasonable to assume that the more similar the noise density convolved with the kernel is to a Gaussian, the better the accuracy. The dashed curves in each plot show the performance of overlapping bins with Gaussian kernels of width $\sigma = 0.8$. As can be seen in the plots, for large number of samples the performances of the two kernels are similar once the dithering noise is above a certain level.

Biological neurons are known to have binary responses (i.e. at a given time instant they either fire or don't fire). They are able to convey graded information by having the rate of firing depend on the sum of the incoming (afferent) signals. This behaviour could be modelled as (temporally local) averaging with noise added before application of a threshold (activation function). If the temporal averaging in the neurons is larger than just a few samples, it would be reasonable to expect that biological neurons implicitly have solved the *inverse dithering problem* of tuning the activation threshold to the noise characteristics, see e.g. [102].

5.6 2D and 3D channel representations

So far we have only discussed channel representations of scalar values. A common situation, which is already dealt with by mean-shift filtering and M-estimation, see chapter 4, is a higher dimensional sample space.

5.6.1 The Kronecker product

The most straight-forward way to extend channel encoding to higher dimensions is by means of the *Kronecker product.* The result of a Kronecker product between two vectors $\mathbf{x}$ and $\mathbf{y}$ is a new vector formed by stacking all possible element-wise products $x_i y_j$. This is related to the (column-wise) vectorisation of an outer-product

$$\mathbf{x} \otimes \mathbf{y} = \text{vec}(\mathbf{y}\mathbf{x}^T) \tag{5.34}$$

$$\begin{pmatrix} x_1 \\ \vdots \\ x_K \end{pmatrix} \otimes \begin{pmatrix} y_1 \\ \vdots \\ y_L \end{pmatrix} = \text{vec} \begin{pmatrix} y_1 x_1 & \dots & y_1 x_K \\ \vdots & \ddots & \vdots \\ y_L x_1 & \dots & y_L x_K \end{pmatrix} = \begin{pmatrix} y_1 x_1 \\ \vdots \\ y_1 x_K \\ \vdots \\ y_L x_1 \\ \vdots \\ y_L x_K \end{pmatrix} .$$

If we have a higher dimensionality than 2, we simply repeat the Kronecker product. E.g. for a 3D space we have

$$\mathbf{x} \otimes \mathbf{y} \otimes \mathbf{z} = \mathrm{vec}(\mathrm{vec}(\mathbf{z}\mathbf{y}^T)\mathbf{x}^T)\,. \tag{5.35}$$

For channel representations in higher dimensions it is additionally meaningful to do encodings of subspaces, such as a line in 2D, and a line or a plane in 3D. We will develop the encoding of a line in 2D in section 5.6.3.

For applications such as channel averaging (see section 4.2.1), the channel representation will become increasingly less practical as the dimensionality of the space to be represented is increased, and methods such as the mean-shift filter (see section 4.2.3) are to be preferred. However, for applications where the sparsity of the channel representation can be utilised in storage of data (e.g. the associative learning in chapter 9) higher dimensions are viable.

5.6.2 Encoding of points in 2D

Since the Gaussian function is separable, the Kronecker product of two 1D Gaussian channel vectors is equivalent to using isotropic 2D Gaussian kernels

$$\mathrm{B}^{k,l}(x,y) = \mathrm{B}^k(x)\mathrm{B}^l(y) = e^{-\dfrac{(x-k)^2+(y-l)^2}{2\sigma^2}}\,. \tag{5.36}$$

For efficiency, we will however still perform the encoding using the Kronecker product of two 1D channel vectors.

5.6.3 Encoding of lines in 2D

A line constraint in 2D is the set of all points (x, y) fulfilling the equation

$$x\cos\phi + y\sin\phi - \rho = 0\,. \tag{5.37}$$

Here $\begin{pmatrix}\cos\phi & \sin\phi\end{pmatrix}^T$ is the line normal, and ρ is the *signed normal distance*, i.e. the projection of an arbitrary point on the line onto the line normal. The distance of a specific point (k, l) to the line is then given by

$$d = ||k\cos\phi + l\sin\phi - \rho||\,. \tag{5.38}$$

This means that we can encode the line constraint, by simply applying the kernel

$$\mathrm{B}^{k,l}(\phi,\rho) = e^{-\dfrac{d^2}{2\sigma^2}} = e^{-\dfrac{(k\cos\phi + l\sin\phi-\rho)^2}{2\sigma^2}}\,. \tag{5.39}$$

5.6.4 Local decoding for 2D Gaussian kernels

In order to capture sample dispersions that are not aligned to the axes the local decoding should model the channel values using a full 2D Gaussian function

$$u^{k,l} = r\mathrm{B}^{k,l}(\mathbf{x}) = re^{-0.5\,(\mathbf{x}-\mathbf{m})^T\,\mathbf{C}^{-1}\,(\mathbf{x}-\mathbf{m})}\,. \tag{5.40}$$

Here $\mathbf{x} = \begin{pmatrix} x & y \end{pmatrix}^T$, $\mathbf{m} = \begin{pmatrix} k & l \end{pmatrix}^T$, and $\mathbf{C}$ is a full covariance matrix. In scalar form we get

$$u^{k,l} = re^{-\dfrac{(x-k)^2\sigma_y^2 + (y-l)^2\sigma_x^2 - 2(x-k)(y-l)\sigma_{xy}}{2(\sigma_x^2\sigma_y^2 - \sigma_{xy}^2)}} \tag{5.41}$$

or, for $\Delta x = x - k$ and $\Delta y = y - l$,

$$u^{k,l} = re^{-\dfrac{\Delta x^2\sigma_y^2 + \Delta y^2\sigma_x^2 - 2\Delta x\Delta y\sigma_{xy}}{2(\sigma_x^2\sigma_y^2 - \sigma_{xy}^2)}} \,. \tag{5.42}$$

If we choose to estimate the parameters in a 3×3 neighbourhood around the position (k, l), we obtain the following system

$$\begin{pmatrix} u^{k-1,l-1} \\ u^{k-1,l} \\ u^{k-1,l+1} \\ u^{k,l-1} \\ u^{k,l} \\ u^{k,l+1} \\ u^{k+1,l-1} \\ u^{k+1,l} \\ u^{k+1,l+1} \end{pmatrix} = \begin{pmatrix} r\mathrm{B}^{k-1,l-1}(x,y) \\ r\mathrm{B}^{k-1,l}(x,y) \\ r\mathrm{B}^{k-1,l+1}(x,y) \\ r\mathrm{B}^{k,l-1}(x,y) \\ r\mathrm{B}^{k,l}(x,y) \\ r\mathrm{B}^{k,l+1}(x,y) \\ r\mathrm{B}^{k+1,l-1}(x,y) \\ r\mathrm{B}^{k+1,l}(x,y) \\ r\mathrm{B}^{k+1,l+1}(x,y) \end{pmatrix} . \tag{5.43}$$

The logarithm of an arbitrary row can be written as

$$\ln u^{k+c,l+d} = \ln r - \frac{(\Delta x - c)^2\sigma_y^2 + (\Delta y - d)^2\sigma_x^2 - 2(\Delta x - c)(\Delta y - d)\sigma_{xy}}{2(\sigma_x^2\sigma_y^2 - \sigma_{xy}^2)} \,. \tag{5.44}$$

This can be factorised as

$$\ln u^{k+c,l+d} = 0.5 \begin{pmatrix} 1 & 2c & 2d & -c^2 & -d^2 & -2cd \end{pmatrix} \mathbf{p} \tag{5.45}$$

for the parameter vector

$$\mathbf{p} = \frac{1}{\sigma_x^2\sigma_y^2 - \sigma_{xy}^2} \begin{pmatrix} 2\ln r(\sigma_x^2\sigma_y^2 - \sigma_{xy}^2) - \Delta x^2\sigma_y^2 - \Delta y^2\sigma_x^2 + 2\Delta x\Delta y\sigma_{xy} \\ \Delta x\sigma_y^2 - \Delta y\sigma_{xy} \\ \Delta y\sigma_x^2 - \Delta x\sigma_{xy} \\ \sigma_y^2 \\ \sigma_x^2 \\ -\sigma_{xy} \end{pmatrix} . \tag{5.46}$$

In the parameters $\mathbf{p}$, we recognise the inverse covariance matrix

$$\mathbf{C}^{-1} = \frac{1}{\sigma_x^2\sigma_y^2 - \sigma_{xy}^2} \begin{pmatrix} \sigma_y^2 & -\sigma_{xy} \\ -\sigma_{xy} & \sigma_x^2 \end{pmatrix} = \begin{pmatrix} p_4 & p_6 \\ p_6 & p_5 \end{pmatrix} . \tag{5.47}$$

Thus we can compute the covariance matrix as

$$\hat{\mathbf{C}} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} = \frac{1}{p_4 p_5 - p_6^2} \begin{pmatrix} p_5 & -p_6 \\ -p_6 & p_4 \end{pmatrix} . \tag{5.48}$$

From the expressions for p_2, and p_3 in (5.46), we obtain the following system

$$\begin{pmatrix} p_2 \\ p_3 \end{pmatrix} = \mathbf{C}^{-1} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \quad \text{with the solution} \quad \begin{pmatrix} \Delta\hat{x} \\ \Delta\hat{y} \end{pmatrix} = \hat{\mathbf{C}} \begin{pmatrix} p_2 \\ p_3 \end{pmatrix} . \tag{5.49}$$

From the solution, we can obtain an estimate of the confidence $\hat{r}$, as

$$\hat{r} = e^{0.5(p_1 + p_4 \Delta\hat{x}^2 + p_5 \Delta\hat{y}^2 - 2p_6 \Delta\hat{x}\Delta\hat{y})} . \tag{5.50}$$

The final peak location is obtained by adding the centre bin location to the peak offset

$$\hat{x} = \Delta\hat{x} + k \quad \text{and} \quad \hat{y} = \Delta\hat{y} + l\,. \tag{5.51}$$

The expectation of the estimated covariance matrix $\hat{\mathbf{C}}$ is the sum of the covariance of the noise, and the covariance of the kernel functions $\mathbf{C_b} = \text{diag}(\sigma, \sigma)$, see (5.36) and (5.39). This means that we can obtain the covariance of the noise as $\hat{\mathbf{C}}_{\text{noise}} = \hat{\mathbf{C}} - \mathbf{C_b}$.

5.6.5 Examples

Clustering of points in 2D is illustrated in figure 5.13. Here we have channel encoded 1000 points, each from one of 5 clusters at random locations. The centre plot shows the average of the channel representation of the points. In the right plot, the decoded modes have their peaks represented as dots, and their covariances as ellipses. This approach to clustering is different from techniques like *K-means clustering* [7] and *mixture model estimation* using *expectation maximisation* (EM) [15, 7], which both require the number of clusters to be known a priori. In order to get a variable number of clusters, such methods have to test all reasonable numbers of clusters, and select one based on some criterion see e.g. [15]. Here we instead assume a scale, specified by the channel distance and kernel width, and directly obtain a variable number of clusters. This makes our clustering technique fall into the same category as mean-shift clustering, see section 4.2.3.

The encoding of line constraints is illustrated in figure 5.14. Here we have channel encoded 4 lines, and averaged their channel representations. In the right plot, the decoded modes have their peaks represented as dots, and their covariances as ellipses. This approach to finding multiple solutions to a line constraint equation was applied in [93] to find multiple solutions to systems of optical flow constraints at motion boundaries.

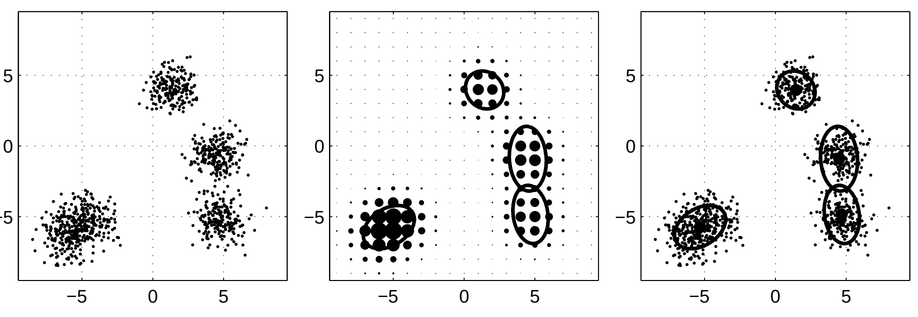


Figure 5.13: Illustration of point decoding. Left: Points to encode. Centre: Average of the corresponding channel representations, the sizes of the filled circles correspond to the channel values. Ellipses show the decoded modes. Right: The points, and the decoded modes.

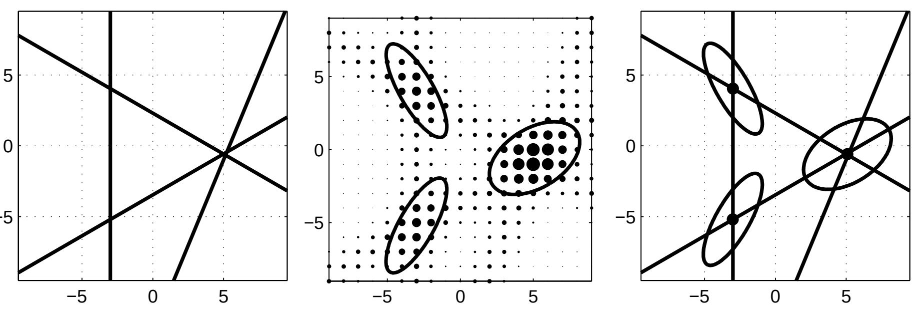


Figure 5.14: Illustration of line constraint decoding. Left: Lines to encode. Centre: Average of the corresponding channel representations, the sizes of the filled circles correspond to the channel values. Ellipses show the decoded modes. Right: The original lines, and the decoded modes.

5.6.6 Relation to Hough transforms

The idea of encoding line constraints, averaging and decoding, is the same as in the *Hough transform*, see e.g. [92]. In the Hough transform, each line constraint contributes either 1 or 0 to cells in an *accumulator array* (corresponding to the channel matrix).

To reduce noise, and avoid detection of multiple maxima for each solution, a smoothing is sometimes applied to the accumulator array *after* it has been computed [92]. Note that channel encoding is a more sound approach, since it corresponds to smoothing *before* sampling. The use of overlapping kernels, and a local decoding, instead of just finding the accumulator cell with the largest contribution, improves the accuracy of the result. Furthermore it avoids the trade-off between noise sensitivity and accuracy of the result inherent in the Hough transform, and all other peak detection schemes using non-overlapping bins.[8] The difference in obtained accuracy is often large, see e.g. [33] for an example where the estimation error is reduced by more than a factor 50.

[8]A larger amount of noise will however still mean that larger kernels should be used, and this will affect the amount of interference between multiple decodings, see section 5.4.

Chapter 6

Channel Smoothing

In this chapter we will introduce the *channel smoothing* technique for image denoising. We will identify a number of problems with the original approach [42], and suggest solutions to them. One of the solutions is the *alpha synthesis* technique. Channel smoothing is also compared to other popular image denoising techniques, such as mean-shift, bilateral filtering, median filtering, and normalized averaging.

6.1 Introduction

In chapter 4 we saw that averaging in the channel representation followed by a local decoding is a way to find simple patterns (clusters) in a data set. In low-level image processing, the data set typically consists of pixels or features in a regular grid. Neighbouring pixels are likely to originate from the same scene structure, and it seems like a good idea to exploit this known relationship and perform spatially localised clustering. This is done in *channel smoothing* [42, 29].

6.1.1 Algorithm overview

Channel smoothing of a grey-scale image $p : \mathbb{Z}^2 \to \mathbb{R}$ can be divided into three distinct steps.

1. **Decomposition**.
   The first step is to channel encode each pixel value, $p(x, y)$, in the grey-scale image with the confidence $r(x, y) = 1$, to obtain a set of *channel images.*

2. **Smoothing**.
   We then perform a spatially local averaging (low-pass filtering) on each of the channel images.

3. **Synthesis**.
   Finally we synthesise an image and a corresponding confidence using a local decoding (see section 3.2.3) in each pixel.

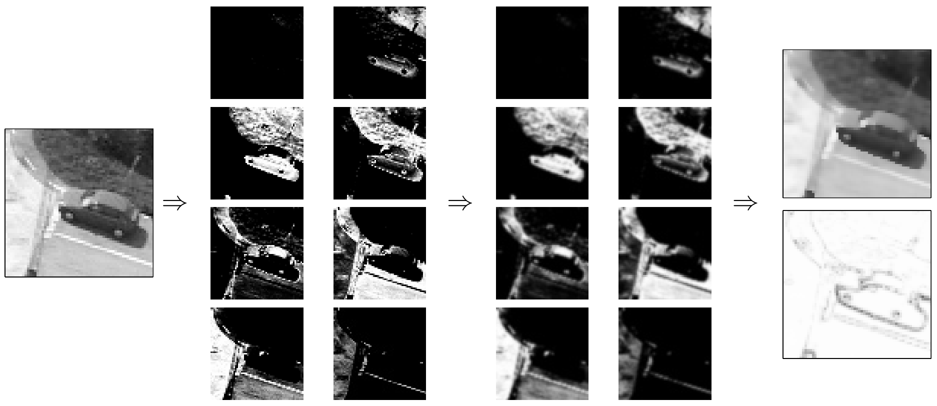


Figure 6.1: Illustration of channel smoothing. Left to right: Input, channel representation, low-passed channels, and local decoding.

At locations where the image is discontinuous we will obtain several valid decodings. The most simple solution is to *select* the decoding with the highest confidence. This synthesis works reasonably well, but it has several problems, such as introduction of jagged edges, and rounded corners. In section 6.3 we will analyse these problems, and suggest solutions. Finally in section 6.4 we will have a look at a more elaborate synthesis method that deals with these problems.

6.1.2 An example

Figure 6.1 shows an example of channel smoothing. In this example we have used $K = 8$ channel images, and averaged each channel image with a separable Gaussian filter of $\sigma = 1.18$ and 7 coefficients per dimension. A channel representation $\mathbf{u} = \begin{pmatrix} H^1(p) & \dots & H^K(p) \end{pmatrix}^T$ can represent measurements $p \in [3/2, K-1/2]$, and thus we have scaled the image intensities $p(x,y) \in [r_l, r_h]$ using a linear mapping $p_x(x,y) = t_1 p(x,y) + t_0$ with

$$t_1 = \frac{K-2}{r_h - r_l} \quad \text{and} \quad t_0 = 3/2 - t_1 r_l \tag{6.1}$$

as described in section 3.3.

6.2 Edge-preserving filtering

The channel smoothing procedure performs averaging of measurements that are similar in both property (here intensity) and location. In this respect, channel smoothing is similar to edge preserving filtering techniques, such as *robust anisotropic diffusion*[8], *selective binomial filtering*[40], *mean-shift filtering*[19], and *non-linear Gaussian filtering*[103, 2, 44] also known as *SUSAN noise filtering*[88], and in case of vector valued signals as *bilateral filtering*[98]. All these methods can be related to *redescending*[1] M-estimators (see section 4.2.4). The relationship between anisotropic diffusion and M-estimation is established in [8], and selective binomial filtering can be viewed as an M-estimation with a cutoff-squares error norm (see section 4.2.4).

In section 6.5 we will compare mean-shift filtering and bilateral filtering to channel smoothing, thus we will now describe these two methods in more detail.

6.2.1 Mean-shift filtering

Mean-shift filtering [19, 16, 43] is a way to cluster a sample set, by moving each sample toward the closest mode. As described in section 4.2.3 this is accomplished by gradient descent on a kernel density estimate. For each sample $\mathbf{p}_n$, an iteration is started in the original sample value i.e. $\bar{\mathbf{p}}_n^0 = \mathbf{p}_n$, and is iterated until convergence. The *generalised mean-shift* iteration [16] is defined by

$$\bar{\mathbf{p}}_n^{i+1} = \frac{\sum_k H(\mathbf{p}_k - \bar{\mathbf{p}}_n^i)\mathbf{p}_k}{\sum_k H(\mathbf{p}_k - \bar{\mathbf{p}}_n^i)} \,. \tag{6.2}$$

Here H is a kernel which is said to be the *shadow* of the kernel in the corresponding kernel density estimator, see section 4.1.1. For a kernel density estimate using the Epanechnikov kernel, the iteration rule becomes an average in a local window, and can thus be computed very quickly [43]. This is what has given the method its name, and averaging in a local window is also the most commonly applied variant of mean-shift filtering.

Mean-shift filtering has been developed by Comaniciu and Meer [19, 20] into algorithms for edge-preserving filtering, and segmentation. In the edge-preserving filter, they apply mean-shift filtering to the parameter vector

$$\mathbf{p}(\mathbf{x}) = \begin{pmatrix}\mathbf{x}/\sigma_s & r(\mathbf{x})/\sigma_r & g(\mathbf{x})/\sigma_r & b(\mathbf{x})/\sigma_r\end{pmatrix}^T \,. \tag{6.3}$$

Here r, g, and b are the three colour bands of an RGB image, and the parameters σ_s, and σ_r allow independent scaling of the spatial and range (colour) vector elements respectively. The convergence point $\bar{\mathbf{p}}^*$ is stored in the position where the iteration was started. The result is thus a hill-climbing on the kernel density estimate.

6.2.2 Bilateral filtering

Bilateral filtering [98] of a signal $\mathbf{p}(\mathbf{x})$ is defined by

$$\mathbf{q}(\mathbf{x}) = \frac{\int \mathbf{p}(\mathbf{y})H((\mathbf{x}-\mathbf{y})/\sigma_s)H((\mathbf{p}(\mathbf{x})-\mathbf{p}(\mathbf{y}))/\sigma_r)d\mathbf{y}}{\int H((\mathbf{x}-\mathbf{y})/\sigma_s)H((\mathbf{p}(\mathbf{x})-\mathbf{p}(\mathbf{y}))/\sigma_r)d\mathbf{y}} \tag{6.4}$$

where H is a multi-dimensional Gaussian, see (4.5). For the special case of a scalar valued image $p(\mathbf{x})$, the expression (6.4) is identical to the earlier *non-linear Gaussian filter* [103, 2, 44], and to the *SUSAN noise filtering* technique [88]. In

[1]A redescending M-estimator has an influence function with monotonically decreasing magnitude beyond a certain distance to the origin.[8]

(6.4) we explicitly distinguish between the spatial position $\mathbf{x}$, and the sample values $\mathbf{f}(\mathbf{x})$, whereas in the generalised mean-shift iteration (6.2) on the parameter vector (6.3) they are treated as one entity.

It is nevertheless possible to relate generalised mean-shift and bilateral filtering, for the case of a separable kernel $H(d)$, such as the Gaussian. If we use $\mathbf{p}$ from (6.3) in (6.2), and identify r, g, b as $\mathbf{f}$ in (6.4), then the result of (6.2) and (6.4) after one iteration are identical in the r, g, b and $\mathbf{f}$ part. The difference is that (6.4) does not update the position as (6.2) does. A similar observation is made in [104], where non-linear Gaussian filtering (i.e. bilateral filtering) is shown to correspond to the first iteration of a gradient descent on an M-estimation problem. Bilateral filtering thus moves the data towards the local M-estimate, but in general gets stuck before it is reached [104].

Tomasi and Manduchi [98] also suggest a scheme for iterated bilateral filtering, by again applying (6.4) to the result $\mathbf{q}(\mathbf{x})$. As noted in [19] this iteration will not converge to a stable clustering. Instead it will eventually erode the image to contain a single constant colour. Thus, neither bilateral filtering nor iterated bilateral filtering are robust techniques in a strict sense.

6.3 Problems with strongest decoding synthesis

The original synthesis in channel smoothing selects the decoding with the strongest confidence as its output.[42, 29] This synthesis has three distinct problems

1 **Jagged edges**.
Since the synthesis *selects* one decoding, edges can become arbitrarily sharp. This means that for an infinitesimal translation of the underlying signal, the synthesis of a pixel may completely change value, and this in turn results in jagged edges.

2 **Rounded corners**.
Corners will tend to be rounded, because the number of pixels voting for the intensity inside the corner (i.e. the confidence) becomes lower than the number of votes for the outside intensity when we are near the tip of the corner.

3 **Patchiness**.
For steep slopes, or large amounts of blurring, when we have inlier noise, the selection tends to generate a patchy output signal.

To illustrate these effects we have devised a test image consisting of a slanted plane, surrounded by a number of triangles with protruding acute angles (see figure 6.2, left for a noise corrupted version). The difference in grey-level from the background is also different for all the triangles, in order to illustrate at what difference the non-linear behaviour starts. The parameters of the channel smoothing have been specifically set to exaggerate the three problems listed above. The result is shown in the right plot of figure 6.2.

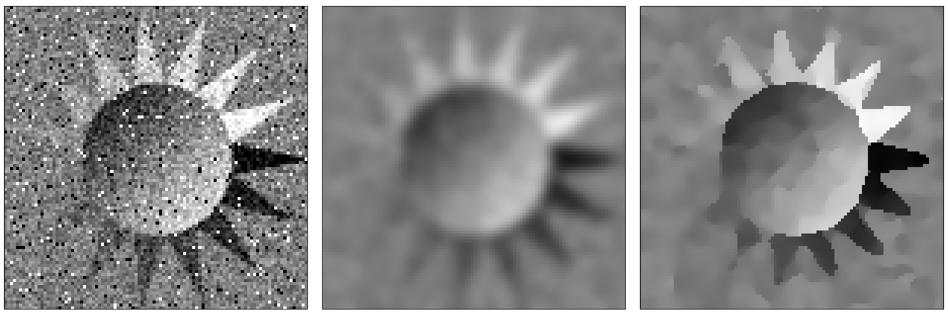


Figure 6.2: Illustration of problems with strongest decoding synthesis. Left to right: Input (100 × 100 pixels, with values in range $[-1, 1]$. Noise is Gaussian with $\sigma = 0.1$, and 5% salt&pepper pixels.), normalized average ($\sigma = 2.2$), channel smoothing ($K = 15$ channels, $\sigma = 2.2$).

In order to demonstrate the non-linear behaviour of the method, a normalized average with the same amount of smoothing is also shown for comparison. Normalized averaging of a signal–confidence pair (p, r) using the kernel g is defined by the quotient

$$q(\mathbf{x}) = \frac{(p \cdot r * g)(\mathbf{x})}{(r * g)(\mathbf{x})} \tag{6.5}$$

where $\cdot$ denotes an element-wise product.

In our example, normalized averaging is equivalent to plain averaging, except near the edges of the image, where it helps maintaining the correct DC-level. See [48, 26] for more on normalized averaging, and the more general framework of *normalized convolution*[26].

6.3.1 Jagged edges

The jagged edges problem can be dealt with using super-sampling techniques common in computer graphics, see e.g. [59] section 4.8. For channel smoothing this can be done by filling in zeros in between the encoded channel values, before smoothing, and thus generate an output at a higher resolution. By modifying the amount of channel smoothing accordingly, we can obtain edges with a higher resolution. This technique is demonstrated in figure 6.3.

Here we have filled in zeroes to obtain 4× the pixel density along each spatial dimension. We have then modified the amount of smoothing according to

$$\sigma_{\text{new}} = \sigma\sqrt{(4^n - 1)/3} \tag{6.6}$$

where $n = 3$ is the octave scale, as suggested in [64]. As a final step we have then blurred and subsampled the high resolution output. This has given us an image without jagged edges (see figure 6.3, right).

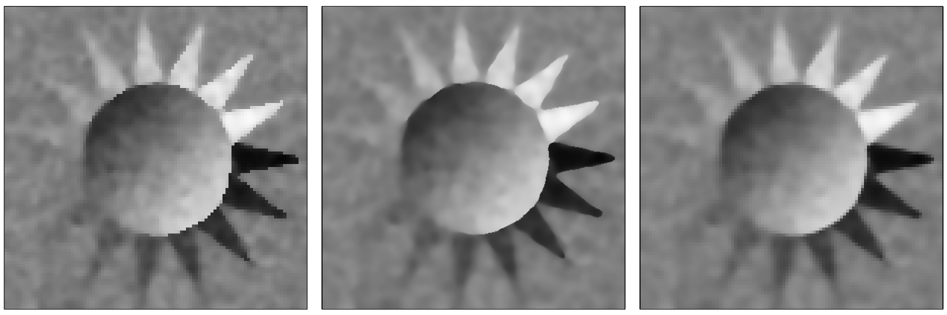


Figure 6.3: Super-sampling from channels. Left to right: Strongest decoding output (using $K = 7$, and $\sigma = 1.3$), Strongest decoding output after 4× up-sampling of the channel images (using $K = 7$, and $\sigma = 1.3 \times \sqrt{(4^3 - 1)/3} = 5.96$), Smoothing and subsampling of the high-resolution decoding ($\sigma = 1.4$). For input image, see figure 6.2.

6.3.2 Rounding of corners

A solution to the rounding of corners was suggested by Spies and Johansson in [94]. They proposed to sometimes select the decoding closest to the original grey-level, instead of the one with the highest confidence. The method in [94] works as follows:

- If the strongest confidence is above a threshold t_h the corresponding decoding is selected.
- If not, all decodings with a confidence above t_l are searched for the decoding closest to the original grey value.

Spies and Johansson suggest using $t_h = 0.9$ and $t_l = 0.1$. A similar behaviour can be obtained by removing all decodings with confidence below a threshold $c_{\min}$, and select the remaining decoding which is closest to the original value. Selecting the closest decoding will make the method correspond roughly to the hill-climbing done by the mean-shift procedure (see section 6.2.1). These two methods (from now on called Spies and Hill-climbing) are compared to the strongest decoding in figure 6.4. As can be seen in the figure, these methods trade preservation of details against removal of outliers which happen to be inliers in nearby structures.

6.3.3 Patchiness

The patchiness problem is caused by a too wide distribution of samples around a mode. More specifically, the exact mode location cannot be determined by examining only channel values inside the decoding window, since the noise has not been completely averaged out, see figure 6.5. Usually more channel smoothing is unable to average out the noise, since there simply are no more samples inside the appropriate grey-level range locally in the image.

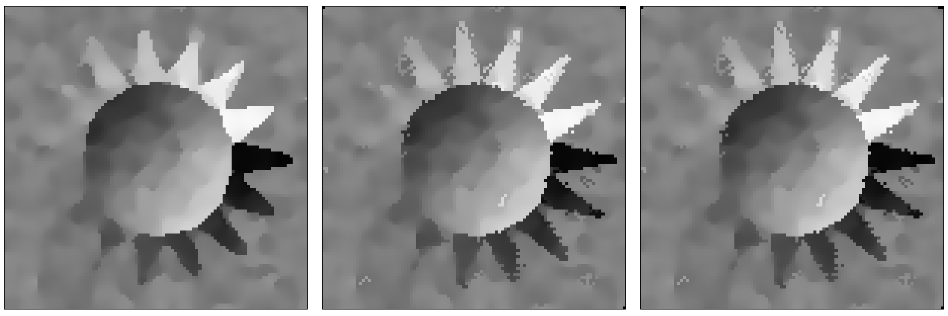


Figure 6.4: Comparison of decoding selection schemes. Left to right: Strongest decoding synthesis, the Spies method ($t_h = 0.7$ $t_l = 0.25$), the Hill-climbing method ($c_{\min} = 0.25$). All three use $K = 15$ channels, and $\sigma = 2.2$.

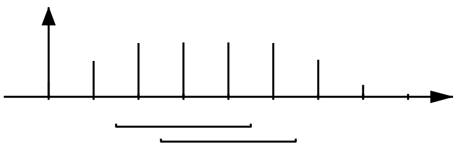


Figure 6.5: Reason for the patchiness problem. For wide distributions, it might be impossible to pick a proper decoding window. Both alternatives indicated here will give wrong results, since the noise has not been completely averaged out.

There are two causes to the wide distributions of samples. The first one is that the noise distribution might be too wide. This can be dealt with by increasing the sizes of the used kernel functions such that more samples can be used in the averaging process. This would however require a modification to the decoding step. A way to obtain larger kernels which does not require a modification of the decoding step is to simply reduce the number of channels and scale them to fit the same interval according to (6.1). Reducing the number of channels will however make the method increasingly more similar to plain linear smoothing.

There is however a second cause to the wide distributions. The channel averaging process implicitly assumes that the signal is piecewise constant. When the signal locally constitutes a ramp, we violate this assumption, and the actual width of the distribution depends on the slope of the ramp. Since the required width of the channels depends both on the amount of noise and on the slope of the signal, a more theoretically sound solution is to use a more advanced decoding scheme, which adapts the size of the decoding window to take more channel values into account when necessary. Alternatively we could replace the locally constant assumption with a locally linear one, and cluster in (μ, dx, dy)-space instead. None of these ideas are investigated further in this thesis.

6.4 Alpha synthesis

We will now instead attack the jagged edges problem, and at the same time make a slight improvement on the performance with respect to the rounding of corners. The reason for the jagged edges in the strongest decoding synthesis is that the decoding is a *selection* of one of the decodings. In many situations a selection is desirable, for instance if we want to teach a robot to navigate around an object. Going either `left` or `right` of the object works fine, but the average: `straight-ahead` is not a valid trajectory. In the present situation however, we want to output an image, and in images we should not have arbitrarily sharp edges.

