Linköping Studies in Science and Technology
Thesis No. 869

Sparse Representations for Medium Level Vision

Per-Erik Forssén

INSTITUTE OF TECHNOLOGY
LINKÖPINGS UNIVERSITET

LIU-TEK-LIC-2001:06

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

Linköping February 2001

Sparse Representations for Medium Level Vision

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

ISBN 91-7219-951-2 ISSN 0280-7971

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

Zen proverb

Abstract

In this thesis a new type of representation for medium level vision operations is explored. We focus on representations that are *sparse* and *monopolar*. The word sparse signifies that information in the feature sets used is not necessarily present at all points. On the contrary, most features will be inactive. The word monopolar signifies that all features have the same sign, e.g. are either positive or zero. A zero feature value denotes "no information", and for non-zero values, the magnitude signifies the relevance.

A sparse scale-space representation of local image structure (lines and edges) is developed.

A method known as the *channel representation* is used to generate sparse representations, and its ability to deal with multiple hypotheses is described. It is also shown how these hypotheses can be extracted in a robust manner.

The connection of *soft histograms* (i.e. histograms with overlapping bins) to the channel representation, as well as to the use of *dithering* in relaxation of quantisation errors is shown. The use of soft histograms for estimation of unknown probability density functions (PDF), and estimation of image rotation are demonstrated.

The advantage with the use of sparse, monopolar representations in associative learning is demonstrated.

Finally we show how sparse, monopolar representations can be used to speed up and improve template matching.

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:

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

The people at the Computer Vision laboratory, for providing a stimulating research environment, for sharing their 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, for his constructive criticism on early versions of this manuscript.

Johan Wiklund, for keeping the computers happy.

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.

Contents

Chapter 1

Introduction

1.1 Motivation

The work presented in this thesis has been performed within the WITAS[1] project [11, 25]. The goal of the WITAS project is, according to the project web page [62]:

> *... to demonstrate, before the end of the year* 2003*, an airborne computer system that is able to make rational decisions about the continued operation of the aircraft, based on various sources of knowledge including pre-stored geographical knowledge, knowledge obtained from vision sensors, and knowledge communicated to it by radio.*

In other words, the goal is to build an autonomous Unmanned Aerial Vehicle (UAV), that is able to deal with visual input. This thesis will focus on a small subset of computer vision for 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.

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

The WITAS project involves all three levels, but as the title of this thesis suggests, we will mainly deal with medium-level vision. More specifically we will deal with medium-level methods using *sparse*, *monopolar* representations. The word sparse signifies that information in the feature sets used is not necessarily present at all points. On the contrary, most features will be inactive. The word monopolar signifies that all features have the same sign, e.g. are either positive or zero. A zero feature value denotes "no information", and for non-zero values, the magnitude signifies the relevance.

An other property of the sparse data we will use is that it is *locally continuous*. This means that it is possible (and indeed quite likely) that two adjacent statements are true at the same time. This bears resemblance to *fuzzy logic*, where several statements can be simultaneously true, to different degrees. For example, the statements "the water is warm", and "the water is hot" may both be true to different degrees, depending on the actual water temperature.

1.2 Overview

The thesis starts with a brief look at the best autonomous vision systems there are today—biological. Chapter 2 contains a description of some aspects of what is known about biological vision today.

Chapter 3 contains a method to obtain a sparse scale-space representation of low-level image structure. The method is focused on obtaining reliable statements in a limited number of points, rather than statements at all positions. This kind of representation is well suited to instance based learning.

Chapter 4 contains a description of how compact variables may be transformed into the *channel representation* [21, 47, 7, 3], a sparse representation developed for computer vision applications at the Computer Vision laboratory. This chapter also describes how the value or values represented in a channel value vector can be retrieved.

Chapter 5 describes an application of the channel representation called *soft histograms*. It also describes how soft histograms (and other histogram techniques with overlapping bins) can be used to detect peaks in a PDF in a much more accurate and robust manner than what is possible with the same data using conventional histograms. Finally a method to estimate image rotation using soft histograms of a local orientation feature is demonstrated.

Chapter 6 contains various experiments and ideas concerning associative learning using the associative networks [22] developed at the Computer Vision laboratory.

Finally, chapter 7 contains a description of sparse template matching. Sparse template matching is a novel technique that makes use of sparse data to speed up and improve template matching. The method is evaluated on data with varying degrees of sparsity, and compared with the commonly used *Sum of Squared Difference* (SSD) method.

1.3 Contributions

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

- A scale-space representation of lines and edges is implemented and described in chapter 3. This chapter is basically an extended version of the conference paper "Sparse feature maps in a scale hierarchy" [16].
- A framework for channel encoding and local reconstruction of scalar values is presented in chapter 4.
- A demonstration of the connection between *dithering* and overlapping bins in histogram creation is given in chapter 5.
- Applications of *soft histograms* (i.e. histograms with overlapping bins) in the field of image analysis are demonstrated in chapter 5. The applications are: accurate peak detection in a PDF estimated with overlapping bins, and a fast method to estimate image rotation using DFT coefficients of soft histograms.
- A feature selection mechanism for associative learning and aspects of learning rules for associative learning are presented in chapter 6.
- A novel technique that makes use of sparse, monopolar data to speed up and improve template matching is presented in chapter 7.

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
$\langle \mathbf{x} \mid \mathbf{y} \rangle$	The scalar product
$\arg \boldsymbol{z}$	Argument of a complex number
$\operatorname{conj} \boldsymbol{z}$	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
$\operatorname{adist}(\varphi_1 - \varphi_2)$	Angular distance of cyclic variables

Chapter 2

Biological vision systems

2.1 Introduction

Many important problems in computer vision still await robust and reliable solutions. Most animals, and many insects are much better at dealing with visual input than the most sophisticated machine vision systems. Since biological and mechanical systems use different kinds of "hardware", there are of course several important differences, but there is still much to be gained by adopting some of the design principles that biological vision systems adhere to.

This chapter gives a short description of some aspects of biological image interpretation that are likely to be useful in machine vision as well. We will attempt to make use of several of the principles mentioned here in the rest of this thesis.

2.2 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.

2.2.1 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 [6]. As pointed out by e.g. O'Regan [49] 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.

2.2.2 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 fairly recent paradigm of *active vision* 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* [6].

2.2.3 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 apply percept-response association [24].

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.

2.3 Information representation

We will now have a look at how biological systems represent visual information. This is by no means an exhaustive presentation, it should merely be seen as background, and motivation for the representations chosen in the following chapters of this thesis.

2.3.1 Low level biological vision operations

Mammalian vision systems receive their inputs through light sensitive cells called *rods* (those mammals capable of colour vision have an additional class of light sensitive cells called *cones*). The signals from these cells are processed by *bipolar*

and *amacrine* cells, and later by *ganglion* cells, which propagate the information from the retina to a region of the thalamus known as LGN.

One important kind of ganglion cells are the M-type cells. They are present in all mammals, and perform a differentiation of responses from two bipolar cells. The bipolar cells integrate responses from light sensitive cells in *receptive fields* that are approximately circular, with weights that decrease with the distance from the centre. The combined operation of bipolar and ganglion cells is usually modelled as a *Difference of Gaussian*, DoG, computation. That is, each response is computed as a difference between two Gaussian filtered versions of the input. Typically one of the bipolar cells has a significantly more concentrated support than the other. The extreme case in the fovea is one light detecting cell per bipolar cell. The differentiation results in two main types of M-type ganglion, or *centre-surround* cells. One kind produces a response if the centre of the receptive field is brighter than the surrounding region, and the other kind works in the opposite manner [60].

The ganglion cell responses from the left and right eyes are collected and modulated in LGN, and further sent to the primary visual cortex, or V1, where they are arranged in left and right visual fields [4]. In V1, responses that resemble Gabor type wavelets are computed in a wide range of scales.

From an evolutionary point of view, these computations must gain the organism an advantage. One such advantage is, as we shall see in chapter 7, that they aid product sum matching.

2.3.2 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 [32].

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. There thus 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 (in-between the bright, and dark levels for the retinal bipolar cells). This results in bipolar detectors being much more sensitive to disturbances [32]. In chapter 6 we will make use of monopolar representations in associative learning.

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 [20].

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

2.3.3 View centred representations

Biological vision systems interpret visual stimuli by generation of image features in several retinotopic maps [4]. 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 [21], 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 [8].

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 [21], and is also what is used by the human mind in abstract reasoning, and in spoken language. In chapter 3 we will generate feature maps of structural information, that can be used to form a view centered object representation.

2.3.4 Local vs distributed coding

There are three main ways to represent a system state using a number of signals. Consider the following simple example given by Thorpe [59]: We have a stimulus that can consist of a horizontal or a vertical bar, and 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.

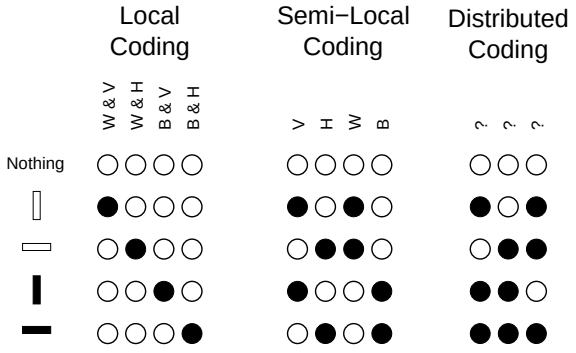


Figure 2.1: Local, semi-local, and distributed coding.
Figure adapted from [59].

One way to represent the state is to assign one signal to each system state. This is called a *local coding* in the figure. One big advantage with local coding is that the system can deal with several hypotheses at once. In the example in the figure, two active responses would mean that there was two bars present in the scene. An other way is to assign one output for each state of the two properties: orientation and colour. This is called *semi-local coding* in the figure. As we move

away from a completely local coding, the ability to deal with several hypotheses gradually disappears. For instance, if we had one vertical, and one horizontal bar, we could deal with them separately using semi-local coding only if they had the same colour.

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

If we view "minimum number of signals" as our goal, we will arrive at a compact coding scheme that is equivalent to data compression.

2.3.5 Sparse signals

In data compression the information in each signal is maximised. But we could also envision an other optimisation goal: maximisation of the information content in the *active* nodes only (see figure 2.2). Something similar to this seems to happen at the lower levels of visual processing in mammals [13]. The result of this kind of optimisation on visual input is a representation that is *sparse*, i.e. most signals are inactive. This is similar to the local, and semi-local coding examples in the previous section.

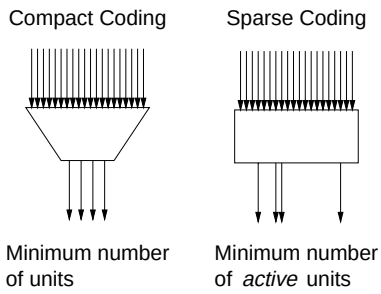


Figure 2.2: Compact vs. sparse coding.
Figure adapted from [13].

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 [13].

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 [13, 40]. 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.

Chapter 3

Lines and edges in scale space

3.1 Background

Biological vision systems are capable of instance recognition in a manner that is vastly superior to current machine vision systems. Perceptual experiments [49, 8] 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. The fact that difference of Gaussians, and Gabor-type wavelets have a correspondence to the first two levels of processing in biological vision systems is widely known [4]. 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 [34]. 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. Another important difference between our approach and others is that we keep different kinds of events separate instead of combining them into one compact feature map.

3.1.1 Classical edge detection

The fact that discontinuities in images convey important information has been known and used for a long time in image processing. One early example that is still widely used are the Sobel edge filters [56]. Another common example is the Canny edge detector [9] that produces visually pleasing binary images. The goal of edge detecting algorithms in image processing is usually to obtain useful input

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

to segmentation algorithms [58], and for this purpose, the ideal step edge detection that the Canny edge detector performs is in general insufficient [53], 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 aid instance recognition), we will discuss conventional edge detection no further.

3.1.2 Phase-gating

The use of characteristic phase as a descriptive image feature originates from the idea of *phase-gating*, originally mentioned in a thesis by Haglund [27]. 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 [27]. Haglund used 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.

3.1.3 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 3.1). 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 [44]. 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*.

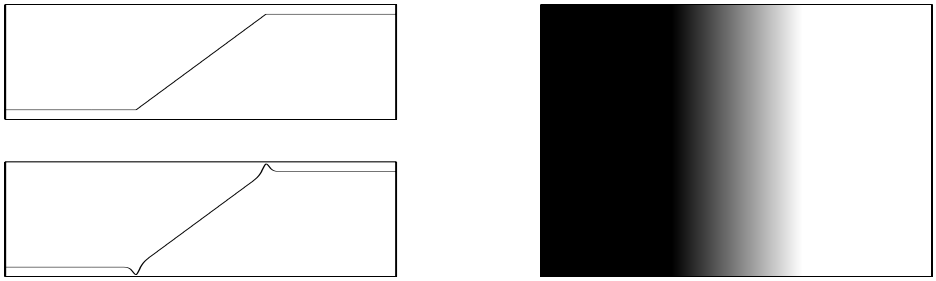


Figure 3.1: Mach bands near a ramp edge.
Top-left: Image intensity profile
Bottom-left: Perceived image intensity *Right: Image*

This observation has lead to the invention of phase congruency feature detectors [37]. 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 [54]. 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 responses in a small spatial range near the characteristic phases.

3.2 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 3.2).

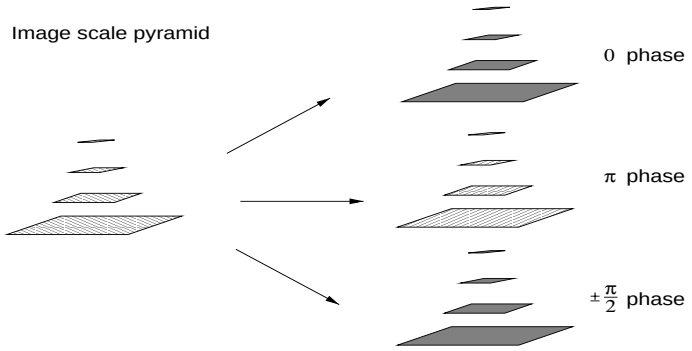


Figure 3.2: Scale hierarchies.

This approach touches the field of scale-space analysis pioneered by Witkin [63]. See [39] for a recent overview of scale space methods. Our approach to scale space analysis is somewhat similar to that of Reisfeld [54]. 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.