The solution to this problem is to generate a continuous transition between the decoded values. Instead of choosing the decoding with the highest confidence, we will now combine all decoded signal–confidence pairs (p_k, r_k) in a non-linear way. In this way it is possible to obtain an output signal where the degree of smoothness is controlled by a parameter. The combination of decoded signal–confidence pairs (p_k, r_k) is done according to

$$p^{\text{out}} = \sum_k p_k w_k \quad \text{where} \quad w_k = \frac{r_k^\alpha}{\sum_l r_l^\alpha} \tag{6.7}$$

and α is a tuning parameter. For $\alpha = 1$ we get a linear combination of the decodings p_k, and for $\alpha \to \infty$ we obtain a selection of the strongest decoding again. The behaviour of (6.7) is illustrated in figure 6.6.

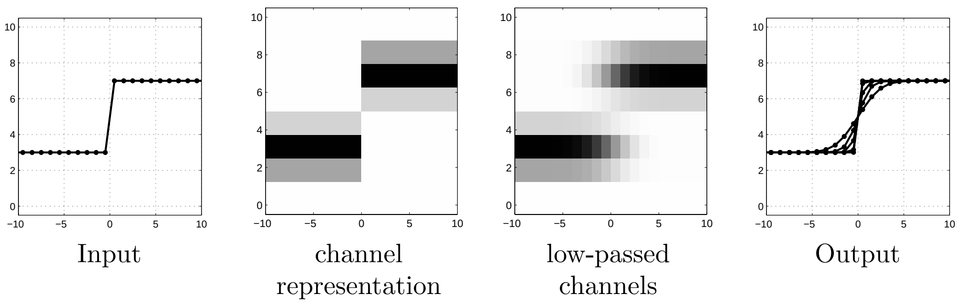


Figure 6.6: Illustration of alpha synthesis in 1D. Outputs for 5 different α values are shown.

What is important to note here is that the actual distance between the decodings p_1 and p_2 plays no part in the synthesis (when the decodings do not interfere). This is not the case for methods like *Histogram filters* [107]. In Histogram filters the noise is removed by regularised inverse filtering of the histogram (i.e. the channel vector). The output is then computed as the *mass centre* of histogram, after raising the histogram bin values to a power γ. The purpose of γ is to control the sharpness of the result, and it thus serves the same purpose as our α. In such an approach, the interference of the two levels will be different when they are close and when they are far apart. Furthermore, raising the histogram to a power is not a shift invariant operation.

The signal in figure 6.6 (left) is expanded into a number of channels, which are low-passed. In the decoding we get two valid signal–confidence pairs (p_1, r_1) and (p_2, r_2). The blurred channels have values that smoothly change from their highest value to zero. Since the confidence computation is a local average of the channel values, the confidence will have the same profile as the channels. The confidence of a valid decoding is exactly 1 when the low-pass kernel is inside one of the flat regions, and as we move across the edge it drops to zero.

6.4.1 Separating output sharpness and channel blurring

The actual value of the confidence is given by a convolution of a step and the used low-pass kernel. In this example we have used a Gaussian low-pass kernel, and thus obtain the following confidences for the valid decodings

$$r_1(x) = ((1 - \text{step}) * g)(x) = \int_{-\infty}^{-x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-0.5(x/\sigma)^2} = \Phi\left(\frac{-x}{\sigma}\right) = 1 - \Phi\left(\frac{x}{\sigma}\right) \tag{6.8}$$

$$r_2(x) = (\text{step} * g)(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-0.5(x/\sigma)^2} = \Phi\left(\frac{x}{\sigma}\right) . \tag{6.9}$$

Where $\Phi(x)$ is the integral of the standard normal PDF. The weights now become

$$w_1 = \frac{r_1^\alpha}{r_1^\alpha + r_2^\alpha} = \frac{1}{1 + \left(\frac{1}{1 - \Phi\left(\frac{x}{\sigma}\right)} - 1\right)^\alpha} \quad \text{and} \quad w_2 = \frac{1}{1 + \left(\frac{1}{\Phi\left(\frac{x}{\sigma}\right)} - 1\right)^\alpha} \tag{6.10}$$

If we look at the decoding for $x = 0$, we get

$$p^{\text{out}}(0) = w_1(0)p_1 + w_2(0)p_2 = \frac{1}{2}p_1 + \frac{1}{2}p_2 = \frac{p_1 + p_2}{2} \tag{6.11}$$

which is desirable, since this point is directly inbetween the two levels p_1 and p_2. If we look at the derivative at $x = 0$, we get

$$\begin{aligned} \frac{\partial p^{\text{out}}}{\partial x}(0) &= \frac{\partial w_1}{\partial x}(0)p_1 + \frac{\partial w_2}{\partial x}(0)p_2 = \dots \\ &= \frac{\alpha}{\sqrt{2\pi}\sigma}(p_2 - p_1) . \end{aligned} \tag{6.12}$$

This motivates setting $\alpha \propto \sigma$. By switching to a parameter $\beta = \alpha/\sigma$ we get a new synthesis formula

$$p^{\text{out}} = \sum_k p_k w_k \quad \text{where} \quad w_k = \frac{r_k^{\beta\sigma}}{\sum_l r_l^{\beta\sigma}} . \tag{6.13}$$

Using the parameter β, we can control the slope of the decoding at the transition point independently of σ. While (6.12) only holds exactly for $x = 0$, a constant α/σ ratio gives very stable slopes for other values of x as well. This is demonstrated experimentally in figure 6.7.

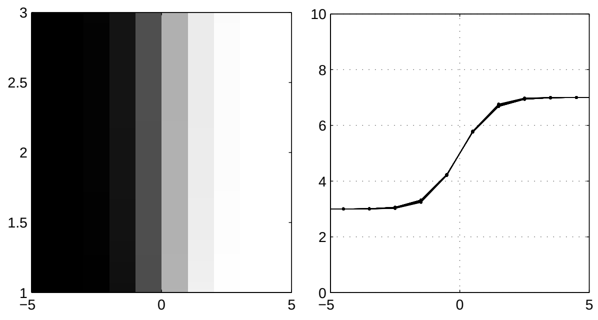


Figure 6.7: Illustration of stability of the $\alpha \propto \sigma$ approximation. Here we have used $\sigma \in \{1, 1.05, \ldots, 3\}$ and set $\beta = 1$. Left: each row is p^{out} for some value of σ. Right: all p^{out} curves superimposed.

Figure 6.8 demonstrates the synthesis according to (6.7) on the test-image in figure 6.2 (left), for varying amounts of blurring (σ) and varied values of β. Each row has a constant β value, and as can be seen they have roughly the same sharpness.

6.4.2 Comparison of super-sampling and alpha synthesis

We will now make a comparison of alpha synthesis and the super-sampling method described in section 6.3.1. Figure 6.9 shows the result. From the figure we can see that the result is qualitatively similar with respect to elimination of jagged edges. After examining details (see bottom row of figure 6.9) we find that alpha synthesis is slightly better at preserving the shape of corners.

In addition to slightly better preservation of corners, alpha synthesis also has the advantage that it is roughly a factor 16 faster, since it does not need to use a higher resolution internally.

6.4.3 Relation to smoothing before sampling

An image from a digital camera is a regular sampling of a projected 3D scene. Before sampling, the projection has been blurred by the *point spread function* of the camera lens, and during sampling it is further smoothed by the spatial size of the detector elements. These two blurrings can be modelled by convolution of the continuous signal $f(x)$ with a blurring kernel $g(x)$. A blurring of a piecewise constant signal $f(x)$, such as the one in figure 6.6, will have transition profiles which look the same, regardless of the signal amplitude. This is evident since for a scalar A, a signal $f(x)$, and a filter $g(x)$ we have

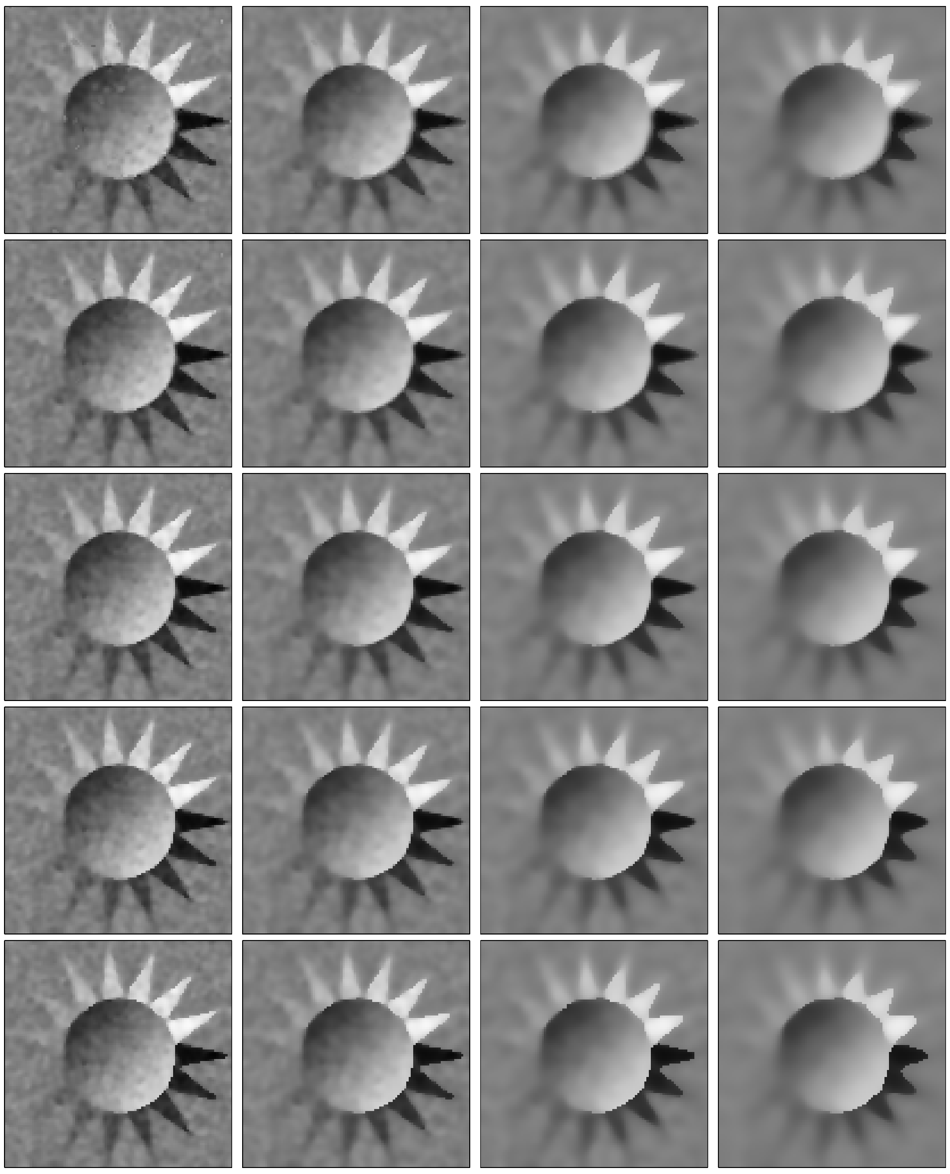


Figure 6.8: σ independent sharpness. Column-wise σ = 1.2, 1.6, 2.4, 3.2. Row-wise β = 1, 2, 4, 8, ∞ (i.e. strongest decoding synthesis). The input image is shown in figure 6.2. $K = 7$ has been used throughout the experiment.

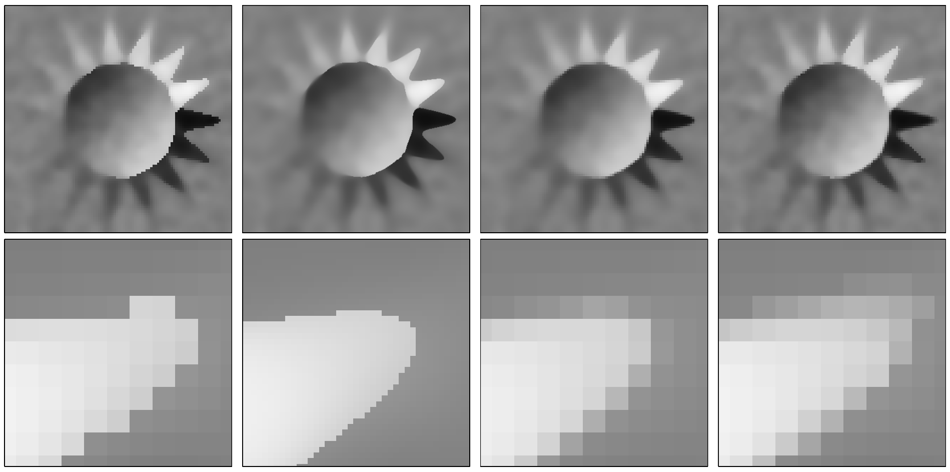


Figure 6.9: Comparison of super-sampling method and alpha synthesis. Left to right: strongest decoding synthesis, strongest decoding with 4× higher resolution, blurred and subsampled, alpha synthesis. Top row shows full images, bottom row shows a detail. $K = 7$, $\sigma = 2.2$ and $\alpha = 3\sigma$ has been used.

$$(Af * g)(x) = A(f * g)(x)\,. \tag{6.14}$$

For a suitable choice of blurring kernel $g(x)$, a small translation of the input signal will result in a small change in the sampled signal.

The behaviour of smoothing before sampling bears resemblance to the alpha synthesis in two respects:

1. For a suitable choice of α, alpha synthesis generates a smooth transition between two constant levels, in such a way that a small translation of the input signal results in a small change in the output signal. This is not the case for e.g. the strongest decoding synthesis, but it is the case for smoothing before sampling.

2. As noted earlier, the actual distance between the two grey-levels in a transition plays no part in the alpha synthesis. This means that a transition with a certain profile will be described with the same number of samples, regardless of scaling (i.e. regardless of the distance between p_1 and p_2). This follows directly from (6.7). In this respect channel smoothing with alpha synthesis behaves like a blurring before sampling for piecewise constant signals. This is different from methods such as the *histogram filter* [107], which control the sharpness of the output by an exponent on the channel values.

We can thus view channel smoothing with alpha synthesis as analogous to smoothing before sampling on a piecewise constant signal model. The analogy is that the channel images represent a continuous image model (for piecewise constant signals), and this model is blurred and sampled by the alpha synthesis.

6.5 Comparison with other denoising filters

We will now compare channel-smoothing with alpha synthesis with a number of other common denoising techniques. The methods are tested on the test-image in figure 6.2, contaminated with two noise types: Gaussian noise only, and Gaussian plus salt&pepper noise. For all methods we have chosen the filter parameters such that the mean absolute error (MAE) between the uncontaminated input and the input contaminated with Gaussian noise is minimised.[2] MAE is defined as

$$\varepsilon_{\mathrm{MAE}} = \frac{1}{N_1 N_2} \sum_{x_1=1}^{N_1} \sum_{x_2=1}^{N_2} |f_{\mathrm{ideal}}(\mathbf{x}) - f_{\mathrm{out}}(\mathbf{x})| \,. \tag{6.15}$$

MAE was chosen above RMSE, since it is more forgiving to outliers. The methods, and their optimal filter parameters are listed below

1. **Normalized averaging**.
   As defined in (6.5). $\sigma = 1.17$.
2. **Median filtering**.
   See e.g. [92]. The MATLAB implementation, with symmetric border extension, and a 5×5 spatial window.
3. **Bilateral filtering**.
   As defined in (6.4). $\sigma_s = 1.64$ and $\sigma_p = 0.30$.
4. **Mean-shift filtering**.
   As defined in (6.2). $\sigma_s = 4.64$ and $\sigma_p = 0.29$.
5. **Channel smoothing**.
   With alpha synthesis, $K = 8$, $\sigma = 1.76$, and $\alpha = 3\sigma$ (not optimised).

The results are reproduced in figure 6.10. As is evident in this experiment, neither bilateral nor mean-shift filtering is able to eliminate outliers. The reason for this is that they both do gradient descent starting in the outlier value. However, none of these methods allow outliers to influence the other pixels as a linear filter does. The average MAE values for 10 instances of the images in figure 6.10 are summarised in the table below.

noise type	input	normaver	median	bilateral	meanshift	chan.sm.
Gaussian	0.0790	0.0305	0.0307	0.0277	0.0226	0.0230
Gaussian+S&P	0.0996	0.0389	0.0320	0.0377	0.0411	0.0234

The winner for Gaussian noise is mean-shift, with channel smoothing close second. For Gaussian+salt&pepper noise, channel smoothing is way ahead of all other methods, with the median filter being the runner up. The perceptual image quality, which is what we would really like to optimise for, is a different matter however.

[2]For stability of the parameter choice, the error was measured on 10 instances of the noise contaminated input.

6.6 Applications of channel smoothing

The image denoising procedure is demonstrated in figure 6.11. The examples are described below, row by row:

row 1 This is a repeat of the experiment in figure 6.2 (left), but with alpha synthesis added, and with the number of channels chosen to match the noise (by making the channels wider).

row 2 This is restoration after contamination with the same type of noise on a natural image.

row 3 This is a duplication of the restoration of an irregularly sampled signal experiment done in [48], but using channel smoothing instead of normalized averaging to interpolate values at the missing positions. Note that this is image denoising and interpolation combined, if just interpolation is sought, normalized averaging is preferable, since the piecewise constant assumption modifies the image content slightly.

row 4 This is an example of simple image editing by means of a confidence map. The right edge of the image is black due to problems with the frame grabber. This has been corrected by setting the confidence at these locations to zero. Additionally two cars have been removed, by setting the confidences of their pixels to zero. Note that more sophisticated methods, such as *texture synthesis*[25] exist for removing objects in an image.

6.6.1 Extensions

Channel smoothing has been extended to directional smoothing by Felsberg in [28]. This is needed if we want to enhance thin periodic patterns, such as is present in fingerprints. Anisotropic channel filtering is however out of the scope of this thesis.

6.7 Concluding remarks

In this chapter we have investigated the channel smoothing technique for image denoising. A number of problems with the straight-forward approach have been identified and solutions have been suggested. Especially the alpha synthesis technique seems promising, and should be investigated further. Channel smoothing has also been compared to a number of common image denoising techniques, and was found to be comparable to, or better than all tested methods in a MAE sense. The real criterion should however be the perceived image quality, and this has not been investigated.

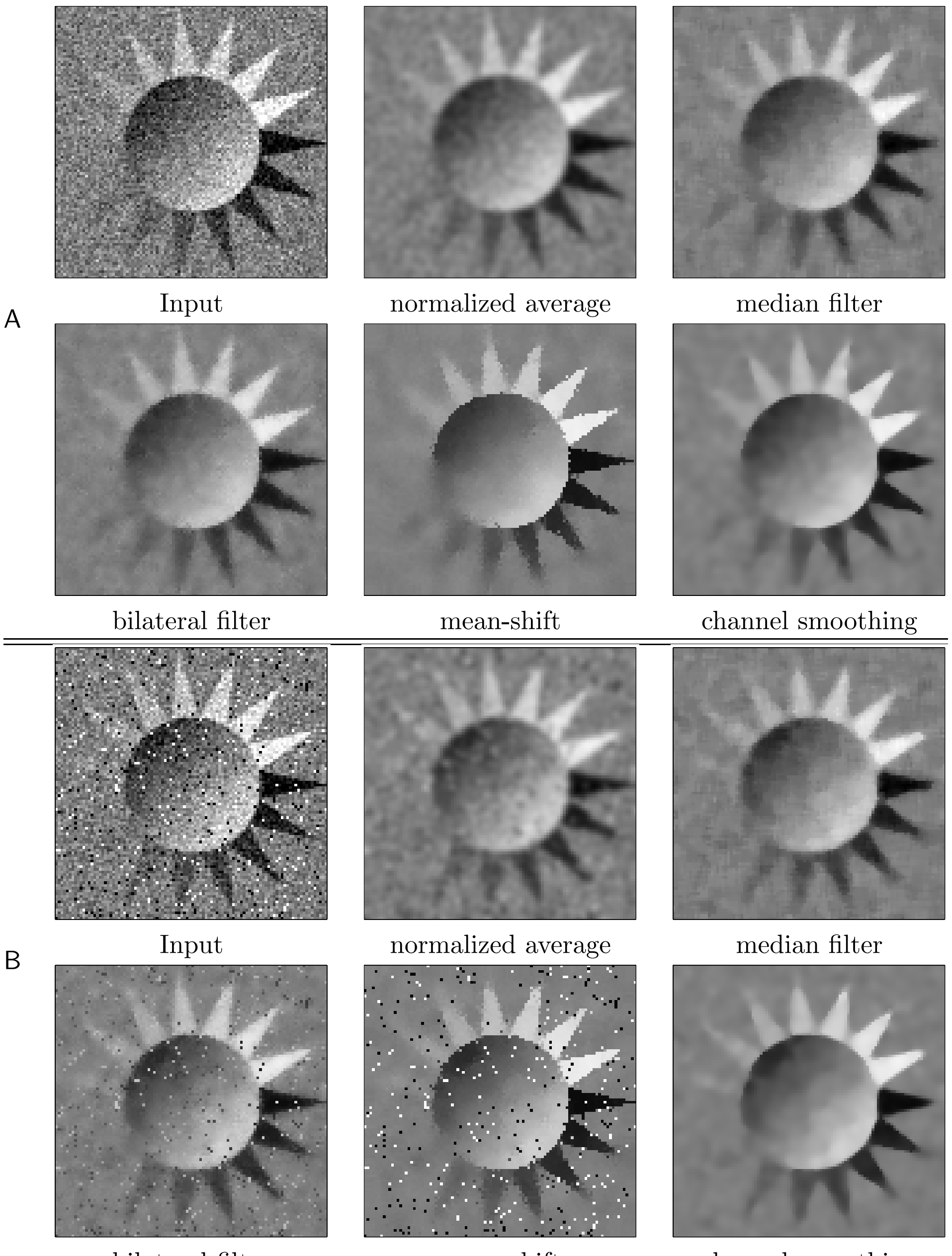


Figure 6.10: Comparison of filters. A Gaussian noise ($\sigma = 0.1$). B Gaussian noise $\sigma = 0.1$, and 5% salt&pepper pixels. Filter parameters: Normalized average: $\sigma = 1.17$, Median filter: symmetric border extension and 5×5 window, Bilateral filter: $\sigma_s = 1.64$, $\sigma_p = 0.30$, Mean-shift filter: $\sigma_s = 4.64$, $\sigma_p = 0.29$, Channel smoothing: $K = 8$, $\sigma = 1.76$, $\alpha = 3\sigma$ (not optimised).

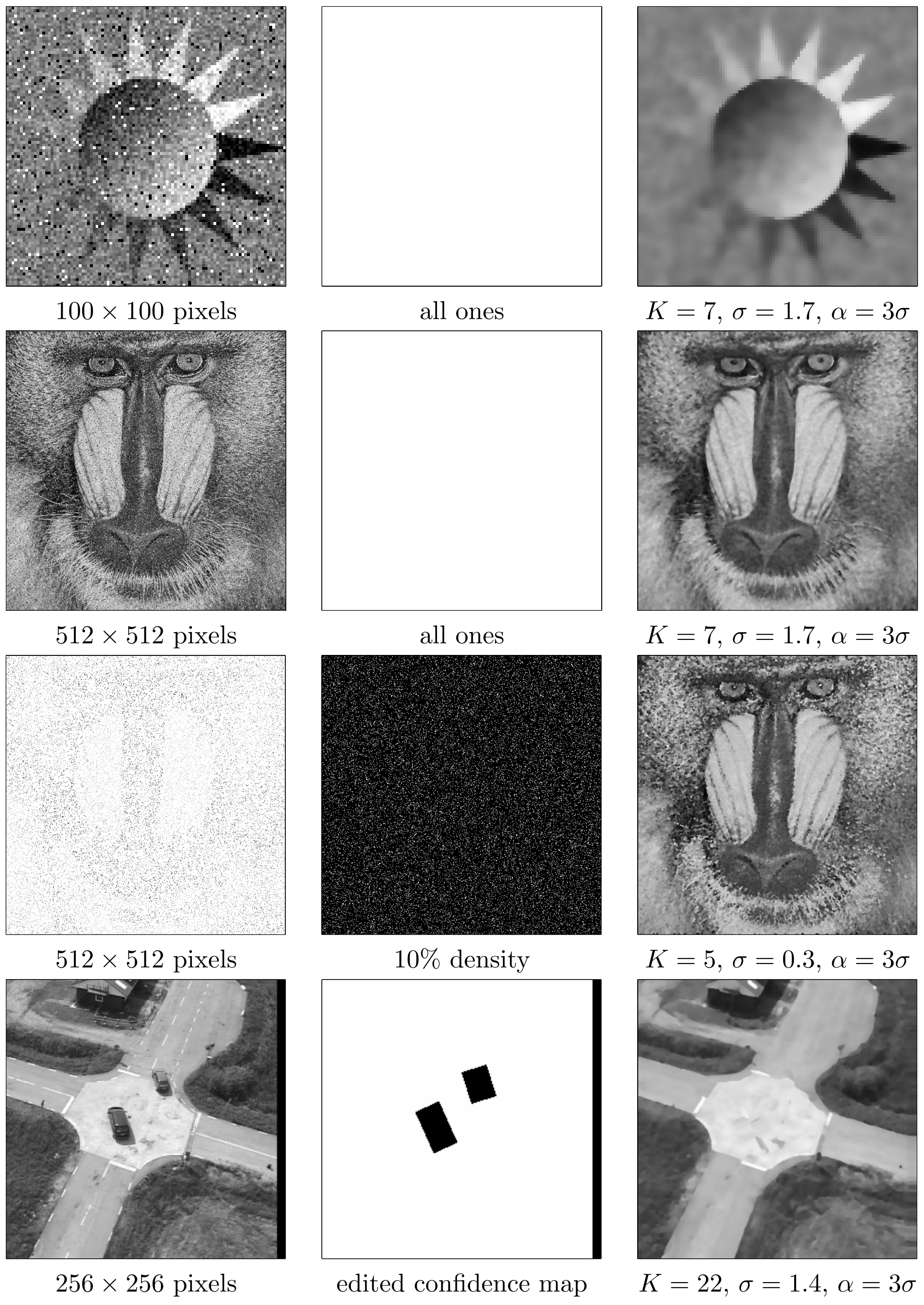

100×100 pixels
all ones
$K = 7$, $\sigma = 1.7$, $\alpha = 3\sigma$
512×512 pixels
all ones
$K = 7$, $\sigma = 1.7$, $\alpha = 3\sigma$
512×512 pixels
10% density
$K = 5$, $\sigma = 0.3$, $\alpha = 3\sigma$
256×256 pixels
edited confidence map
$K = 22$, $\sigma = 1.4$, $\alpha = 3\sigma$

Figure 6.11: Examples of channel smoothing. Columns left to right: Input images, input confidence, output. The first two images have been contaminated with Gaussian noise with $\sigma = 0.1$ (intensities $\in [0, 1]$), and 5% salt&pepper pixels.

Chapter 7

Homogeneous Regions in Scale-Space

In this chapter we develop a hierarchical blob feature extraction method. The method works on vector fields of arbitrary dimension. We demonstrate that the method can be used in wide baseline matching to align images. We also extend the method to cluster constant slopes for grey-scale images.

7.1 Introduction

For large amounts of smoothing, the channel smoothing output looks almost like a segmented image (see e.g. figure 6.2). This is what inspired us to develop the simple blob detection algorithm[1] to be presented in this chapter. The channel smoothing operation is non-linear, and fits into the same category as clustering and robust estimation techniques. The smoothing operation performed on each channel is however linear, and thus relates to linear scale-space theory.

7.1.1 The scale-space concept

When applying successive low-pass filtering to a signal, fine structures are gradually removed. This is formalised in the theory of *scale space*[106, 72]. Scale space is the extension of a signal $f(\mathbf{x})$, by means of a blurring kernel $g(\mathbf{x}, \sigma)$ into a new signal

$$f_s(\mathbf{x}, \sigma) = (f * g(\sigma))(\mathbf{x})\,. \tag{7.1}$$

The parameter σ is the scale coordinate. The original signal is embedded in the new signal since $f_s(\mathbf{x}, 0) = f(\mathbf{x})$. The kernel $g(\mathbf{x}, \sigma)$ is typically a Gaussian, but Poisson kernels have also been used [27]. Figure 7.1 contains an illustration of the scale-space concept.

[1]The method was originally presented in [41]. Here we have extended the method to deal with colour images, and also made a few other improvements.

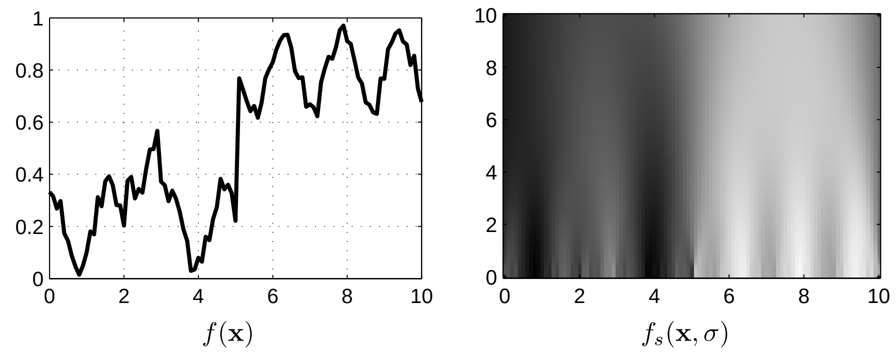


Figure 7.1: Gaussian scale space of a 1D-signal.

The fact that fine structures are removed by blurring motivates the use of a lower sample density at coarser scales, as is done in a scale pyramid.

7.1.2 Blob features

Homogeneity features are called *blobs* in scale-space theory [72]. In comparison with segmentation, blob detection has a more modest goal—we do not attempt to segment out exact shapes of objects, instead we want to extract robust and repeatable features. The difference between segmentation and blob detection is illustrated by the example in figure 7.2. As can be seen, the blob representation discards exact shapes, and thin connections between patches are neglected.

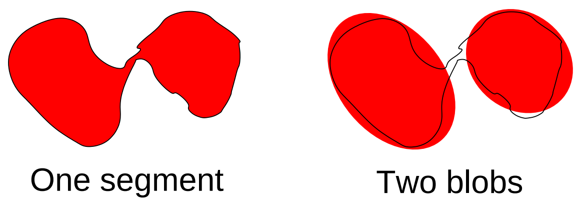


Figure 7.2: Difference between segmentation and blob detection.

Blob features have been used as texture descriptors [73] and as features for image database search [6]. For a discussion of the similarities and differences of other approaches and the one presented here, the reader is directed to [40].

Blob features are related to *maximally stable extremal regions*(MSER)[82], and to *affinely invariant neighbourhoods*[99]. MSER features are regions grown around an intensity extrema (max or min) and are used to generate affine invariant frames, which are then used for view-based object recognition [82]. Affinely invariant neighbourhoods are found by starting at intensity extrema and finding the nearest extrema along rays emanating from the point. These extrema are then linked to form a closed curve, which is used to define an affine invariant [99].

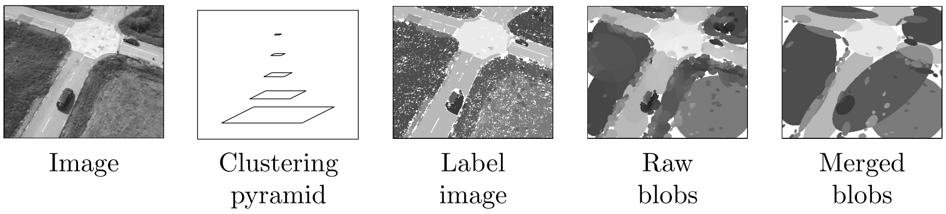


Figure 7.3: Steps in blob extraction.

7.1.3 A blob feature extraction algorithm

The blob estimation procedure uses a scale pyramid. Each position and scale in the pyramid contains a measurement–confidence pair $(\mathbf{p}, r)$.[2] The confidence is a binary variable, i.e. $r \in \{0, 1\}$. It signifies the absence or presence of a *dominant* measurement in the local image region. When $r = 1$, the dominant measurement is found in $\mathbf{p}$. This representation is obtained by non-linear means, in contrast to most scale-space methods which are linear [72], and thus obtain the *average* measurement.

The pyramid is used to generate a label image, from which we can compute the moments of each region. The shape of each region is approximated by its moments of orders 0, 1 and 2. These moments are conveniently visualised as ellipses, see right part of figure 7.3. Finally we merge blobs which are adjacent and of similar colour using an agglomerative clustering scheme. The following sections will each in turn describe the different steps of the algorithm, starting with the generation of the clustering pyramid.

7.2 The clustering pyramid

When building the clustering pyramid, we view each image pixel as a measurement $\mathbf{p}$ with confidence $r = 1$. We then expand the image into a set of channel images. For each of these channel images we generate a low-pass pyramid. The channels obtained from the input image constitute scale 1. Successively coarser scales are obtained by low-pass filtering, followed by sub-sampling. The low-pass filter used consists of a horizontal and a vertical 4-tap binomial kernel [1 3 3 1]/8.

Since the filter sums to 1, the confidence values of the decoded $(\mathbf{p}, r)$ pairs correspond to fractions of the *area* covered by the filter. Thus, we construct the low-pass pyramids, and decode the dominant $(\mathbf{p}, r)$ pair in each position. Finally we make the confidence binary by setting values below r_{min} to zero, and values above or equal to r_{min} to 1. Typically we use the area threshold $r_{\text{min}} = 0.5$. Figure 7.4 shows such a pyramid for an aerial image.

In principle this pyramid generation method can also be applied to vector field images, by extending the channel representation as described in section 5.6. The number of channels required for vector fields of dimension higher than 2 does however make this approach rather expensive with respect to both computational

[2]For generality we will have a vector notation of $\mathbf{p}$, although we will initially use scalar valued images.

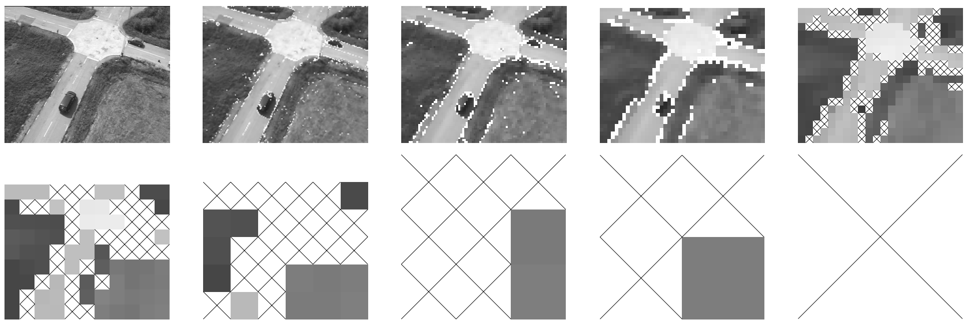


Figure 7.4: Clustering pyramid created using $K = 26$ channels, spaced according to (6.1). Positions with confidence $r = 0$ are indicated with crosses.

and storage requirements. For example a channel representation of an RGB colour image with 26 channels per colour band will give us $26^3 = 17576$ channel images to filter.

7.2.1 Clustering of vector fields

To perform clustering on vector fields, we will replace channel filtering with another robust estimation technique. The representation at the next scale, $\mathbf{p}^*$, is now generated as the solution to the weighted robust estimation problem

$$\arg\min_{\mathbf{p}^*} \sum_k w_k r_k \rho(||\mathbf{p}_k - \mathbf{p}^*||)\,. \tag{7.2}$$

Here w_k are weights from the outer product of two binomial kernels, and $\rho(d)$ is a robust error norm. In contrast to linear filter theory we cannot use a separable optimisation, so for a 1D binomial kernel [1 3 3 1]/8 we will have to take all 16 pixels in a local neighbourhood into account.

Note that for most choices of $\rho(d)$, problem (7.2) is not convex, and thus local minima exist. We are however looking for the global min, and will employ a technique known as *successive outlier rejection* (SOR) to find it. An iteration of SOR is computed as

$$\mathbf{p}^*_{\text{est}} = \frac{1}{N_r}\sum_k \mathbf{p}_k r_k w_k o_k \quad \text{where} \quad N_r = \sum_k r_k w_k o_k\,. \tag{7.3}$$

Here o_k are outlier rejection weights, which are initially set to 1. After each iteration we find the pixel with largest residual $d_k = ||\mathbf{p}^*_{\text{est}} - \mathbf{p}_k||$. If $d_k > d_{\max}$, we remove this pixel by setting $o_k = 0$. This procedure is iterated until there are no outliers left. The found solution will be a fixed point of a gradient descent on (7.2), for the *cut-off squares error norm*[3]. Furthermore, provided that more than half the data supports it, the solution will be either the global min, or close

[3] $\rho(d) = d^2$ for $|d| < d_{\max}$, and $d^2_{\max}$ otherwise.

to the global min. The SOR approach thus effectively solves the initialisation problem which exists in the commonly applied *iterated reweighted least squares* (IRLS) technique, see section 4.2.4, or e.g. [109, 95]. Initialisation is also the reason why mean-shift filtering [19, 20] is unsuitable for generation of a clustering pyramid. The importance of initialisation is demonstrated in figure 7.5. Here we have applied SOR to minimise (7.2), and compared the result with mean-shift filtering, and IRLS. The IRLS method was, just like mean-shift, initiated with the original pixel value. As can be seen, only SOR is able to reject outliers. Note that since each iteration either removes one outlier or terminates, there is an upper limit to the number of required iterations (e.g. 16 for a 4×4 neighbourhood).

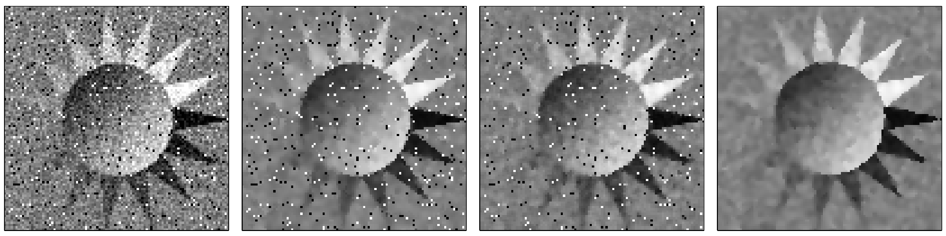


Figure 7.5: Importance of initialisation. Left to right: Input image, IRLS output (6×6 window), mean-shift output (spatial radius 3), SOR output (6×6 window).

As mentioned, the SOR method is closely related to (7.2), for the cut-off squares error norm. The sharp boundary between acceptance and rejection in cut-off squares is usually undesirable. As a remedy to this we add a few extra IRLS iterations with a more smooth error norm. An iteration of IRLS on (7.2) has the same form as (7.3), but with the outlier rejection weights replaced with $o_k = \rho'(d_k)/d_k$. If the error norms are similar in shape and extent, we should already be close to the solution, since we are then minimising similar functions. We will use the weighting function

$$o(d_k) = \begin{cases} (1 - (d_k/d_{\max})^2)^2 & \text{if } |d_k| < d_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

which corresponds to the biweight error norm [109]. The $d_{\max}$ parameter defines the scale of the error norm, and is used to control the sensitivity of the algorithm. To compute $\mathbf{p}^*$ and r^* in each position in the pyramid, we thus first find $\mathbf{p}^*$ with SOR, followed by IRLS. The output confidence r^* is then computed as

$$r^* = \begin{cases} 1 & \text{if} \quad \sum_k r_k w_k o_k \geq r_{\min} \sum_k w_k \\ 0 & \text{otherwise.} \end{cases} \tag{7.5}$$

Here o_k are the weights used in the last IRLS iteration, see (7.3), and just like in section 7.2, $r_{\min}$ is a threshold which corresponds to the minimum required area fraction belonging to the cluster. The clustering process is repeated for successively coarser scales until the pyramid has been filled.

7.2.2 A note on winner-take-all vs. proportionality

There is a distinct difference between the channel approach and the SOR+IRLS approach with respect to propagation of information through the pyramid. For the channel variant, all pixels in the input image will have contributed to the value of the channel vector at the top of the pyramid. For the SOR+IRLS variant, each pixel at each scale only describes the dominant colour component, and thus all the other values will be suppressed. This is the same approach as is taken in elections in the UK and the US, where each constituency only gets to select one (or one kind of) representative to the next level. The channel approach on the other hand corresponds to the Swedish system of propagating the proportions of votes for the different parties to the next level.

The number of channels K, and the maximum property distance $d_{\max}$ can be related according to $d_{\max} = 2t_1 = 2(K + 1 - N)/(r_u - r_l)$, where N, r_u, and r_l are defined in section 3.3. The relationship comes from $2t_1$ being the approximate boundary between averaging and rejection, see section 5.4.

The two methods are quite similar in behaviour, but even for grey-scale images on conventional computer architectures, the SOR+IRLS variant is a factor 5 or more faster. This is due to the small support (4×4) of the parameter estimation. On hardware with higher degrees of parallelism this could however be different.

7.3 Homogeneous regions

The generation of the clustering pyramid was bottom up, and we will now continue by doing a top-down pass over the pyramid. Note that the region propagation described here is different from the one in [41]. The changes since [41] have speeded up the algorithm and also made the result after region merging more stable.

We start at the top scale and generate an empty label image. Each time we encounter a pixel with a property distance above $d_{\max}$) from the pixels above it, we will use it as a seed for a new region. Each new seed is assigned an integer label. We allow each pixel to propagate its label to the twelve nearest pixels below it, provided that they are sufficiently similar see figure 7.6 (left). For a pixel on the scale below, this means that we should compare it to three pixels above it, see figure 7.6 (right). If several pixels on the scale above have a property distance below $d_{\max}$, we propagate the label of the one with the smallest distance.

The algorithm for label image generation is summarised like this:

step 1 Generate an empty label image at the top scale.

step 2 Assign new labels to all unassigned pixels with $r = 1$.

step 3 Move to the next scale.

step 4 Compare each pixel with $r = 1$ to the three nearest pixels on the scale above. Propagate the label of the pixel with the smallest property distance, if smaller than $d_{\max}$.

step 5 If we are at the finest scale we are done, otherwise go back to **step 2**.

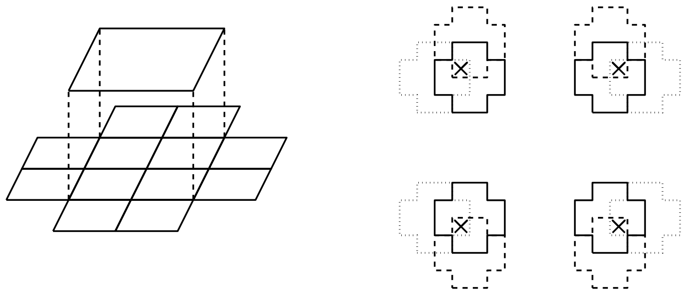


Figure 7.6: Label propagation. Left: A pixel can propagate its label to twelve pixels at the scale below. Right: This can be implemented by comparing each pixel with three pixels on the scale above.

The result of this algorithm is shown in figure 7.7. As can be seen, this is an *oversegmentation*, i.e. image patches which are homogeneous have been split into several regions.

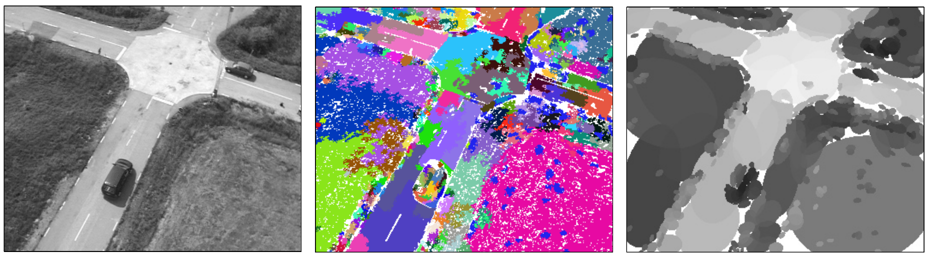


Figure 7.7: Result of label image generation. Left to right: Input image, label image, blobs from regions.

7.3.1 Ellipse approximation

The label image $l(\mathbf{x}) : \mathbb{Z}^2 \rightarrow \mathbb{N}$ is a compact representation of a set of binary masks

$$v_n(\mathbf{x}) = \begin{cases} 1 & \text{if } l(\mathbf{x}) = n \\ 0 & \text{otherwise.} \end{cases} \tag{7.6}$$

The raw moments of such a binary mask $v_n(\mathbf{x}) : \mathbb{Z}^2 \rightarrow \{0, 1\}$ are defined by the weighted sum

$$\mu_{kl} = \sum_{x_1} \sum_{x_2} x_2^k x_1^l v_n(\mathbf{x}). \tag{7.7}$$

For a more extensive discussion on moments of binary masks, see e.g. [92]. For all the regions $\{v_n\}_1^N$ in the image we will now compute the raw moments of order 0 to 2, i.e. μ_{00}, μ_{01}, μ_{10}, μ_{02}, μ_{11}, and μ_{20}. This can be done using only one `for`-loop over the label image.

The raw moments are then converted to measures of the area a_n, the centroid vector $\mathbf{m}_n$, and the inertia matrix $\mathbf{I}_n$ according to

$$a_n = \mu_{00}, \quad \mathbf{m}_n = \frac{1}{\mu_{00}} \begin{pmatrix} \mu_{01} \\ \mu_{10} \end{pmatrix} \quad \text{and} \quad \mathbf{I}_n = \frac{1}{\mu_{00}} \begin{pmatrix} \mu_{02} & \mu_{11} \\ \mu_{11} & \mu_{20} \end{pmatrix} - \mathbf{m}_n \mathbf{m}_n^T. \tag{7.8}$$

Using the input image $\mathbf{p}(x, y)$ we also compute and store the average measurements for all regions,

$$\mathbf{p}_n = \frac{1}{\mu_{00}} \sum_{x_1} \sum_{x_2} \mathbf{p}(x_1, x_2) v_n(x_1, x_2). \tag{7.9}$$

If a region has the shape of an ellipse, its shape can be retrieved from the inertia matrix, see theorem C.1 in the appendix. Even if the region is not a perfect ellipse, the ellipse corresponding to the inertia matrix is a convenient approximation of the region shape. From the eigenvalue decomposition $\mathbf{I} = \lambda_1 \hat{\mathbf{e}}_1 \hat{\mathbf{e}}_1^T + \lambda_2 \hat{\mathbf{e}}_2 \hat{\mathbf{e}}_2^T$ with $\lambda_1 \geq \lambda_2$ we can find the axes of the ellipse as $2\sqrt{\lambda_1}\hat{\mathbf{e}}_1$ and $2\sqrt{\lambda_2}\hat{\mathbf{e}}_2$ respectively, see theorem C.2 in the appendix. Since $\mathbf{I} = \mathbf{I}^T$ each blob can be represented by $1 + 2 + 3 + N$ parameters where N is the dimensionality of the processed vector field. I.e. we have 7 parameters for grey-scale images, and 9 for RGB images etc. A visualisation of the regions as ellipses is shown in figure 7.7 (right).

7.3.2 Blob merging

As stated before the result of the label image generation is an oversegmentation, and thus the final stage of the algorithm is a merging of adjacent blobs.

Due to the linearity of the raw moments (7.7) the moments of a combined mask can be computed from the moments of its parts. I.e. if we have $v = v_1 + v_2$ we get $\mu_{ij}(v) = \mu_{ij}(v_1) + \mu_{ij}(v_2)$. For the corresponding property vectors we get $\mathbf{p}(v) = (\mu_{00}(v_1)\mathbf{p}(v_1) + \mu_{00}(v_2)\mathbf{p}(v_2))/\mu_{00}(v)$.

Candidates for merging are selected based on a count of pixels along the border of two regions with similar colours. We define an adjacency matrix $\mathbf{M}$ with M_{ij} signifying the number of pixels along the common border of blobs i and j. The adjacency matrix is computed by modifying the region propagation algorithm in section 7.3, such that **step 4** at the finest scale computes the count. Whenever two pixels (with different labels) at the coarser scale match a pixel at the finer scale, we are at a region boundary, and thus add 1 to the corresponding position in the adjacency matrix. Since $\mathbf{M}$ is symmetric, only the upper triangle need to be stored. Using $\mathbf{M}$ we can now select candidates for merging according to

$$M_{ij} > m_{\text{thr}} \sqrt{\min(\mu_{00}(v_i), \mu_{00}(v_j))} \tag{7.10}$$

where m_{thr} is a tuning threshold. Typically we use $m_{\text{thr}} = 0.5$. This choice results in a lot of mergers, but thin strands of pixels are typically not allowed to join two blobs into one, see figure 7.2. The square root in (7.10) is motivated by M_{ij} being a length, and μ_{00} being an area.

All merging candidates are now placed in a list. They are successively merged pairwise, starting with the most similar pair according to the property distance $d = ||\mathbf{p}_i - \mathbf{p}_j||$. After merging, the affected property distances are recomputed, and a new smallest distance pair is found. The pairwise merging is repeated until none of the candidate mergers have a property distance below $d_{\max}$. This clustering scheme falls into the category of *agglomerative clustering* [63].

Finally we remove blobs with areas (i.e. μ_{00}) below a threshold $a_{\min} = 20$.

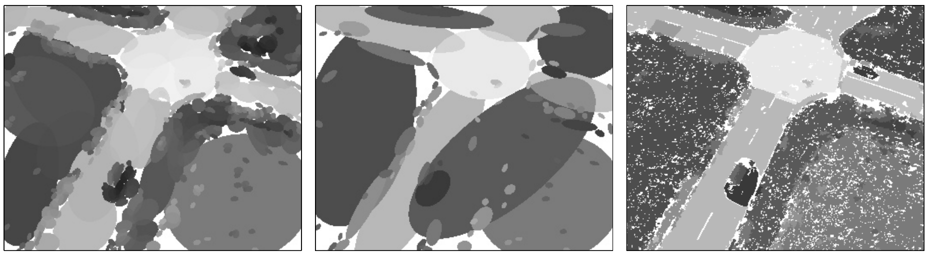


Figure 7.8: Result of blob merging. Left to right: blobs from regions (384 blobs), merged blobs (92 blobs, $m_{\text{thr}} = 0.5$), reprojection of blob colours onto label image regions.

Figure 7.8 shows blob representations of the image in figure 7.7 (left), before and after merging. In order to show the quality of the segmentation, the blob colours have been used to paint the corresponding regions in the label image, see figure 7.8 (right). In this example, the clustering pyramid is created using $K = 26$ channels that are spaced according to (6.1). The blob representation initially contains 384 blobs, which after merging and removal of small blobs drops to 92. This gives a total of 644 parameters for the entire image. Compared to the 348×287 input image pixels this is a factor 155 of data reduction.

7.4 Blob features for wide baseline matching

The usefulness of an image feature depends on the application, and should thus be evaluated in a proper context. We intend to use the blob features for view based object recognition, wide-baseline matching, and aerial navigation. All of these topics are however out of the scope of this thesis, and we will settle for a very simple demonstration. Using pairs of images captured from a helicopter, we detect blobs, and store their centroids in two lists $\{\mathbf{m}_k\}_1^K$, and $\{\mathbf{n}_l\}_1^L$. We then find pairs of points that correspond given a homographic model

$$h \begin{pmatrix} \hat{\mathbf{n}}_k \\ 1 \end{pmatrix} = \mathbf{H} \begin{pmatrix} \mathbf{m}_k \\ 1 \end{pmatrix} \quad \text{and} \quad h \begin{pmatrix} \hat{\mathbf{m}}_l \\ 1 \end{pmatrix} = \mathbf{H}^{-1} \begin{pmatrix} \mathbf{n}_l \\ 1 \end{pmatrix} . \tag{7.11}$$

Two points are said to correspond when the residual

$$\delta_{kl} = \sqrt{(\hat{\mathbf{n}}_k - \mathbf{n}_l)^T (\hat{\mathbf{n}}_k - \mathbf{n}_l)} + \sqrt{(\hat{\mathbf{m}}_l - \mathbf{m}_k)^T (\hat{\mathbf{m}}_l - \mathbf{m}_k)} \tag{7.12}$$

is below a given threshold.

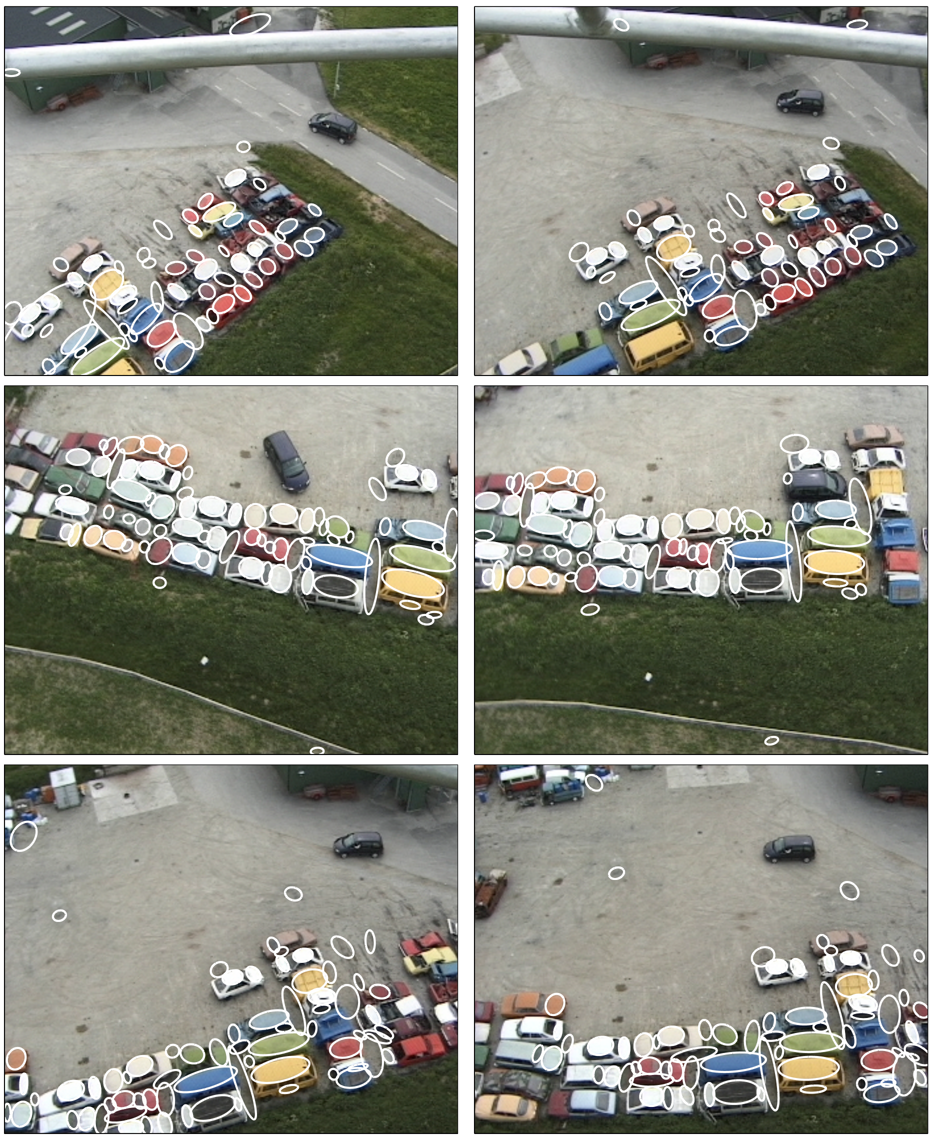


Figure 7.9: Correspondences for blobs from a video sequence. Each row shows a pair of matched images.

The correspondence is found using a RANSAC [109] like method. We start by selecting 4 random points in image 1. For each of these we select a random point in the other image among those with a property distance $d_{kl} = ||\mathbf{p}_{m,k} - \mathbf{p}_{n,l}||$ below $d_{\max}$. For each such random correspondence, we compute the pairwise geometric residuals (7.12) for all point pairs and keep the pairs with distances below a threshold $\delta_{\max}$. We stop once we get more than 15 matches. The found matches are then used to estimate a new homography using *scaled total least squares* (STLS) with the scaling described in [57]. We then find correspondences given the new homography and recompute the residuals. This procedure is iterated until convergence, which usually is reached after three to four cycles.

Figure 7.9 shows the found correspondences for three image pairs from a video sequence. The number of blobs N_b, the number correspondences N_m, and the inlier fraction $\varepsilon = N_m/N_b$ are listed in the table below.

Frame	1	2	3	4	5	6
N_b	142	163	151	139	161	176
N_m	61	61	76	76	70	70
ε	0.43	0.37	0.50	0.55	0.43	0.40

7.4.1 Performance

A C-implementation of the blob feature extraction algorithm takes about 4 seconds to process a 360 × 288 RGB image on a Sun Ultra 60 (296MHz). Moving the implementation to an Intel Pentium 4 (1.9GHz) platform resulted in computation times below one second per frame.

7.4.2 Removal of cropped blobs

Since an image only depicts a window of a scene, some of the homogeneous regions in the scene will only partially be contained in the image. Such cropped regions will give rise to blobs which change shape as the camera moves, in a manner which does not correspond to camera movement. Thus, if the blobs are to be used for image matching, we will probably gain stability in the matching by removing cropped blobs. Most such blobs can be removed by calculating the bounding box of the ellipse corresponding to the blob shape, and removing those blobs which have their bounding boxes partially outside the image.

In appendix C, theorem C.3, the outline of an ellipse is shown to be given by a parameter curve. To find a bounding box for an ellipse, we rewrite this curve into two parameter curves

$$\mathbf{x} = \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix} \begin{pmatrix} a\cos t \\ b \sin t \end{pmatrix} + \mathbf{m} = \begin{pmatrix} ar_{11}\cos t + br_{12}\sin t \\ ar_{21}\cos t + br_{22}\sin t \end{pmatrix} + \mathbf{m} \tag{7.13}$$

$$= \begin{pmatrix} \sqrt{a^2r_{11}^2 + b^2r_{12}^2}\sin(t+\varphi_1) \\ \sqrt{a^2r_{21}^2 + b^2r_{22}^2}\sin(t+\varphi_2) \end{pmatrix} + \mathbf{m} \tag{7.14}$$

Since $\sin(t+\varphi)$ assumes all values in the range $[-1,1]$ during one period, the bounding box is given by the amplitudes

$$A_1 = \sqrt{a^2r_{11}^2 + b^2r_{12}^2} \quad \text{and} \quad A_2 = \sqrt{a^2r_{21}^2 + b^2r_{22}^2}\,. \tag{7.15}$$

The bounding box becomes

$$x_1 \in [m_1 - A_1, m_1 + A_1] \quad \text{and} \quad x_2 \in [m_2 - A_2, m_2 + A_2]\,. \tag{7.16}$$

7.4.3 Choice of parameters

The blob feature extraction algorithm has three parameters, *spatial kernel width*, *area threshold* $a_{\min}$, and *range kernel width* $d_{\max}$. We will now give some suggestions on how they should be set.

1. **Spatial kernel width** A larger spatial kernel will make the method less sensitive to translations of the grid. At the same time however, it will also reduce the amount of detail obtained. A trade-off that seems to work well for a wide range of images is the choice of a 4×4 neighbourhood. To speed up the algorithm, we can even consider using the 12 pixel neighbourhood consisting of the central pixels in the 4×4 region, and omit the spatial weights.

2. **Area threshold** $r_{\min}$. Characteristic for this parameter is that low values give more mergers of non-connected but adjacent regions. This results in fewer features. High values will cause less information to be propagated to the higher levels of the pyramid. This will lead to fewer mergers, but also to less information being propagated to higher levels in the pyramid. Thus we obtain more regions at the lowest level, and consequently higher computational times. Typically we will use the intermediate value $r_{\min} = 0.5$.

3. **Range kernel witdh** $d_{\max}$. This parameter decides whether two colours should be considered the same or not. A small value will give lots of blobs, while a large value gives few blobs. A suitable choice of $d_{\max}$ value depends on the input images.

Typically we will thus mainly modify the $d_{\max}$ parameter and let the others have the fixed values suggested above.

7.5 Clustering of planar slopes

The implicit assumption in the previous clustering methods has been that the image consists of piecewise constant patches. We could instead assume a constant slope. This is reasonable e.g. in depth maps from indoor scenes and of various man-made objects. We now assume a scalar input image, i.e. $f : \mathbb{Z}^2 \to \mathbb{R}$, and a local image model of the type

$$f(\mathbf{x}) = \begin{pmatrix} 1 & x_1 - m_1 & x_2 - m_2 \end{pmatrix} \mathbf{p}(\mathbf{x})\,. \tag{7.17}$$

Here $\mathbf{m} = \begin{pmatrix} m_1 & m_2 \end{pmatrix}^T$ is the centre spatial position of the local model. The parameter vector in each pixel can be interpreted as $\mathbf{p}(\mathbf{x}) = (\text{mean}, \text{slope}_x, \text{slope}_y)^T$.

When building the first level of the clustering pyramid we thus have to go from the image f to a parametric representation $\mathbf{p}$. In principle this could be done by applying plain derivative filters. This would however distort the measurements at the edges of each planar patch, and thus make the size of the detected regions smaller, as well as limiting the sizes of objects that can be detected. Instead we will estimate the parametric representation using a robust linear model

$$\arg\min_{\mathbf{p}^*} \sum_k w_k r_k \rho(||f_k - \begin{pmatrix} 1 & x_{1,k} - m_1 & x_{2,k} - m_2 \end{pmatrix} \mathbf{p}^*||) \tag{7.18}$$

where w_k are weights from a binomial kernel, typically in a 4×4 neighbourhood. Like in the colour clustering method, see section 7.2, we solve (7.18) by SOR followed by a few M-estimation steps. For the 4×4 region, we define the following quantities

$$\mathbf{M} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix} - 2.5 \quad \mathbf{Y} = \mathbf{X}^T \tag{7.19}$$

$$\mathbf{B} = \begin{pmatrix} | & | & | \\ \text{vec}(\mathbf{M}) & \text{vec}(\mathbf{X}) & \text{vec}(\mathbf{Y}) \\ | & | & | \end{pmatrix}\,. \tag{7.20}$$

We also define a weight matrix $\mathbf{W}$ with diagonal elements $(\mathbf{W})_{kk} = o_k w_k r_k$. Here o_k are outlier rejection weights, w_k are spatial binomial weights, and r_k are the input confidences. For the 4×4 region $\mathbf{f}$, the iterations of the model parameter estimation are now performed as

$$\mathbf{p}^*_{\text{est}} = (\mathbf{WB})^{\dagger}\mathbf{W}\text{vec}(\mathbf{f})\,. \tag{7.21}$$

Since we have just 3 parameters to estimate, we could use a (slightly less reliable) matrix inversion instead

$$\mathbf{p}^*_{\text{est}} = (\mathbf{B}^T\mathbf{WWB})^{-1}\mathbf{B}^T\mathbf{WW}\text{vec}(\mathbf{f})\,. \tag{7.22}$$

This is faster since the matrix inverse can be computed in a non-iterative way. At the start all outlier rejection weights are set to 1. After each iteration we find the pixel with the largest residual

$$d_k = \left| f_k - \begin{pmatrix} 1 & x_{1,k} - m_1 & x_{2,k} - m_2 \end{pmatrix} \mathbf{p}^*_{\mathrm{est}} \right| . \tag{7.23}$$

If $d_k > d_{\max}$, we remove this pixel by setting $o_k = 0$. By iterating until convergence we will find a fixed point of (7.18), for the cut-off squares error norm. Furthermore, if more than half the data supports the solution, we are guaranteed to be close to the global min. Again, we polish the result by a few M-estimation steps with a smoother kernel. The corresponding IRLS iteration will have the same form as the iteration above, i.e (7.21) or (7.22), with the exception that the outlier rejection weights are replaced with $o_k = \rho'(d_k)/d_k$.

To obtain $\mathbf{p}^*$ and r^* for the first scale in the pyramid, we thus compute $\mathbf{p}^*$ according to the above equations, and r^* according to (7.5).

We typically set the area threshold to $r_{\min} = 0.85$, which is considerably higher than in the colour method. Pixels on the boundaries between two different slopes will typically have a parameter estimate that does not correspond to any of the slopes. In order to get zero confidence for such pixels, we could either reduce the $d_{\max}$ threshold, or increase the $r_{\min}$ parameter. It turns out that the latter option is preferable, since reducing $d_{\max}$ will also cause grey-level quantisation in the input to propagate to the slope parameters. The result of the parameter estimation step for a simple test image is shown in figure 7.10.

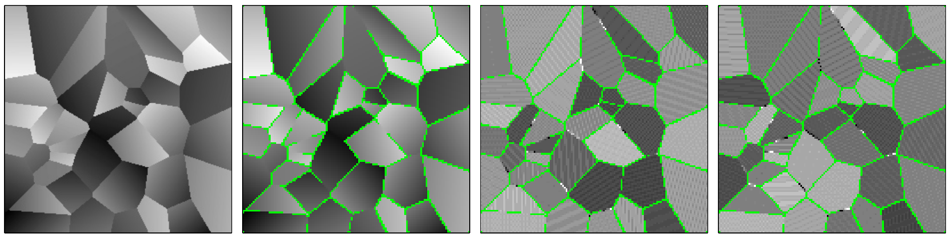


Figure 7.10: Result of parameter estimation. Left to right: Input, mean estimate, x-slope estimate, y-slope estimate.