3.2.1 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 it's quadrature complement (odd, denoted f_o). These two filters are usually combined into a complex valued *quadrature filter*, $\boldsymbol{f} = f_e + \boldsymbol{i} f_o$ [23]. 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, $\boldsymbol{q}(x)$, 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.

$$R_i(\rho) = \begin{cases} e^{-\frac{\ln^2(\rho/\rho_i)}{\ln 2}} & \text{if } \rho > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

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]

$$f_{e,i}(x) = \mathrm{Re}(\mathrm{IDFT}\{R_i(\rho)\}) \tag{3.2}$$
$$f_{o,i}(x) = \mathrm{Im}(\mathrm{IDFT}\{R_i(\rho)\}) \tag{3.3}$$

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.

3.2.2 Characteristic phase

By *characteristic phase* we mean phases that are consistent over a range of scales, and thus characterise the local image region. In practise this only happens at local magnitude peaks of the responses from the even and odd filters.[3] In other words, the characteristic phases are always one of 0, π, and $\pm\pi/2$.

Only some occurrences of these phases are consistent over scale though (see figure 3.3). 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

[2]Note that there are other ways to obtain spatial filters from frequency descriptions that, in many ways produce better filters [35].

[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.

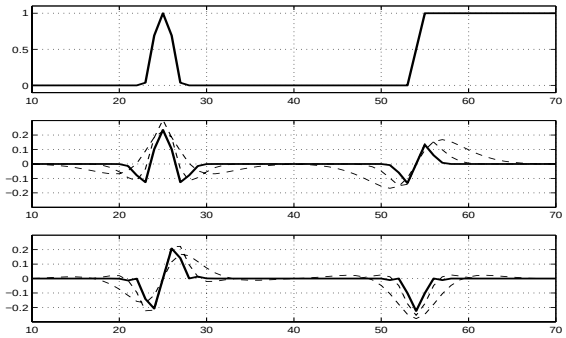


Figure 3.3: Line and edge filter responses in 1D.
Top: *A one-dimensional signal.*
Centre: *Line responses at $\rho_i = \pi/2$ (solid), and $\pi/4$ and $\pi/8$ (dashed)*
Bottom: *Edge responses at $\rho_i = \pi/2$ (solid), and $\pi/4$ and $\pi/8$ (dashed)*

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, and consequently they do not correspond to characteristic phases. 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, are not consistent over scale. Instead they will move as we change frequency bands. This phenomenon can be used to sort out the desired peaks.

3.2.3 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*:

$$p_{1,i} = \max(0, q_{e,i}) \tag{3.4}$$
$$p_{2,i} = \max(0, -q_{e,i}) \tag{3.5}$$
$$p_{3,i} = \mathrm{abs}(q_{o,i}) \tag{3.6}$$

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.

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

$$p_{1,i} = \max(0, q_{e,i} \cdot q_{e,i-1}/a_{i-1}) \cdot \max(0, \mathrm{sign}(q_{e,i})) \tag{3.7}$$
$$p_{2,i} = \max(0, q_{e,i} \cdot q_{e,i-1}/a_{i-1}) \cdot \max(0, \mathrm{sign}(-q_{e,i})) \tag{3.8}$$
$$p_{3,i} = \max(0, q_{o,i} \cdot q_{o,i-1}/a_{i-1}) \tag{3.9}$$

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 equations 3.7-3.9 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 3.4.

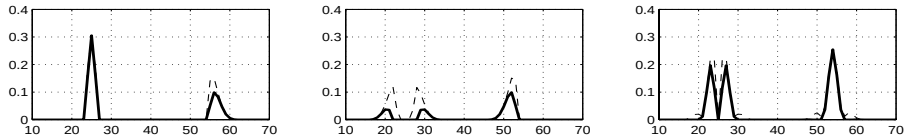


Figure 3.4: Consistent phase in 1D. ($\rho_i = \pi/4$)
$p_{1,i}$, $p_{2,i}$, $p_{3,i}$ according to equations 3.4-3.6 (dashed), and equations 3.7-3.9 (solid)

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:

$$c_{1,i} = \max(0, p_{1,i} - \alpha \text{abs}(q_{o,i-2})) \tag{3.10}$$
$$c_{2,i} = \max(0, p_{2,i} - \alpha \text{abs}(q_{o,i-2})) \tag{3.11}$$
$$c_{3,i} = \max(0, p_{3,i} - \alpha \text{abs}(q_{e,i-2})) \tag{3.12}$$

We have chosen an amount of inhibition $\alpha = 2$, and the base scale, $\rho_i = \pi/4$. With these values we successfully remove the edge responses at the line event, and a 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 3.5.

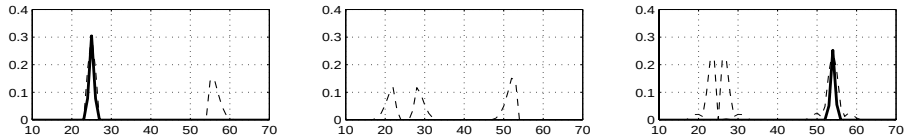


Figure 3.5: Phase channels in 1D. ($\rho_i = \pi/4$, $\alpha = 2$)
$p_{1,i}$, $p_{2,i}$, $p_{3,i}$ according to equations 3.4-3.6 (dashed), and equations 3.10-3.12 (solid).

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 3.6.

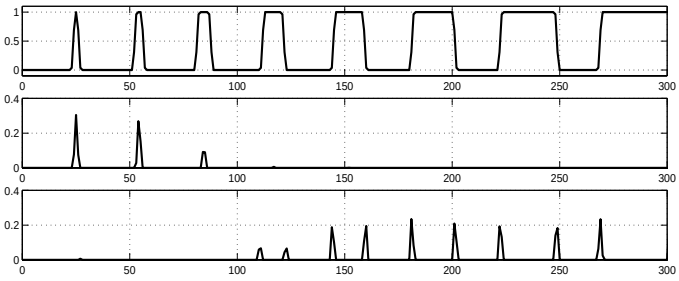


Figure 3.6: Transition between line and edge description. ($\rho_i = \pi/4$)
Top: Signal Centre: $c_{1,i}$ phase channel
Bottom: $c_{3,i}$ phase channel.

3.2.4 Local orientation information

The filters we employ in 2D will be the extension of the lognorm filter function (equation 3.1) to 2D [23]:

$$\mathrm{F_{ki}}(\mathbf{u}) = \mathrm{R_i}(\rho)\mathrm{D_k}(\hat{\mathbf{u}}) \tag{3.13}$$

Where

$$\mathrm{D_k}(\hat{\mathbf{u}}) = \begin{cases} (\hat{\mathbf{u}} \cdot \hat{\mathbf{n}}_k)^2 & \text{if } \mathbf{u} \cdot \hat{\mathbf{n}}_k > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

We will use four filters, with directions $\hat{\mathbf{n}}_1 = (\ 0 \quad 1\ )^t$, $\hat{\mathbf{n}}_2 = (\ \sqrt{0.5} \quad \sqrt{0.5}\ )^t$, $\hat{\mathbf{n}}_3 = (\ 1 \quad 0\ )^t$, and $\hat{\mathbf{n}}_4 = (\ \sqrt{0.5} \quad -\sqrt{0.5}\ )^t$. 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 [23].

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

$$(f_{e,ki} * p_{ki})(\mathbf{x}) \approx \mathrm{Re}(\mathrm{IDFT}\{\mathrm{F_{ki}}(\mathbf{u})\}) \tag{3.15}$$

$$(f_{o,ki} * p_{ki})(\mathbf{x}) \approx \mathrm{Im}(\mathrm{IDFT}\{\mathrm{F_{ki}}(\mathbf{u})\}) \tag{3.16}$$

[4] Note that the fact that both the line, and the edge statements are low near the fourth event (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 [35] to separate 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* [23].

$$\boldsymbol{z}(\mathbf{x}) = \sum_k a_{ki}(\hat{n}_{k1} + \boldsymbol{i}\hat{n}_{k2})^2 = a_{1i}(\mathbf{x}) - a_{3i}(\mathbf{x}) + \boldsymbol{i}(a_{2i}(\mathbf{x}) - a_{4i}(\mathbf{x})) \tag{3.17}$$

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

3.2.5 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 3.6, rotated around the origin (see figure 3.7).

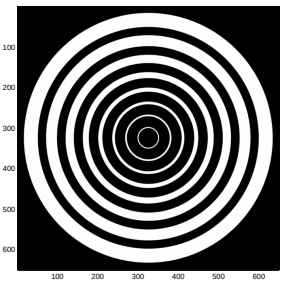


Figure 3.7: A 2D test pattern.

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 [23], 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]

$$\hat{\mathbf{n}}_z = (\ \mathrm{Re}(\sqrt{\boldsymbol{z}}) \quad \mathrm{Im}(\sqrt{\boldsymbol{z}})\ )^t \tag{3.18}$$

[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.

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

$$w_k = \hat{\mathbf{n}}_k^t \hat{\mathbf{n}}_z \tag{3.19}$$

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

$$q_{e,i} = \sum_k q_{e,i,k} \text{abs}(w_k) \tag{3.20}$$

$$q_{o,i} = \sum_k q_{o,i,k} w_k \tag{3.21}$$

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 (equations 3.7-3.9, 3.10-3.12), we now compute three phase channels, $c_{1,i}$, $c_{2,i}$, and $c_{3,i}$, in each scale.

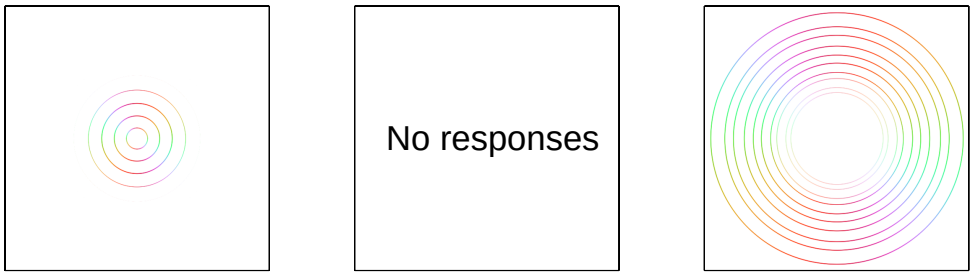


Figure 3.8: Characteristic phase channels in 2D. ($\rho_i = \pi/4$)
Left to right: Characteristic phase channels $c_{1,i}$, $c_{2,i}$, and $c_{3,i}$, according to equations 3.10-3.12 ($\alpha = 2$). The colours indicate the locally dominant orientation.

The characteristic phase channels are shown in figure 3.8.[6] As we can see, the channels exhibit a smooth transition from describing the white regions in the test pattern (see figure 3.7) 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 [54].

3.2.6 Local orientation and characteristic phase

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

[6] The magnitude of lines this thin can be difficult to reproduce in print. However, the magnitudes in this plot *should* vary just like in figure 3.6.

$$\boldsymbol{z}_n(\mathbf{x}) = \begin{cases} 0 & \text{if } c_n(\mathbf{x}) = 0 \\ \dfrac{c_n(\mathbf{x}) \cdot \boldsymbol{z}(\mathbf{x})}{|\boldsymbol{z}(\mathbf{x})|} & \text{otherwise} \end{cases} \tag{3.22}$$

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 3.9. The colours in the figure indicate the locally dominant orientation, just like in figure 3.8. 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.

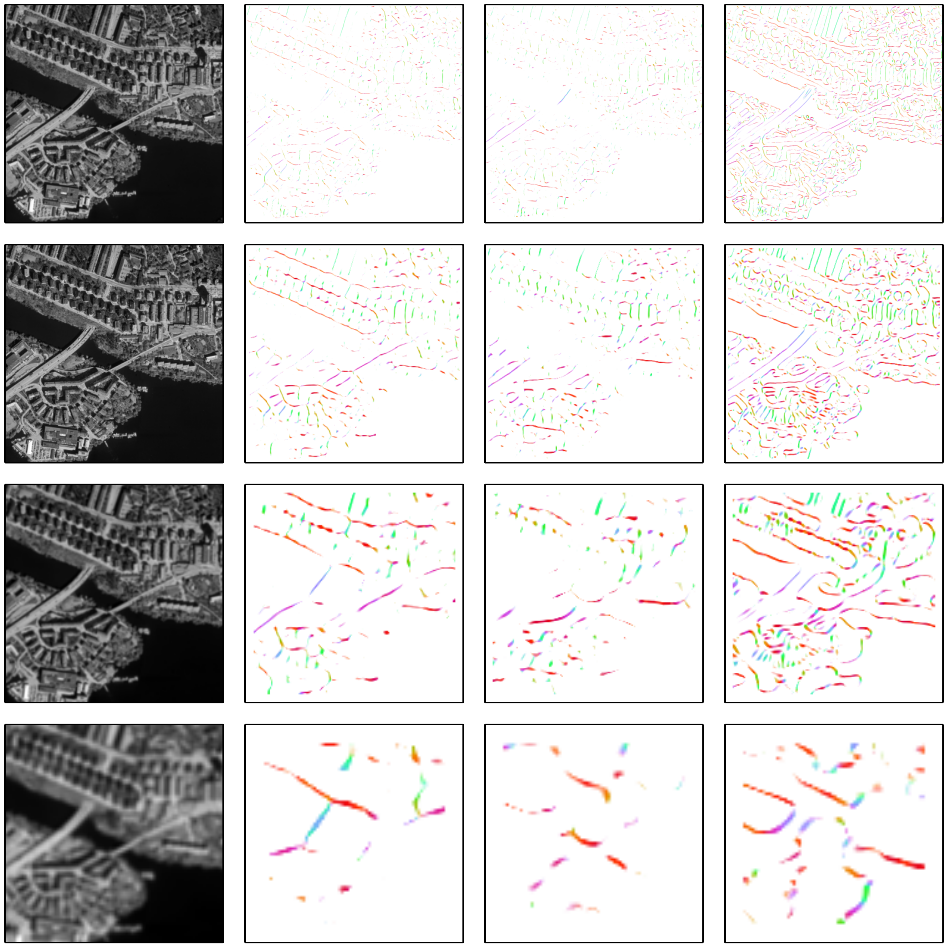


Figure 3.9: Sparse feature hierarchy. ($\rho_i = \{\pi/2, \pi/4, \pi/8, \pi/16\}$)

3.2.7 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 4

Channel representation

4.1 Channel coding

4.1.1 Compact representations

Compact representations (see chapter 2) such as numbers, 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 as input for a system that should learn a complex and unknown relationship between two sets of data. 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. A better approach is to study the problem locally. An other motivation for local learning is that most complex functions can be sufficiently well approximated as locally linear, and linear relationships are easy to learn (see chapter 6 for more on local learning).