7.5.1 Subsequent pyramid levels

After we have obtained the parametric representation, we can generate the other steps in the pyramid using almost the same technique as in the colour method. The main difference is that we have to take the centre of the local neighbourhood into account, and adjust the local mean level accordingly. That is, we now have a robust estimation problem of the form

$$\arg\min_{\mathbf{p}^*} \sum_k w_k r_k \rho(||\tilde{\mathbf{p}}_k - \mathbf{p}^*||) \tag{7.24}$$

where $\tilde{\mathbf{p}}_k$ is a vector with the mean level adjusted

$$\tilde{\mathbf{p}}_k = \begin{pmatrix} p_{1,k} - ((x_{1,k} - m_1)p_{2,k} + (x_{2,k} - m_2)p_{3,k})s \\ p_{2,k} \\ p_{3,k} \end{pmatrix} . \tag{7.25}$$

Here s is a factor that compensates for the fact that the pixel distance at scale 2 is twice that at scale 1 and so on. In other words we have $s = 2^{\text{scale}-1}$. The residuals for SOR are now computed as

$$d_k = \sqrt{(\tilde{\mathbf{p}}_k - \mathbf{p}^*_{\text{est}})^T \mathbf{W} (\tilde{\mathbf{p}}_k - \mathbf{p}^*_{\text{est}})} \tag{7.26}$$

where $\mathbf{W}$ is a weight matrix of the form $\text{diag}(\mathbf{W}) = \begin{pmatrix} 1 & w_d & w_d \end{pmatrix}$. The parameter w_d allows us to adjust the relative importance of error in mean and error in slope. Typically we set $w_d = 200$.

7.5.2 Computing the slope inside a binary mask

To estimate the mean and slopes inside each mask v_n, we will now assume a local signal model centred around the point $\mathbf{m} = \begin{pmatrix} m_1 & m_2 \end{pmatrix}^T$

$$f(\mathbf{x}) = \begin{pmatrix} 1 & x_1 - m_1 & x_2 - m_2 \end{pmatrix} \mathbf{p} = p_1 + p_2 x_1 - p_2 m_1 + p_3 x_2 - p_3 m_2 \, . \tag{7.27}$$

We define the moments η_{kl} of $f(\mathbf{x})$ inside the mask v_n as

$$\eta_{kl} = \frac{1}{N} \sum_{x_1,x_2} v_n(x_1,x_2) f(x_1,x_2) x_1^k x_2^l \tag{7.28}$$

where $N = \mu_{00}$ is the number of elements inside the mask v_n. For the model (7.27) we now get

$$\eta_{00} = p_1 + p_2 \frac{1}{N} \sum_{x_1,x_2} v_n(x_1,x_2) x_1 - p_2 m_1 + p_3 \frac{1}{N} \sum_{x_1,x_2} v_n(x_1,x_2) x_2 - p_3 m_2 \tag{7.29}$$

$$= p_1 + p_2 m_1 - p_2 m_1 + p_3 m_2 - p_3 m_2 = p_1 \tag{7.30}$$

as expected. For the first moments we obtain

$$\eta_{10} = p_1 m_1 + p_2 \left(\frac{1}{N} \sum_{x_1,x_2} v_n(x_1,x_2) x_1^2 - m_1^2 \right)$$
$$+ p_3 \left(\frac{1}{N} \sum_{x_1,x_2} v_n(x_1,x_2) x_1 x_2 - m_1 m_2 \right) \tag{7.31}$$
$$= p_1 m_1 + p_2 I_{11} + p_3 I_{12} \tag{7.32}$$

and

$$\begin{aligned}\eta_{01} &= p_1 m_2 + p_2 \left(\frac{1}{N} \sum_{x_1,x_2} v_n(x_1,x_2) x_2 x_1 - m_2 m_1 \right) \\ &\quad + p_3 \left(\frac{1}{N} \sum_{x_1,x_2} v_n(x_1,x_2) x_2^2 - m_2^2 \right) \qquad (7.33) \\ &= p_1 m_2 + p_2 I_{21} + p_3 I_{22} \, . \qquad (7.34)\end{aligned}$$

This can be summarised as the system

$$\begin{pmatrix}\eta_{01}\\ \eta_{10}\end{pmatrix} = p_1 \mathbf{m} + \mathbf{I} \begin{pmatrix}p_2\\ p_3\end{pmatrix} \qquad (7.35)$$

where **m** and **I** are obtained according to (7.7) and (7.8). We thus first compute **m** and **I**. We then compute p_1 from η_{00}, see (7.30), and finally p_2 and p_3 as

$$\begin{pmatrix}p_2\\ p_3\end{pmatrix} = \mathbf{I}^{-1} \begin{pmatrix}\eta_{01} - p_1 m_1\\ \eta_{10} - p_1 m_2\end{pmatrix} \, . \qquad (7.36)$$

7.5.3 Regions from constant slope model

The results of blob feature extraction on the test image in figure 7.10 are shown in figure 7.11. As can be seen in this figure, most of the regions have successfully been detected. The result is also compared with the output from clustering using the piecewise constant assumption.

We stress that these are just first results. Some of the regions obtained before merging (see figure 7.11 left) have striped structure in contrast to the case in the locally constant clustering (see figure 7.7, left). This suggests that the clustering strategy might not be optimal. It is well known that clustering of lines using the model

$$\begin{pmatrix}x & y & 1\end{pmatrix}^T \begin{pmatrix}\cos\phi & \sin\phi & -\rho\end{pmatrix} = 0 \qquad (7.37)$$

is preferable to using the model

$$y = kx + m \qquad (7.38)$$

since estimation of very steep slopes (large k) becomes unstable. See section 5.6.3 for an example of use for (7.37). This preference for a normal representation of lines suggests that we should view the grey-level image as a surface, and cluster surface normals instead.

The algorithm presented here is intended as a post processing to a depth-from-stereo vision algorithm. In depth maps of indoor scenes, a region with constant slope could correspond to a floor, a wall, or a door etc. A compact description of such features will hopefully be useful for autonomous indoor navigation.

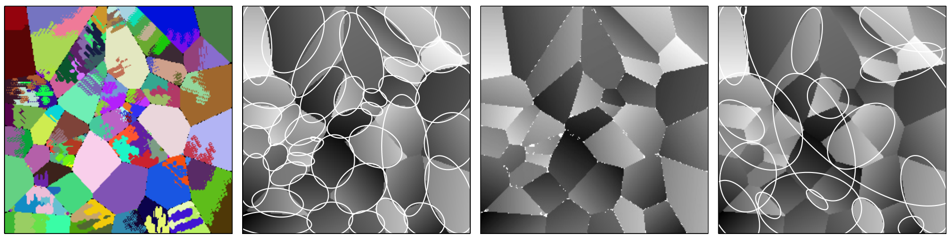


Figure 7.11: Blobs from piecewise linear assumption. Left to right: label image, detected blobs, reprojection of blob slopes to the corresponding label image regions, and blobs from piecewise constant assumption ($d_{\max} = 0.16$).

7.6 Concluding Remarks

In this chapter we have introduced a blob feature detection algorithm that works on vector fields of arbitrary dimension. The usefulness of the algorithm has been demonstrated on a wide baseline matching task. We have also extended the blob feature detection to cluster constant slopes instead of locally constant colour. The slope clustering has however not been fully evaluated, and some design choices, such as the chosen representation of the slopes might not be optimal. Specifically the option to cluster surface normals instead should be tested.

Chapter 8

Lines and Edges in Scale-Space

In this chapter we develop a representation of lines and edges in a scale hierarchy. By using three separate maps, the identity of lines and edges are kept separate. Further, the maps are made sparse by inhibition from coarser scales.

8.1 Background

Biological vision systems are capable of instance recognition in a manner that is vastly superior to current machine vision systems. Perceptual experiments [83, 12] are consistent with the idea that they accomplish this feat by remembering a sparse set of features for a few views of each object, and are able to interpolate between these (see discussion in chapter 2). What features biological systems use is currently not certain, but we have a few clues. It is a widely known fact that difference of Gaussians, and Gabor-type wavelets are useful models of the first two levels of processing in biological vision systems [5]. There is however no general agreement on how to proceed from these simple descriptors, toward more descriptive and more sparse features.

One way might be to detect various kinds of image symmetries such as circles, star-patterns, and divergences (such as corners) as was done in [65, 64]. Two very simple kinds of symmetries are lines and edges[1], and in this chapter we will see how extraction of lines and edges can be made more selective, in a manner that is locally continuous both in scale and spatially. An important difference between our approach and other line-and-edge representations, is that we keep different kinds of events separate instead of combining them into one compact feature map.

[1]To be strict, an edge is better described as an anti-symmetry.

8.1.1 Classical edge detection

Depth discontinuities in a scene often lead to intensity discontinuities in images of that scene. Thus, discontinuities such as lines and edges in an image tend to correspond to object boundaries. This fact been known and used for a long time in image processing. One early example that is still widely used are the Sobel edge filters [91]. Another common example is the Canny edge detector [14] that produces visually pleasing binary images. The goal of edge detecting algorithms in image processing is often to obtain useful input to segmentation algorithms [92], and for this purpose, the ideal step edge detection that the Canny edge detector performs is in general insufficient [85], since a step edge is just one of the events that can divide the areas of a physical scene. Since our goal is quite different (we want a sparse scene description that can be used in view-based object recognition), we will discuss conventional edge detection no further.

8.1.2 Phase-gating

Lines and edges correspond to specific local phases of the image signal. Line and edge features are thus related to the local phase feature. Our use of local phase originates from the idea of *phase-gating*, originally mentioned in a thesis by Haglund [55]. Phase-gating is a postulate that states that an estimate from an arbitrary operator is valid only in particular places, where the relevance of the estimate is high [55]. Haglund uses this idea to obtain an estimate of size, by only using the even quadrature component when estimating frequency, i.e. he only propagated frequency estimates near 0 and π phase.

8.1.3 Phase congruency

Mach bands are illusory peaks and valleys in illumination that humans, and other biological vision systems perceive near certain intensity profiles, such as ramp edges (see figure 8.1). Morrone et al. have observed that these illusory lines, as well as perception of actual lines and edges, occur at positions where the sum of Fourier components above a given threshold have a corresponding peak [78]. They also note that the sum of the squared output of even and odd symmetric filters always peaks at these positions, which they refer to as points of *phase congruency.*

This observation has lead to the invention of phase congruency feature detectors [68]. At points of phase congruency, the phase is spatially stable over scale. This is a desirable property for a robust feature. However, phase congruency does not tell us which kind of feature we have detected; is it a line, or an edge? For this reason, phase congruency detection has been augmented by Reisfeld to allow discrimination between line, and edge events [86]. Reisfeld has devised what he calls a *Constrained Phase Congruency Detector* (CPCT for short), that maps a pixel position and an orientation to an energy value, a scale, and a symmetry phase (0, $\pm\pi/2$ or π). This approach is however not quite suitable for us, since the map produced is of a semi discrete nature; each pixel is either of 0, $\pm\pi/2$ or π phase, and only belongs to the scale where the energy is maximal. The features we want should on the contrary allow a slight overlap in scale space, and have

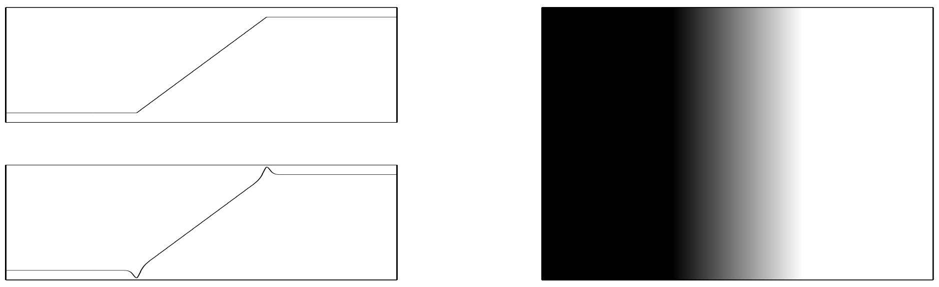


Figure 8.1: Mach bands near a ramp edge.
Top-left: Image intensity profile
Bottom-left: Perceived image intensity *Right: Image*

responses in a small spatial range near the characteristic phases.

8.2 Sparse feature maps in a scale hierarchy

Most feature generation procedures employ filtering in some form. The outputs from these filters tell quantitatively more about the filters used than the structures they were meant to detect. We can get rid of this excessive load of data, by allowing only certain phases of output from the filters to propagate further. These *characteristic phases* have the property that they give invariant structural information rather than all the phase components of a filter response.

We will now generate feature maps that describe image structure in a specific scale, and at a specific phase. The distance between the different scales is one octave (i.e. each map has half the centre frequency of the previous one.) The phases we detect are those near the characteristic phases 0, π, and $\pm\pi/2$. Thus, for each scale, we will have three resultant feature maps (see figure 8.2).

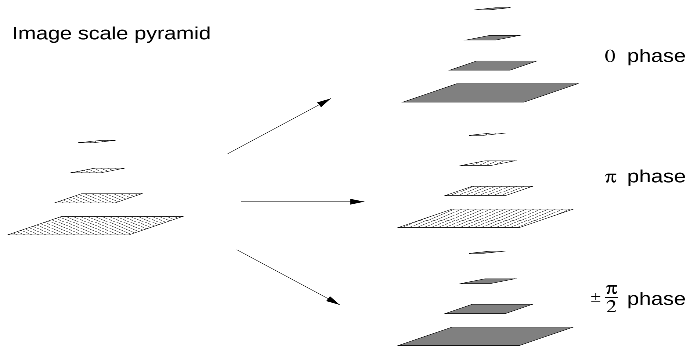


Figure 8.2: Scale hierarchies.

This approach touches the field of scale-space analysis pioneered by Witkin [106]. See [72] for a recent overview of scale space methods. Our approach to scale space analysis is somewhat similar to that of Reisfeld [86]. Reisfeld has defined what he calls a *Constrained Phase Congruency Transform* (CPCT), that maps a pixel position and an orientation to an energy value, a scale, and a symmetry phase (0, π, $\pm\pi/2$, or none). We will instead map each image position, at a given scale, to three complex numbers, one for each of the characteristic phases. The argument of the complex numbers indicates the dominant orientation of the local image region at the given scale, and the magnitude indicates the local signal energy when the phase is near the desired one. As we move away from the characteristic phase, the magnitude will go to zero. This representation will result in a number of complex valued images that are quite sparse, and thus suitable for pattern detection.

8.2.1 Phase from line and edge filters

For signals containing multiple frequencies, the phase is ambiguous, but we can always define the *local phase* of a signal, as the phase of the signal in a narrow frequency range.

The local phase can be computed from the ratio between a band-pass filter (even, denoted f_e) and its quadrature complement (odd, denoted f_o). These two filters are usually combined into a complex valued *quadrature filter*, $\boldsymbol{f} = f_e + \boldsymbol{i} f_o$ [48]. The real and imaginary parts of a quadrature filter correspond to line, and edge detecting filters respectively. The local phase can now be computed as the argument of the filter response, $\boldsymbol{q}(x) = (s * \boldsymbol{f})(x)$, or if we use the two real-valued filters separately, as the four quadrant inverse tangent; $\arctan(q_o(x), q_e(x))$.

To construct the quadrature pair, we start with a discretised lognormal filter function, defined in the frequency domain

$$R_i(\rho) = \begin{cases} e^{-\dfrac{\ln^2(\rho/\rho_i)}{\ln 2}} & \text{if } \rho > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{8.1}$$

The parameter ρ_i determines the peak of the lognorm function, and is called the centre frequency of the filter. We now construct the even and odd filters as the real and imaginary parts of an inverse discrete Fourier transform of this filter[2]

$$f_{e,i}(x) = \mathrm{Re}(\mathrm{IDFT}\{R_i(\rho)\}) \tag{8.2}$$

$$f_{o,i}(x) = \mathrm{Im}(\mathrm{IDFT}\{R_i(\rho)\})\,. \tag{8.3}$$

We write a filtering of a sampled signal, $s(x)$, with a discrete filter $f_k(x)$ as $q_k(x) = (s * f_k)(x)$, giving the response signal the same indices as the filter that produced it.

8.2.2 Characteristic phase

By *characteristic phase* we mean phases that are consistent over a range of scales, and thus characterise the local image region. For natural images this mainly

[2]Note that there are other ways to obtain spatial filters from frequency descriptions that, in many ways produce better filters [67].

happens at local magnitude peaks of the responses from the even and odd filters.[3] In other words, the characteristic phases are almost always one of 0, π, and $\pm\pi/2$. This motivates our restriction of the phase to these three cases.

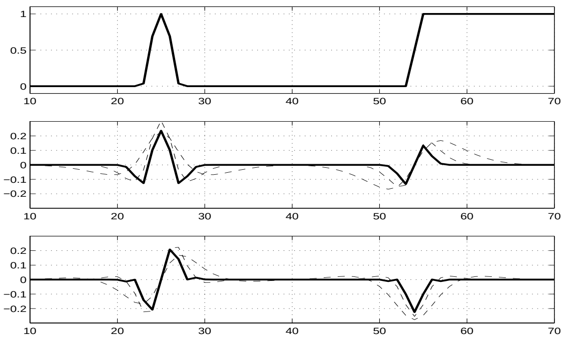


Figure 8.3: Line and edge filter responses in 1D.
Top: *A one-dimensional signal.*
Centre: *Line responses at* $\rho_i = \pi/2$ *(solid), and* $\pi/4$ *and* $\pi/8$ *(dashed)*
Bottom: *Edge responses at* $\rho_i = \pi/2$ *(solid), and* $\pi/4$ *and* $\pi/8$ *(dashed)*

Only some occurrences of these phases are consistent over scale though (see figure 8.3). First, we can note that band-pass filtering always causes ringings in the response. For isolated line and edge events this will mean one extra magnitude peak (with the opposite sign) at each side of the peak corresponding to the event. These extra peaks will move when we change frequency bands, in contrast to those peaks that correspond to the line and edge features. Second, we can note that each line event will produce one magnitude peak in the line response, and two peaks in the edge response. The peaks in the edge response, however, will also move when we change frequency bands. We can thus use stability over scale as a criterion to sort out the desired peaks.

8.2.3 Extracting characteristic phase in 1D

Starting from the line and edge filter responses at scale i: $q_{e,i}$, and $q_{o,i}$, we now define three *phase channels*

$$p_{1,i} = \max(0, q_{e,i}) \tag{8.4}$$
$$p_{2,i} = \max(0, -q_{e,i}) \tag{8.5}$$
$$p_{3,i} = \text{abs}(q_{o,i})\,. \tag{8.6}$$

That is, we let $p_{1,i}$ constitute the positive part of the line filter response, corresponding to 0 phase, $p_{2,i}$, the negative part, corresponding to π phase, and $p_{3,i}$ the magnitude of the edge filter response, corresponding to $\pm\pi/2$ phase.

[3]A peak in the even response will always correspond to a zero crossing in the odd response, and vice versa, due to the quadrature constraint.

Phase invariance over scale can be expressed by requiring that the phase at the next lower octave has the same sign

$$p_{1,i} = \max(0, q_{e,i} \cdot q_{e,i-1}/a_{i-1}) \cdot \max(0, \text{sign}(q_{e,i})) \tag{8.7}$$

$$p_{2,i} = \max(0, q_{e,i} \cdot q_{e,i-1}/a_{i-1}) \cdot \max(0, \text{sign}(-q_{e,i})) \tag{8.8}$$

$$p_{3,i} = \max(0, q_{o,i} \cdot q_{o,i-1}/a_{i-1}) \,. \tag{8.9}$$

The first max operation in the equations above will set the magnitude to zero whenever the filter at the next scale has a different sign. This operation will reduce the effect of the ringings from the filters. In order to keep the magnitude near the characteristic phases proportional to the local signal energy, we have normalised the product with the signal energy at the next lower octave $a_{i-1} = \sqrt{q_{e,i-1}^2 + q_{o,i-1}^2}$. The result of the operation in (8.7)-(8.9) can be viewed as a phase description at a scale in between the two used. These channels are compared with the original ones in figure 8.4.

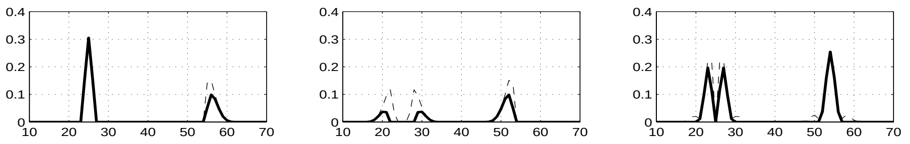


Figure 8.4: Consistent phase in 1D. ($\rho_i = \pi/4$)
$p_{1,i}$, $p_{2,i}$, $p_{3,i}$ *according to* (8.4)-(8.6) *(dashed), and* (8.7)-(8.9) *(solid)*

We will now further constrain the phase channels in such a way that only responses consistent over scale are kept. We do this by inhibiting the phase channels with the complementary response in the third lower octave

$$c_{1,i} = \max(0, p_{1,i} - \alpha \text{abs}(q_{o,i-2})) \tag{8.10}$$

$$c_{2,i} = \max(0, p_{2,i} - \alpha \text{abs}(q_{o,i-2})) \tag{8.11}$$

$$c_{3,i} = \max(0, p_{3,i} - \alpha \text{abs}(q_{e,i-2})) \,. \tag{8.12}$$

We have chosen an amount of inhibition $\alpha = 2$, and the base scale, $\rho_i = \pi/4$. With this value we successfully remove the edge responses at the line event, and at the same time keep the rate of change in the resultant signal below the Nyquist frequency. The resultant characteristic phase channels will have a magnitude corresponding to the energy at scale i, near the corresponding phase. These channels are compared with the original ones in figure 8.5.

As we can see, this operation manages to produce channels that indicate lines and edges without any unwanted extra responses. An important aspect of this operation is that it results in a gradual transition between the description of a signal as a line or an edge. If we continuously increase the thickness of a line, it will gradually turn into a bar that will be represented as two edges.[4] This phenomenon is illustrated in figure 8.6.

[4]Note that the fact that both the line, and the edge statements are low near the fourth event

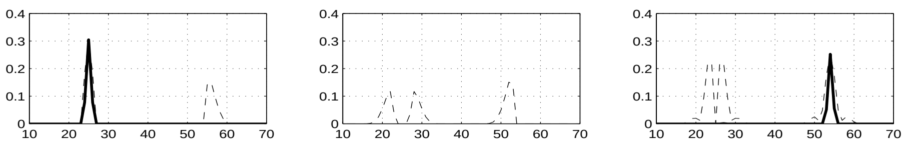


Figure 8.5: Phase channels in 1D. ($\rho_i = \pi/4$, $\alpha = 2$)
$p_{1,i}$, $p_{2,i}$, $p_{3,i}$ *according to* (8.4)-(8.6) *(dashed), and* (8.10)-(8.12) *(solid).*

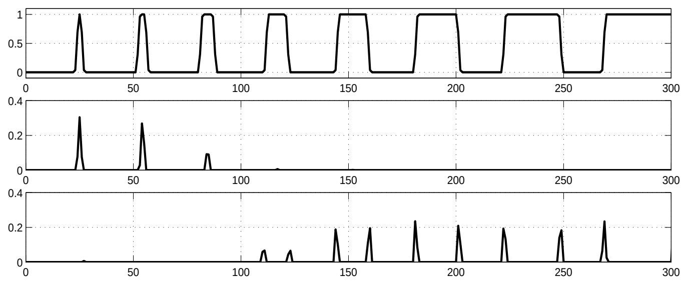


Figure 8.6: Transition between line and edge description. ($\rho_i = \pi/4$)
Top: Signal Centre: $c_{1,i}$ *phase channel Bottom:* $c_{3,i}$ *phase channel.*

8.2.4 Local orientation information

The filters we employ in 2D will be the extension of the lognorm filter function (8.1) to 2D [48]

$$\mathrm{F_{ki}}(\mathbf{u}) = \mathrm{R_i}(\rho)\mathrm{D_k}(\hat{\mathbf{u}}) \tag{8.13}$$

where

$$\mathrm{D_k}(\hat{\mathbf{u}}) = \begin{cases} (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}_k)^2 & \text{if } \mathbf{u} \cdot \hat{\mathbf{n}}_k > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{8.14}$$

We will use four filters, with directions $\hat{\mathbf{n}}_1 = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$, $\hat{\mathbf{n}}_2 = \begin{pmatrix} \sqrt{0.5} & \sqrt{0.5} \end{pmatrix}^T$, $\hat{\mathbf{n}}_3 = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$, and $\hat{\mathbf{n}}_4 = \begin{pmatrix} \sqrt{0.5} & -\sqrt{0.5} \end{pmatrix}^T$. These directions have angles that are uniformly distributed modulo π. Due to this, and the fact that the angular function decreases as $\cos^2 \varphi$, the sum of the filter-response magnitudes will be orientation invariant [48].

Just like in the 1D case, we will perform the filtering in the spatial domain

$$(f_{e,ki} * p_{ki})(\mathbf{x}) \approx \mathrm{Re}(\mathrm{IDFT}\{\mathrm{F_{ki}}(\mathbf{u})\}) \tag{8.15}$$

$$(f_{o,ki} * p_{ki})(\mathbf{x}) \approx \mathrm{Im}(\mathrm{IDFT}\{\mathrm{F_{ki}}(\mathbf{u})\})\,. \tag{8.16}$$

(positions 105 to 125) does not mean that this event will be lost. The final representation will also include other scales of filters, which will describe these events better.

Here we have used a filter optimisation technique [67] to factorise the lognorm quadrature filters into two approximately one-dimensional components. The filter $p_{ki}(\mathbf{x})$, is a smoothing filter in a direction orthogonal to $\hat{\mathbf{n}}_k$, while $f_{e,ki}(\mathbf{x})$, and $f_{o,ki}(\mathbf{x})$ constitute a 1D lognorm quadrature pair in the $\hat{\mathbf{n}}_k$ direction.

Using the responses from the four quadrature filters, we can construct a *local orientation* image. This is a complex valued image, in which the magnitude of each complex number indicates the signal energy when the neighbourhood is locally one-dimensional, and the argument of the numbers denote the local orientation, in the *double angle representation* [48]

$$\boldsymbol{z}(\mathbf{x}) = \sum_k a_{ki}(\hat{n}_{k1} + \boldsymbol{i}\hat{n}_{k2})^2 = a_{1i}(\mathbf{x}) - a_{3i}(\mathbf{x}) + \boldsymbol{i}(a_{2i}(\mathbf{x}) - a_{4i}(\mathbf{x})) \tag{8.17}$$

where $a_{ki}(\mathbf{x})$, the signal energy, is defined as $a_{ki} = \sqrt{q_{e,ki}^2 + q_{o,ki}^2}$.

8.2.5 Extracting characteristic phase in 2D

To illustrate characteristic phase in 2D, we need a new test pattern. We will use the 1D signal from figure 8.6, rotated around the origin (see figure 8.7).

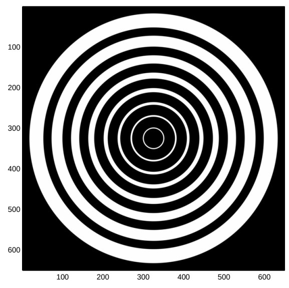


Figure 8.7: A 2D test pattern.

When extracting characteristic phases in 2D we will make use of the same observation as the local orientation representation does: Since visual stimuli can locally be approximated by a simple signal in the dominant orientation [48], we can define the *local phase* as the phase of the dominant signal component.

To deal with characteristic phases in the dominant signal direction, we first synthesise responses from a filter in a direction, $\hat{\mathbf{n}}_z$, compatible with the local orientation[5]

$$\hat{\mathbf{n}}_z = \begin{pmatrix}\mathrm{Re}(\sqrt{\boldsymbol{z}}) & \mathrm{Im}(\sqrt{\boldsymbol{z}})\end{pmatrix}^T . \tag{8.18}$$

The filters will be weighted according to the value of the scalar product between the filter direction, and this orientation compatible direction

[5]Since the local orientation, $\boldsymbol{z}$, is represented with a double angle argument, we could just as well have chosen the opposite direction. Which one of these we choose does not really matter, as long as we are consistent.

$$w_k = \hat{\mathbf{n}}_k^T \hat{\mathbf{n}}_z \,. \tag{8.19}$$

Thus, in each scale we synthesise one odd, and one even response projection as

$$q_{e,i} = \sum_k q_{e,i,k} \mathrm{abs}(w_k) \tag{8.20}$$

$$q_{o,i} = \sum_k q_{o,i,k} w_k \,. \tag{8.21}$$

This will change the sign of the odd responses when the directions differ more than π, but since the even filters are symmetric, they should always have a positive weight. In accordance with our findings in the 1D study (8.7)-(8.9), (8.10)-(8.12), we now compute three phase channels, $c_{1,i}$, $c_{2,i}$, and $c_{3,i}$, in each scale.

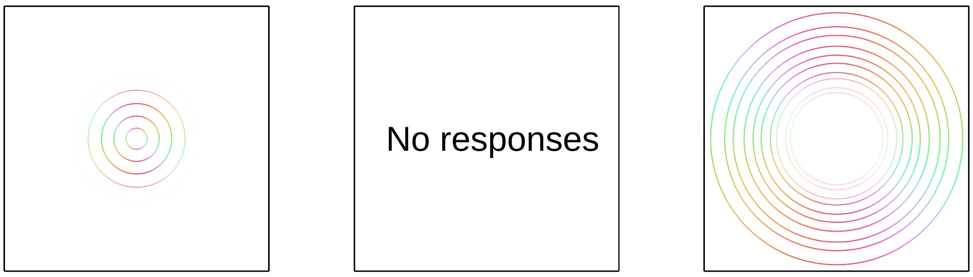


Figure 8.8: Characteristic phase channels in 2D. ($\rho_i = \pi/4$)
Left to right: Characteristic phase channels $c_{1,i}$*,* $c_{2,i}$*, and* $c_{3,i}$*, according to* (8.10)-(8.12) *(*$\alpha = 2$*). The colours indicate the locally dominant orientation.*

The characteristic phase channels are shown in figure 8.8.[6] As we can see, the channels exhibit a smooth transition from describing the white regions in the test pattern (see figure 8.7) as lines, and as two edges. Also note that the phase statements actually give the phase in the dominant orientation, and not in the filter directions, as was the case for CPCT [86].

8.2.6 Local orientation and characteristic phase

An orientation image can be be gated with a phase channel, $c_n(\mathbf{x})$, in the following way

$$\boldsymbol{z}_n(\mathbf{x}) = \begin{cases} 0 & \text{if } c_n(\mathbf{x}) = 0 \\ \dfrac{c_n(\mathbf{x}) \cdot \boldsymbol{z}(\mathbf{x})}{|\boldsymbol{z}(\mathbf{x})|} & \text{otherwise.} \end{cases} \tag{8.22}$$

We now do this for each of the characteristic phase statements $c_{1,i}(\mathbf{x})$, $c_{2,i}(\mathbf{x})$, and $c_{3,i}(\mathbf{x})$, in each scale. The result is shown in figure 8.9. The colours in the

[6]The magnitude of lines this thin can be difficult to reproduce in print. However, the magnitudes in this plot *should* vary in the same way as in figure 8.6.

figure indicate the locally dominant orientation, just like in figure 8.8. Notice for instance how the bridge near the centre of the image changes from being described by two edges, to being described as a bright line, as we move through scale space.

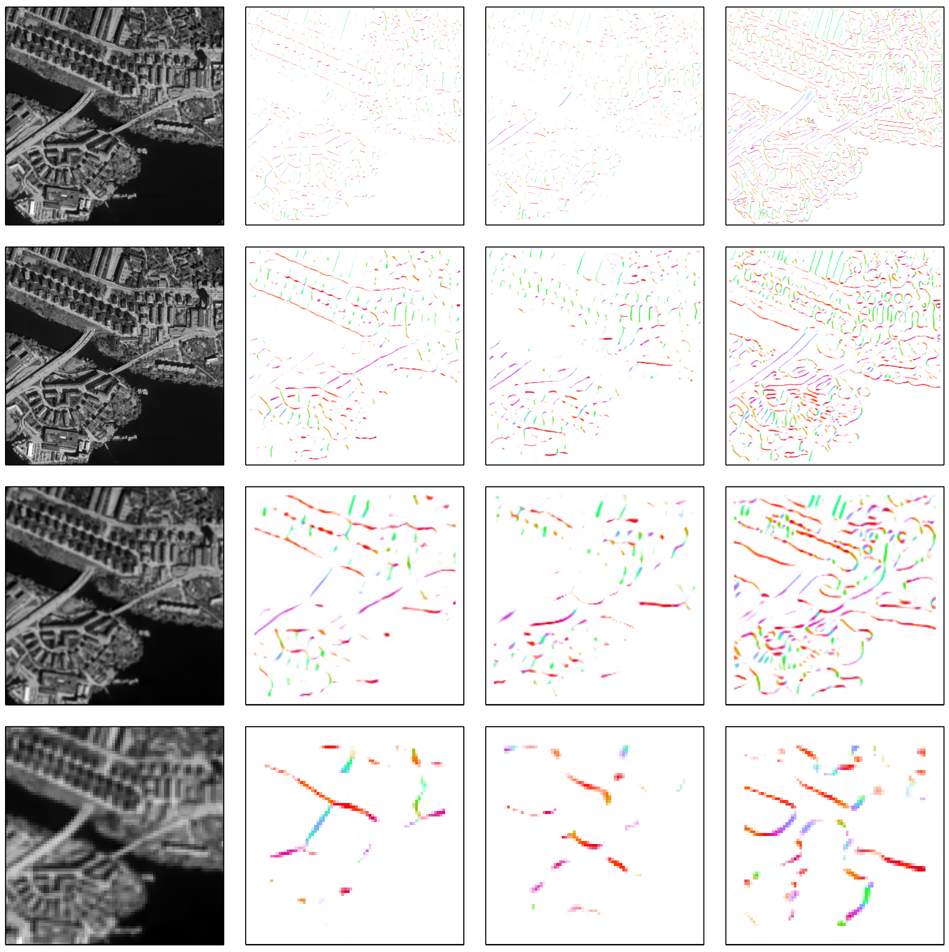


Figure 8.9: Sparse feature hierarchy. ($\rho_i = \pi/2, \pi/4, \pi/8, \pi/16$)

8.3 Concluding remarks

The strategy of this approach for low-level representation is to provide sparse, and reliable statements as much as possible, rather than to provide statements in all points.