4.1.2 Channel representation of scalars

The channel representation [21, 7, 47] is a good first step towards a better representation of the inputs. When moving from a compact, numerical representation to the channel representation, we project our number onto a set of band-pass functions, $\psi_n(s)$. These functions are zero along most of the real axis, and raise smoothly to 1 near a specific scalar value n:

$$\psi_n(s) = \begin{cases} \cos^2(\omega(s-n)) & |s-n| < \frac{\pi}{2\omega} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

If we distribute our basis functions with unit distance, i.e. $n \in \mathrm{N}$, the parameter ω can be used to control the correlation (or overlap), between neighbouring channel values. For this reason the ω parameter is called the *channel overlap*.

A concrete example is always illustrative, and we will thus now encode the scalar $s = 5.23$, with $\omega = \pi/3$ (See figure 4.1).

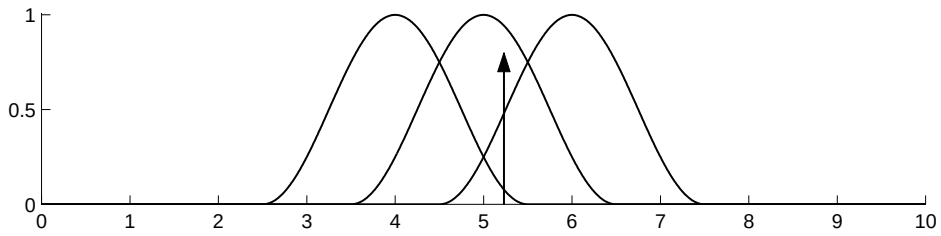


Figure 4.1: Channel encoding.
The basis functions $\psi_4(s)$ through $\psi_6(s)$ are plotted, along with the scalar s ($\omega = \pi/3$).

We will place the resultant coefficients in a *channel value vector* $\mathbf{c}$:

$$\begin{aligned}\mathbf{c} &= \begin{pmatrix} \psi_1(s) & \psi_2(s) & \psi_3(s) & \psi_4(s) & \psi_5(s) & \psi_6(s) & \psi_7(s) & \psi_8(s) \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & 0.0778 & 0.9431 & 0.4791 & 0 & 0 \end{pmatrix}\end{aligned}$$

Since the values of the basis functions only depend on the distance between the scalar s, and the channel centres (i.e. the basis functions are symmetric), we could also view the process as a sampling of a basis function with the centre at the scalar value (See figure 4.2).

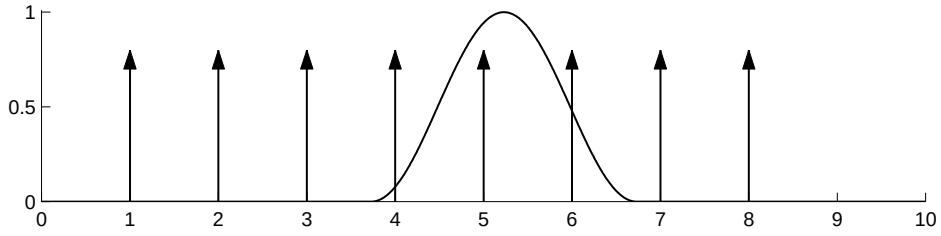


Figure 4.2: Channel encoding.
A basis function $\psi_s(x)$ is sampled at the channel centres $x = 1, 2, 3 \ldots$.

As we can see in figure 4.2, the basis function will always activate several samples. As long as there is more than one active sample, reconstruction of the scalar should be possible. This corresponds to situations when the frequency of the wave function, *within it's non-zero window*, is below the Nyquist frequency. Using the sampling theorem as a heuristic we can conclude that reconstruction is possible as long as $\omega \leq \pi/2$.[1] In practise however, a higher degree of redundancy is preferable, since this will improve our robustness to noise in the reconstruction.

[1]The reason for this is that $\cos^2(\omega(x-n)) = \frac{1}{2}(1 + cos(2\omega(x-n)))$ i.e. the frequency is $f = \frac{2\omega}{\pi}$. The frequency requirement $f \leq 1$ gives $\omega \leq \frac{\pi}{2}$. See also theorem A.2 for a relation between the number of activated channels, N, and the overlap, ω.

The fact that reconstruction is possible is important, since this guarantees that we have not destroyed any information when encoding our scalar.

Each of the channel values in $\mathbf{c}$ states something much more specific than the original scalar s did. Mere activity of a channel means that we know approximately where s lies. This fact makes the channel representation very useful in associative learning, as we will see in chapter 6.

4.1.3 Metamerism

Since each scalar will only activate channels in a limited range, most of the channels in a channel value vector will usually be zero. This means that for a large channel value 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.

Consider the case where you have trained a system to estimate the horizontal position of a face in an image. What happens when this system encounters two faces in the same image? A compact representation can only give one response, and in theory it could choose to respond with either of the two locations, but in practise it will most likely return their average. And what should the system do when there is no face in the image?

Both these problems are dealt with in an elegant manner by the channel representation. If the faces are far enough apart, the system could return two responses in the same channel value vector. If, on the other hand, there was no face in the image, the channel values would simply drop to zero.

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*, and the states (one poking finger, or two close poking fingers) that cannot be distinguished in the given representation are called *metamers*.

The smallest distance between sensations that the system can handle is called the *metameric distance*, and is limited by the distance between the sensors (or in our case, the channels).

4.2 Local reconstruction

In order to evaluate the performance of a learning system, we need to be able to perform reconstruction. That is, we need to be able to tell what scalar, or scalars, the channel value vector represents.

4.2.1 Reconstruction using wavelet theory

If we use wavelet terminology, the encoding of a scalar as a channel value vector can be seen as a set of scalar products with *analysing wavelets* or *dual basis functions*. If we define the scalar to be encoded as a Dirac,

$$\mathrm{g_s}(x) = \delta(x - s) \tag{4.2}$$

and use the scalar product

$$\langle\, \mathrm{f}(x) \,|\, \mathrm{g}(x) \,\rangle = \int \mathrm{f}(x)\mathrm{g}(x)\; dx \tag{4.3}$$

the scalar product between the scalar function, $\mathrm{g_s}(x)$, and an analysing wavelet $\psi_k(x)$ becomes

$$\langle\, \mathrm{g_s}(x) \,|\, \psi_k(x) \,\rangle = \int \delta(x - s)\psi_k(x)\; dx = \psi_k(s) \tag{4.4}$$

The channel encoding can thus be expressed as

$$\mathbf{c} = \begin{pmatrix} \langle\, \mathrm{g_s}(x) \,|\, \psi_1(x) \,\rangle & \langle\, \mathrm{g_s}(x) \,|\, \psi_2(x) \,\rangle & \dots & \langle\, \mathrm{g_s}(x) \,|\, \psi_K(x) \,\rangle \end{pmatrix} \tag{4.5}$$

Since the scalar encoding process in general is a non-orthogonal transform (except when $\omega = \pi/2$) reconstruction of the scalar should be performed through a weighted summation of the channel values, and the *reconstruction wavelets* or *basis functions*[2], $\tilde{\psi}_n(x)$:

$$\mathrm{g}(x) = \sum_{n=1}^{N} u_n \tilde{\psi}_n(x) \tag{4.6}$$

The basis functions, $\tilde{\psi}_n(x)$, can be computed as linear combinations of the dual basis functions, $\psi_n(x)$:

$$\begin{pmatrix} \tilde{\psi}_1(x) \\ \tilde{\psi}_2(x) \\ \vdots \\ \tilde{\psi}_N(x) \end{pmatrix} = \mathbf{G}^{-1} \begin{pmatrix} \psi_1(x) \\ \psi_2(x) \\ \vdots \\ \psi_N(x) \end{pmatrix} \tag{4.7}$$

The weights in the linear combination are given by:

$$\mathbf{G} = \begin{pmatrix} \langle\, \psi_1 \,|\, \psi_1 \,\rangle & \langle\, \psi_1 \,|\, \psi_2 \,\rangle & \dots & \langle\, \psi_1 \,|\, \psi_N \,\rangle \\ \langle\, \psi_2 \,|\, \psi_1 \,\rangle & \langle\, \psi_2 \,|\, \psi_2 \,\rangle & \dots & \langle\, \psi_2 \,|\, \psi_N \,\rangle \\ \vdots & \vdots & \ddots & \\ \langle\, \psi_N \,|\, \psi_1 \,\rangle & \langle\, \psi_N \,|\, \psi_2 \,\rangle & \dots & \langle\, \psi_N \,|\, \psi_N \,\rangle \end{pmatrix} \tag{4.8}$$

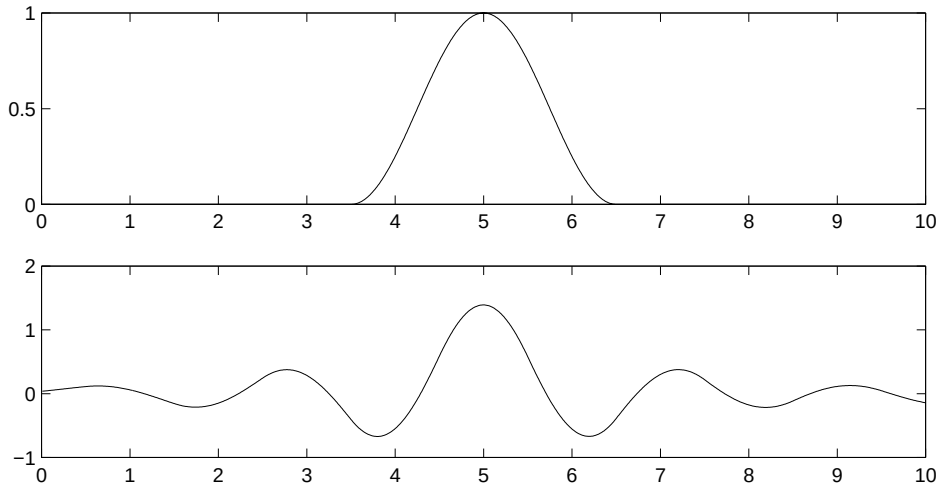


Figure 4.3: An analysing wavelet, and the corresponding reconstructing wavelet.
Top: *An envelope function with $\omega = \pi/3$.*
Bottom: *The corresponding reconstruction function.*

For an orthogonal basis, only the elements in the diagonal of $\mathbf{G}$ will be non-zero, since all other scalar products are zero by definition. Thus, for orthogonal bases, the basis is identical to the dual basis, except for a scaling.

An example of a reconstruction function is shown in figure 4.3. Here a set of envelope functions, $\psi_k(s)$, with $\omega = \pi/3$, were constructed, and corresponding reconstruction functions were computed according to equations 4.7 and 4.8. As we can see, the resultant reconstruction functions have much larger support than the original functions.

The reason for this is that the channel envelope function (see figure 4.2) has unlimited frequency content due to its finite spatial support, and thus cannot be represented by conventional sampling. The reconstruction function in figure 4.3 will thus reconstruct the projection of the envelope function onto the subspace of band-limited functions.

The envelope functions can in fact never be represented well using this terminology since they are non-zero only in a limited interval, and thus cannot be band limited.

4.2.2 The need for a local inverse

Reconstruction using wavelet theory can be made to work when the channel values correspond to one event. However, we want to be able to store several values within a channel set, and the reconstruction should naturally be able to extract

[2]The channel values are coordinates in this basis. Since our annotations are based on the channel values, this is the basis, and the analysing wavelets constitute the dual basis.

them all. As soon as there is more than one event, the events will interfere and cause reconstruction errors, no matter how far apart the events are.

We can easily see that there is a better way, if we look at how a scalar is projected onto the basis functions. For any value of the overlap we can decompose the real axis into non-overlapping intervals in which exactly N basis functions are active at one time (see figure 4.4). Since only these N basis functions are needed to describe this local region, the reconstruction of a scalar *within that region* need only consider these N basis functions.

In fact, if we want to allow several hypotheses, we *should* only consider these N basis functions, since we will otherwise introduce unnecessary interference between the hypotheses.

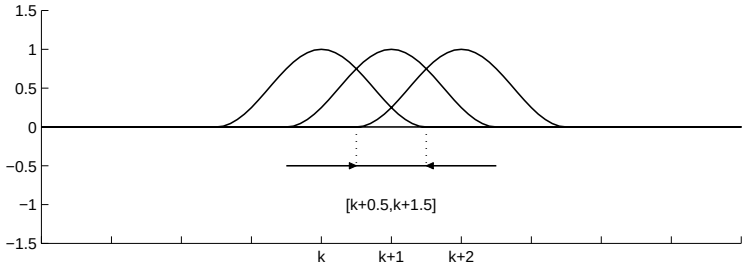


Figure 4.4: Valid range for local inverse. ($\omega = \pi/3$)

We will term the operation of performing reconstruction within one of these intervals a *local inverse*, a reconstruction that is only valid within a limited range of scalar values.

4.2.3 Computing a local inverse

The local inverse can be computed using an idea illustrated in figure 4.5. The channel values are now seen as samples from an envelope function which peaks at the scalar value s. Before we can present the solution however, we have to introduce some notations.

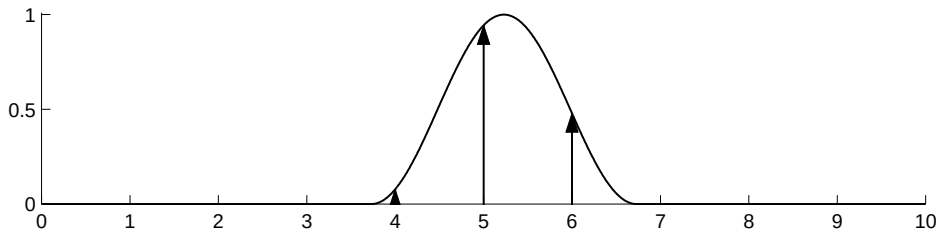


Figure 4.5: Example of channel values.
In this example, $\omega = \pi/3$*, and* $s = 5.23$

The first active channel will be called k (in the figure we have $k = 4$), and the number of active channels will be called N (in the figure we have $N = 3$).