Traditionally, the trend has been to produce compact, descriptive components as much as possible; mainly to reduce storage and computation. As the demands on performance are increasing it is no longer clear why components signifying different phenomena should be mixed. An edge is something separating two regions with different properties, and a line is something entirely different.

The use of sparse data representations in computation leads to a mild increase in data volume for separate representations, compared to combined representations.

Although the representation is given in discrete scales, this can be viewed as a conventional sampling, although in scale space, which allows interpolation between these discrete scales, with the usual restrictions imposed by the sampling theorem. The requirement of a good interpolation between scales determines the optimal relative bandwidths of filters to use.

Chapter 9

Associative Learning

This chapter introduces an associative network architecture using the channel representation. We describe the descriptive properties of the networks, and illustrate their behaviour using a set of experiments. We will also relate the associative networks to the techniques Radial Basis Function (RBF) networks, Support Vector Machines (SVM) and Fuzzy Control.

9.1 Architecture overview

In the proposed architecture, the choice of information representation is of fundamental importance. The architecture makes use of the channel information representation introduced in chapter 3. The channel representation implies a mapping of signals into a higher-dimensional space, in such a way that it introduces locality in the information representation with respect to all dimensions; geometric space as well as property space. The obtained locality gives two advantages:

- Nonlinear functions and combinations can be implemented using linear mappings
- Optimisation in learning converges much faster.

Figure 9.1 gives an intuitive illustration of how signals are represented as local fragments, which can be freely assembled to form an output. The system is moving along a state space trajectory. The state vector $\mathbf{x}$ consists of both internal and external system parameters. The response space is typically a subset of those parameters, e.g. orientation of an object, position of a camera sensor in navigation, or actions of a robot. Response channels and feature channels measure local aspects of the state space. The response channels and feature channels define response channel vectors $\mathbf{u}$ and feature vectors $\mathbf{a}$ respectively.

The processing mode of the architecture is association where the mapping of features a^h onto desired responses u^k is learned from a representative training set of observation pairs $\{\mathbf{a}_n, \mathbf{u}_n\}_{n=1}^N$, see figure 9.1(b). The feature vector $\mathbf{a}$ may contain some hundred thousand components, while the output vector $\mathbf{u}$ may contain some

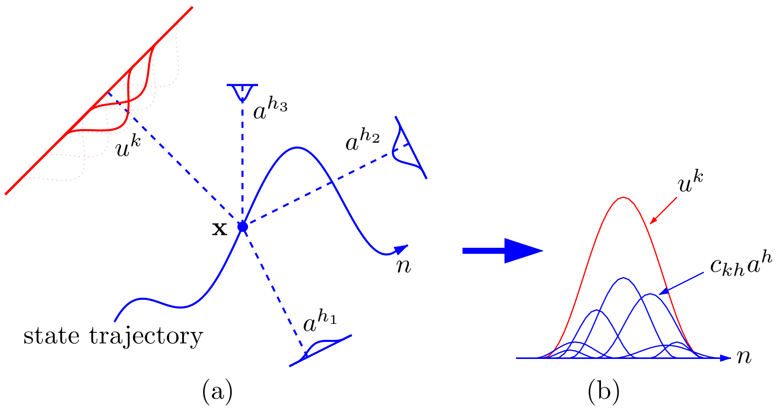


Figure 9.1: Architecture overview. (a) The system is moving along a state space trajectory. Response channels, u^k, and feature channels, a^h, measure different (local) aspects of the state vector $\mathbf{x}$. (b) The response channels and the feature channels define localised functions along the trajectory. A certain response channel is associated with some of the feature channels with appropriate weights c_{kh}. Figure borrowed from [51].

thousand components. For most features of interest, only limited parts of the domain will have non-zero contributions. This provides the basis for a sparse representation, which gives improved efficiency in storage and better performance in processing.

The model of the system is in the standard version a linear mapping from a feature vector $\mathbf{a}$ to a response vector $\mathbf{u}$ over a linkage matrix $\mathbf{C}$,

$$\mathbf{u} = \mathbf{C}\mathbf{a}\,. \tag{9.1}$$

In some training process, a set with N samples of output vectors $\mathbf{u}$ and corresponding feature vectors $\mathbf{a}$ are obtained. These form a response matrix $\mathbf{U} = \begin{pmatrix}\mathbf{u}_1 & \ldots & \mathbf{u}_N\end{pmatrix}$ and a feature matrix $\mathbf{A} = \begin{pmatrix}\mathbf{a}_1 & \ldots & \mathbf{a}_N\end{pmatrix}$. The training implies finding a solution matrix $\mathbf{C}$ to

$$\mathbf{U} = \mathbf{C}\mathbf{A}\,. \tag{9.2}$$

The linkage matrix is computed as a solution to a least squares problem with a monopolar constraint $\mathbf{C} \geq \mathbf{0}$. This constraint has a regularising effect, and in addition it gives a sparse linkage matrix. The monopolar representation together with locality, allows a fast optimisation, as it allows a parallel optimisation of a large number of loosely coupled system states.

We will compare the standard version (9.1) to models where the mapping is made directly to the response subset of the state parameters, i.e. typically what would be used in regular kernel machines. We will in these cases use a modified model with various normalisations of $\mathbf{a}$.

9.2 Representation of system output states

For a system acting in a continuous environment, we can define a *state space* $\mathcal{X} \subset \mathbb{R}^M$. A state vector, $\mathbf{x} \in \mathcal{X}$, completely characterises the current situation for the system, and $\mathcal{X}$ is thus the set of all situations possible for the system. The state space has two parts termed *internal* and *external.* Internal states describe the system itself, such as its position and its orientation. External states describe a subset of the total states of the environment, which are to be incorporated in the system's knowledge, such as the position, orientation and size of a certain object.

The estimation of external states requires a coupling to internal states, which can act as a known reference in the learning process. In general it is desirable to estimate either a state, or a *response* that changes the state, i.e. a system behaviour. For simplicity we will in this chapter assume that the desired response variables are components of the state vector $\mathbf{x}$.

We assume that the system is somehow induced to change its state, such that it covers the state space of interest for the learning process. For an agent acting in the physical world, the system state change has to take place in a continuous way (due to the inertia caused by limited power for displacement of a certain mass, see [49] for a more extensive discussion). It is thus reasonable to view the set of states $\{\mathbf{x}_n\}_1^N$ used in the learning process as a *system state trajectory.* We can express this system state trajectory as a matrix $\mathbf{X} = \begin{pmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_N\end{pmatrix}$.

9.2.1 Channel representation of the state space

The normal form of output for the structure is in channel representation. It is advantageous to represent the scalar state variables in a regular channel vector form, as this allows multiple outputs when the mapping from input to output is ambiguous, see section 3.2.1. The channel representation also forms the basis for learning of discontinuous phenomena, as will be demonstrated in section 9.6.

A *response channel* vector $\mathbf{u}^m$ is a channel representation of one of the components x^m of the state vector $\mathbf{x}$, see (3.1). The vector $\mathbf{u}^m$ is thus a non-ambiguous representation of position in a *response state space* $\mathcal{R}^m = \{x^m : \mathbf{x} \in \mathcal{X}\}$.

With this definition, a response channel will be non-zero only in a very limited region of the state space. The value of a channel can be viewed as a confidence in the hypothesis that the current state is near a particular prototype state. When a specific channel is non-zero it is said to be *active*, and the subspace where a specific channel is active is called the *active domain* of that channel. As the active domain is always much smaller than the inactive domain, an inactive channel will convey almost no information about position in state space. The small active domain is also what makes the representation sparse.

The response channel vectors $\mathbf{u}_n^m$ can be put into response channel matrices $\mathbf{U}^m = \begin{pmatrix}\mathbf{u}_1^m & \mathbf{u}_2^m & \ldots & \mathbf{u}_N^m\end{pmatrix}$. All such response channel matrices are stacked row-wise to form the response channel matrix $\mathbf{U}$. While $\mathbf{U}$ will have a much larger number of rows than the original state matrix $\mathbf{X}$ due to the increase of dimensionality in the representation, the sparsity of the representation will imply a moderate increase of the amount of data (typically a factor 3).

9.3 Channel representation of input features

It is assumed that the system can obtain at least partial knowledge about its state from a set of observed feature variables, $\{a^h\}$, forming a feature vector $\mathbf{a} = \left(a^1 \ a^2 \ \dots \ a^H\right)^T$. In order for an association or learning process to be meaningful, there has to be a sufficiently unique and repeatable correspondence between system states and observed features. One way to state this requirement is as follows: The *sensor space*, $\mathcal{A}$, of states that the feature channels can represent, should allow an unambiguous mapping $f : \mathcal{A} \rightarrow \mathcal{R}$. The situation where this requirement is violated is known in learning and robotics as *perceptual aliasing*, see e.g. [17].

A generative model for $\{a^h\}$, useful for systems analysis, can be expressed as localised, non-negative kernel functions $\mathrm{B}^h(\mathbf{x})$. These are functions of a weighted distance between the state vector $\mathbf{x}$ and a set of prototype states $\mathbf{x}^h \in \mathcal{X}$. We exemplify this with the $\cos^2$-kernel,

$$a^h = \mathrm{B}^h(\mathbf{x}) = \begin{cases} \cos^2(\mathrm{d}(\mathbf{x}, \mathbf{x}^h)) & \text{if } \mathrm{d}(\mathbf{x}, \mathbf{x}^h) \le \pi/2 \\ 0 & \text{otherwise.} \end{cases} \tag{9.3}$$

The used distance function is defined as

$$d(\mathbf{x}, \mathbf{x}^h) = \sqrt{(\mathbf{x} - \mathbf{x}^h)^T \mathbf{M}^h (\mathbf{x} - \mathbf{x}^h)}. \tag{9.4}$$

The matrix $\mathbf{M}^h$ is positive semidefinite, and describes the similarity measure for the distance function around state $\mathbf{x}^h$, allowing a scaling with different sensitivities with respect to different state variables. Equation (9.3) indicates that a^h will have a maximal value of 1 as $\mathbf{x} = \mathbf{x}^h$. It will go monotonically to zero as the weighted distance increases to $\pi/2$. Normally, neither $\mathbf{x}^h$, nor $\mathbf{M}^h$ are explicitly known, but emerge implicitly from the properties of the set of sensors used in the actual case. These are generally different from one sensor or filter to another, which motivates the notion of *channel representation*, as each channel has its specific identity, the identification of which is part of the learning process.

In general, there is no requirement for a regular arrangement of channels, be it on the input side or on the output side. The prescription of an orderly arrangement at the output comes from the need to interface the structure to the environment, e.g. to determine its performance. In such a case it will be desirable to map the response channel variables back into scalar variables in order to compare them with the reference, something which is greatly facilitated by a regular arrangement.

Similarly to the state variables, we denote the observation at sample point n by a vector $\mathbf{a}_n = \left(a_n^1 \ a_n^2 \ \dots \ a_n^H\right)^T$. These observation or feature vectors can be put into a feature matrix $\mathbf{A} = \left(\mathbf{a}_1 \ \mathbf{a}_2 \ \dots \ \mathbf{a}_N\right)$.

9.3.1 Feature generation

The feature vectors $\mathbf{a}$, input to the associative structure, may derive directly from the preprocessing parts of a computer vision system, representing local image properties such as orientation, curvature, colour, etc. Unless the features emerge as monopolar quantities, we will channel encode them. If the properties have a confidence measure, it is natural to weight the channel features with this, see discussion in chapter 3.

Often, combinations or functions comb($\mathbf{a}$) of a set of features $\mathbf{a}$, will be used as input to the associative structure. A common way to increase specificity in the percept space is to generate product pairs of the feature vector components, or a subset of them, i.e.

$$\mathrm{comb}(\mathbf{a}) = \mathrm{vec}(\mathbf{a}\mathbf{a}^T)\,. \tag{9.5}$$

The symbol vec signifies the trivial transformation of concatenating rows or columns of a matrix into a vector, see section 5.6.1. For simplicity of notation, we will express this as a substitution,

$$\mathbf{a} \leftarrow \mathrm{comb}(\mathbf{a})\,. \tag{9.6}$$

If we find a linear feature–response mapping in the training phase using the feature combination (9.5), it will correspond to a quadratic mapping from the original features to the responses. A network with this kind of feature expansion is called a *higher order network* [7].

The final vector $\mathbf{a}$, going into the associative structure will generally be considerably longer than the corresponding size of the sensor channel array. As we are dealing with sparse feature data, the increase of the data volume will be moderate.

9.4 System operation modes

The channel learning architecture can be run under two different operation modes, providing output in two different representations:

1. position encoding for *discrete event* mapping
2. magnitude encoding for *continuous function* mapping.

The first variety, discrete event mapping, is the mode which maximally exploits the advantage of the information representation, to allow implementation and learning of highly non-linear transfer functions, using a linear mapping.

The second variety is similar to more traditional function approximation methods.

9.4.1 Position encoding for discrete event mapping

In this mode, the structure is trained to map onto a set of channel representations of the system response state variables, as discussed in subsection 9.2.1. Thus each response will have a non-zero output only within limited regions of the definition range. The major issue is that a multi-dimensional, fragmented feature set is mapped onto a likewise fragmented, version of the system response state space. See figure 9.2 for an illustration.

There are a number of characteristics of the discrete event mode:

- Mapping is made to sets of response channels, whose response functions may be partially overlapping to allow the reconstruction of a continuous variable.

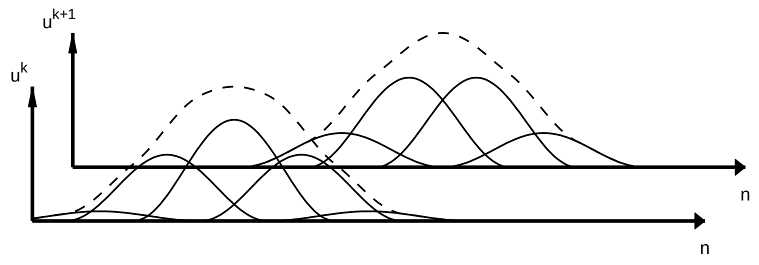


Figure 9.2: Illustration of discrete event mapping. Solid curves are weighted input feature functions $c_{kh}a^h(t)$ along the state space trajectory. Dashed curves are the responses $u^k(t) = \sum_h c_{kh}a^h(t)$.

- Output channels are assumed to assume some standard maximum value, say 1, but are expected to be zero most of the time, to allow a sparse representation.
- The system state is not given by the magnitude of a single output channel, but is given by the relation between outputs of adjacent channels.
- Relatively few feature functions, or sometimes only a single feature function, are expected to map onto a particular output channel.
- The channel representation of a signal allows a unified representation of signal value and of signal confidence, where the relation between channel values represents value, and the magnitude represents confidence. Since the discrete event mode implies that both the feature and response state vectors are in the channel representation, the confidence of the feature vector will be propagated to the response vector if the mapping is linear.

The properties just listed, allows the structure to be implemented as a purely linear mapping,

$$\mathbf{u} = \mathbf{C}\mathbf{a}\,. \tag{9.7}$$

9.4.2 Magnitude encoding for continuous function mapping

The continuous function mapping mode is used to generate the response state variables directly, rather than a set of channel functions for position decoding. The response state vector, $\mathbf{x}$, is approximated by a weighted sum of channel feature functions, see figure 9.3 for an illustration.

This mode corresponds to classical function approximation objectives. The mode is used for accurate representation of a scalar continuous function, which is often useful in control systems.

The approximation will be good if the feature functions are sufficiently local, and sufficiently dense. There are a number of characteristics for the continuous function mapping:

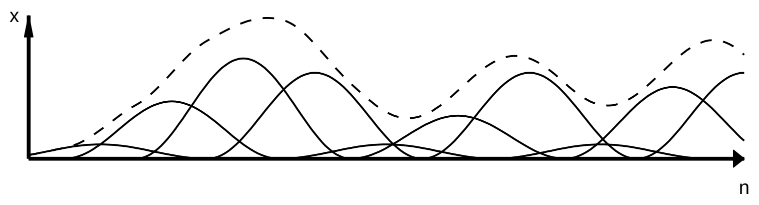


Figure 9.3: Illustration of continuous function mapping. Solid curves are weighted input feature functions $c_h a^h(t)$ along the state space trajectory. Dashed curve is the response $x(t) = \sum_h c_h a^h(t)$.

- It uses rather complete sets of feature functions, compared to the mapping onto a single response in discrete event mode. The structure can still handle local feature dropouts without adverse effects upon well behaved regions.
- Mapping is made onto continuous response variables, which may have a magnitude which varies over a large range.
- A high degree of accuracy in the mapping can be obtained if the feature vector is normalised, as stated below.

In this mode, however, it is not possible to represent both a state value $\mathbf{x}$, and a confidence measure r, unless it is done explicitly.

For a channel vector, the vector sum corresponds to the confidence, see section 3. As a first assumption we could thus assume that the feature vector sum corresponds to the confidence measure. Assuming a linear feature–response mapping, this will imply that the confidence is propagated to the response,

$$r\mathbf{x} = \mathbf{C}\mathbf{a}\,. \tag{9.8}$$

By dividing the feature vector $\mathbf{a}$ with r we can normalise with the amount of confidence, or certainty, in $\mathbf{a}$. This is related to the theory of normalized averaging, see e.g. [48].

If we use this model, we have additionally made the assumption that all features have the same confidence in each sample. To be slightly more flexible, we will instead assume a linear model for the confidence measure

$$r = \mathbf{w}^T\mathbf{a}\,, \tag{9.9}$$

where $\mathbf{w} > \mathbf{0}$ is a suitable weight vector. We now obtain the following response model for continuous function mode:

$$\mathbf{x} = \mathbf{C}\frac{1}{\mathbf{w}^T\mathbf{a}}\mathbf{a}\,. \tag{9.10}$$

Note that $\mathbf{w}^T\mathbf{a}$ is a weighted l_1-norm of $\mathbf{a}$, since $\mathbf{a}$ is non-negative. An unweighted l_1-norm, $\mathbf{w} = \mathbf{1}$, is often used in RBF networks and probabilistic mixture models, see [58, 77]. Other choices of weighting $\mathbf{w}$ will be discussed in section 9.5.2.

9.5 Associative structure

We will now turn to the problem of estimating the linkage matrix $\mathbf{C}$ in (9.10) and in (9.7). We take on a unified approach for the two system operation modes. The models can be summarised into

$$\mathbf{u} = \mathbf{C}\frac{1}{s(\mathbf{a})}\mathbf{a}\,, \tag{9.11}$$

where $s(\mathbf{a})$ is a normalisation function, and $\mathbf{u}$ denotes a scalar or a vector, representing either the explicit state variable/variables, or a channel representation thereof. In continuous function mode (9.10) $\mathbf{u} = \mathbf{x}$ and $s(\mathbf{a}) = \mathbf{w}^T\mathbf{a}$. In discrete event mode (9.7) $\mathbf{u}$ is a channel representation of $\mathbf{x}$ and $s(\mathbf{a}) \equiv 1$.

In the subsequent discussion, we will limit the scope to a supervised learning framework. Still, the structure can advantageously be used as a core in systems for other strategies of learning, such as reinforcement learning, with a proper embedding [96]. This discussion will assume *batch mode training.* This implies that there are N observation pairs of corresponding feature vectors $\mathbf{a}_n$ and state or response vectors $\mathbf{u}_n$. Let $\mathbf{A}$ and $\mathbf{U}$ denote the matrices containing all feature vector and response vector samples respectively, i.e.

$$\begin{cases} \mathbf{U} = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_N \\ | & | & & | \end{pmatrix} = \begin{pmatrix} - & \mathbf{u}^1 & - \\ - & \mathbf{u}^2 & - \\ & \vdots & \\ - & \mathbf{u}^K & - \end{pmatrix} \\ \\ \mathbf{A} = \begin{pmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_N \\ | & | & & | \end{pmatrix} = \begin{pmatrix} - & \mathbf{a}^1 & - \\ - & \mathbf{a}^2 & - \\ & \vdots & \\ - & \mathbf{a}^H & - \end{pmatrix} \end{cases} \tag{9.12}$$

For a set of observation samples collected in accordance with (9.12), the model in (9.11) can be expressed as

$$\mathbf{U} = \mathbf{C}\mathbf{A}\mathbf{D}_s\,, \tag{9.13}$$

where

$$\mathbf{D}_s = \mathrm{diag}^{-1}\left(s(\mathbf{a}_1) \quad s(\mathbf{a}_2) \quad \dots \quad s(\mathbf{a}_N)\right)\,. \tag{9.14}$$

The linkage matrix $\mathbf{C}$ is computed as a solution to a weighted least squares problem, with the constraint $\mathbf{C} \geq \mathbf{0}$. This constraint has a regularising effect on the mapping, and also ensures a sparse linkage matrix $\mathbf{C}$. For a more extensive discussion on the monopolar constraint, see the article [51].

9.5.1 Optimisation procedure

The procedure to optimise the associative networks is mainly the work of Granlund and Johansson [51]. It is described in this section for completeness. The linkage matrix $\mathbf{C}$ is computed as the solution to the constrained weighted least-squares problem

$$\min_{\mathbf{C}\geq\mathbf{0}} \; e(\mathbf{C})\,, \tag{9.15}$$

where

$$\begin{aligned} e(\mathbf{C}) &= \|\mathbf{U}-\mathbf{CAD}_s\|^2_{\mathbf{W}} \\ &= \operatorname{trace}(\mathbf{U}-\mathbf{CAD}_s)\mathbf{W}(\mathbf{U}-\mathbf{CAD}_s)^T\,. \end{aligned} \tag{9.16}$$

The weight matrix $\mathbf{W}$, which controls the relevance of each sample, is chosen as $\mathbf{W}=\mathbf{D}_s^{-1}$. The minimisation problem (9.15) does not generally have a unique solution, as it can be under-determined or over-determined.

The proposed solution to (9.15) is the fixed point of the sequence

$$\begin{cases} \mathbf{C}(0) &= \mathbf{0} \\ \mathbf{C}(i+1) &= \max\left(\mathbf{0}, \mathbf{C}(i)-\nabla e(\mathbf{C}(i))\mathbf{D}_f\right)\,, \end{cases} \tag{9.17}$$

where $\mathbf{D}_f$ is the positive definite diagonal matrix

$$\mathbf{D}_f = \operatorname{diag}(\mathbf{v})\operatorname{diag}^{-1}(\mathbf{AD}_s\mathbf{A}^T\mathbf{v}) \quad \text{for some } \mathbf{v}>\mathbf{0}\,. \tag{9.18}$$

Since $\mathbf{W}=\mathbf{D}_s^{-1}$ we have

$$\begin{aligned} \nabla e(\mathbf{C}) &= (\mathbf{CAD}_s-\mathbf{U})\mathbf{WD}_s\mathbf{A}^T \\ &= (\mathbf{CAD}_s-\mathbf{U})\mathbf{A}^T\,, \end{aligned} \tag{9.19}$$

and we rewrite sequence (9.17) as

$$\mathbf{C}(i+1) = \max\big(\mathbf{0}, \mathbf{C}(i)-(\mathbf{C}(i)\mathbf{AD}_s-\mathbf{U})\mathbf{A}^T\mathbf{D}_f\big)\,. \tag{9.20}$$

We can interpret $\mathbf{D}_s$ and $\mathbf{D}_f$ as normalisations in the sample and feature domain respectively, see section 9.5.2 for further details. We will consequently refer to $\mathbf{D}_s$ as *sample domain normalisation* and $\mathbf{D}_f$ as *feature domain normalisation.*

9.5.2 Normalisation modes

This normalisation can be put in either of the two representation domains, the *sample domain* or the *feature domain*, but with different effects upon convergence, accuracy, etc. For each choice of sample domain normalisation $\mathbf{D}_s$ there are non-unique choices of feature domain normalisations $\mathbf{D}_f$ such that sequence (9.20) converges to a solution of problem (9.15). $\mathbf{D}_f$ can for example be computed from (9.18). The choice of normalisation depends on the operation mode, i.e. continuous function mapping or discrete event mapping. There are some choices that are of particular interest. These are discussed below.

Discrete event mapping

Discrete event mode (9.7) corresponds to a sample domain normalisation matrix

$$\mathbf{D}_s = \mathbf{I}\,. \tag{9.21}$$

Choosing $\mathbf{v} = \mathbf{1} = (1\ 1\ \dots\ 1)^T$ in (9.18) gives

$$\mathbf{D}_f = \operatorname{diag}^{-1}(\mathbf{A}\mathbf{A}^T\mathbf{1}) = \begin{pmatrix} \frac{1}{\mathbf{a}^1\mathbf{m}_\mathrm{f}^T} & & \\ & \frac{1}{\mathbf{a}^2\mathbf{m}_\mathrm{f}^T} & \\ & & \ddots \end{pmatrix}, \tag{9.22}$$

where $\mathbf{m}_\mathrm{f} = \sum_h \mathbf{a}^h$ is proportional to the mean in the feature domain. As $\mathbf{D}_s$ does not contain any components of $\mathbf{A}$ there is no risk that it turns singular in domains of samples having all feature components zero.

This choice of normalisation will be referred to as *Normalisation entirely in the feature domain.*

Continuous function mapping

There are several ways to choose $\mathbf{w}$ in the continuous function model (9.10), depending on the assumptions of error models, and the resulting choice of confidence measure s. One approach is to assume that all training samples have the same confidence, i.e. $s \equiv 1$, and compute $\mathbf{C} \geq \mathbf{0}$ and $\mathbf{w} \geq \mathbf{0}$ such that

$$\begin{cases} \mathbf{1} & \approx & \mathbf{w}^T\mathbf{A} \\ \mathbf{X} & \approx & \mathbf{C}\mathbf{A}\,. \end{cases} \tag{9.23}$$

Sometimes it may be desirable to have an individual confidence measure for each training sample. Another approach is to design a suitable $\mathbf{w}$ and then compute $\mathbf{C}$ using the optimisation framework in section 9.5.1 with $s(\mathbf{a}) = \mathbf{w}^T\mathbf{a}$.

There are two specific designs of $\mathbf{w}$ that are worth emphasising. The channel representation implies that large feature channel magnitudes indicate a higher confidence than low values. We can consequently use the sum of the feature channels as a measure of confidence:

$$s(\mathbf{a}) = \mathbf{1}^T\mathbf{a} \quad \Rightarrow \quad \mathbf{x} = \mathbf{C}\frac{1}{\mathbf{1}^T\mathbf{a}}\mathbf{a}\,. \tag{9.24}$$

As mentioned before, this model is often used in RBF-networks and probabilistic mixture models, see [58, 77]. The corresponding sample domain normalisation matrix is

$$\mathbf{D}_s = \operatorname{diag}^{-1}(\mathbf{A}^T\mathbf{1}) = \begin{pmatrix} \frac{1}{\mathbf{a}_1^T\mathbf{1}} & & \\ & \frac{1}{\mathbf{a}_2^T\mathbf{1}} & \\ & & \ddots \end{pmatrix}, \tag{9.25}$$

and if we choose $\mathbf{v} = \mathbf{1}$ in (9.18) we get

$$\mathbf{D}_f = \operatorname{diag}^{-1}(\mathbf{A}\mathbf{1}) = \begin{pmatrix} \frac{1}{\mathbf{a}^1\mathbf{1}} & & \\ & \frac{1}{\mathbf{a}^2\mathbf{1}} & \\ & & \ddots \end{pmatrix}. \tag{9.26}$$

This choice of model will be referred to as *Mixed domain normalisation.*

It can also be argued that a feature element which is frequently active should have a higher confidence than a feature element which is rarely active. This can be included in the confidence measure by using a weighted sum of the features, where the weight is proportional to the mean in the sample domain:

$$s(\mathbf{a}) = \mathbf{m}_\mathrm{s}^T \mathbf{a} \quad \text{where} \quad \mathbf{m}_\mathrm{s} = \mathbf{A1} = \sum_n \mathbf{a}_n \,. \tag{9.27}$$

This corresponds to the sample domain normalisation matrix

$$\mathbf{D}_s = \mathrm{diag}^{-1}(\mathbf{A}^T\mathbf{A1}) = \begin{pmatrix} \frac{1}{\mathbf{m}_\mathrm{s}^T\mathbf{a}_1} & & \\ & \frac{1}{\mathbf{m}_\mathrm{s}^T\mathbf{a}_2} & \\ & & \ddots \end{pmatrix} , \tag{9.28}$$

and by using $\mathbf{v} = \mathbf{A1}$ in (9.18) we get

$$\mathbf{D}_f = \mathbf{I} \,. \tag{9.29}$$

This choice of model will be referred to as *Normalisation entirely in the sample domain.*

9.5.3 Sensitivity analysis for continuous function mode

We will now make some observations concerning the insensitivity to noise of the system, under the assumption of sample normalisation in continuous function mode. That is, a response state estimate $\hat{\mathbf{x}}_n$ is generated from a feature vector $\mathbf{a}$ according to model (9.10), i.e.

$$\hat{\mathbf{x}}_n = \mathbf{C}\frac{1}{\mathbf{w}^T\mathbf{a}_n}\mathbf{a}_n \,. \tag{9.30}$$

We observe that regardless of choice of normalisation vector $\mathbf{w}^T$, the response will be independent of any global scaling of the features, i.e.

$$\mathbf{C}\frac{1}{\mathbf{w}^T\mathbf{a}_n}\mathbf{a}_n = \mathbf{C}\frac{1}{\mathbf{w}^T\gamma\mathbf{a}_n}\gamma\mathbf{a}_n \,. \tag{9.31}$$

If multiplicative noise is applied, represented by a diagonal matrix $\mathbf{D}_\gamma$, we get

$$\hat{\mathbf{x}}_n = \mathbf{C}\frac{1}{\mathbf{w}^T\mathbf{D}_\gamma\mathbf{a}_n}\mathbf{D}_\gamma\mathbf{a}_n \,. \tag{9.32}$$

If the choice of weights in $\mathbf{C}$ and $\mathbf{w}$ is consistent, i.e. if the weights used to generate a response at a sample n were to obey the relation $\mathbf{C} = \hat{\mathbf{x}}_n\mathbf{w}^T$, the network is perfectly invariant to multiplicative noise. As we shall see in the experiments to follow, the normalisation comes close to this ideal for the entire sample set, provided that the response signal varies slowly. For such situations, the network suppresses multiplicative noise well.

Similarly, a sensitivity analysis can be made for discrete event mode. We will in this presentation only refer to the discussion in chapter 3 for the invariances available in the channel representation, and to the results from the experimental verification in the following section.

9.6 Experimental verification

We will in this section analyse the behaviour and the noise sensitivity of several variants of associative networks, both in continuous function mode and in discrete event mode. A generalisation of the common CMU twin spiral pattern [18] has been used, as this is often used to evaluate classification networks. We have chosen to make the pattern more difficult in order to show that the proposed learning machinery can represent both continuous function mappings (regression) and mappings to discrete classes (classification). The robustness is analysed with respect to three types of noise: additive, multiplicative, and impulse noise on the feature vector.

9.6.1 Experimental setup

In the experiments, a three dimensional state space $\mathcal{X} \subset \mathbb{R}^3$ is used. The sensor space $\mathcal{A} \subset \mathbb{R}^2$, and the response space $\mathcal{R} \subset \mathbb{R}$ are orthogonal projections of the state space. The network is trained to perform the mapping $f : \mathcal{A} \to \mathcal{R}$ which is depicted in figure 9.4. Note that this mapping can be seen as a surface of points $\mathbf{x} \in \mathbb{R}^3$, with $x_3 = f(x_1, x_2)$. The analytic expression for $f(x_1, x_2)$ is:

$$f(r,\varphi) = \begin{cases} f_s(r,\varphi) & \text{if} \quad \mathrm{mod}(\varphi + \sqrt{1000r}, 2\pi) < \pi \\ \mathrm{sign}(f_s(r,\varphi)) & \text{otherwise,} \end{cases} \tag{9.33}$$

$$\text{where} \quad f_s(r,\varphi) = (1/\sqrt{2} - r)\cos(\varphi + \sqrt{1000r})\,.$$

Variables $r = \sqrt{x_1^2 + x_2^2}$ and $\varphi = \tan^{-1}(x_1, x_2)$ are the polar coordinates in sensor space $\mathcal{A}$. As can be seen in the figure, the mapping contains both smooth parts (given by the cos function) and discontinuities (introduced by the sign function). The pattern is intended to demonstrate the following properties:

1. The ability to approximate piecewise continuous surfaces.
2. The ability to describe discontinuities (i.e. assignment into discrete classes).
3. The transition between interpolation and representation of a discontinuity.
4. The inherent approximation introduced by the sensor channels.