In the computation of the local inverse, we will allow the channel values to be scaled by a factor α, since this will increase the robustness of the scalar reconstruction. For the N non-zero channels we can now formulate the following equation:

$$\mathbf{c} = \frac{1}{\alpha}\begin{pmatrix} c_k \\ c_{k+1} \\ \vdots \\ c_{k+N-1} \end{pmatrix} = \begin{pmatrix} \psi_k(s) \\ \psi_{k+1}(s) \\ \vdots \\ \psi_{k+N-1}(s) \end{pmatrix} \tag{4.9}$$

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

$$c_{k+d} = \psi_k(s) = \cos^2(\omega(s-k-d)) \tag{4.10}$$

$$c_{k+d} = 0.5 + 0.5\cos(2\omega(s-k-d)) \tag{4.11}$$

$$2c_{k+d} = 1 + \cos(2\omega(s-k))\cos(2\omega d) + \sin(2\omega(s-k))\sin(2\omega d) \tag{4.12}$$

$$2c_{k+d} - 1 = \begin{pmatrix} \cos(2\omega d) & \sin(2\omega d) \end{pmatrix}\begin{pmatrix} \cos(2\omega(s-k)) \\ \sin(2\omega(s-k)) \end{pmatrix} \tag{4.13}$$

And thus the entire equation system can be written as:

$$\underbrace{\begin{pmatrix} 2c_k - 1 \\ 2c_{k+1} - 1 \\ \vdots \\ 2c_{k+N-1} - 1 \end{pmatrix}}_{\mathbf{b}} = \underbrace{\begin{pmatrix} \cos(2\omega 0) & \sin(2\omega 0) \\ \cos(2\omega 1) & \sin(2\omega 1) \\ \vdots & \vdots \\ \cos(2\omega(N-1)) & \sin(2\omega(N-1)) \end{pmatrix}}_{\mathbf{A}}\begin{pmatrix} \cos(2\omega(s-k)) \\ \sin(2\omega(s-k)) \end{pmatrix} \tag{4.14}$$

This system can be solved using a least-squares fit:

$$\begin{pmatrix} \cos(2\omega(s-k)) \\ \sin(2\omega(s-k)) \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = (\mathbf{A}^t\mathbf{A})^{-1}(\mathbf{A}^t\mathbf{b}) \tag{4.15}$$

Finally, the sought scalar can be computed as:

$$s = k + \frac{1}{2\omega}\arg\left[d_1 + \boldsymbol{i}d_2\right] \tag{4.16}$$

Now we have to remember that this is a *local inverse*. The solution is thus only valid in a limited range. In theorem A.1 in the appendix the valid range is shown to be $k + N - 1 - \frac{\pi}{2\omega} \leq s \leq k + \frac{\pi}{2\omega}$.

The scaling factor α in equation 4.9 will be reflected in the magnitude of the vector $d_1 + \boldsymbol{i}d_2$:

$$\alpha = |d_1 + \boldsymbol{i}d_2| \tag{4.17}$$

If the channel value vector is the output of an associative learning network, this value can be used as a confidence measure for this local solution.

4.2.4 Local bases

We will now reconnect to wavelet theory by viewing the local inverse (equation 4.16) as a projection of coordinates onto basis functions.

The transpose of the matrix $\mathbf{A}$ (equation 4.14) can be seen as the dual basis matrix for the coordinates $\mathbf{b}$.[3] Each row in $\mathbf{A}$ can be seen as a complex exponential. The full dual basis can thus be visualised as vectors on a spiral (see figure 4.6) where each function only sees other functions within its horizon, or local range. The local dual basis (i.e. $\mathbf{A}^t$) is thus a cut-out of this spiral, corresponding to the interval we want to investigate for a solution.

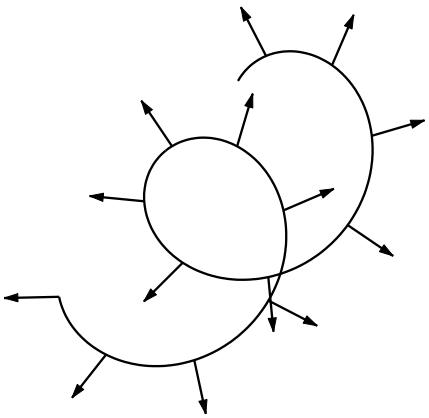


Figure 4.6: Complex exponentials viewed as vectors on a spiral.

The matrix $(\mathbf{A}^t\mathbf{A})^{-1}\mathbf{A}^t$ from equation 4.15 can be viewed as consisting of the basis vectors corresponding to the coordinates $\{b_1,\ b_2,\ \ldots\ b_N\}$. These can be transformed into complex vectors $\boldsymbol{v}_d$ as follows:

$$\mathbf{B} = (\mathbf{A}^t\mathbf{A})^{-1}\mathbf{A}^t\ , \quad \boldsymbol{v}_d = B_{1,d} + \boldsymbol{i}B_{2,d} \tag{4.18}$$

The local inverse can now be written as:

[3] The coordinates $\mathbf{b}$ correspond to our channel values as $\mathbf{b} = 2\mathbf{c} - 1$ (see equation 4.14).

$$s = k + \frac{1}{2\omega}\arg\left[\sum_{d=1}^{N} b_d \boldsymbol{v}_d\right] \tag{4.19}$$

For all local inverses that depend on the same number of channel values, the matrix $\mathbf{B}$ will be identical, so it only has to be computed once.

4.2.5 A local tight frame

There is an interesting special case for which the local inverse becomes very simple. Whenever $\omega = \pi/N$ for integers $N \geq 3$, $\mathbf{A}^t\mathbf{A}$ becomes $\frac{N}{2}\mathbf{I}$. This means that, the basis, $\mathbf{B}$, is a mere scaling of the dual basis $\mathbf{A}^t$, more precisely $\mathbf{B} = \frac{2}{N}\mathbf{A}^t$.

A basis is called a *tight frame* if it equals its dual globally, except for a scaling [10]. In the same spirit we will call this a *local tight frame*. Since the norm of the channel values is constant for $\omega = \pi/N$ where N is an integer $N \geq 3$ (see theorem A.4), our local tight frames are true tight frames as well, as long as there is only one scalar encoded in the channel vector.

Since $\mathbf{B} = \frac{2}{N}\mathbf{A}^t$, we can compute the local inverse as a local weighted summation of the original basis functions:

$$\mathrm{h}(s) = \sum_{n=k}^{k+N-1} u_n \psi_n(s) \tag{4.20}$$

Now the complex numbers $\boldsymbol{v}_d$ in equation 4.19 can be expressed directly as exponentials, using the definition of $\mathbf{A}$ in equation 4.14:

$$\boldsymbol{v}_d = \cos(2\omega d) + \boldsymbol{i}\sin(2\omega d) = e^{\boldsymbol{i}2\omega d} \tag{4.21}$$

This approach is quite similar to the scalar reconstruction used in [47]. However, our reconstruction is local whereas the one described in [47] is global, and thus only allows one hypothesis.

An other important thing that occurs when $\omega = \frac{\pi}{N}$ is that all valid ranges for groups of N channel values become the same size, and they only overlap at a single scalar value (see theorem A.1). This means that we could implement the reconstruction as a `for` loop where all consecutive groups of N channel values are checked for a solution.

4.3 Some other local model techniques

We will now have a look at two classes of similar techniques that have evolved in parallel to the channel representation. The description of the two techniques (Radial Basis Function networks, and adaptive fuzzy control) are not meant to be exhaustive, the purpose of the presentation is merely to acknowledge their existence, and to highlight how they differ from the channel representation.

4.3.1 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 [50, 29]. 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 in principle is equivalent to the channel representation. The advantage with this 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.

Related to RBF networks are hierarchies of local Gaussian models. Such networks have been investigated by for instance Landelius in [38]. His setup allows new models to be added where needed, and unused models to be removed.

4.3.2 Adaptive fuzzy control

To reduce learning time it is advantageous to define the localised inputs beforehand. One example of pre-defined sets of local inputs in learning is adaptive controllers in the field of *fuzzy control*, see for instance [51] for and overview of fuzzy control. In fuzzy control a set of local rules between measurements, and desired outputs are established. These are 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*. The `IF-THEN` rules can be learned by a neural network, see for instance [42]. This kind of learning is however only able to solve function-approximation-like problems, since the methods implicitly assume one global response in the scalar reconstruction (or *defuzzification*) phase. They also require that each local state on the input side can be defined by a single feature function (or `IF`-part membership function).

Chapter 5

Soft histograms

We will now describe an application of the channel representation called *soft histograms*. Soft histograms is a special case of the *kernel density estimators* developed by Parzen and Rosenblatt [55, 52]. The presentation given here is not meant to be a rigorous theoretical framework, but rather a description of how soft histograms can be used in engineering-like situations.

5.1 Background

The purpose of a histogram is to estimate how the values of a variable is distributed across a certain range, i.e. to estimate a probability density function (PDF). The most common use of histograms in image analysis is estimation of the intensity distribution.

When computing a conventional histogram, the range of values for the data is separated into a set of disjoint *bins*. For each bin one counts the number of samples that fall into its range. If we call the bin centres m_k, and the bin width (and bin distance) d, the histogram value for bin number k can be written as:

$$h_k = \sum_{n=1}^{N} H_k(s_n) \quad \text{where} \quad H_k(s_n) = \begin{cases} 1 & \text{if } |s_n - m_k| < d/2 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Here s_n are samples of the variable under study, and N is the number of samples of this variable. The histogram creation procedure can be seen as an initial quantisation of the samples s_n, followed by a summation. Unless the variable under study is already quantised (as is normally the case for image intensities), the histogram creation introduces an effect similar to aliasing. We can see this by viewing the histogram creation as a band limitation of the PDF, followed by a sampling. The equivalent of a band-limitation function is $H_k(s)$, which corresponds to a sinc() in the Fourier domain.

The fact that the above described histogram creation in some sense violates the sampling theorem limits the uses of a histogram. We will now describe a method

to generate more useful histograms, and to illustrate some ways they can be put to use.

5.1.1 Dithering

Since the histogram creation process contains an inherent quantisation, it could probably benefit from *dithering* [26]. Dithering is the process of adding a small amount of noise (with certain characteristics) to a signal prior to the quantisation. Dithering is commonly used in image reproduction with a small number of available intensities or colours, as well as in quality improvement of digital audio [31].

The initial probability for a sample to fall into a certain bin is 1 inside the bin interval (see figure 5.1, left). However, when we add triangular noise, or *TPDF noise*[1] (see figure 5.1, right) we end up with *stochastic bins*, with PDFs that are smooth, and slightly overlapping (see figure 5.1, centre).

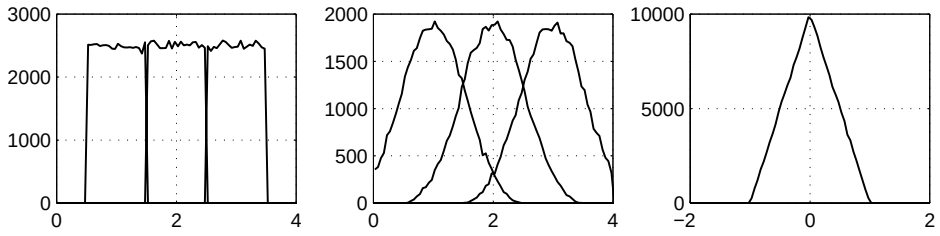


Figure 5.1: Stochastic bins.
Left: *Estimated PDFs of bins* $1 \ldots 3$.
Centre: *Estimated PDFs when noise is added before quantisation.*
Right: *Estimated PDF of added noise.*

5.1.2 Overlapping bins

It is interesting to note that the shape of the stochastic bins in figure 5.1 show a strong resemblance to the $\cos^2()$ envelope functions in the previous section. However, they are not quite identical. The theoretical shape of the PDF is that of a rectangular PDF convolved with a triangular one, and this shape can be represented using piecewise second order polynomials. They are however of similar shape, and are both spatially limited.

In quantisation we are usually forced to choose one bin, due to the compact representations of numbers. For such situations, changing the bins into stochastic bins is a good idea, since on the average we will get overlapping bins, and thus reduced amount of "aliasing".

However, since we are now creating histograms, we could just as well generate deterministic, but overlapping bins. For this purpose we will employ the channel

[1] TPDF noise is designed to de-correlate the power spectrum of the quantisation error and that of the signal [31]. It can be generated by summation of two uniformly distributed random variables.

representation, and compute what is called *soft histograms*.

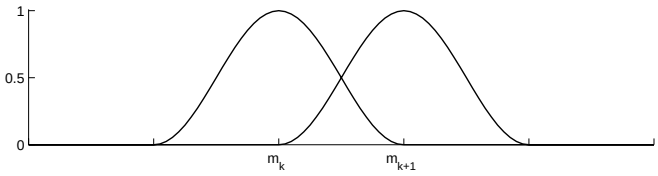


Figure 5.2: Overlapping "bins".

Instead of letting each sample fall into one of the bins, we will allow it to fall into two or more neighbouring bins, but only partially (see figure 5.2). Using the same notation as in equation 5.1, the histogram value for bin (or rather channel) number k is now written as:

$$h_k = \sum_{n=1}^{N} \psi_k(s_n) \quad \text{where} \quad \psi_k(s_n) = \begin{cases} \cos^2\left(\frac{\pi}{2w}(s_n - m_k)\right) & \text{if } |s_n - m_k| < d \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

This approach falls into the class of *kernel density estimators* [55, 52] developed in the late 50's and early 60's by Rosenblatt and Parzen. Parzen prescribes a general kernel function instead of the rectangular function $\mathrm{H}_k(s)$ in equation 5.1.

In relation to the original channel definition in equation 4.1, equation 5.2 has the equivalent of a channel overlap of $\omega = \frac{\pi}{2}$.

Note that, contrary to the conventional approach we have not destroyed any information when letting s_n fall into two bins. We can see this by reconstructing the sample s_n from the two non-zero $\psi_k(s_n)$ contributions. If the two bins with non-zero contributions are denoted k, and $k+1$, the difference between their contributions can be written as:

$$\begin{aligned} \psi_k(s_n) - \psi_{k+1}(s_n) &= \cos^2\left(\frac{\pi}{2d}(s_n - m_k)\right) - \cos^2\left(\frac{\pi}{2d}(s_n - m_{k+1})\right) = \\ &= \cos^2\left(\frac{\pi}{2d}(s_n - m_k)\right) - \sin^2\left(\frac{\pi}{2d}(s_n - m_k)\right) = \\ &= \cos\left(\frac{\pi}{d}(s_n - m_k)\right) \end{aligned}$$

since $m_{k+1} = m_k + d$. The sample value can thus be reconstructed as:

$$s_n = m_k + \frac{d}{\pi} \arccos(\psi_k(s_n) - \psi_{k+1}(s_n)) \tag{5.3}$$

As soon as we start to use more than one sample in equation 5.2 however, there is no way back.

If we already have computed a full conventional histogram, with bins $h_1, h_2, \ldots h_K$, and corresponding bin centres $m_1, m_2, \ldots m_K$, it can easily be converted into a soft histogram $c_1, c_2, \ldots c_L$, according to:

$$c_l = \sum_{k=1}^{K} h_k \psi_l(m_k) \tag{5.4}$$

That is, we compute the value of each envelope function, $\psi_l(x)$, once for each bin in the conventional histogram, and multiply the result with the number of times this bin was visited. This is normally much faster than a full soft histogram computation, since the number of bins in the full histogram, K, usually is much smaller than the number of samples, N. If we ensure that K is significantly larger than the number of bins, L, the quantisation effects in the original histogram will be negligible.

5.1.3 Aliasing in conventional histograms

Figure 5.3 illustrates the "aliasing effect" of conventional histogram computation. Both graphs show 10 superimposed histograms (with 50 bins each) that only differ in construction by a displacement. As can be seen, the soft histogram varies smoothly with the bin-centre displacement, while conventional histograms exhibit abrupt jumps in the bin values. This effect is quite similar to what happens during sampling with insufficient band-limitation.

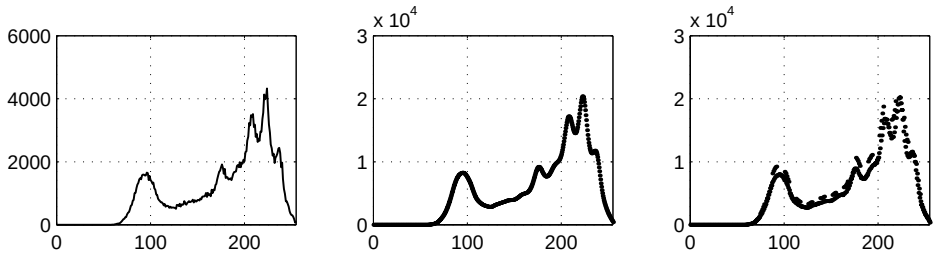


Figure 5.3: Comparison of conventional and soft histograms.
Left: *Image histogram with 256 bins.*
Centre: 10 *superimposed soft histograms with* 50 *bins each.*
Right: 10 *superimposed conventional histograms with* 50 *bins each.*

The extent of the aliasing effects depend highly on the characteristics of the underlying distribution. Cases where the effect is either smaller or more severe than shown in figure 5.3 are easy to find.

5.2 Finding peaks in a soft histogram

In many applications it is of interest to know at which values we have local peaks in a distribution. Due to the aliasing-like effects in conventional histograms, a local peak in the histogram does not necessarily correspond to a peak in the distribution. Provided that we have detected a peak, m_k, in the histogram, we can only say that

it is *likely* that the PDF peak lies somewhere in the range $[m_k - d/2, m_k + d/2]$. If we were using conventional histograms, we would have to reduce the bin sizes and increase the number of samples in order to improve the accuracy, something which is not always possible.

With soft histograms however, the location of the peaks can be found with an accuracy that is higher than the bin distance. The bin distance mainly limits how close two peaks may be. If they are too close (within the *metameric distance*, see section 4.1.3), they will tend to interfere, and eventually their average will be found instead.

As we saw in equation 5.3 it would be sufficient to use two neighbouring bin values for reconstruction *if we only had one sample of the distribution*. However, for other situations we have to consider at least three consecutive bin values. We only have to consider bin triplets for which the middle bin value is larger than its neighbours, and for those cases we can find the peak by modelling a local interval of the PDF as a second order polynomial:

$$h = l_1 s^2 + l_2 s + l_3 \tag{5.5}$$

For convenience, we label the position of the middle bin in the triplet m_p, and centre the polynomial around it, by using the variable $m = s - m_p$. This gives us the bin positions $-d$, 0, and d. We can now find the unknown parameters l_1, l_2, and l_3, using the positions and values of the histogram bins $p-1$, p, and $p+1$. This gives us the following equation system:

$$\begin{pmatrix} h_{p-1} \\ h_p \\ h_{p+1} \end{pmatrix} = \begin{pmatrix} d^2 & -d & 1 \\ 0 & 0 & 1 \\ d^2 & d & 1 \end{pmatrix} \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix} \tag{5.6}$$

This system has the following closed solution:

$$\begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{1}{2d^2} & \frac{-1}{d^2} & \frac{1}{2d^2} \\ \frac{-1}{2d} & 0 & \frac{1}{2d} \\ 0 & 1 & 0 \end{pmatrix}}_{\mathbf{D}} \begin{pmatrix} h_{p-1} \\ h_p \\ h_{p+1} \end{pmatrix} \tag{5.7}$$

The advantage of this formulation is that the matrix $\mathbf{D}$ can be reused as long as d stays constant, i.e. at least throughout the entire histogram.

The peak of the polynomial defined by l_1, l_2, and l_3 can be found by detecting when the derivative is zero:

$$h = m^2 l_1 + m l_2 + l_3 \quad \Rightarrow \quad \frac{\partial h}{\partial m} = 2 m l_1 + l_2 = 0 \quad \Rightarrow \quad m = -\frac{l_2}{2 l_1}$$

We can now locate the extreme point to:

$$\hat{s} = m + m_p \tag{5.8}$$

Since we have limited our search to bin triplets where the middle bin is larger than its neighbours, we know that this is a maximum.

Figure 5.4 illustrates the application of this peak detection scheme to soft histograms with 13, 25, 50, and 100 bins respectively. Note how the peaks near $s = 215$ are seen as one peak in the first histogram, and gradually change into three peaks in the last one.

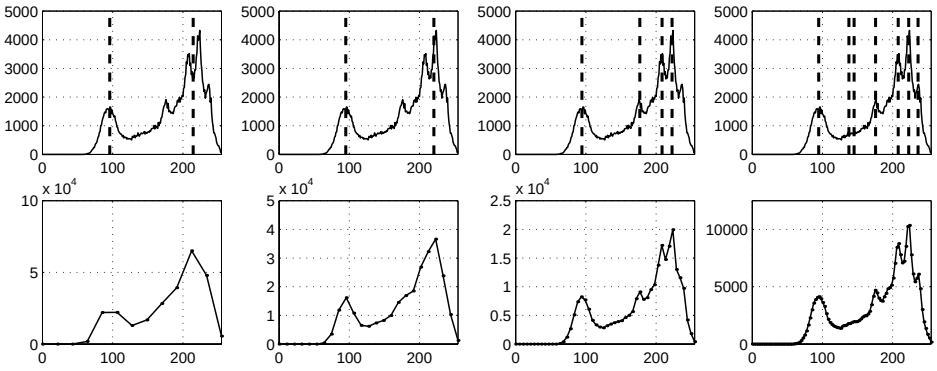


Figure 5.4: Detection of histogram peaks using soft histograms.
Top row: *Detected peaks plotted in a 256-bin histogram.*
Bottom row: *Soft histograms used (*13*,* 25*,* 50*, and* 100 *bins respectively).*

The accuracy of the result is of course highly dependent on the number of samples, N. Given the same set of samples, the accuracy should however always be at least as good as that of a conventional histogram.

5.3 Soft histograms of vector fields

An other application of soft histograms is detection of image rotation. This can be done using a property that is equivariant with rotation, such as local orientation (see chapter 3), or the DIV features described in [34]. We will now illustrate the method using soft histograms of local orientation, but DIV features work just as well.

We can construct a soft histogram (with K bins) of the double-angle local orientation feature $\boldsymbol{z}_n = m_n e^{\boldsymbol{i}\varphi_n}$ as follows:

$$h_k = \sum_{n=1}^{N} m_n \psi_k(\varphi_n) \tag{5.9}$$

$$\text{where} \quad \psi_k(\varphi_n) = \begin{cases} \cos^2\left(\frac{\omega}{d}(\varphi_n - \alpha_k)\right) & \text{if adist}(\varphi_n - \alpha_k) < \frac{\pi d}{2\omega} \\ 0 & \text{otherwise} \end{cases}$$

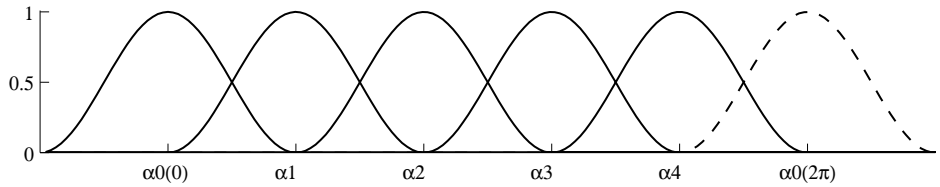


Figure 5.5: Modular channels ($\omega = \pi/2$).

That is, we weight the contribution from each orientation value, φ_n, with the local orientation magnitude m_n. Since the angle φ_n only assumes values in the range $[0, 2\pi[$, we specify our bin distances as:

$$d = \frac{2\pi}{K} \quad \text{and our bin centres as} \quad \alpha_k = d(k-1)$$

Note that since φ_n is a modular variable, we have replaced the absolute distance in equation 5.2 with the angular distance. This implies that the first, and the last channels are neighbours (see figure 5.5).

If we have two such orientation histograms from two images that differ only by a rotation, we can compute this rotation from the phases of the discrete Fourier transforms of the two histograms. For this application it is important that the channel overlap ω is chosen to be $\omega = \pi/n$ where n is an integer $n \geq 2$, since this will ensure that the total amount of contribution is independent of the actual sample location (see theorem A.3), and thus that the energy of the histogram is rotation independent.

Note that finding the correct rotation of the histogram, only means that we have found the image rotation modulo π, due to the double angle nature of local orientation. Full 360° rotation can however be found using soft histograms of the DIV operator described in [34].

5.3.1 Alignment of cyclic histograms

We can find out how we should rotate a cyclic histogram in order to align it with another one using the discrete Fourier transforms of the two histograms. The method to be described produces results with a resolution considerably higher than what is indicated by the number of bins in the two histograms. The idea is a derivative of the disparity estimation method described in *Signal Processing for Computer Vision* [23], section 7.4. Alignment of cyclic histograms has been used to speed up fractal image coding, as documented in [17].

The method utilises the Fourier coefficients in the positive half of the Fourier domain, i.e. for a cyclic histogram $\{h_1, h_2, \ldots, h_N\}$ we will use the coefficients

$$\boldsymbol{w}_k(h) = \sum_{n=1}^{N} h_n e^{-\boldsymbol{i} 2\pi k(n-1)/N} \quad \text{for } k = 1, 2 \ldots \lfloor N/2 \rfloor \tag{5.10}$$

If we have two histograms $h_1, \ldots h_N$, and $g_1, \ldots g_N$, with similar shape, but different shifts, the phase difference of the Fourier coefficients can give an estimate of how much the histograms have been shifted. However, since only the first coefficient pair can give a $[0, 2\pi]$ estimate (the others are given modulo π, modulo $\pi/2$ and so on), we will have to use it to move the others into place. The shift as seen by the first coefficient pair appears as the argument of the following complex number:

$$\boldsymbol{c}_1 = \boldsymbol{w}_1(h) \operatorname{conj}(\boldsymbol{w}_1(g)) \tag{5.11}$$

The next shift estimate can be moved into place (or *unwrapped*) using this estimate as follows:

$$\tilde{\boldsymbol{c}}_2 = \boldsymbol{w}_2(h) \operatorname{conj}(\boldsymbol{w}_2(g)) \tag{5.12}$$

$$\varphi_2 = \varphi_1 + \arg(\tilde{\boldsymbol{c}}_2 e^{\boldsymbol{i} 2\varphi_1})/2 \tag{5.13}$$

$$\boldsymbol{c}_2 = |\tilde{\boldsymbol{c}}_2| e^{\boldsymbol{i}\varphi_2} \tag{5.14}$$

Here φ_1 and φ_2 are the arguments of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ respectively. We keep the magnitude of the complex number, since it contains a measure of the two signal energies at the current frequency. Using the two shift estimates $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, we can construct an estimate that takes the energy at the two frequencies into account:

$$\theta_2 = \arg(\boldsymbol{c}_1 + \boldsymbol{c}_2) \tag{5.15}$$

For the next pair of Fourier coefficients it would thus be better to use this estimate to unwrap the phase. We will thus use the following algorithm:

$$\tilde{\boldsymbol{c}}_k = \boldsymbol{w}_k(h) \operatorname{conj}(\boldsymbol{w}_k(g)) \tag{5.16}$$

$$\varphi_k = \theta_{k-1} + \arg(\tilde{\boldsymbol{c}}_k e^{\boldsymbol{i} k\theta_{k-1}})/k \tag{5.17}$$

$$\boldsymbol{c}_k = |\tilde{\boldsymbol{c}}_k| e^{\boldsymbol{i}\varphi_k} \tag{5.18}$$

$$\theta_k = \arg\left[\sum_{l=1}^{k} \boldsymbol{c}_l\right] \tag{5.19}$$

The final value of θ_k will be our estimate of the rotation.

5.4 Experiments

We will now evaluate the performance of the rotation estimation, and tune some of its parameters. We will compute orientation histograms of pairs of images that differ only by a rotation, and compare the estimated and the known rotations.

5.4.1 Band limitation

One of the images is computed from the other one using bi-cubic interpolation. In order to avoid aliasing effects due to the fact that higher frequencies can be represented in diagonal directions, we have to band limit the image to π. In the experiments to follow, we have applied a Gaussian low-pass filter with $\sigma = 1.5$ before using the images.

5.4.2 Example

Figure 5.6 shows an example of orientation histogram computation. Two circular regions, their local orientation images, and the corresponding histograms are shown. As can be seen, $K = 9$ soft bins have been used in this example.

In figure 5.7 we have computed the rotation according to equations 5.16-5.19, and rotated the second image such that the estimated rotation is undone. This means that we rotate the image an angle $\varphi = \theta_k/2$, since the orientation representation uses the double angle. As can be seen in the difference image, we do not quite get the original image back.

5.4.3 Comparison between soft and conventional histograms

Our first experiment is meant to illustrate the advantage with overlapping bins in rotation estimation. As input feature to the soft histograms we will use local orientation vectors computed from the orientation tensors described in [12]. The tensor estimation uses 9 one-dimensional filters oriented along the horizontal and vertical axes, all with a spatial extent of 9 pixels. We thus have a total of 81 coefficients. The standard deviation of the Gaussian applicability is $\sigma = 1.23$.