As sensor channels, a variant of the channels prescribed in expression (9.3) is used:

$$\mathrm{B}^h(\mathbf{x}) = \begin{cases} \cos^2(\omega d) & \text{if} \quad \omega d \leq \frac{\pi}{2} \\ 0 & \text{otherwise,} \end{cases} \tag{9.34}$$

$$\text{where} \quad d = \sqrt{(\mathbf{x} - \mathbf{x}^h)^T \mathbf{M} (\mathbf{x} - \mathbf{x}^h)}, \tag{9.35}$$

and $\mathbf{M} = \mathrm{diag}(1\ 1\ 0)$. In the experiments $H = 2000$ such sensors are used, with random positions $\{\mathbf{x}^h\}_1^H$ inside the box $([-0.5, 0.5], [-0.5, 0.5]) \subset \mathcal{A}$. The sensors

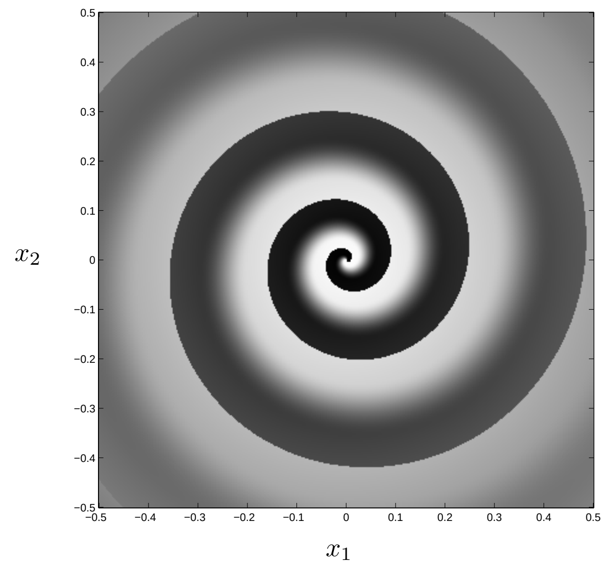


Figure 9.4: Desired response function. Black to White correspond to values of $x_3 \in [-1, 1]$.

have channel widths of $\omega = \pi/0.14$ giving each an active domain with radius 0.07. Thus, for each state $\mathbf{x}_n$, a feature vector $\mathbf{a}_n = \begin{pmatrix} \mathrm{B}^1(\mathbf{x}_n) & \mathrm{B}^2(\mathbf{x}_n) & \dots & \mathrm{B}^H(\mathbf{x}_n) \end{pmatrix}^T$ is obtained.

During training, random samples of the state vector $\mathbf{x}_n \in \mathcal{X}$ on the surface $f : \mathcal{A} \to \mathcal{R}$ are generated. These are used to obtain pairs $\{f_n,\ \mathbf{a}_n\}$ using (9.33) and (9.34). The training sets are stored in the matrices $\mathbf{f}$, and $\mathbf{A}$ respectively. The performance is then evaluated on a regular sampling grid. This has the advantage that performance can be visualised as an image. Since real valued positions $\mathbf{x} \in \mathcal{X}$ are used, the training and evaluation sets are disjoint.

The mean absolute error (MAE) between the network output and the ground truth (9.33), is used as a performance measure,

$$\varepsilon_{\text{MAE}} = \frac{1}{N} \sum_{n=1}^{N} |f(\mathbf{x}_n) - \mathbf{c}\mathbf{a}_n| \,, \tag{9.36}$$

or, for discrete event mode

$$\varepsilon_{\text{MAE}} = \frac{1}{N} \sum_{n=1}^{N} |f(\mathbf{x}_n) - \mathrm{dec}(\mathbf{C}\mathbf{a}_n)| \,. \tag{9.37}$$

The rationale for using this error measure is that it is roughly proportional to the number of misclassifications along the black-to-white boundary, in contrast to RMSE which is proportional to the number of misclassifications squared.

9.6.2 Associative network variants

We will demonstrate the behaviour of the following five variants of associative networks:

1. **Mixed domain normalisation bipolar network**
   This network uses the model
   $$\hat{f} = \frac{1}{\mathbf{1}^T\mathbf{a}}\mathbf{ca}\,.$$
   This model is often used in RBF-networks and probabilistic mixture models, see [58, 77]. This network is optimised according to
   $$\mathbf{c} = \arg\min_{\mathbf{c}} \|\mathbf{f} - \mathbf{cAD}_s\|^2 + \gamma\|\mathbf{c}\|^2\,. \tag{9.38}$$
   In the experiments, the explicit solution is used, i.e.
   $$\mathbf{c} = \mathbf{fAD}_s(\mathbf{AD}_s\mathbf{D}_s^T\mathbf{A}^T + \gamma\mathbf{I})^{-1}\,. \tag{9.39}$$
   Note that for larger systems, it is more efficient to replace (9.39) with a gradient descent method.

2. **Mixed domain normalisation monopolar network**
   Same as above, but with a monopolar constraint on $\mathbf{c}$, instead of the Tikhonov regularization used above.

3. **Sample domain normalisation monopolar network**
   This network uses the model
   $$\hat{f} = \frac{1}{\mathbf{m}_s^T\mathbf{a}}\mathbf{ca}\,,$$
   where $\mathbf{m}_s$ is computed from the training set sensor channels according to $\mathbf{m}_s = \mathbf{A1}$.

4. **Uniform sample confidence monopolar network**
   This network uses the model
   $$\hat{f} = \frac{1}{\mathbf{w}^T\mathbf{a}}\mathbf{ca}\,,$$
   where the mapping $\mathbf{w}$ is trained to produce the response 1 for all samples, see (9.23).

5. **Discrete event mode monopolar network**
   This network uses the model
   $$\hat{\mathbf{u}} = \mathbf{Ca} \quad \Leftrightarrow \quad \hat{f} = \mathrm{dec}(\mathbf{Ca})\,,$$
   with $K = 7$ channels. The responses should describe the interval $[-1, 1]$ so the decoding step involves a linear mapping, see (3.5).

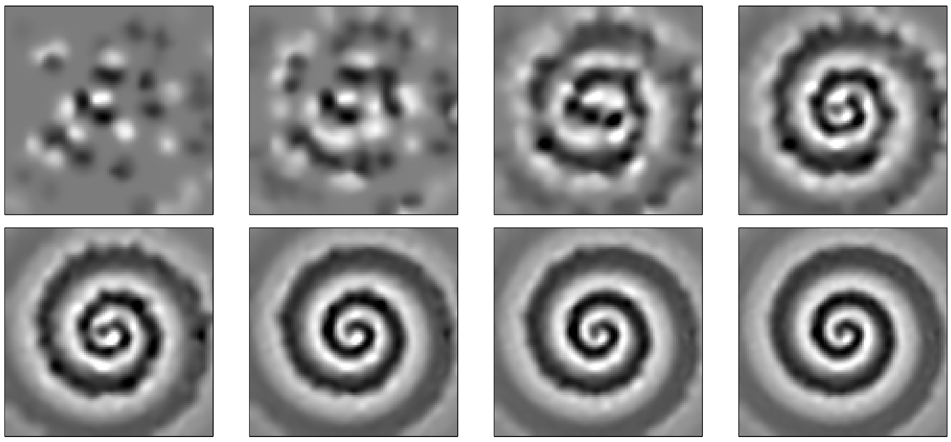


Figure 9.5: Performance of bipolar network (#1) under varied number of samples. Top left to bottom right: $N = 63$, 125, 250, 500, 1000, 2000, 4000, 8000.

9.6.3 Varied number of samples

As a demonstration of the generalisation abilities of the networks we will first vary the number of samples. The monopolar networks are optimised according to section 9.5, with 50 iterations. For the bipolar network we have used $\gamma = 0.005$. This value is chosen to give the same error on the training set as in network #2 using $N = 500$ samples.

The performance on the regular grid is demonstrated in figure 9.5 for the bipolar network (#1), and in figure 9.6 for the discrete event network (#5).

If we look at the centre of the spiral, we see that both networks fail to describe the fine details of the spiral, although #1 is doing slightly better. For the discrete event network, the failure is a direct consequence of the feature channel sizes. For the bipolar network it is a combined consequence of the size and density of the feature channels.

We can also observe that the discrete event network is significantly better at dealing with the discontinuities. This is also reflected in the error measures, see figure 9.7. For very low numbers of samples, when both networks clearly fail, the bipolar network is slightly better. We have also plotted the performance of the monopolar mappings in continuous function mode. As can be seen in the plot, these are all slightly worse off than the bipolar network. All three monopolar continuous function mode variants have similar performances on this setup. Differences appear mainly when the sample density becomes non-uniform (not shown here).

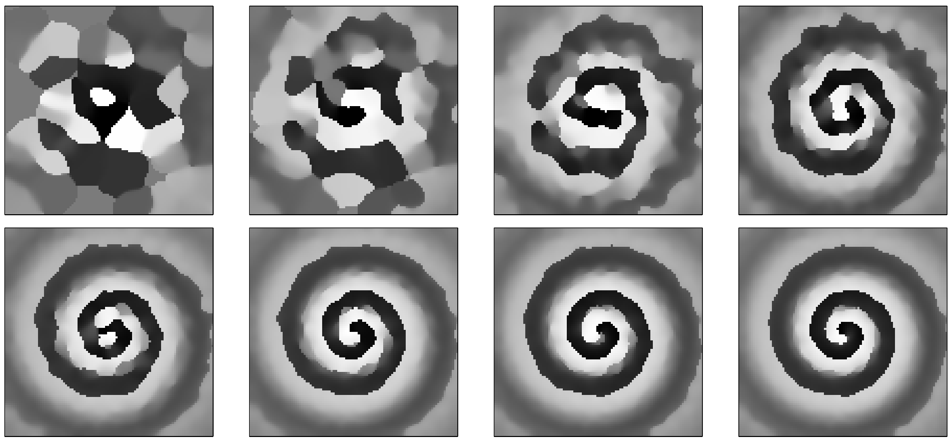


Figure 9.6: Performance of discrete event network (#5) under varied number of samples. Top left to bottom right: N = 63, 125, 250, 500, 1000, 2000, 4000, 8000.

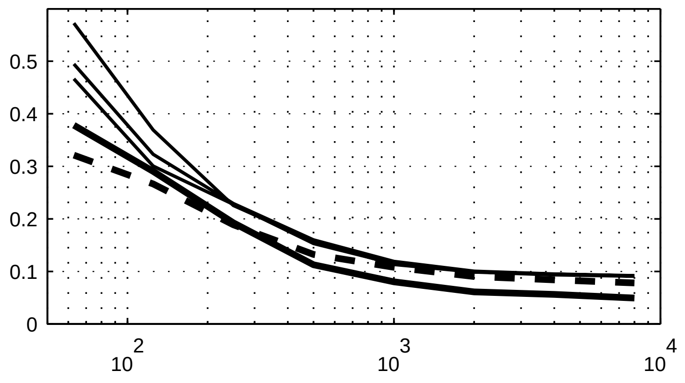


Figure 9.7: MAE under varied number of samples. Solid thick is #5, and dashed is #1. Solid thin are #2,#3, and #4. For low number of samples the variants are ordered #2, #3, #4 with #4 being the best one.

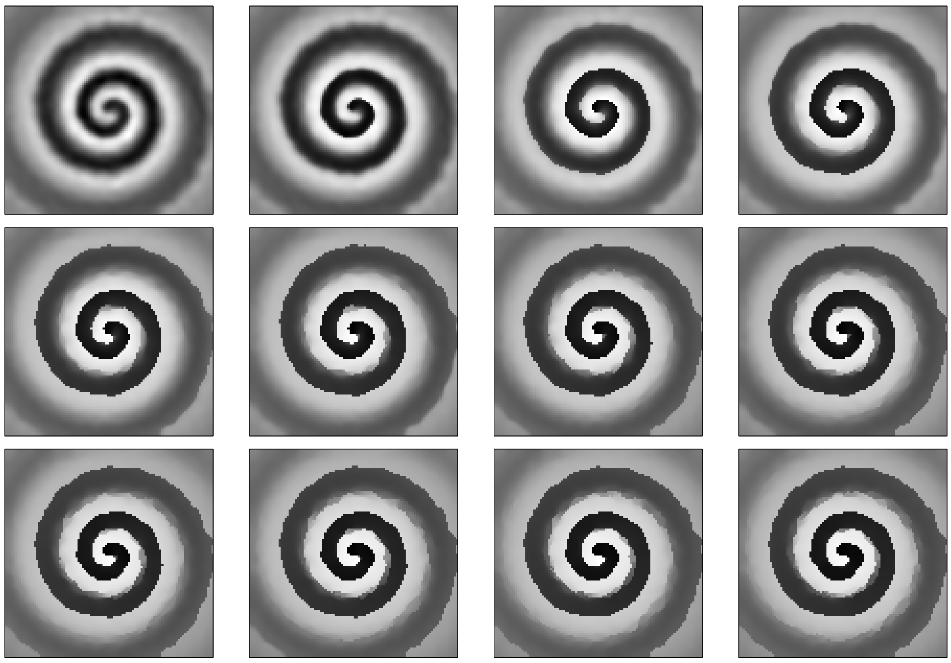


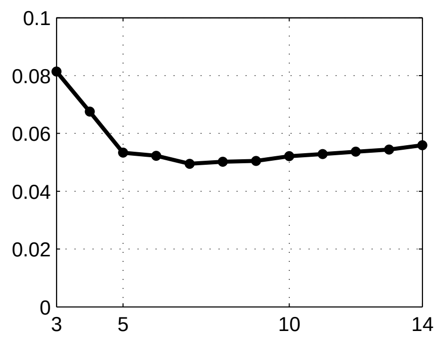
Figure 9.8: Performance of discrete event network (#5) under varied number of channels. Top left to bottom right: $K = 3$ to $K = 14$.

9.6.4 Varied number of channels

The relationship between the sizes of feature and response channels is important for the performance of the network. The distance between the channels also determines where the decision between interpolation and introduction of a discontinuity is made. We will now demonstrate these two effects by varying the number of channels in the range $[3 \dots 14]$, and keeping the number of samples high, $N = 8000$.

As can be seen in figure 9.8, a low number of channels gives a smooth response function. For $K = 3$ no discontinuity is introduced at all, since there is only one interval for the local reconstruction (see section 3.2.3). As the number of channels is increased, the number of discontinuities increases. Initially this is an advantage, but for a large number of channels, the response function becomes increasingly patchy (see figure 9.8). In practice, there is thus a trade-off between description of discontinuities, and patchiness. This trade-off is also evident if MAE is plotted against the number of channels, see figure 9.9 left.

In figure 9.9, right part, error curves for smaller numbers of samples have been plotted. It can be seen that, for a given number of samples, the optimal choice of channels varies. Better performance is obtained for a small number of channels, when fewer samples are used. The standard way to interpret this result is that a high number of response channels allows a more complex model, which requires



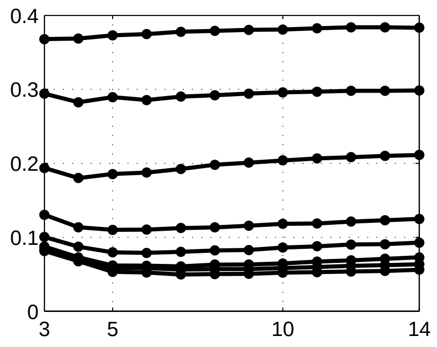


Figure 9.9: MAE under varied number of channels. Left MAE for $N = 8000$. Right MAE for $N = 63$, 125, 250, 500, 1000, 2000, 4000, and 8000.

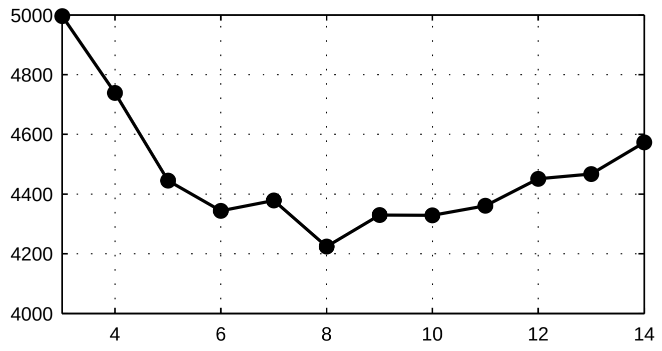


Figure 9.10: Number of non-zero coefficients under varied number of channels. Compare this with 2000 non-zero coefficients for the continuous function networks.

more samples.

If we plot the number of non-zero coefficients in the linkage matrix **C**, we also see that there is an optimal number of channels, see figure 9.10. Note that although the size of **C** is between 3 and 14 times larger than in continuous function mode, the number of links only increases by a factor 2.1 to 2.5.

9.6.5 Noise sensitivity

We will now demonstrate the performance of the associative networks when the feature set is noisy. We will use the following noise models:

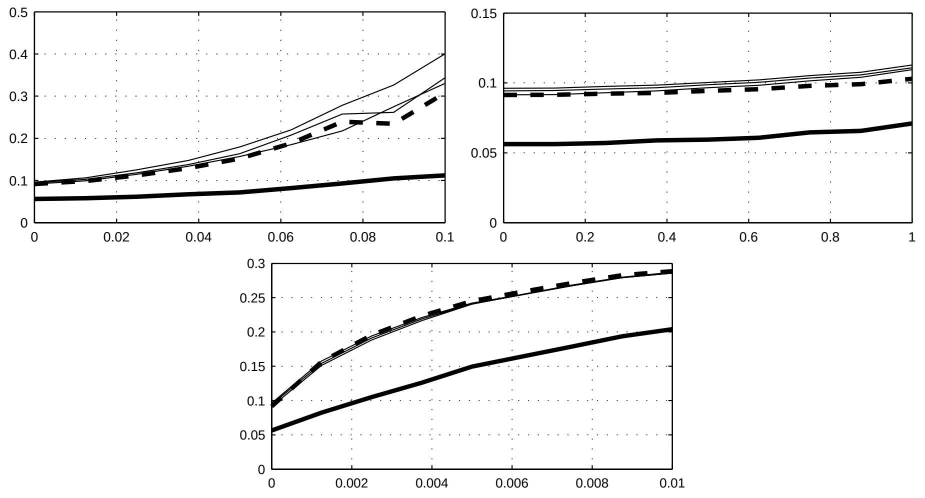


Figure 9.11: Noise sensitivity. Top left: additive noise, top right: multiplicative noise, bottom: impulse noise. Solid thick is #5, and dashed is #1. Solid thin are #2,#3, and #4.

1. **Additive noise**: A random value is added to each feature value, i.e.

$$\mathbf{a}_n^* = \mathbf{a}_n + \boldsymbol{\eta}\,,$$

with $\eta^k \in \text{rect}[-p, p]$, and the parameter p is varied in the range $[0, 0.1]$.

2. **Multiplicative noise**: Each feature value is multiplied with a random value, i.e.

$$\mathbf{a}_n^* = \mathbf{D}_\eta \mathbf{a}_n\,,$$

where $\mathbf{D}_\eta$ is a diagonal matrix with $(\mathbf{D}_\eta)_{kk} = \eta^k \in \text{rect}[1-p, 1+p]$, and the parameter p is varied in the range $[0, 1]$.

3. **Impulse noise**: A fraction of the features is set to 1, i.e.

$$a_n^{k,*} = \begin{cases} 1 & \text{if } f_r < p \text{ where } f_r \in \text{rect}(0,1) \\ a_n^k & \text{otherwise,} \end{cases}$$

and the parameter p is varied in the range $[0, 0.01]$.

The results of the experiments are shown in figure 9.11. We have consistently used $N = 4000$ samples for evaluation, and corrupted them with noise according to the discussion above. In order to make the amount of regularisation comparable we have optimised the γ parameter for network #1 to give the same error on the training set as network #2 at $N = 4000$ samples. This gave $\gamma = 0.08$.

As can be seen from the additive noise experiment, network #5 has a different slope for its dependence upon noise level. The other networks are comparable, and differences are mainly due to how well the networks are able to represent the pattern in the first place (see section 9.6.3). For the multiplicative noise case, we see that the slope is similar for all networks. Thus we can conclude that the multiplicative noise behaviour is comparable for all tested networks. For the impulse noise case we can see that for small amounts of noise, network #5 has a less steep slope than the others. For larger amounts of noise however, all networks seem to behave in a similar manner.

The purpose of these experiments has been to demonstrate the abilities of the associative networks to generalise, and to cope with various kinds of sensor noise. Several experiments using image features as inputs have been made, but have to be excluded from this presentation. For details of such experiments, the reader is directed to [53, 34, 80].

9.7 Other local model techniques

We will now have a look at three classes of techniques similar to the associative networks presented in this chapter. The descriptions of the techniques, Radial Basis Function (RBF) networks, Support Vector Machines (SVM), and adaptive fuzzy control, are not meant to be exhaustive, the purpose of the presentation is merely to describe the similarities and differences between them and the associative networks.

9.7.1 Radial Basis Function networks

The fact that an increased input dimensionality with localised inputs simplifies learning problems has also been exploited in the field of Radial Basis Function (RBF) networks [77, 58]. RBF networks have a hidden layer with localised Gaussian models, and an output layer which is linear. In effect this means that RBF networks learn a hidden representation which works like a channel encoding. The advantage with this approach is that the locations, and sizes of the channels (or RBFs) adapt to the data. The obvious disadvantage compared to using a fixed set of localised inputs is of course longer training time, since the network has two layers that have to be learned.

Typically the RBF positions are found using a clustering scheme such as K-means [77], or, if the number of traing data is low, one RBF is centered around each traning data. Related to RBF networks are hierarchies of local Gaussian models. Such networks have been investigated by for instance Landelius in [69]. His setup allows new models to be added where needed, and unused models to be removed. Compared to the associative networks presented in this chapter, we also note that the response from an RBF network is a continuous function, and not a channel representation. This means that RBF networks cannot properly deal with multiple hypotheses.

9.7.2 Support Vector Machines

Support Vector Machines (SVM) is another kernel technique that avoids mapping into a high-dimensional space alltogether. For a SVM it is required that the used kernel is positive definite. For such cases, *Mercers theorem* states that the kernel function is equivalent to an inner product in a high-dimensional space [58].

Obvious differences between the associative networks and SVM are that a SVM has a low dimensional feature space, and maps either to a binary variable (classification SVM), or to a continuous function (regression SVM). An associative network on the other hand uses a high-dimensional, sparse representation of the feature space, and maps to a set of response channels. Since SVMs do not use responses in the channel representation, they are unable to deal with multiple hypotheses.

9.7.3 Adaptive fuzzy control

Adaptive fuzzy control is a technique for learning locally linear functional relationships, see e.g. [84] for an overview. In fuzzy control a set of local fuzzy inference rules between measurements, and desired outputs are established. These are often in a form suitable for linguistic communication, for instance: `IF` temperature(warm) `THEN` power(reduce). The linguistic states ("warm" and "reduce" in our example) are defined by localised *membership functions*, corresponding to the kernels in the channel representation. Each input variable is *fuzzified* into a set of membership degrees, which are in the range $[0, 1.0]$. Groups of one membership function per input variable are connected to an output membership function in a fuzzy inference rule. Each fuzzy inference rule only fires to a certain degree, which is determined by the amount of input activations. The result of the fuzzy inference is a weighted linear combination of the output membership functions, which can be used to decode a response in a *defuzzification* step, which is typically a global moment (centroid) computation. The `IF-THEN` inference rules can be learned by a neural network, see for instance [76]. Typically the learning adjusts the shape and positions of the membership functions, while the actual set of `IF-THEN` rules stays fixed. Thus a fuzzy-inference can be thought of as an associative network with adaptive feature and response channels, and a static, binary linkage matrix $\mathbf{C}$.

There are several differences between the associative networks, and fuzzy control. In fuzzy control the implicit assumption is that there is only one value per feature dimension activating the membership functions at the input side. As shown in this chapter, this is not the case in associative learning using the channel representation. Furthermore fuzzy control only allows one response, since the defuzzification is a global operation. In contrast, representation of multiple values is an important aspect of the channel representation, see section 3.2.1.

9.8 Concluding remarks

In this chapter we have demonstrated that the channel learning architecture running in discrete event mode is able to describe simultaneously continuous and transiential phenomena, while still being better than or as good as a linear network at suppressing noise. An increase in the number of response channels does not cause an explosion in the number of used links. Rather, it remains fairly stable at approximately twice the number of links required for a continuous function mapping. This is a direct consequence of the monopolar constraint.

The training procedure shows a fast convergence. In the experiments described, a mere 50 iterations have been required. The fast convergence is due to the monopolar constraint, locality of the features and responses, and the choice of feature domain normalisation.

The learning architecture using channel information also deals properly with the perceptual aliasing problem, that is, it does not attempt to merge or average conflicting statements, but rather passes them on to the next processing level. This allows a second processing stage to resolve the perceptual aliasing, using additional information not available at the lower level.

The ability of the architecture to handle a large number of models in separate or loosely coupled domains of the state space, promises systems with a combination of the continuous mapping of control systems with the state complexity we have become familiar with from digital systems. Such systems can be used for the implementation of extremely complex, contextually controlled mapping model structures. One such application is for view based object recognition in computer vision [53].

Chapter 10

An Autonomous Reactive System

This chapter describes how a world model for successive recognition can be learned using associative learning. The learned world model consists of a linear mapping that successively updates a high-dimensional system state, using performed actions and observed percepts. The actions of the system are learned by rewarding actions that are good at resolving state ambiguities. As a demonstration, the system is used to resolve the localisation problem in a labyrinth.

10.1 Introduction

During the eighties a class of robotic systems known as *reactive robotic systems* became popular. The introduction of system designs such as the *subsumption architecture* [11] caused a small revolution due to their remarkably short response times. Reactive systems are able to act quickly since the actions they perform are computed as a direct function of the sensor readings, or *percepts*, at a given time instant. This design principle works surprisingly well in many situations despite its simplicity. However, a purely reactive design is sensitive to a fundamental problem known as *perceptual aliasing*, see e.g. [17].

Perceptual aliasing is the situation when the percepts are identical in two situations when the system should perform different actions. There are two main solutions to this problem:

- The first is to add more sensors to the system such that the two situations can be distinguished.
- The second is to give the system an internal state. This state is estimated such that it is different in the two situations, and can thus be used to guide the actions.

This chapter will deal with the latter solution, which further on will be called *successive state estimation.* We note here that the introduced state can be tailor-made to resolve the perceptual aliasing.

Successive state estimation is called *recursive parameter estimation* in signal processing, and *on-line filtering* in statistics [101]. Successive recognition could potentially be useful to computer vision systems that are to navigate in a known environment using visual input, such as the autonomous helicopter in the WITAS project [52].

10.1.1 System outline

Successive state estimation is an important component of an active perception system. The system design to be described is illustrated in figure 10.1. The state estimation, which is the main topic of this chapter, is performed by the *state transition* and *state narrowing* boxes.

The state transition box updates the state using information about which action the system has taken, and the state narrowing box successively resolves ambiguities in the state by only keeping states that are consistent with the observed stimulus.

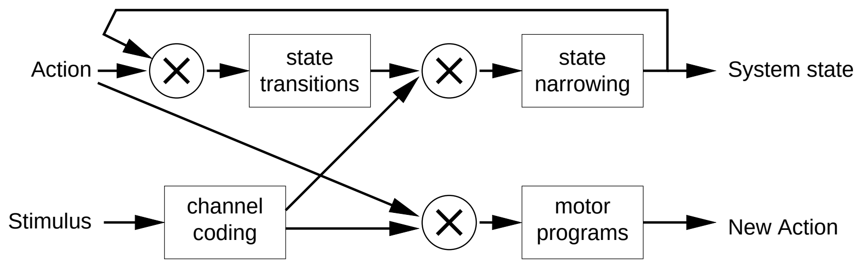


Figure 10.1: System outline.

The system consistently uses the *channel representation* (see chapter 3) to represent states and actions. This implies that information is stored in channel vectors of which most elements are zero. Each channel is non-negative, and its magnitude signifies the relevance of a specific hypothesis (such as a specific system state in our case), and thus a zero value represents “no information”. This information representation has the advantage that it enables very fast associative learning methods to be employed [50], and improves product sum matching [34].

The *channel coding* box in figure 10.1 converts the percepts into a channel representation. Finally, the *motor program* box is the subsystem that generates the actions of the system. The complexity of this box is at present kept at a minimum.

10.2 Example environment

To demonstrate the principle of successive state estimation, we will apply it on the problem shown in figure 10.2. The arrow in the figure symbolises an autonomous agent that is supposed to successively estimate its position and gaze direction by performing actions and observing how the percepts change. This is known as the *robot localisation problem* [101]. The labyrinth is a known environment, but the initial location of the agent is unknown, and thus the problem consists of learning (or designing) a world model that is useful for successive recognition.

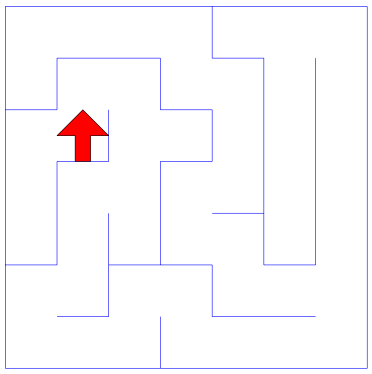


Figure 10.2: Illustration of the labyrinth navigation problem.

The stimulus constitutes a three element binary vector, which tells whether there are walls to the left, in front, or to the right of the agent. For the situation in the figure, this vector will look like this:

$$\mathbf{m} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T .$$

This stimulus is converted to percept channels in one of two ways

$$\begin{aligned} \mathbf{p}_1 &= \begin{pmatrix} m_1 & m_2 & m_3 & 1-m_1 & 1-m_2 & 1-m_3 \end{pmatrix}^T \quad \text{or} \\ \mathbf{p}_2 &= \begin{pmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \end{pmatrix}^T , \end{aligned} \tag{10.1}$$

where

$$p_h = \begin{cases} 1 & \text{if} \quad \mathbf{m} = \mathbf{m}^h \\ 0 & \text{otherwise,} \end{cases}$$

and $\{\mathbf{m}^h\}_1^8$ is the set of all possible stimuli. This expansion is needed since we want to train an associative network [50] to perform the state transitions, and since the network only has non-negative coefficients, we must have a non-zero input vector whenever we want a response.

The two variants $\mathbf{p}_1$ and $\mathbf{p}_2$ will be called *semi-local*, and *local* percepts respectively. For the semi-local percepts, correlation serves as a similarity measure, or *metric*, but for the local percepts we have no metric—the correlation is either 1 or 0.

The system has three possible actions $\mathbf{a}^1 =$ `TURN_LEFT`, $\mathbf{a}^2 =$ `TURN_RIGHT`, and $\mathbf{a}^3 =$ `MOVE_FORWARD`. These are also represented as a three element binary vector, with only one non-zero element at a time. E.g. `TURN_RIGHT` is represented as

$$\mathbf{a}^2 = \begin{pmatrix}0 & 1 & 0\end{pmatrix}^T .$$

Each action will either turn the agent 90° clockwise or anti clockwise, or move it forward to the next grid location (unless there is a wall in the way).

As noted in section 10.1, the purpose of the system state is to resolve perceptual aliasing. For the current problem this is guaranteed by letting the state describe both agent location and absolute orientation. This gives us the number of states as

$$N_s = \text{rows} \times \text{cols} \times \text{orientations} . \tag{10.2}$$

For the labyrinth in figure 10.2 this means $7 \times 7 \times 4 = 196$ different states.

10.3 Learning successive recognition

If the state is in a local representation, that is, each component of the state vector represents a local interval in state space, successive recognition can be obtained by a linear mapping. For the environment described in section 10.2, we will thus use a state vector with N_s components.

The linear mapping will recursively estimate the state, $\mathbf{s}$, from an earlier state, the performed action, $\mathbf{a}$, and an observed percept $\mathbf{p}$. I.e.

$$\mathbf{s}(t+1) = \mathbf{C}\left[\mathbf{s}(t) \otimes \mathbf{a}(t) \otimes \mathbf{p}(t+1)\right] \tag{10.3}$$

where $\otimes$ is the *Kronecker product*, which generates a vector containing all product pairs of the elements in the involved vectors (see section 5.6.1). The sought linear mapping $\mathbf{C}$ is thus of dimension $N_s \times N_s N_a N_p$ where N_a and N_p are the sizes of the action and percept vectors respectively.

In order to learn the mapping we supply examples of $\mathbf{s}$, $\mathbf{a}$, and $\mathbf{p}$ for all possible state transitions. This gives us a total of $N_s N_a$ samples. The coefficients of the mapping $\mathbf{C}$ are found using a least squares optimisation with non-negative constraint

$$\begin{aligned} &\arg \min_{c_{ij}>0} ||\mathbf{u} - \mathbf{C}\mathbf{f}||^2 \quad \text{where} \\ \mathbf{u} &= \mathbf{s}(t+1) \\ \mathbf{f} &= \mathbf{s}(t) \otimes \mathbf{a}(t) \otimes \mathbf{p}(t+1) . \end{aligned}$$

For details of the actual optimisation see section 9.5.1.