From the orientation tensors, we compute a complex valued double angle image $\boldsymbol{z}(\mathbf{x})$ as follows:

$$\boldsymbol{z}(\mathbf{x}) = t_{11} - t_{22} + \boldsymbol{i}2t_{12}$$

Where t_{ij} are the components of the 2×2 tensor. This expression is equivalent to $(\lambda_1 - \lambda_2)\hat{\boldsymbol{z}}_1$ where $\lambda_1 \geq \lambda_2$ are the two eigenvalues, and $\hat{\boldsymbol{z}}_1 = e^{\boldsymbol{i}2\varphi}$ is a double angle representation of the principal subspace.

We compute local orientation of an image, and the same image rotated varying amounts about the origin. We then use all orientation responses for which $x^2+y^2 = r^2 < 90$ to compute modular soft histograms according to equation 5.9 (see figure 5.6).

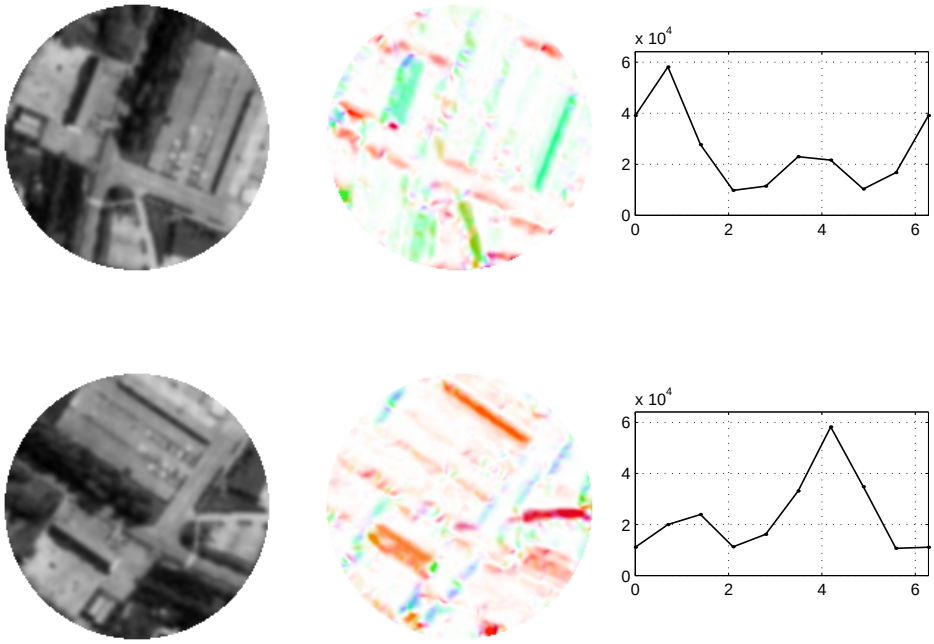


Figure 5.6: Soft histograms of local orientation.
Left to right: Intensity images, Local orientation images $\boldsymbol{z}$, and soft histograms of two rotations of an image.

Figure 5.7: Images after alignment, and the difference image.
The colourmap in the difference image goes from black for large negative values to white for large positive values.

We will use an overlap of $\omega = \pi/3$, and vary the number of soft bins K. As a performance measure, we will use the absolute estimation error averaged over 45 different rotations in the range $(-\pi/2, \pi/2)$, and 6 different image regions. The result is compared with conventional histograms in the diagrams of figure 5.8. As

can be seen, the accuracy is between 20 and 100 times better for the same number of bins.

An interesting observation is that the error is very small for soft histograms with just 3 bins. This is because the bins used span the entire range of the rotation angle, resulting in a histogram that only contains one single frequency component. For more realistic situations this kind of bins will not be useful, since they require an almost exact correspondence between the two images to match.

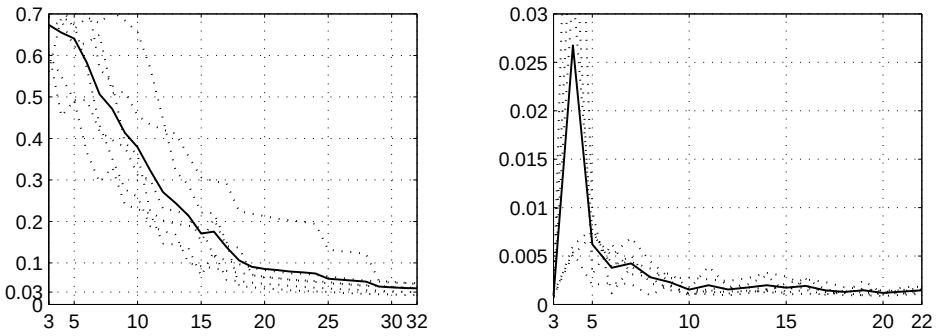


Figure 5.8: Errors in estimation of rotation angle (radians).
Rotation estimation using conventional histograms (left) and soft histograms (right). Single image errors (dotted) and average (solid). Note the different scalings of the axes.

5.4.4 Soft histograms with varied overlap

In order to find a good choice of bin overlap we will now try four different values of ω. In all other respects, this experiment is identical to the one in section 5.4.3. The result is plotted in figure 5.9. Note that for $\omega = \pi/q$ we have to use at least $K = q$ bins. As can be seen in the plots, we have quite small errors each time $K = q$. As mentioned in section 5.4.3 this is in general not a good thing, since these histograms will only contain one frequency component, and the estimation will thus require that the two images are identical except for the rotation.

The most notable difference between the plots is the much increased average error when the overlap is set to $\omega = \pi/2$. We can also see that we should avoid using fewer than $K = 10$ bins when $\omega = \pi/3$.

Higher degrees of overlap will lead to less varied frequency content in the histograms, and thus less total information content. A good choice thus seems to be a value of $\omega = \pi/3$ with $K \geq 10$. This also corresponds to the overlaps obtained for the stochastic bins in dithering (see section 5.1.1).

5.4.5 Evaluation of orientation estimations

We now evaluate three different methods to estimate local orientation. We will vary the number of bins. We will also look at what happens if we do not band-limit

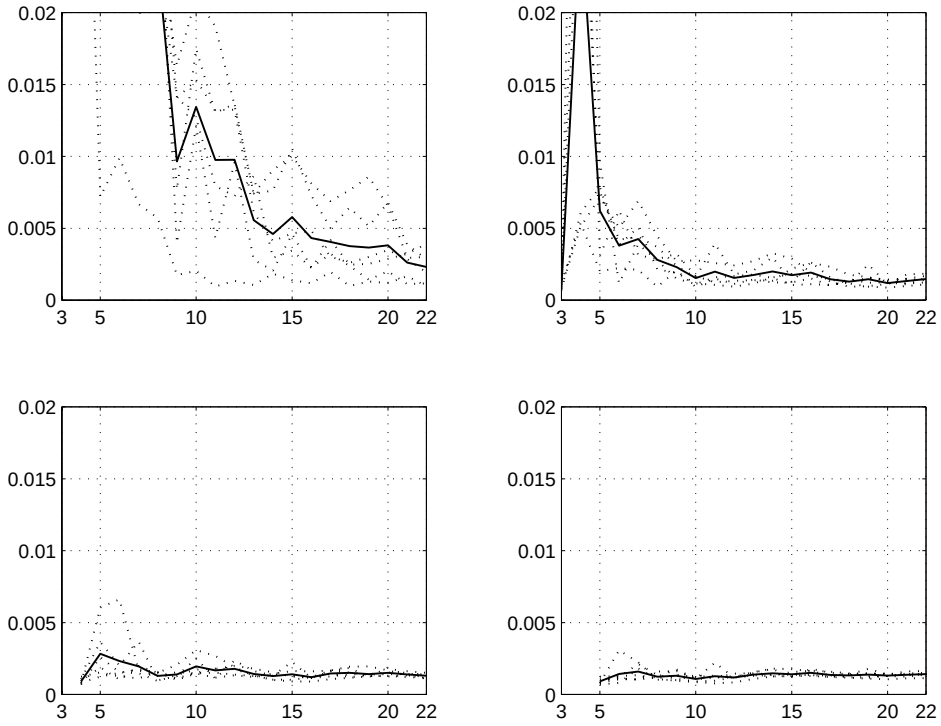


Figure 5.9: Errors for different overlaps (radians).
Top: $\omega = \pi/2$ *and* $\pi/3$. *Bottom:* $\omega = \pi/4$ *and* $\pi/5$.

our image before rotation. All used orientation estimations have approximately the same spatial extent, in order to ensure that they are comparable. We will now briefly introduce the three methods.

- The first method computes an estimate of the double angle vector using edge filters only:

$$\begin{aligned} \boldsymbol{z}_0(\mathbf{x}) &= (s * e_x * g_y)(\mathbf{x}) + \boldsymbol{i}(s * e_y * g_x)(\mathbf{x}) \\ \boldsymbol{z}(\mathbf{x}) &= \boldsymbol{z}_0(\mathbf{x})^2 / |\boldsymbol{z}_0(\mathbf{x})| \end{aligned}$$

  Here e_x and e_y are differentiating Gaussian filters in the horizontal and vertical directions respectively, and g_x, and g_y are Gaussian low-pass filters. The filters have a standard deviation of $\sigma = 1.23$, and each filter has 9 real valued coefficients. We thus have a total of $4 \times 9 = 36$ coefficients for this method. This orientation estimation was also used in [17].

- The second orientation estimation method is the one suggested in chapter 3 of [23]. This method is also used in chapter 3 of this thesis:

$$\boldsymbol{z}(\mathbf{x}) = \sum_{k=1}^{4} |(s * \boldsymbol{f}_k)(\mathbf{x})| e^{\boldsymbol{i}(k-1)\frac{\pi}{2}}$$

Here $\boldsymbol{f}_k$ is a set of four log-normal quadrature filters with a centre frequency of 1.11 and a relative bandwidth of $B = 2$. The filters are made separable using a filter optimisation technique described in [35]. The horizontal and vertical quadrature filters consist of complex filters with 2×9 coefficients plus real filters with 7 coefficients. The diagonal filters use complex filters with 2×19 coefficients, and real filters with 5 coefficients. The total number of coefficients is thus $2(2 \times 9 + 7) + 2(2 \times 19 + 5) = 136$.

- The third method is the polynomial method used in the experiment in section 5.4.3. As mentioned earlier this method uses 81 coefficients.

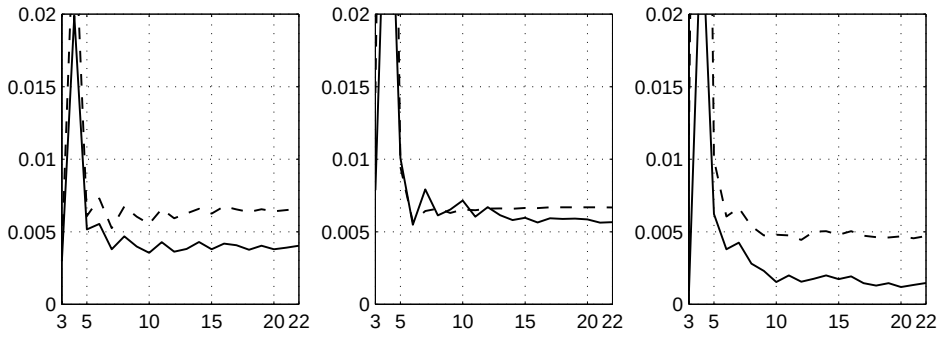


Figure 5.10: Mean absolute error as function of number of bins.
Left: Edge filters. Centre: Log-normal quadrature filters.
Right: Polynomial filters.
Dashed curves show the error when no band-limitation has been made.

As can be seen in figure 5.10 the estimation using polynomial filters gives the most accurate result in this application. It is also significantly faster than the quadrature filter method, since it uses less than two thirds the number of coefficients, but slower than the edge filter method which uses less than half of its number of coefficients.

Clues to what is causing the small errors in the estimation appear if we instead compute the average for each rotation angle (see figure 5.11). There are probably several factors causing the periodic variation seen in these plots. Part of the periodic variations are probably due to the inexactness of the bi-cubic interpolation used when rotating the images.

There is also an other likely cause: In the edge filter method we have used horizontal and vertical filters only, but in the quadrature filter method we have used four filters in evenly distributed directions. As we can see in the figure, the

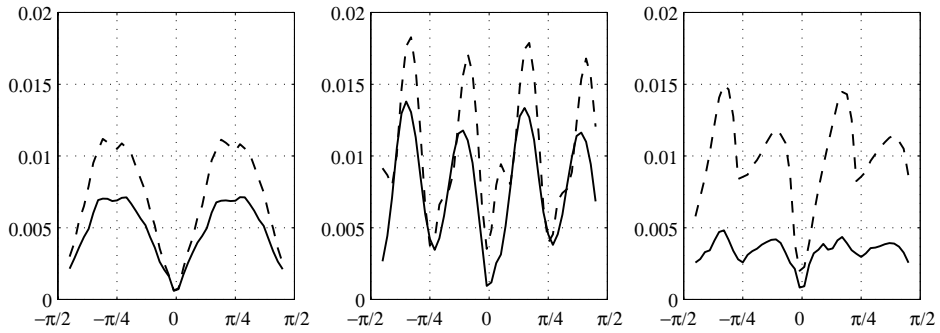


Figure 5.11: Mean absolute error as function of angle.
Left: Edge filters. Centre: Log-normal quadrature filters.
Right: Polynomial filters.
Dashed curves show the error when no band-limitation has been made.

periodic variation of the error seems to be, at least partially determined by the number of filter directions.

All these errors are however quite small. The average error of 0.0015rad obtained in the polynomial filter method corresponds to 0.086°.

Chapter 6

Associative learning

6.1 A linear network with localised inputs

This chapter contains some experiments on the performance and behaviour of associative networks with inputs and responses that are monopolar and localised. First however, we will introduce the associative network concept.

6.1.1 A two phase system

A system that is able to learn through interaction with its environment should ideally be able to learn continuously, and behave in such a way that it *actively* explores new situations that it decides to learn. In this chapter we will however assume the more modest goal of a *batch mode learning system*. This kind of system interacts with the world in two phases:

- **Exploratory phase**. Initially the system will perform a guided exploration of the external world. This includes finding out how the sensors and actuators work. The result of this phase is an estimated functional relationship between the stimuli and responses.
- **Reactive phase**. After the initial phase, the learning will end, and the system will act as a stimuli-response automata. That is, it will receive stimuli, and generate responses (act) according to these.

Note that this setup is not an adequate basis for designing a completely autonomous system, but it can be used as a crude model of the two *modes of operation* that an autonomous system must have [24]. A fully autonomous system will have to be able to continuously explore, and react interchangeably. The above mentioned setup will however allow us to explore many aspects of learning for autonomous systems. It also has many useful applications in itself since it allows design of systems through examples of desired behaviour, instead of through explicit behavioural specification (i.e. programming).

There are evidences that subsystems of biological organisms work in this two phase manner. For instance the information processing taking place in the retina

and low levels of the visual system of mammals is learned at an early stage of development, and once learnt, it stays constant [4].

6.1.2 Input representations in learning

Inputs to artificial learning systems are often in the form of compact variables. For instance an input representation of temperature might be a single node with a signalling strength proportional to the actual temperature. The reasons for this input representation is mainly convenience (most computer systems have support for this representation built in, and data usually arrives to the system in this form), and memory utilisation. There are however several problems associated with the use of plain variable inputs:

- Unless the problem to be solved is linear, we will have to use multiple layers in our network. This was shown by Minsky and Papert in their book *Perceptrons*[41].
- The more layers we add to a network, the longer it takes for it converge.
- For good performance, the size of a multi-layer network will have to be adjusted to make sure that the number of nodes in the second (hidden) layer is right.

These problems can be overcome if we change the representation of our stimuli. If we ensure that our stimuli are represented in such a way that they are localised in the response domain, we can solve problems that are locally linear, but globally non-linear, using a single network layer (see figure 6.1).

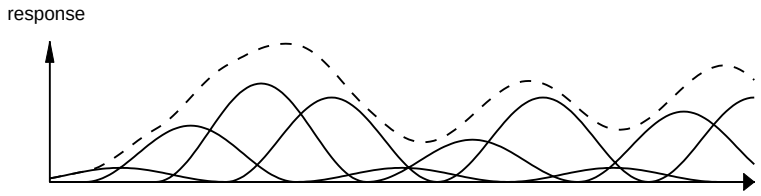


Figure 6.1: Weighted inputs (solid), and their sum (dashed).

6.1.3 Linear networks

In the reactive phase, the system will thus generate *response functions* $\{\mathrm{u_k}(s)\}_1^K$ as a weighted sum of *feature functions*, $\{\mathrm{a_h}(s)\}_1^H$:

$$\mathrm{u_k}(s) = \sum_{h=1}^{H} c_{kh}\mathrm{a_h}(s) \quad , \quad k = 1, 2 \ldots K$$

Or in matrix notation:

$$u_k(s) = \mathbf{c}_k \mathbf{a}(s) \tag{6.1}$$

The argument, s, is meant to indicate that a_h and u_k are views, or representations of the underlying state of the external world. The matrix $\mathbf{c}_k$ implements a static reactive memory, and contains a set of weights that will be generated during the training phase. The response variable, $u_k(s)$, and the reactive memory matrix, $\mathbf{c}_k$, are indexed, since the network will usually have several outputs.

6.1.4 Localised inputs

As stated earlier, a linear, single-layer network can solve non-linear problems if we ensure that the inputs are localised in the response dimension (see figure 6.1). If we know that the input data is in compact form, we can increase the locality by applying channel coding (see chapter 4).

For situations where we can expect several simultaneously active features, and features that are active in more than one local region of the response dimension, it might also be advantageous to look at coincidences, or *covariant combinations* of sensor channels. Covariant combinations are constructed from the *sensor channel vector*, $\mathbf{x}$, which is simply a list of the sensor channel values $\{x_s\}_1^S$. The *feature vector*, $\mathbf{a}$, is constructed from $\mathbf{x}$ as all unique combinations of sensor outputs:

$$\{a_h\}_1^H = \{x_i x_j : \quad 1 \leq i < j \leq S\} \tag{6.2}$$

This corresponds to the top right half of the outer product matrix:

$$\mathbf{x}\mathbf{x}^t = \begin{pmatrix} x_1x_1 & \mathbf{x_1x_2} & \mathbf{x_1x_3} & \dots & \mathbf{x_1x_S} \\ x_2x_1 & x_2x_2 & \mathbf{x_2x_3} & \dots & \mathbf{x_2x_S} \\ x_3x_1 & x_3x_2 & x_3x_3 & \dots & \mathbf{x_3x_S} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_Sx_1 & x_Sx_2 & x_Sx_3 & \dots & x_Sx_S \end{pmatrix}$$

For a network where features are constructed as covariant combinations of sensor channel values, the number of features, H, is related to the number of sensor channels, S, as:

$$H = S(S-1)/2$$

6.1.5 Localised outputs

The example of response generation in figure 6.1 works fine if the stimuli originate from a channel coded compact variable. However, the fact that the response we have generated is a compact variable could lead to complications.

If the system is an object recognition system with image features as stimuli, each response state could be seen as a description of the object at a certain view. By prescribing a compact response, we would force the system to merge all views at varying strengths. This will lead to poor robustness, and sensitivity to noise.

We could instead use localised responses by channel coding the desired response. Each response can now be seen as a template for a specific view of the object.

As discussed in chapter 4, channel coding of a response is an elegant way to design a system that can deal with multiple hypotheses, as well as give a "no response" statement when appropriate. The channel coded responses will be grouped in a vector $\mathbf{u}$ of size K. This vector is called a *response mode*, and there will usually be several such response modes within one system.

By specifying that the responses should be localised, we can improve the robustness of the system, since only a few features will be used to define each response. This means that a random disturbance on the input side will only have a local influence on the response. An other reason why the robustness is improved is that when the response functions are overlapping, the actual response statement will be distributed across a few of the responses. This means that a random error can be averaged out during the scalar reconstruction, as we will see in the experiments to follow.

6.2 Batch mode training setup

During the exploratory phase (see section 6.1.1) we passively collect training data, which will be used to find the weights in the $\mathbf{C}$ matrix during a *batch mode training*. During the batch mode training, we can thus assume that we have access to a set of training data that enables the system to learn how to react in all situations it might encounter:

$$\mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_N \\ | & & | \end{pmatrix} \quad \mathbf{A} = \begin{pmatrix} | & & | \\ \mathbf{a}_1 & \cdots & \mathbf{a}_N \\ | & & | \end{pmatrix}$$

The rows in $\mathbf{A}$ and $\mathbf{U}$ can be seen as samples from feature functions, $\mathrm{a_h}(s)$, and response functions, $\mathrm{u_k}(s)$. Each row in $\mathbf{A}$ will be denoted $\mathbf{a}_h$, and each row in $\mathbf{U}$, $\mathbf{u}_k$. These vectors contain samples that describe the mapping we want to find. Individual elements of the matrices $\mathbf{U}$ and $\mathbf{A}$ are denoted a_{hn} and u_{kn} respectively.

6.2.1 Learning as a search

The batch mode training can be seen as a search for a solution to the equation:

$$\mathbf{U} = \mathbf{CA} \tag{6.3}$$

The solution we want to find is, as we shall see, not the conventional least squares solution.

In order to reduce the amount of data we have to handle simultaneously, we will exploit the fact that this problem can be solved one response at a time, i.e:

$$\mathbf{u}_k = \mathbf{c}_k \mathbf{A} \tag{6.4}$$

This sub-problem is visualised in figure 6.2: We want to model a response function as a weighted sum of feature functions.

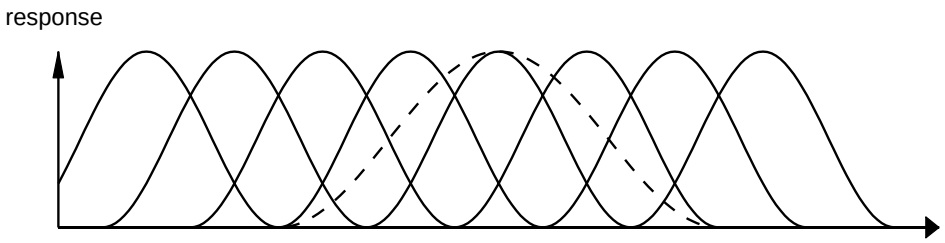


Figure 6.2: Desired response (dashed), and the feature functions (solid).

Note however that figure 6.2 does not show the whole truth. Since the space in which a feature function is defined generally is multidimensional, figure 6.2 just shows the special case of a one-dimensional trajectory in stimulus space.

The solution to the training problem is visualised in figure 6.3. Note that since the feature functions have a slight overlap in this example, the individual feature functions are always lower in magnitude than the response function.

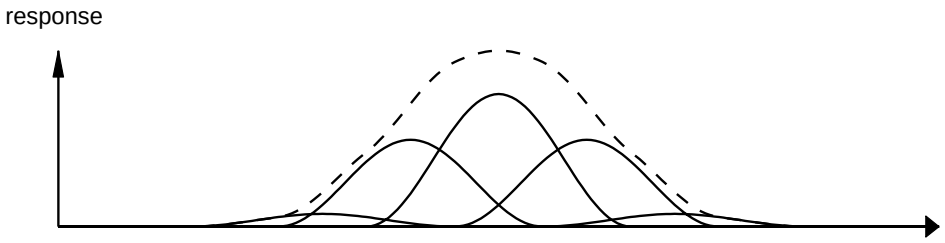


Figure 6.3: Weighted feature functions (solid) and their sum (dashed).

6.2.2 Sparse and non-negative coefficients

If we should decide to solve the optimisation problem (equation 6.3) as a regular *least square fit*, we would end up with a matrix, $\mathbf{C}$, that is filled with mostly non-zero coefficients, of both excitatory (positive), and inhibitory (negative) nature.

The coefficients in the matrix will also typically be quite large, since adjacent features of excitatory and inhibitory nature will compete. While such a matrix would usually produce a very accurate result for the samples within the training set, the interpolating abilities of the network is not necessarily that good.

The ability to interpolate can be improved by restricting the coefficients in $\mathbf{C}$ to positive values. This will in effect exclude all inhibition, and thus remove the competition between excitatory and inhibitory features. An other advantage with non-negative coefficients is that we typically end up with fewer non-zero coefficients than we do when we allow negative weights. For typical learning situations the amount of non-zero coefficients in $\mathbf{C}$ will be below 10%.

One problem is still left though. We can easily end up with quite large values for some coefficients. This typically indicates that the samples of the feature function in question lie far from the receptive field centre. In other words, the samples we have do not sufficiently well describe the shape of the feature, and for samples outside the training set, we could encounter responses that are way above the desired response, specified during training. This phenomenon is referred to as *supernormal stimuli* in biology. To reduce this effect, we simply impose an upper limit on the coefficients.

$$\min_{0 \leq \mathbf{c}_k \leq \varepsilon} \|\mathbf{u}_k - \mathbf{c}_k \mathbf{A}\| \tag{6.5}$$

We have now arrived at a *bounded least-squares problem* (see equation 6.5). Such a problem can be solved by a bounded least-squares optimiser, such as `sblsreg` [1].

6.2.3 Notes on system size

In principle there are three parameters that influence the size of a training problem: The number of samples N, the number of features H, and the number of responses K.

In practise, we have to make sure that the number of samples is at least as large as the number of responses, i.e. $N \geq K$. There is however no such restriction relating the number of features with the number of samples, since we will usually have lots of features active simultaneously.

6.3 Feature selection

The number of sensor channels needed to describe an aspect of an image sufficiently well will usually result in a large number of features. For a network where features are constructed as covariant combinations of sensor channel values, the number of features, H, is related to the number of sensor channels, S, as $H = S(S-1)/2$.

Thus, even for one-dimensional input data, the size of the feature vector $\mathbf{a}$ can be quite large (most learning problems of interest have H values that are above 1000).

Due to the large size of the learning problems, direct application of a bounded least-squares optimiser will in many cases be very time consuming. Thus, we will now present a method to reduce the size of the optimisation problem.

If we look at figures 6.2 and 6.3, it is immediately apparent that those feature functions that lie outside the non-zero range for the response function should always have zero coefficients. It is even reasonable to assume that those features that have the major part of their non-zero values outside the non-zero range for the response function are not useful to model the response. We will now put this simple observation to practical use, as a means of reducing the optimisation problem (equation 6.5).

Using the step function:

$$\mathrm{us}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.6}$$

we can express how large a part of a feature function $\mathrm{a_h}(s)$ is within the non-zero range of the response function $\mathrm{u_k}(s)$ that we want to model:

$$m_{h+} = \int \mathrm{us}(\mathrm{u_k}(s))\mathrm{a_h}(s) \tag{6.7}$$

In the same way we can find out how large a part of the feature function $\mathrm{a_h}(s)$ is outside this range (see figure 6.4):

$$m_{h-} = \int (1 - \mathrm{us}(\mathrm{u_k}(s)))\mathrm{a_h}(s) \tag{6.8}$$

Using the samples from the feature and response functions, we estimate m_{h+} and m_{h-} as:

$$m_{h+} \approx \sum \mathrm{us}(u_{kn})a_{hn} = \mathrm{us}(\mathbf{u}_k)\mathbf{a}_h^T \tag{6.9}$$

$$m_{h-} \approx \sum (1 - \mathrm{us}(u_{kn}))a_{hn} = (1 - \mathrm{us}(\mathbf{u}_k))\mathbf{a}_h^T \tag{6.10}$$

Now we have a measure of which features we can safely ignore when solving equation 6.5; we simply ignore all features for which m_{h-} is larger than m_{h+} (see figure 6.4).