10.3.1 Notes on the state mapping

The first thing to note about usage of the mapping, $\mathbf{C}$, is that the state vector obtained by the mapping has to be normalised at each time step, i.e.

$$\begin{cases} \tilde{\mathbf{s}}(t+1) &= \mathbf{C}\left[\mathbf{s}(t) \otimes \mathbf{a}(t) \otimes \mathbf{p}(t+1)\right] \\ \mathbf{s}(t+1) &= \dfrac{\tilde{\mathbf{s}}(t+1)}{\sum_k \tilde{s}_k(t+1)}\,. \end{cases} \tag{10.4}$$

In the environment described in section 10.2, we obtain exactly the same behaviour when we use two separate maps:

$$\begin{cases} \mathbf{s}^*(t+1) &= \mathbf{C}_1\left[\mathbf{s}(t) \otimes \mathbf{a}(t)\right] \\ \tilde{\mathbf{s}}(t+1) &= \mathbf{C}_2\left[\mathbf{s}^*(t+1) \otimes \mathbf{p}(t+1)\right]\,. \end{cases} \tag{10.5}$$

These two maps correspond to the boxes *state transition* and *state narrowing* in figure 10.1. An interesting parallel to *on-line filtering* algorithms in statistics is that $\mathbf{C}_1$ corresponds to the stochastic *transition model*

$$\mathbf{s}^*(t+1) \sim p(x(t+1)|\mathbf{s}(t), \mathbf{a}(t)) \tag{10.6}$$

where x is the unknown current state. Additionally, $\mathbf{C}_2$ is related to the stochastic *observation model* $p(\mathbf{p}(t)|s(t))$. A probabilistic interpretation of $\mathbf{C}_2$ would be

$$\mathbf{s}(t+1) \sim p(x(t+1)|\mathbf{s}^*(t+1), \mathbf{p}(t+1))\,. \tag{10.7}$$

See for instance [101] for a system which makes use of this framework.

The mappings have sizes $N_s \times N_sN_a$ and $N_s \times N_sN_p$, and this gives us at most $N_s^2(N_a + N_p)$ coefficients compared to $N_s^2N_aN_p$ in the single mapping case. Thus the split into two maps is advantageous, provided that the behaviour is not affected (which in our case it is not).

Aside from the gain in number of coefficients, the split into two maps will also simplify the optimisation of the mappings considerably. If we during the optimisation supply samples of $\mathbf{s}^*(t{+}1)$ that are identical to $\mathbf{s}(t{+}1)$ we end up with a mapping, $\mathbf{C}_2$, that simply weights the state vector with the correlations between the observed percept and those corresponding to each state during optimisation. In other words (10.5) is equivalent to

$$\tilde{\mathbf{s}}(t+1) = \mathrm{diag}(\mathbf{P}\mathbf{p}(t+1))\mathbf{C}_1\left[\mathbf{s}(t) \otimes \mathbf{a}(t)\right]\,. \tag{10.8}$$

Here $\mathbf{P}$ is a matrix with row n containing the percept observed at state n during the training, and diag() generates a matrix with the argument vector in the diagonal.

10.3.2 Exploratory behaviour

How quickly the system is able to recognise it's location is of course critically dependent on which actions it takes. A good exploratory behaviour should strive to observe new percepts as often as possible, but how can the system know that shifting its attention to something new when it does not yet know where it is?

In this system the actions are chosen using a *policy*, where the probabilities for each action are conditional on the previous action $\mathbf{a}(t-1)$ and the observed percept $\mathbf{p}(t)$. I.e. the action probabilities can be calculated as

$$p(\mathbf{a}(t) = \mathbf{a}^h) = \mathbf{c}^h\left[\mathbf{a}(t-1) \otimes \mathbf{p}_2(t)\right] \tag{10.9}$$

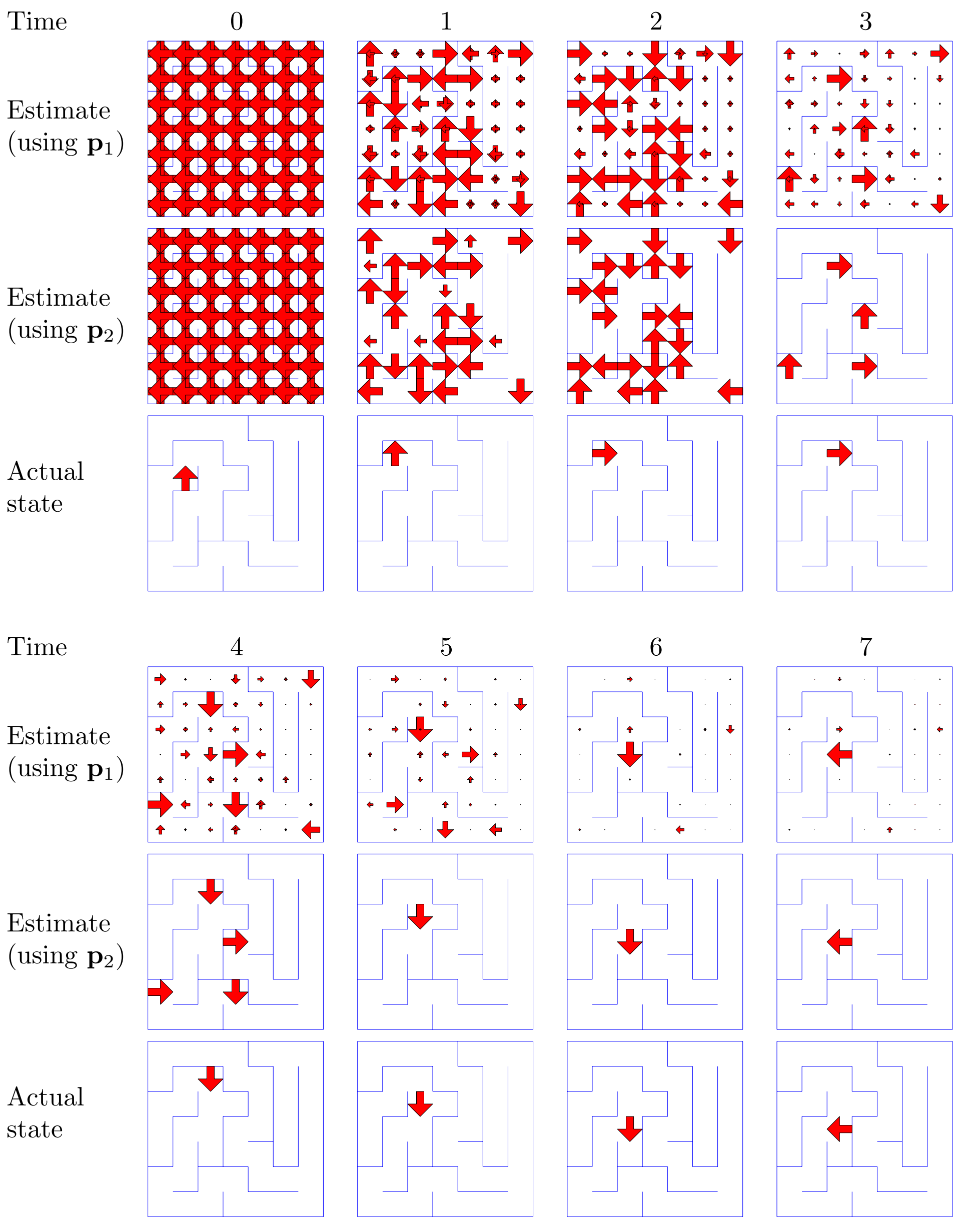

Time
0
1
2
3
Estimate (using $\mathbf{p}_1$)
Estimate (using $\mathbf{p}_2$)
Actual state
Time
4
5
6
7
Estimate (using $\mathbf{p}_1$)
Estimate (using $\mathbf{p}_2$)
Actual state

Figure 10.3: Illustration of state narrowing.

where $\{\mathbf{a}^h\}_1^3$ are the three possible actions (see section 10.2). The coefficients in the mappings $\{\mathbf{c}^h\}_1^3$ should be defined such that $\sum_h p(\mathbf{a}(t) = \mathbf{a}^h) = 1$.

Initially we define the policy $\{\mathbf{c}^h\}_1^3$, manually. A random run of a system with a fixed policy is demonstrated in figure 10.3. The two different kinds of percepts $\mathbf{p}_1$ and $\mathbf{p}_2$ are those defined in (10.1).

10.3.3 Evaluating narrowing performance

The performance of the localisation process may be evaluated by observing how the estimated state vector $\mathbf{s}(t)$ changes over time. As a measure of how *narrow* a specific state vector is we will use

$$n(t) = \frac{\sum_k s_k(t)}{\max_k\{s_k(t)\}} \, . \tag{10.10}$$

If all state channels are activated to the same degree, as is the case for $t = 0$, we will get $n(t) = N_s$, and if just one state channel is activated we will get $n(t) = 1$. Thus $n(t)$ can be seen as a measure of how many possible states are still remaining.

Figure 10.4 (top) shows a comparison of systems using local and semi-local percepts for 50 runs of the network. For each run the true initial state is selected at random, and $\mathbf{s}(0)$ is set to $\mathbf{1}/N_s$.

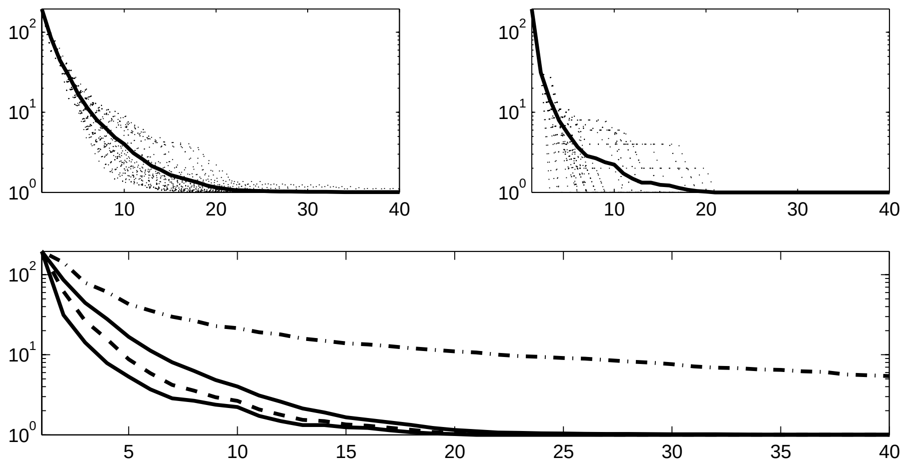


Figure 10.4: Narrowing performance.
Top left: $n(t)$ for a system using $\mathbf{p}_1$. Top right: $n(t)$ for a system using $\mathbf{p}_2$. Each graph shows 50 runs (dotted). The solid curves are averages. Bottom: Solid: $\mathrm{n}(t)$ *for $\mathbf{p}_1$ and $\mathbf{p}_2$. Dashed: $\mathbf{p}_1$ using* $\mathrm{f}_1()$*. Dash-dotted: $\mathbf{p}_1$ using* $\mathrm{f}_2()$*. Each curve is an average over* 50 *runs.*

Since the only thing that differs between the two upper plots in figure 10.4 is the percepts, the difference in convergence has to occur in step 2 of (10.5). We can further demonstrate what influence the feature correlation has on the convergence by modifying the correlation step in equation 10.8 as follows

$$\tilde{\mathbf{s}}(t+1) = \mathrm{diag}(\mathrm{f}(\mathbf{P}\mathbf{p}(t+1)))\mathbf{C}_1\left[\mathbf{s}(t) \otimes \mathbf{a}(t)\right] \, . \tag{10.11}$$

We will try the following two choices of f() on correlations of the semi-local percepts

$$\mathrm{f}_1(c) = \sqrt{c} \quad \text{and} \quad \mathrm{f}_2(c) = \begin{cases} 1 & \text{if } c > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{10.12}$$

All four kinds of systems are compared in the lower graph of figure 10.4. As can be seen, the narrowing behaviour is greatly improved by a sharp decay of the percept correlation function. However, for continuous environments there will most likely be a trade off between sharp correlation functions and state interpolation and the number of samples required during training.

10.3.4 Learning a narrowing policy

The conditional probabilities in the policy defined in section 10.3.2 can be learned using *reinforcement learning* [96]. A good exploratory behaviour is found by giving rewards to conditional actions $\{\mathbf{a}(t)|\mathbf{p}(t), \mathbf{a}(t-1)\}$ that reduce the narrowing measure (10.10), and by having the action probability density $p(\mathbf{a}(t) = \mathbf{a}^h|\mathbf{p}(t), \mathbf{a}(t-1))$ gradually increase for conditional actions with above-average rewards. This is called a *pursuit method* [96].

In order for the rewards not to die out, the system state is regularly reset to all ones, for instance when t mod $30 = 0$. The first attempt is to define the reward as a plain difference of the narrowing measure (10.10), i.e.

$$r_1(t) = n(t-1) - n(t)\,. \tag{10.13}$$

With this reward, the agent easily gets stuck into sub-optimal policies, such as constantly trying to move into a wall. Better behaviour is obtained by also looking at the narrowing difference one step into the future, i.e.

$$r_2(t) = r_1(t) + r_1(t+1) = n(t-1) - n(t+1)\,. \tag{10.14}$$

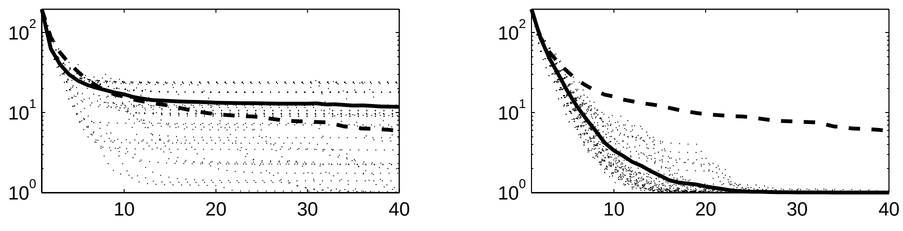


Figure 10.5: Narrowing performance.
Left: $n(t)$ *for a policy learned using* $r_1(t)$*. Right:* $n(t)$ *for a policy learned using* $r_2(t)$*. Each graph shows* 50 *runs (dotted). The thick curves are averages. Dashed curves show average narrowing for a completely random walk.*

The behaviours learned using (10.13) and (10.14) are compared with a random walk in figure 10.5.

10.4 Concluding remarks

The aim of this chapter has not been to describe a useful application, but instead to show how the principle of successive recognition can be used. Compared to a real robot navigation task, the environment used is way too simple to serve as a model world. Further experiments will extend the model to continuous environments, with noisy percepts and actions.

Chapter 11

Conclusions and Future Research Directions

In this chapter we conclude the thesis by summarising the results. We also indicate open issues, which point to research directions that can be pursued further.

11.1 Conclusions

This thesis is the result of asking the question “What can be done in the channel representation, which cannot be accomplished without it?”. We started by deriving expressions for channel encoding scalars, and retrieving them again using a *local decoding*. We then investigated what the simple operation of averaging in the channel representation resulted in.

The result that several modes of a distribution can be treated in parallel is of fundamental importance in perception. Perception in the presence of noise is a difficult problem. One especially persistent kind of noise is a competing nearby feature. By making use of the channel representation, we can make this problem go away, by simultaneously estimating all present features in parallel. We can select significant features after estimation, by picking one or several of the *local decodings*. In principle this would allow the design of a perception system similar to that of the bat described in section 2.1.4.

Channel representation is also useful for response generation. The associative networks in chapter 9 was shown to be able to learn piecewise continuous mappings, without blurring discontinuities. Such an ability is useful in response generation, such as navigation with obstacle avoidance. If we encounter an obstacle in front of us, it might be possible to pass it on both the left and the right side, so both these options are valid responses. Their average however is not, and thus a system that learns obstacle avoidance will need to use some kind of channel representation in order to avoid such inappropriate averaging.

For all levels in a perception system it is crucial that not all information is processed at each position. In order not to be flooded with data we need to exploit *locality*, i.e. restricting the amount of information available at each position to a local context. This can however lead to problems such as *perceptual aliasing*. As was demonstrated in chapter 9, intermediate responses in channel representation is a proper way to deal with perceptually aliased states. The channel representation solves the perceptual aliasing problem by not trying to merge states, but instead passing them on to the next processing level, where hopefully more context will be available to resolve the aliasing.

11.2 Future research

As is common in science, this thesis answered some questions, but at the same time it raised several new ones. We will now mention some questions which might be worthwhile to pursue further.

This far, the only operations considered in channel spaces are averaging and non-negative projections. Are there other meaningful operations in channel spaces? One option is to adapt the averaging to the local image structure, see Felsberg's paper [28] for some first results in this area.

The clustering of constant slopes developed in chapter 7 is, as mentioned just a first result. It could probably benefit from changing the representation of the slopes to cluster.

11.2.1 Feature matching and recognition

Two currently active areas in computer vision are *wide baseline matching*, see e.g. [99] and *parts based object recognition*, see e.g. [70, 66, 82]. Both of these areas are possible applications for the blob features developed in chapter 7.

11.2.2 Perception action cycles

Active vision is an important aspect of robotics. Apart from the simple example in chapter 10, this thesis has not dealt with closed perception–action loops. One direction to pursue is to explore the visual servoing idea in connection with the methods and representations developed in this thesis. One way to do this is to apply the successive recognition system in chapter 10 to more realistic problems, but other architectures and approaches could also prove useful.

It is by logic we prove, it is by intuition that we invent.
Henri Poincaré, 1904

Appendices

A Theorems on $\cos^2$ kernels

Theorem A.1 *For* $\cos^2$ *kernels with* $\omega = \pi/N$*, a group of* N *consecutive channels starting at index* k *has a common active domain of*

$$S_k^N = \,]k - 1 + N/2, k + N/2[\, .$$

Proof: The *active domain* (non-zero domain, or support) of a channel is defined as

$$S_k = \{x : \mathrm{B}^k(x) > 0\} = \,]L_k, U_k[\, . \tag{A.1}$$

Since the kernels should go smoothly to zero (as discussed in section 3.2.2), this is always an open interval, as indicated by the brackets. For the $\cos^2$ kernel (3.2) we have domains of the type

$$S_k = \,]k - \pi/2/\omega, k + \pi/2/\omega[\, . \tag{A.2}$$

For $\omega = \pi/N$ this becomes

$$S_k = \,]k - N/2, k + N/2[\, . \tag{A.3}$$

The common active domain of N channels, S_k^N becomes

$$S_k^N = S_k \cap S_{k+1} \cap \ldots \cap S_{k+N-1} = \,]L_{k+N-1}, U_k[\, = \tag{A.4}$$

$$= \,]k + N - 1 - N/2, k + N/2[\, = \,]k - 1 + N/2, k + N/2[\, . \tag{A.5}$$

This concludes the proof. □

Theorem A.2 *For* $\cos^2$ *kernels with* $\omega = \pi/N$*, and a local decoding using* N *channels, the represented domain of a* K *channel set becomes*

$$R_K^N = \,]N/2, K + 1 - N/2]\, .$$

Proof: If we perform the local decoding using groups of N channels with $\omega = \pi/N$, we will have decoding intervals according to theorem A.1. Note that we need to have $N \in \mathbb{N}/\{0, 1\}$ in order to have a proper decoding. These intervals are all of length 1, and thus they do not overlap. We now modify the upper end of the intervals

$$S_k^N = \,]k - 1 + N/2, k + N/2] \tag{A.6}$$

in order to be able to join them. This makes no practical difference, since all that happens at the boundary is that one channel becomes inactive. For a channel representation using K channels (with $K \geq N$) we get a represented interval of type

$$R_K^N = S_1^N \cup S_2^N \cup \ldots \cup S_{K-N+1}^N =]L_{1+N-1}, U_{K-N+1}] \tag{A.7}$$

$$=]N/2, K - N + 1 + N/2] =]N/2, K + 1 - N/2] \,. \tag{A.8}$$

This concludes the proof. □

Theorem A.3 *The sum of a channel value vector* $\begin{pmatrix} B^1(x) & B^2(x) & \ldots & B^K(x) \end{pmatrix}^T$ *for* $\omega = \pi/N$*, where* $N \in \mathbb{N}/\{0,1\}$*, is invariant to the value of* x *when* $x \in R_K^N$*.*

Proof: According to theorem A.1, groups of N consecutive channels with $\omega = \pi/N$ have mutually non-overlapping active domains S_k^N. This means that for a given channel vector, the value x will fall into exactly one of these domains, S_k^N. Thus the sum over the entire channel set is equal to the sum over the channels belonging to S_k^N, for some value of k

$$\sum_{n=0}^{K} \mathrm{B}^n(x) = \sum_{n=k}^{k+N-1} \mathrm{B}^n(x) = \sum_{n=0}^{N-1} \mathrm{B}^{k+n}(x) \,. \tag{A.9}$$

We now define a complex valued function

$$\boldsymbol{v}_k(x) = e^{\boldsymbol{i}2\omega(x-k)} \,. \tag{A.10}$$

This allows us to write the kernel function $\mathrm{B}^k(x)$ as

$$\begin{aligned} \mathrm{B}^k(x) &= \cos^2(\omega(x-k)) = 0.5 + 0.5\cos(2\omega(x-k)) = \\ &= 0.5 + 0.5\mathrm{Re}\left[\mathbf{v}_k(x)\right] \,. \end{aligned}$$

Now the sum in (A.9) becomes

$$\sum_{n=0}^{N-1} \mathrm{B}^{k+n}(x) = \frac{N}{2} + \frac{1}{2}\mathrm{Re}\left[\sum_{n=0}^{N-1} \mathbf{v}_{k+n}(x)\right] \,. \tag{A.11}$$

The complex sum in this expression can be rewritten as

$$\sum_{n=0}^{N-1} \mathbf{v}_{k+n}(x) = \sum_{n=0}^{N-1} e^{\boldsymbol{i}2\omega(x-k-n)} = e^{\boldsymbol{i}2\omega(x-k)} \sum_{n=0}^{N-1} \left(e^{-\boldsymbol{i}2\omega}\right)^n \,. \tag{A.12}$$

For[1] $e^{-\boldsymbol{i}2\omega} \neq 1$ this geometric sum can be written as

[1]The case $e^{-\boldsymbol{i}2\omega} = 1$ never happens, since it is equivalent to $\omega = n\pi$, where $n \in \mathbb{N}$, and our assumption was $\omega = \pi/N$, $N \in \mathbb{N}/\{0,1\}$.

$$\sum_{n=0}^{N-1}\left(e^{-\boldsymbol{i}2\omega}\right)^n = \frac{1-e^{-\boldsymbol{i}2\omega N}}{1-e^{-\boldsymbol{i}2\omega}}\,. \tag{A.13}$$

The numerator of this expression is zero exactly when $\omega N = n\pi$, $n \in \mathbb{N}$. Since our assumption was $\omega = \pi/N$, $N \in \mathbb{N}/\{0,1\}$, it is always zero. From this follows that the exponential sum in equation A.11 also equals zero. We can now reformulate equation A.11 as

$$\sum_{n=0}^{N-1} \mathrm{B}^{k+n}(x) = \frac{N}{2} \quad \text{for } \omega = \pi/N \text{ where } N \in \mathbb{N}/\{0,1\}. \tag{A.14}$$

This in conjunction with (A.9) proves the theorem. □

Theorem A.4 *The sum of a squared channel value vector* $\begin{pmatrix} B^1(x)^2 & B^2(x)^2 & \dots & B^K(x)^2\end{pmatrix}^T$ *for* $\omega = \pi/N$, *where* $N \in \mathbb{N}/\{0,1,2\}$ *is invariant to the value of* x *when* $x \in R_K^N$.

Proof: The proof of this theorem is similar to the proof of theorem A.3. With the same reasoning as in (A.9) we get

$$\sum_{n=0}^{K} \mathrm{B}^n(x)^2 = \sum_{n=k}^{k+N-1} \mathrm{B}^n(x)^2 = \sum_{n=0}^{N-1} \mathrm{B}^{k+n}(x)^2\,. \tag{A.15}$$

We now rewrite the squared kernel function as

$$\mathrm{B}^k(x)^2 = \cos^4(\omega(x-k)) = \frac{3}{8} + \frac{1}{2}\cos(2\omega(x-k)) + \frac{1}{8}\cos(4\omega(x-k))\,.$$

This allows us to rewrite (A.9) as

$$\sum_{n=0}^{N-1} \mathrm{B}^{k+n}(x)^2 = \frac{3N}{8} + \frac{1}{2}\mathrm{Re}\left[\sum_{n=0}^{N-1} \mathbf{v}_{k+n}(x)\right] + \frac{1}{8}\mathrm{Re}\left[\sum_{n=0}^{N-1} \mathbf{v}_{k+n}^2(x)\right]\,. \tag{A.16}$$

The first complex sum in this expression is zero for $\omega = \pi/N$, where $N \in \mathbb{N}/\{0,1\}$ (see equations A.12 and A.13).

The second sum can be written as

$$\sum_{n=0}^{N-1} \mathbf{v}_{k+n}^2(x) = \sum_{n=0}^{N-1} e^{\boldsymbol{i}4\omega(x-k-n)} = e^{\boldsymbol{i}4\omega(x-k)} \sum_{n=0}^{N-1}\left(e^{-\boldsymbol{i}4\omega}\right)^n\,. \tag{A.17}$$

For $e^{-\boldsymbol{i}4\omega} \neq 1$ (that is, $\omega \neq n\pi/2$ where $n \in \mathbb{N})^2$, this geometric sum can be written as

[2] In effect this excludes the solutions $N = 1$, and $N = 2$.

$$\sum_{n=0}^{N-1} \left(e^{-\boldsymbol{i}4\omega}\right)^n = \frac{1-e^{-\boldsymbol{i}4\omega N}}{1-e^{-\boldsymbol{i}4\omega}}\,. \tag{A.18}$$

The numerator of this expression is zero exactly when $\omega = \frac{n\pi}{2N}$, for integers n, and N, but again our premise was $\omega = \pi/N$, $N \in \mathbb{N}/\{0,1,2\}$, so it is always zero. The constraints on equation A.18 requires us to exclude the cases $N \in \{0,1,2\}$.

We can now reformulate equation A.16 as

$$\sum_{n=0}^{N-1} \mathrm{B}^{k+n}(x)^2 = \frac{3N}{8} \quad \text{for } \omega = \pi/N \text{ where } N \in \mathbb{N}/\{0,1,2\}. \tag{A.19}$$

This in conjunction with (A.15) proves the theorem. □

Observation A.5 *We will now derive a local decoding for the* $\cos^2$ *when* $\omega = \pi/2$.

For the case $\omega = \pi/2$ we can also define a local decoding, but it is more difficult to decide whether the decoding is valid or not. We now have the system

$$\begin{pmatrix} u^l \\ u^{l+1} \end{pmatrix} = \begin{pmatrix} r\mathrm{B}^l(x) \\ r\mathrm{B}^{l+1}(x) \end{pmatrix} = \begin{pmatrix} r\cos^2(\pi/2(x-l)) \\ r\cos^2(\pi/2(x-l-1)) \end{pmatrix} \tag{A.20}$$

since $\cos(x-\pi/2) = \sin(x)$ we have

$$\begin{pmatrix} u^l \\ u^{l+1} \end{pmatrix} = \begin{pmatrix} r\cos^2(\pi/2(x-l)) \\ r\sin^2(\pi/2(x-l)) \end{pmatrix}\,. \tag{A.21}$$

We now see that

$$\hat{x} = l + \frac{2}{\pi}\arg\left[\sqrt{u^l} + \boldsymbol{i}\sqrt{u^{l+1}}\right] = l + \frac{2}{\pi}\tan^{-1}\left(\sqrt{u^{l+1}/u^l}\right) \tag{A.22}$$

and

$$\hat{r}_1 = |u^l + \boldsymbol{i}u^{l+1}| \quad \text{and} \quad \hat{r}_2 = u^l + u^{l+1}\,. \tag{A.23}$$

In order to select valid decodings, we cannot simply check if $\hat{x}$ is inside the common support, since this is always the case. One way to avoid giving invalid solutions is to require that $\hat{r}_2(l) \geq \hat{r}_2(l+1)$ and $\hat{r}_2(l) \geq \hat{r}_2(l-1)$. □

Theorem A.6 *The* $\cos^2$ *local decoding is an unbiased estimate of the mean, if the PDF* $f(x)$ *is even, and restricted to the decoding support* S_l^N.

$$f(x) = f(2\mu - x) \quad \text{and} \quad \mathrm{supp}\{f\} \subset S_l^N \quad \Rightarrow \quad E\{\hat{x}\} = E\{x_n\}\,.$$

The local decoding of a $\cos^2$ channel representation consists of two steps: a linear parameter estimation and a non-linear combination of the parameters into estimates of the mode location and the confidence. The expected value of the linear parameter estimation is

$$E\{\mathbf{p}\} = \begin{pmatrix} \int_{S_l^N} \cos(2\omega(x-l))f(x)dx \\ \int_{S_l^N} \sin(2\omega(x-l))f(x)dx \\ \int_{S_l^N} f(x)dx \end{pmatrix} \tag{A.24}$$

if we require that $\text{supp}\{f\} \subset S_l^N$, see section 4.2.2. We now simplify the notation, by denoting $c(x) = \cos(2\omega x)$, and $s(x) = \sin(2\omega x)$. Further, we assume that f is even about a point μ, i.e. $f(x) = f(2\mu - x)$. This allows us to rewrite $E\{p_1\}$ as

$$E\{p_1\} = \int_{S_l^N} f(x)c(x-l)dx = \int_{S_l^N} f(x)c(x-\mu+\mu-l)dx \tag{A.25}$$

$$= \int_{S_l^N} f(x)\left[c(x-\mu)c(\mu-l) - s(x-\mu)s(\mu-l)\right]dx \tag{A.26}$$

$$= c(\mu-l)\int_{S_l^N} f(x)c(x-\mu)dx - s(\mu-l)\underbrace{\int_{S_l^N} f(x)s(x-\mu)dx}_{=0} \tag{A.27}$$

$$= c(\mu-l)\int_{S_l^N} f(x)c(x-\mu)dx \tag{A.28}$$

where one of the integrals becomes zero due to antisymmetry about μ. In a similar way we can rewrite $E\{p_2\}$ as

$$E\{p_2\} = s(\mu-l)\int_{S_l^N} f(x)c(x-\mu)dx\,. \tag{A.29}$$

We now denote the integral by α

$$\alpha = \int_{S_l^N} f(x)c(x-\mu)dx\,. \tag{A.30}$$

This allows us to write

$$E\{p_1\} = \alpha\cos(2\omega(\mu-l)) \tag{A.31}$$

$$E\{p_2\} = \alpha\sin(2\omega(\mu-l))\,. \tag{A.32}$$

Finally we insert these two expressions into the non-linear step of the decoding

$$E\{\hat{x}\} = l + \frac{1}{2\omega}\arg\left[E\{p_1\} + \boldsymbol{i}E\{p_2\}\right] \tag{A.33}$$

$$= l + \frac{1}{2\omega}\arg\alpha e^{\boldsymbol{i}2\omega(\mu-l)} = l + \frac{1}{2\omega}\left(2\omega(\mu-l)\right) \tag{A.34}$$

$$= \mu\,. \tag{A.35}$$

For a density that is even about μ, we also get

$$E\{x_n\} = \int xf(x)dx \tag{A.36}$$

$$= \int(2\mu-x)f(2\mu-x)dx = 2\mu - \int xf(2\mu-x)dx = 2\mu - \int xf(x)dx\,. \tag{A.37}$$

Setting the right-hand of (A.36) equal to the right-hand of (A.37) gives

$$E\{x_n\} = \int xf(x)dx = \mu\,. \tag{A.38}$$

Together with (A.35) this gives $E\{\hat{x}\} = E\{x_n\}$, which concludes the proof. $\square$

B Theorems on B-splines

Theorem B.1 *The sum of integer shifted B-splines is independent of the encoded scalar for any degree n.*

$$\sum_k B_k^n(x) = 1 \quad \forall x,\ n \in \mathbb{N}$$

Proof: B-splines of degree zero are defined as

$$\mathrm{B}_k^0(x) = \begin{cases} 1 & k - 0.5 \leq x < k + 0.5 \\ 0 & \text{otherwise.} \end{cases}$$

From this trivially follows that the zeroth degree sum is constant, since exactly one B-spline is non-zero, and equal to 1 at a time. That is

$$\sum_k \mathrm{B}_k^0(x) = 1 \quad \forall x\,. \tag{B.1}$$

Using the recurrence relation (5.10), we can express the sum of an arbitrary degree as

$$\sum_k \mathrm{B}_k^n(x) = \tag{B.2}$$

$$= \sum_k \left(\frac{x - k + (n+1)/2}{n} \mathrm{B}_{k-1/2}^{n-1}(x) + \frac{(n+1)/2 - x + k}{n} \mathrm{B}_{k+1/2}^{n-1}(x) \right) \tag{B.3}$$

$$= \sum_k \left(\frac{x - k + (n+1)/2}{n} \mathrm{B}_{k-1/2}^{n-1}(x) \right) + \sum_k \left(\frac{(n+1)/2 - x + k}{n} \mathrm{B}_{k+1/2}^{n-1}(x) \right) \tag{B.4}$$

$$= \sum_l \left(\frac{x - l + n/2}{n} \mathrm{B}_l^{n-1}(x) \right) + \sum_l \left(\frac{n/2 + l - x}{n} \mathrm{B}_l^{n-1}(x) \right) \tag{B.5}$$

$$= \sum_l \left(\frac{x - l + n/2}{n} + \frac{n/2 - x + l}{n} \right) \mathrm{B}_l^{n-1}(x) \tag{B.6}$$

$$= \sum_l \mathrm{B}_l^{n-1}(x)\,. \tag{B.7}$$

That is, the sum of B-splines of degree n is equal to the sum of degree $n-1$. This in conjunction with (B.1) proves theorem B.1 by induction. □

Theorem B.2 *The first moment of integer shifted B-splines is equal to the encoded scalar for any degree $n \geq 1$.*

$$\sum_k k B_k^n(x) = x \quad \forall x,\ n \in \mathbb{N}^+$$

Proof: Using (5.11), the first moment of B-splines of degree one can be written as

$$\sum_k k\mathrm{B}_k^1(x) = \sum_k k\left[(x-k+1)\mathrm{B}_{k-1/2}^0(x) + (1-x+k)\mathrm{B}_{k+1/2}^0(x)\right] \tag{B.8}$$

$$= \sum_k k(x-k+1)\mathrm{B}_{k-1/2}^0(x) + \sum_k k(1-x+k)\mathrm{B}_{k+1/2}^0(x) \tag{B.9}$$

$$= \sum_l (l+1/2)(x-l+1/2)\mathrm{B}_l^0(x) + \sum_l (l-1/2)(1/2-x+l)\mathrm{B}_l^0(x) \tag{B.10}$$

$$= \sum_l (1/4-l^2+lx+x/2)\mathrm{B}_l^0(x) + \sum_l (l^2-1/4-lx+x/2)\mathrm{B}_l^0(x) \tag{B.11}$$

$$= \sum_l x\mathrm{B}_l^0(x) = x\sum_l \mathrm{B}_l^0(x) = x\,. \tag{B.12}$$

We now make an expansion of the first moment using the recurrence relation (5.10),

$$\sum_k k\mathrm{B}_k^n(x) = \tag{B.13}$$

$$= \sum_k k\left[\frac{x-k+(n+1)/2}{n}\mathrm{B}_{k-1/2}^{n-1}(x) + \frac{(n+1)/2-x+k}{n}\mathrm{B}_{k+1/2}^{n-1}(x)\right] \tag{B.14}$$

$$= \sum_k k\frac{x-k+(n+1)/2}{n}\mathrm{B}_{k-1/2}^{n-1}(x) + \sum_k k\frac{(n+1)/2-x+k}{n}\mathrm{B}_{k+1/2}^{n-1}(x) \tag{B.15}$$

$$= \sum_l \frac{(l+1/2)(x-l+n/2)}{n}\mathrm{B}_l^{n-1}(x) + \sum_l \frac{(l-1/2)(n/2-x+l)}{n}\mathrm{B}_l^{n-1}(x) \tag{B.16}$$

$$= \sum_l \frac{2lx-2l^2+ln+x-l+n/2}{2n}\mathrm{B}_l^{n-1}(x) \tag{B.17}$$

$$+ \sum_l \frac{ln-2lx+2l^2-n/2+x-l}{2n}\mathrm{B}_l^{n-1}(x) \tag{B.18}$$

$$= \sum_l \frac{ln+x-l}{n}\mathrm{B}_l^{n-1}(x)\,. \tag{B.19}$$

By applying theorem B.1 we get

$$\sum_k k\mathrm{B}_k^n(x) = \sum_l \frac{ln + x - l}{n}\mathrm{B}_l^{n-1}(x) \tag{B.20}$$

$$= \frac{x}{n} + \left(1 - \frac{1}{n}\right)\sum_l l\mathrm{B}_l^{n-1}(x)\,. \tag{B.21}$$

If we assume the theorem holds for $n-1$, we get

$$\sum_k k\mathrm{B}_k^n(x) = \frac{x}{n} + \left(1 - \frac{1}{n}\right)\sum_l l\mathrm{B}_l^{n-1}(x) \tag{B.22}$$

$$= \frac{x}{n} + \left(1 - \frac{1}{n}\right) x = x\,. \tag{B.23}$$

This in conjunction with (B.12) proves theorem B.2 by induction. □

C Theorems on ellipse functions

Theorem C.1 *The matrix* $\mathbf{A}$ *describing the shape of an ellipse* $(\mathbf{x}-\mathbf{m})^T\mathbf{A}(\mathbf{x}-\mathbf{m}) \leq 1$ *is related to the inertia matrix* $\mathbf{I}$ *of the same ellipse according to*

$$\mathbf{I} = \frac{1}{4}\mathbf{A}^{-1} \quad or \quad \mathbf{A} = \frac{1}{4}\mathbf{I}^{-1}\,.$$

Proof: A surface patch in the shape of an ellipse is the set of points $\mathbf{x} = (x_1\ x_2)^T$ fulfilling the relation $(x_1/a)^2 + (x_2/b)^2 \leq 1$. This can be rewritten as

$$\mathbf{x}^T\mathbf{D}\mathbf{x} \leq 1 \quad \text{for} \quad \mathbf{D} = \begin{pmatrix} 1/a^2 & 0 \\ 0 & 1/b^2 \end{pmatrix}\,. \tag{C.1}$$

In order to describe an ellipse with arbitrary position and orientation, we add a rotation $\mathbf{R} = (\mathbf{r}_1\ \mathbf{r}_2)$, and a translation $\mathbf{m} = (m_1\ m_2)^T$ and obtain

$$\big(\mathbf{x}-\mathbf{m}\big)^T\,\mathbf{A}\,\big(\mathbf{x}-\mathbf{m}\big) \leq 1 \quad \text{where} \quad \mathbf{A} = \mathbf{R}^T\mathbf{D}\mathbf{R}\,. \tag{C.2}$$

Note that the square root of the left hand expression is a *Mahalanobis distance* between $\mathbf{m}$ and $\mathbf{x}$, with $\mathbf{A}$ defining the metric, see e.g. [7]. $\mathbf{A}$ often corresponds to the inverse covariance of a data set.

For the ellipse described by $\mathbf{A}$, and $\mathbf{m}$, we can define a binary mask,

$$\mathrm{v}(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad (\mathbf{x}-\mathbf{m})^T\,\mathbf{A}\,(\mathbf{x}-\mathbf{m}) \leq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{C.3}$$

The mask $\mathrm{v}(\mathbf{x})$ has moments that in the continuous case are given by

$$\mu_{kl} = \int_{\mathbb{R}^2} x_1^k x_2^l \mathrm{v}(\mathbf{x}) d\mathbf{x} = \int_{(\mathbf{x}-\mathbf{m})^T \mathbf{A}(\mathbf{x}-\mathbf{m}) \leq 1} x_1^k x_2^l d\mathbf{x} \tag{C.4}$$

$$= \int_{\mathbf{x}^T \mathbf{R}^T \mathbf{D} \mathbf{R} \mathbf{x} \leq 1} (x_1 + m_1)^k (x_2 + m_2)^l d\mathbf{x} \tag{C.5}$$

$$\begin{bmatrix} \mathbf{y} = \mathbf{R}\mathbf{x} \\ d\mathbf{x} = d\mathbf{y} \end{bmatrix} = \int_{\mathbf{y}^T \mathbf{D} \mathbf{y} \leq 1} (\mathbf{r}_1^T \mathbf{y} + m_1)^k (\mathbf{r}_2^T \mathbf{y} + m_2)^l d\mathbf{y} = \tag{C.6}$$

$$\begin{bmatrix} \mathbf{x} = \mathbf{D}^{\frac{1}{2}} \mathbf{y} \\ d\mathbf{y} = |\mathbf{D}^{-\frac{1}{2}}| d\mathbf{x} \end{bmatrix} = \int_{\mathbf{x}^T \mathbf{x} \leq 1} (\mathbf{r}_1^T \mathbf{D}^{-\frac{1}{2}} \mathbf{x} + m_1)^k (\mathbf{r}_2^T \mathbf{D}^{-\frac{1}{2}} \mathbf{x} + m_2)^l |\mathbf{D}^{-\frac{1}{2}}| d\mathbf{x} = \tag{C.7}$$

$$\begin{bmatrix} \mathbf{x} = \rho \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix} = \rho \hat{\mathbf{n}} \\ d\mathbf{x} = \rho d\rho d\varphi \end{bmatrix} = \int_{-\pi}^{\pi} \int_0^1 (\mathbf{r}_1^T \mathbf{D}^{-\frac{1}{2}} \rho \hat{\mathbf{n}} + m_1)^k (\mathbf{r}_2^T \mathbf{D}^{-\frac{1}{2}} \rho \hat{\mathbf{n}} + m_2)^l ab\rho d\rho d\varphi\,. \tag{C.8}$$

If we define the rotation $\mathbf{R}$ to be

$$\mathbf{R} = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} = \begin{pmatrix} \mathbf{r}_1 & \mathbf{r}_2 \end{pmatrix} \tag{C.9}$$

we can simplify this to

$$\begin{aligned} \mu_{kl} = \int_{-\pi}^{\pi} \int_0^1 & (\rho a \cos\phi \cos\varphi - \rho b \sin\phi \sin\varphi + m_1)^k \times \\ & (\rho a \cos\phi \cos\varphi + \rho b \sin\phi \sin\varphi + m_2)^l ab\rho d\rho d\varphi\,. \end{aligned} \tag{C.10}$$

We can now verify the expressions for the low order moments

$$\mu_{00} = \int_{-\pi}^{\pi} \int_0^1 ab\rho d\rho d\varphi = \pi ab \tag{C.11}$$

$$\mu_{10} = \int_{-\pi}^{\pi} \int_0^1 (\rho a \cos\phi \cos\varphi - \rho b \sin\phi \sin\varphi + m_1) ab\rho d\rho d\varphi = m_1 \pi ab \tag{C.12}$$

$$\mu_{01} = \int_{-\pi}^{\pi} \int_0^1 (\rho a \cos\phi \cos\varphi + \rho b \sin\phi \sin\varphi + m_2) ab\rho d\rho d\varphi = m_2 \pi ab \tag{C.13}$$

$$\begin{aligned}
\mu_{20} &= \int_{-\pi}^{\pi}\int_{0}^{1}(\rho a\cos\phi\cos\varphi - \rho b\sin\phi\sin\varphi + m_1)^2 ab\rho d\rho d\varphi \\
&= \pi ab\left(\frac{1}{4}(a^2\cos^2\phi + b^2\sin^2\phi) + m_1^2\right) \qquad \text{(C.14)} \\
\mu_{02} &= \int_{-\pi}^{\pi}\int_{0}^{1}(\rho a\cos\phi\cos\varphi + \rho b\sin\phi\sin\varphi + m_2)^2 ab\rho d\rho d\varphi \\
&= \pi ab\left(\frac{1}{4}(a^2\cos^2\phi + b^2\sin^2\phi) + m_2^2\right) \qquad \text{(C.15)} \\
\mu_{11} &= \int_{-\pi}^{\pi}\int_{0}^{1}(\rho a\cos\phi\cos\varphi - \rho b\sin\phi\sin\varphi + m_1)\times \\
&\qquad (\rho a\cos\phi\cos\varphi + \rho b\sin\phi\sin\varphi + m_2) ab\rho d\rho d\varphi \\
&= \pi ab\left(\frac{1}{4}(a^2\cos^2\phi - b^2\sin^2\phi) + m_1 m_2\right)\,. \qquad \text{(C.16)}
\end{aligned}$$

We now group the three second order moments into a matrix

$$\begin{aligned}
\begin{pmatrix}\mu_{20} & \mu_{11}\\ \mu_{11} & \mu_{02}\end{pmatrix} &= \frac{\pi ab}{4}\begin{pmatrix}\mathbf{r}_1^T\mathbf{D}^{-1}\mathbf{r}_1 & \mathbf{r}_1^T\mathbf{D}^{-1}\mathbf{r}_2\\ \mathbf{r}_2^T\mathbf{D}^{-1}\mathbf{r}_1 & \mathbf{r}_2^T\mathbf{D}^{-1}\mathbf{r}_2\end{pmatrix} + \pi ab\mathbf{m}\mathbf{m}^T \qquad \text{(C.17)} \\
&= \frac{\pi ab}{4}\mathbf{R}^T\mathbf{D}^{-1}\mathbf{R} + \pi ab\mathbf{m}\mathbf{m}^T\,. \qquad \text{(C.18)}
\end{aligned}$$

By division with μ_{00}, see (C.11), we get

$$\frac{1}{\mu_{00}}\begin{pmatrix}\mu_{02} & \mu_{11}\\ \mu_{11} & \mu_{20}\end{pmatrix} = \frac{1}{4}\mathbf{R}^T\mathbf{D}^{-1}\mathbf{R} + \mathbf{m}\mathbf{m}^T\,. \qquad \text{(C.19)}$$

By subtraction of $\mathbf{m}\mathbf{m}^T$ we obtain the definition of the inertia matrix

$$\mathbf{I} = \frac{1}{\mu_{00}}\begin{pmatrix}\mu_{02} & \mu_{11}\\ \mu_{11} & \mu_{20}\end{pmatrix} - \mathbf{m}\mathbf{m}^T = \frac{1}{4}\mathbf{R}^T\mathbf{D}^{-1}\mathbf{R}\,. \qquad \text{(C.20)}$$

Here we recognise the inverse of the ellipse matrix $\mathbf{A}^{-1} = \mathbf{R}^T\mathbf{D}^{-1}\mathbf{R}$, see (C.2), and thus the ellipse matrix $\mathbf{A}$ is related to $\mathbf{I}$ as

$$\mathbf{I} = \frac{1}{4}\mathbf{A}^{-1} \quad \text{or} \quad \mathbf{A} = \frac{1}{4}\mathbf{I}^{-1} \qquad \text{(C.21)}$$

which was what we set out to prove. □

Theorem C.2 *The axes, and the area of an ellipse can be extracted from its inertia matrix* $\mathbf{I}$ *according to*

$$\{a,b\} = \{2\sqrt{\lambda_1}, 2\sqrt{\lambda_2}\} \quad \text{and} \quad \text{Area} = 4\pi\sqrt{\det \mathbf{I}}\,.$$

Proof: For positive definite matrices, the eigenvectors constitute a rotation, and thus (C.20) is an eigenvalue decomposition of $\mathbf{I}$. In other words $\mathbf{I} = \lambda_1\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^T + \lambda_2\hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^T$, has its eigenvalues in the diagonal of $\frac{1}{4}\mathbf{D}^{-1}$, $\{\lambda_1,\lambda_2\} = \{a^2/4, b^2/4\}$. From this follows that $\{a,b\} = \{2\sqrt{\lambda_1}, 2\sqrt{\lambda_2}\}$.

Since $\det \mathbf{I} = \lambda_1\lambda_2 = \frac{1}{16}a^2b^2$, we can find the ellipse area as $\pi ab = 4\pi\sqrt{\det \mathbf{I}}$.

Also note that of all shapes with a given inertia matrix, the ellipse is the one that is best concentrated around $\mathbf{m}$. This means that in the discrete case, the above area measure will always be an overestimate of the actual area, with exception of the degenerate case when all pixels lie on a line. $\square$

Theorem C.3 *The outline of an ellipse is given by the parameter curve*

$$\mathbf{x} = \mathbf{R}^T\mathbf{D}^{-1/2}\begin{pmatrix}\cos t\\ \sin t\end{pmatrix} + \mathbf{m} \quad \text{for} \quad t \in [0, 2\pi[\,. \tag{C.22}$$

Proof: An ellipse is the set of points $\mathbf{x} \in \mathbb{R}^2$ fulfilling relation (C.2). By inserting the parameter curve (C.22) into the quadratic form of (C.2) we obtain

$$\begin{aligned}
(\mathbf{x}-\mathbf{m})^T\mathbf{A}(\mathbf{x}-\mathbf{m}) &= (\mathbf{x}-\mathbf{m})^T\mathbf{R}^T\mathbf{D}\mathbf{R}(\mathbf{x}-\mathbf{m}) && \text{(C.23)}\\
&= \begin{pmatrix}\cos t & \sin t\end{pmatrix}\mathbf{D}^{-1/2}\mathbf{R}\mathbf{R}^T\mathbf{D}\mathbf{R}\mathbf{R}^T\mathbf{D}^{-1/2}\begin{pmatrix}\cos t\\ \sin t\end{pmatrix} && \text{(C.24)}\\
&= \cos^2 t + \sin^2 t = 1\,. && \text{(C.25)}
\end{aligned}$$

Thus all points in (C.22) belong to the ellipse outline. Note that (C.22) is a convenient way to draw the ellipse outline. $\square$

Bibliography

[1] Y. Aloimonos, I. Weiss, and A. Bandopadhay. Active vision. *Int. Journal of Computer Vision*, 1(3):333–356, 1988.

[2] V. Aurich and J. Weule. Non-linear gaussian filters performing edge preserving diffusion. In *17:th DAGM-Symposium*, pages 538–545, Bielefeld, 1995.

[3] R. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):996–1005, August 1988.

[4] D. H. Ballard. Animate vision. In *Proc. Int. Joint Conf. on Artificial Intelligence*, pages 1635–1641, 1989.

[5] M. F. Bear, B. W. Connors, and M. A. Paradiso. *Neuroscience: Exploring the Brain.* Williams & Wilkins, 1996. ISBN 0-683-00488-3.

[6] S. Belongie, C. Carson, H. Greenspan, and J. Malik. Color- and texture-based image segmentation using EM and its application to content based image retrieval. In *Proceedings of the Sixth International Conference on Computer Vision*, pages 675–682, 1998.

[7] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995. ISBN 0-19-853864-2.

[8] M. Black, G. Sapiro, D. Marimont, and D. Heeger. Robust anisotropic diffusion. *IEEE Transactions on Image Processing*, 7(3):421–432, March 1998.

[9] A. Blake. *The Handbook of Brain Theory and Neural Networks*, chapter Active Vision, pages 61–63. MIT Press, 1995. M. A. Arbib, Ed.

[10] M. Borga. *Learning Multidimensional Signal Processing.* PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, 1998. Dissertation No 531, ISBN 91-7219-202-X.

[11] R. Brooks. A robust layered control system for a mobile robot. *IEEE Trans. on Robotics and Automation*, 2(1):14–23, March 1986.

[12] H. H. Bülthoff, S. Y. Edelman, and M. J. Tarr. How are three-dimensional objects represented in the brain? A.I. Memo No. 1479, April 1994. MIT AI Lab.

[13] F. W. Campbell and J. G. Robson. Application of Fourier analysis to the visibility of gratings. *J. Physiol.*, 197:551–566, 1968.

[14] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):255–274, November 1986.

[15] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Blobworld: Image segmentation using expectation-maximisation and its application to image querying. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8):1026–1038, August 2002.

[16] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, August 1995.

[17] L. Chrisman. Reinforcement Learning with Perceptual Aliasing: The Perceptual Distinctions Approach. In *National Conference on Artificial Intelligence*, pages 183–188, 1992.

[18] CMU Neural Networks Benchmark Collection, `http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/bench/cmu/`.

[19] D. Comaniciu and P. Meer. Mean shift analysis and applications. In *Proceedings of ICCV'99*, pages 1197–1203, Corfu, Greece, Sept 1999.

[20] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

[21] G. Dahlquist and Å. Björck. *Numerical Methods and Scientific Computation*, chapter Interpolation and related subjects. SIAM, Philadelphia, 2003. In press.

[22] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Trans. on Information Theory*, 36(5):961–1005, September 1990.

[23] R. Dawkins. *The Blind Watchmaker*. Penguin Books, 1986.

[24] P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund. The WITAS Unmanned Aerial Vehicle Project. In W. Horn, editor, *ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence*, pages 747–755, Berlin, August 2000.

[25] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV99*, pages 1033–1038, Corfu, Greece, September 1999.

[26] G. Farnebäck. *Polynomial Expansion for Orientation and Motion Estimation*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, 2002. Dissertation No 790, ISBN 91-7373-475-6.

[27] M. Felsberg. *Low Level Image Processing with the Structure Multivector*. PhD thesis, Christian-Albrechts-Universität, Kiel, March 2002.

[28] M. Felsberg and G. Granlund. Anisotropic channel filtering. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, LNCS 2749, pages 755–762, Gothenburg, Sweden, June-July 2003.

[29] M. Felsberg, H. Scharr, and P.-E. Forssén. The B-spline channel representation: Channel algebra and channel based diffusion filtering. Technical Report LiTH-ISY-R-2461, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, September 2002.

[30] M. Felsberg, H. Scharr, and P.-E. Forssén. Channel smoothing. *IEEE PAMI*, 2004. Submitted.

[31] D. J. Field. What is the goal of sensory coding? *Neural Computation*, 1994.

[32] P.-E. Forssén. Updating Camera Location and Heading using a Sparse Displacement Field. Technical Report LiTH-ISY-R-2318, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, November 2000.

[33] P.-E. Forssén. Image Analysis using Soft Histograms. In *Proceedings of the SSAB Symposium on Image Analysis*, pages 109–112, Norrköping, March 2001. SSAB.

[34] P.-E. Forssén. Sparse Representations for Medium Level Vision. Lic. Thesis LiU-Tek-Lic-2001:06, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, February 2001. Thesis No. 869, ISBN 91-7219-951-2.

[35] P.-E. Forssén. Window Matching using Sparse Templates. Technical Report LiTH-ISY-R-2392, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, September 2001.

[36] P.-E. Forssén. Observations Concerning Reconstructions with Local Support. Technical Report LiTH-ISY-R-2425, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, April 2002.

[37] P.-E. Forssén. Successive Recognition using Local State Models. In *Proceedings SSAB02 Symposium on Image Analysis*, pages 9–12, Lund, March 2002. SSAB.

[38] P.-E. Forssén. Channel smoothing using integer arithmetic. In *Proceedings SSAB03 Symposium on Image Analysis*, Stockholm, March 2003. SSAB.

[39] P.-E. Forssén and G. Granlund. Sparse Feature Maps in a Scale Hierarchy. In *AFPAC, Algebraic Frames for the Perception Action Cycle*, Kiel, Germany, September 2000.

[40] P.-E. Forssén and G. Granlund. Blob Detection in Vector Fields using a Clustering Pyramid. Technical Report LiTH-ISY-R-2477, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, November 2002.

[41] P.-E. Forssén and G. Granlund. Robust multi-scale extraction of blob features. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, LNCS 2749, pages 11–18, Gothenburg, Sweden, June-July 2003.

[42] P.-E. Forssén, G. Granlund, and J. Wiklund. Channel Representation of Colour Images. Technical Report LiTH-ISY-R-2418, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, March 2002.

[43] K. Fukunaga and L. D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[44] F. Godtliebsen, E. Spjøtvoll, and J. Marron. A nonlinear gaussian filter applied to images with discontinuities. *J. Nonpar. Statist.*, 8:21–43, 1997.

[45] G. H. Granlund. Magnitude Representation of Features in Image Analysis. In *The 6th Scandinavian Conference on Image Analysis*, pages 212–219, Oulu, Finland, June 1989.

[46] G. H. Granlund. The complexity of vision. *Signal Processing*, 74(1):101–126, April 1999. Invited paper.

[47] 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.

[48] G. H. Granlund and H. Knutsson. *Signal Processing for Computer Vision*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9530-1.

[49] G. Granlund. Does Vision Inevitably Have to be Active? In *Proceedings of the 11th Scandinavian Conference on Image Analysis*, Kangerlussuaq, Greenland, June 7–11 1999. SCIA. Also as Technical Report LiTH-ISY-R-2247.

[50] 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.

[51] G. Granlund, P.-E. Forssén, and B. Johansson. HiperLearn: A high performance channel learning architecture. *IEEE Transactions on Neural Networks*, 2003. Submitted.

[52] G. Granlund, K. Nordberg, J. Wiklund, P. Doherty, E. Skarman, and E. Sandewall. WITAS: An Intelligent Autonomous Aircraft Using Active Vision. In *Proceedings of the UAV 2000 International Technical Conference and Exhibition*, Paris, France, June 2000. Euro UVS.

[53] G. H. Granlund and A. Moe. Unrestricted recognition of 3-D objects using multi-level triplet invariants. In *Proceedings of the Cognitive Vision Workshop*, Zürich, Switzerland, September 2002. URL: http://www.vision.ethz.ch/cogvis02/.

[54] R. M. Gray. Dithered quantizers. *IEEE Transactions on Information Theory*, 39(3):805–812, 1993.

[55] L. Haglund. *Adaptive Multidimensional Filtering*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, October 1992. Dissertation No 284, ISBN 91-7870-988-1.

[56] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel. *Robust Statistics: The approach based on influence functions*. John Wiley and Sons, New York, 1986.

[57] R. I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, June 1997.

[58] S. Haykin. *Neural Networks–A comprehensive foundation*. Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 1999. ISBN 0-13-273350-1.

[59] D. Hearn and P. Baker. *Computer Graphics, 2nd ed.* Prentice Hall International, 1994. ISBN 0-13-159690-X.

[60] C. M. Hicks. The application of dither and noise-shaping to nyquist-rate digital audio: an introduction. Technical report, Communications and Signal Processing Group, Cambridge University Engineering Department, United Kingdom, 1995.

[61] I. P. Howard and B. J. Rogers. *Binocular Vision and Stereopsis*. Oxford Psychology Series, 29. Oxford University Press, New York, 1995. ISBN 0195084764.

[62] P. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(73-101), 1964.

[63] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, Sept 1999.

[64] B. Johansson. Multiscale curvature detection in computer vision. Lic. Thesis LiU-Tek-Lic-2001:14, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, March 2001. Thesis No. 877, ISBN 91-7219-999-7.

[65] B. Johansson and G. Granlund. Fast selective detection of rotational symmetries using normalized inhibition. In *Proceedings of the 6th European Conference on Computer Vision*, volume I, pages 871–887, Dublin, Ireland, June 2000.

[66] B. Johansson and A. Moe. Patch-duplets for object recognition and pose estimation. Technical Report LiTH-ISY-R-2553, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, November 2003.

[67] H. Knutsson, M. Andersson, and J. Wiklund. Advanced Filter Design. In *Proceedings of the 11th Scandinavian Conference on Image Analysis*, Greenland, June 1999. SCIA. Also as report LiTH-ISY-R-2142.

[68] P. Kovesi. Image features from phase congruency. Tech. Report 95/4, University of Western Australia, Dept. of CS, 1995.

[69] T. Landelius. *Reinforcement Learning and Distributed Local Model Synthesis*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, 1997. Dissertation No 469, ISBN 91-7871-892-9.

[70] B. Leibe and B. Schiele. Analyzing appearance and contour based methods for object categorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03)*, June 2003.

[71] M. W. Levine and J. M. Shefner. *Fundamentals of sensation and perception*. Addison-Wesley, 1981.

[72] T. Lindeberg. *Scale-space Theory in Computer Vision*. Kluwer Academic Publishers, 1994. ISBN 0792394186.

[73] T. Lindeberg and J. Gårding. Shape from texture in a multi-scale perspective. In *Proc. 4th International Conference on Computer Vision*, pages 683–691, Berlin, Germany, May 1993.

[74] D. Marr. *Vision*. W. H. Freeman and Company, New York, 1982.

[75] C. Meunier and J. P. Nadal. *The Handbook of Brain Theory and Neural Networks*, chapter Sparsely Coded Neural Networks, pages 899–901. MIT Press, 1995. M. A. Arbib, Ed.

[76] S. Mitaim and B. Kosko. Adaptive joint fuzzy sets for function approximation. In *International Conference on Neural Networks (ICNN-97)*, pages 537–542, June 1997.

[77] J. Moody and C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–293, 1989.

[78] M. C. Morrone, J. R. Ross, and R. A. Owens. Mach bands are phase dependent. *Nature*, 324:250–253, 1986.

[79] A. Nieder, D. Freedman, and E. Miller. Representation of the quantity of visual items in the primate prefrontal cortex. *Science*, 297:1708–1711, 6 September 2002.

[80] K. Nordberg, G. Granlund, and H. Knutsson. Representation and Learning of Invariance. In *Proceedings of IEEE International Conference on Image Processing*, Austin, Texas, November 1994. IEEE.

[81] K. Nordberg, P. Doherty, G. Farnebäck, P.-E. Forssén, G. Granlund, A. Moe, and J. Wiklund. Vision for a UAV helicopter. In *Proceedings of IROS'02, workshop on aerial robotics*, Lausanne, Switzerland, October 2002.

[82] S. Obdrzalek and J. Matas. Object recognition using local affine frames on distinguished regions. In *Proceedings of the British Machine Vision Conference*, pages 113–122, London, 2002. ISBN 1-901725-19-7.

[83] J. K. O'Regan. Solving the 'real' mysteries of visual perception: The world as an outside memory. *Canadian Journal of Psychology*, 46:461–488, 1992.

[84] R. Palm, H. Hellendoorn, and D. Driankov. *Model Based Fuzzy Control.* Springer-Verlag, Berlin, 1996. ISBN 3-540-61471-0.

[85] P. Perona and J. Malik. Detecting and localizing edges composed of steps, peaks and roofs. In *Proceedings of ICCV*, pages 52–57, 1990.

[86] D. Reisfeld. The constrained phase congruency feature detector: simultaneous localization, classification, and scale determination. *Pattern Recognition letters*, 17(11):1161–1169, 1996.

[87] H. Scharr, M. Felsberg, and P.-E. Forssén. Noise adaptive channel smoothing of low-dose images. In *CVPR Workshop: Computer Vision for the Nano Scale*, June 2003.

[88] S. M. Smith and J. M. Brady. SUSAN - a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, 1997.

[89] H. Snippe and J. Koenderink. Discrimination thresholds for channel-coded systems. *Biological Cybernetics*, 66:543–551, 1992.

[90] H. Snippe and J. Koenderink. Information in channel-coded systems: correlated receivers. *Biological Cybernetics*, 67:183–190, 1992.

[91] I. Sobel. Camera models and machine perception. Technical Report AIM-21, Stanford Artificial Intelligence Laboratory, Palo Alto, California, 1970.

[92] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision.* International Thomson Publishing Inc., 1999. ISBN 0-534-95393-X.

[93] H. Spies and P.-E. Forssén. Two-dimensional channel representation for multiple velocities. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, LNCS 2749, pages 356–362, Gothenburg, Sweden, June-July 2003.

[94] H. Spies and B. Johansson. Directional channel representation for multiple line-endings and intensity levels. In *Proceedings of IEEE International Conference on Image Processing*, Barcelona, Spain, September 2003.

[95] C. V. Stewart. Robust parameter estimation in computer vision. *SIAM Review*, 41(3):513–537, 1999.

[96] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction.* MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-19398-1.

[97] S. Thorpe. *The Handbook of Brain Theory and Neural Networks*, chapter Localized Versus Distributed representations, pages 549–552. MIT Press, 1995. M. A. Arbib, Ed.

[98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the 6th ICCV*, 1998.

[99] T. Tuytelaars and L. V. Gool. Matching widely separated views based on affinely invariant neighbourhoods. *International Journal of Computer Vision*, 2003. To appear.

[100] M. Unser. Splines: A perfect fit for signal and image processing. *IEEE Signal Processing Magazine*, pages 22–38, November 1999.

[101] N. Vlassis, B. Terwijn, and B. Kröse. Auxiliary particle filter robot localization from high-dimensional sensor observations. Technical Report IAS-UVA-01-05, Computer Science Institute, University of Amsterdam, The Netherlands, September 2001.

[102] M. Volgushev and U. T. Eysel. Noise Makes Sense in Neuronal Computing. *Science*, 290:1908–1909, december 2000.

[103] J. Weule. *Iteration nichtlinearer Gauss-Filter in der Bildverarbeitung.* PhD thesis, Heinrich-Heine-Universität Düsseldorf, 1994.

[104] G. Winkler and V. Liebscher. Smoothers for discontinuous signals. *J. Nonpar. Statistics*, 14(1-2):203–222, 2002.

[105] WITAS web page.
`http://www.ida.liu.se/ext/witas/`.

[106] A. Witkin. Scale-space filtering. In *8th Int. Joint Conf. Artificial Intelligence*, pages 1019–1022, Karlsruhe, 1983.

[107] A. Wrangsjö and H. Knutsson. Histogram filters for noise reduction. In C. Rother and S. Carlsson, editors, *Proceedings of SSAB'03*, pages 33–36, 2003.

[108] R. Zemel, P. Dayan, and A. Pouget. Probabilistic interpretation of population codes. *Neural Computation*, 2(10):403–430, 1998.

[109] Z. Zhang. Parameter estimation techniques: A tutorial. Technical Report 2676, INRIA, October 1995.