In order to reduce the equation system, and to be able to expand it again in a simple way, we first create an identity mapping of size $H \times H$:

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & \cdots \\ 0 & 1 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

From this matrix we now remove those rows in which the difference $m_{h+} - m_{h-}$ is negative. We will call the resultant matrix a *feature selection matrix*:

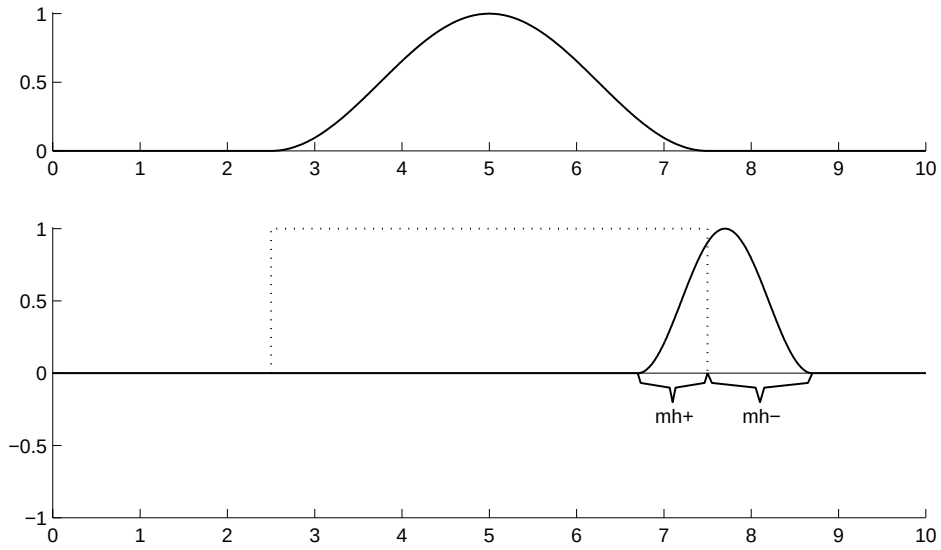


Figure 6.4: Criterion for feature selection.
The top graph shows a response, and the bottom graph shows how the feature function is split in two halves (with integrals m_{h+} and m_{h-}).

$$\begin{pmatrix} \vdots \\ \mathbf{m}_{h+} - \mathbf{m}_{h-} \\ \vdots \end{pmatrix} = \begin{pmatrix} -0.12 \\ \cdot \\ -0.05 \\ -0.82 \\ \vdots \end{pmatrix} \quad \Rightarrow \quad \mathbf{R}_k = \begin{pmatrix} \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cdots \\ 0 & 1 & 0 & 0 & \cdots \\ \cancel{0} & \cancel{0} & \cancel{1} & \cancel{0} & \cdots \\ \cancel{0} & \cancel{0} & \cancel{0} & \cancel{1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

The problem we had to solve for each response (equation 6.5) can now be reduced to:

$$\min_{0 \le \mathbf{c}_k^R \le \varepsilon} \left\| \mathbf{u}_k - \mathbf{c}_k^R \mathbf{A}_k^R \right\| \tag{6.11}$$

Where $\mathbf{c}_k^R$ and $\mathbf{A}_k^R$ are reduced variants of $\mathbf{c}_k$ and $\mathbf{A}$ created using the feature selection matrix, $\mathbf{R}_k$:

$$\mathbf{c}_k^R = \mathbf{c}_k \mathbf{R}_k^T \quad , \quad \mathbf{A}_k^R = \mathbf{R}_k \mathbf{A}$$

Once the equation solver has found a solution $\mathbf{c}_k^R$ we can inflate this to find the corresponding row in $\mathbf{C}$:

$$\mathbf{c}_k = \mathbf{c}_k^R \mathbf{R}_k$$

If the system setup is such that we have an abundance of features, we could even afford to remove all features that are active outside the non-zero range of the response. That is, we could remove all features for which $\mathbf{m}_{h-} > 0$.

6.4 The Hebb rule

We will now present a quick, and very simple way of finding an approximate solution to the weights in the $\mathbf{C}$ matrix, based on the *Hebb rule*. The Hebb rule is the principle that if two cells are active simultaneously their connections should be strengthened [30]. In our case this means that if a feature and a response are active simultaneously, the response can be modelled using the feature.

Most implementations of the Hebb rule involve making an adjustment to the weights proportional to the product of the feature and the response. This is commonly referred to as the *outer product rule* or the *generalised Hebb rule* [2]. If we assume that one feature and response sample ($\mathbf{u}$, $\mathbf{a}$) is enough to calculate the weight matrix, $\mathbf{C}$, we can derive the generalised Hebb rule as:

$$\mathbf{u} = \mathbf{C}\mathbf{a} \tag{6.12}$$

$$\mathbf{u}\mathbf{a}^T = \mathbf{C}\mathbf{a}\mathbf{a}^T = \mathbf{C}\|\mathbf{a}\|^2 \tag{6.13}$$

$$\mathbf{C} = \frac{\mathbf{u}\mathbf{a}^T}{\|\mathbf{a}\|^2} \tag{6.14}$$

When combining several samples, we simply average contributions of this form:

$$\mathbf{C} = \sum_{n=1}^{N} \eta \mathbf{u}_n \mathbf{a}_n^T = \eta \mathbf{U}\mathbf{A}^T \tag{6.15}$$

That is, we sum a set of Hebbian contributions, each scaled by η. The scaling has to be computed afterwards, since it, amongst other things depends on the number of samples, N. For this reason this variant of the Hebb rule is clearly inappropriate as an on-line rule. However, in batch mode we could still use equation 6.15 as a coarse initial solution.

The scaling, η, can be found in a fairly straightforward manner once we have made the summation. We can then simply make a "test run" of the system to find out how we should scale the coefficients in $\mathbf{C}$:

$$\mathbf{C}^* = \mathbf{U}\mathbf{A}^T$$
$$\mathbf{U}^* = \mathbf{C}^*\mathbf{A}$$

If we assume that this response $\mathbf{U}^*$ is correct, except for the scaling η, we can now find η through a least-squares fit for each response:

$$\mathbf{u}_k = \eta \mathbf{u}_k^*$$
$$\eta = \frac{\mathbf{u}_k \mathbf{u}_k^{*T}}{\mathbf{u}_k^* \mathbf{u}_k^{*T}} \tag{6.16}$$

6.4.1 Uneven sample and feature density

The application of the generalised Hebb rule has the advantage that the solution is very quickly computed. Often however, it can produce results that do not resemble the shape of the response signal very well. Three common situations when this happens are:

1. when features are unevenly spread out across the response domain, with much higher density in some places. (See figure 6.5, left)
2. when the samples are unevenly distributed among the features. (See figure 6.5, right)
3. when there are too few samples, especially when there are no samples near the peaks of the feature functions.

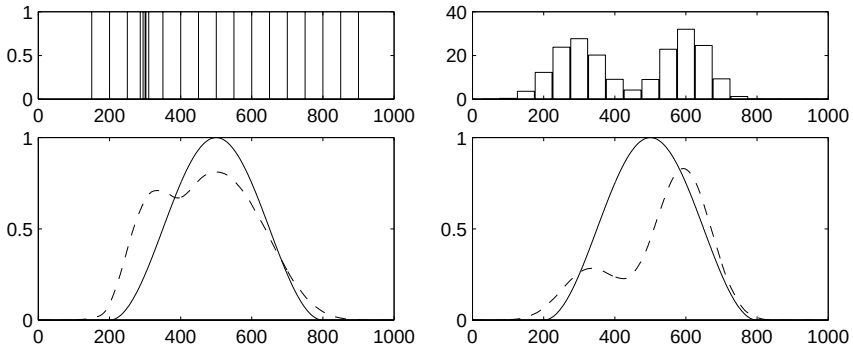


Figure 6.5: Uneven feature density, and uneven sample density.
Left: Top graph shows the positions of the feature function peaks. Bottom graph shows the desired response (solid), and the actual response (dashed) with the feature function density above.
Right: Top graph shows the sum of all samples for each feature. Bottom graph shows the response obtained from the outer product rule (dashed) compared with the desired response (solid) with the sample sums above.

Application of a bounded least squares solver does not have these problems, and we shall see in a later section how we might extend the outer product rule approach to account for these situations as well.

However, the existence of these phenomena (at least cases 2 and 3) might not be all bad though. If the magnitude of the response is used to encode the certainty

of the response, the outer product rule is actually very accurate. When we have a high density of feature functions, we actually know with higher certainty that the response is correct, and a larger magnitude in the response can be used to reflect this.

We actually know very little about those features that have no samples near their peaks. For instance, in the one-dimensional case we do not know whether the peak is to the right, or to the left of the sample. Choosing not to include this feature much (as will happen with the outer product rule) might actually be a wise move, since the feature might otherwise invoke a much stronger response for stimuli we have not yet encountered.

If our system was an active autonomous system, it could detect the lack of samples, and then actively explore the response state in question.

6.4.2 Correlation and causation

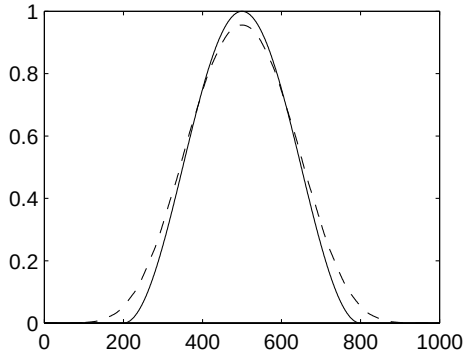


Figure 6.6: The flattening phenomenon when applying the outer product rule. *The response obtained using the outer product rule (dashed) compared with the desired response (solid).*

An other problem with the generalised Hebb rule is that even for the most simple cases the net will produce response curves that are slightly flattened out. (See figure 6.6).

The reason for the flattening is that a feature-response correlation does not always imply that a feature is useful for modelling a response. This is a well known phenomenon in statistics, and is usually formulated as follows: **Correlation does not imply causation**.

The problem is most easily understood if we take a look at the *cause-effect diagram* (See figure 6.7). The outer product rule considers only the case $\{A, A\}$, when both feature and response are active simultaneously. However, stating that a feature can model a response involves all four quadrants of the diagram. For instance, a feature might occasionally be active simultaneously with the response (case $\{A, A\}$) but is also active when the response is not (case $\{I, A\}$).

The conclusion to be drawn from the cause-effect diagram, is that choosing to

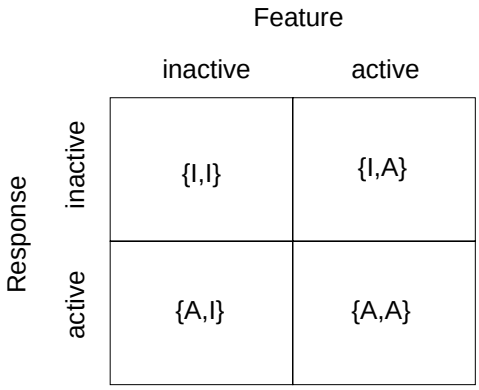


Figure 6.7: The cause-effect diagram.

use a feature to model a response should also consider the case when the response is inactive and the feature is active. A feature-response relationship can always be said to either belong more to case $\{A, A\}$ or more to case $\{I, A\}$. In the former case we should have a weight in the corresponding row of $\mathbf{C}$, in the latter case we should not.

The other two cases ($\{I, I\}$ and $\{A, I\}$) need not bother us now, we can only hope that an other feature can model the response in those cases.

The flattening phenomenon can be dealt with using the feature selection described in section 6.3, or using an iterative optimisation scheme.

6.4.3 An iterative learning scheme

The learning scheme used in the experiments to follow has been developed by Granlund [19]. Basically it's an iterative scheme where the response errors are "sampled" by the feature functions in the same way as the response is "sampled" by the feature functions in the generalised Hebb rule (see equation 6.15). The optimisation is quite fast, provided that the sparsity of the inputs and outputs is utilised, and in general it converges after a few hundred iterations.

6.5 Experiments with COIL-100

The experiments described in this section use the COIL-100 database [45]. This database (see `http://www.cs.columbia.edu/CAVE`) contains 100 objects of varying shape and colour (see figure 6.8). Each object has been placed on a motorised turntable, and has been photographed at 72 different views, 5° apart, and stored as 128×128 pixel RGB bitmaps. The COIL-100 database has previously [45, 64] been used to evaluate object recognition systems.

Figure 6.8: Images in the COIL-100 database.
Objects are numbered row by row, from 1 *to* 100*, starting at the top-left corner.*

Rather than discrimination between different objects, we have chosen to illustrate a more subtle problem within view centred modelling: Continuous description of the orientation of an object.

None of the experiments have utilised the colour information present in the images, in order to demonstrate the power of the shape features used.

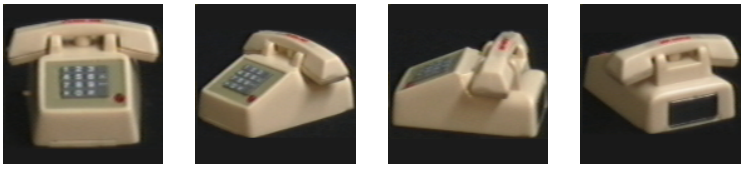


Figure 6.9: Four views of COIL-100 object #60.

Figure 6.10: Input DIV features.
Left: Magnitude of DIV responses for the first view. Centre: Interpretation of DIV response argument. Right: The complex number at local maxima is shown as a vector.

6.5.1 Initial experiment

To illustrate the performance of the associative network, we have used object #60, a telephone (see figure 6.9).

In the training phase, the system is shown every fourth view of the object, and learns to associate the features at each view with a desired response, which in this case is the rotation angle of the turntable. Object #60 was chosen since it is fairly complex, which also makes each view sufficiently different from the others.

6.5.2 Input features

As input to the associative network we have used responses from a DIV operator [34] designed to find high curvature points in image transforms describing local orientation. This is a descriptor which is well suited to associative learning, as it is sparse and very robust.

The spatial size the operator used is $\sigma = 5.5$ (The σ parameter corresponds to the standard deviation of a Gaussian kernel applied on the original image). These responses are down-sampled to 32×32, yielding a total of 1024 feature values. The response for the first view of object #60 is shown in figure 6.10. The response from the DIV operator is a complex number, with the magnitude describing the amount of curvature in the local image region. The argument defines a vector that points in the corner direction.

The DIV responses are then channel coded, by separating each complex number in the real and imaginary parts, and further into positive and negative parts. This gives us a total of $4 \times 1024 = 4096$ input features to the associative network. These features are stored in a vector $\mathbf{a}$. This may appear like a great deal of data, but it really is not, since the responses from the DIV operator are *sparse*, and thus

most points do not require any values to be stored. The sparse matrix toolbox in MATLAB has been used to utilise the sparsity of the data.

6.5.3 Specified responses

The desired response for the system is the angle from which we view the object, φ. This response is also channel coded as $\cos^2()$ responses in a modular arrangement. Initially we will use $K = 5$ channels with an angular overlap of $\omega = \pi/3$:

$$u_k(\varphi) = \begin{cases} \cos^2(\omega d(\varphi - \varphi_k)) & \text{adist}(\varphi, \varphi_k) < \frac{\pi}{2\omega d} \\ 0 & \text{otherwise} \end{cases}$$

where

$$d = \frac{K}{2\pi} \quad \text{and} \quad \varphi_k = \frac{k-1}{d}$$

The parameter d specifies the channel distance, and φ_k are the channel centres, distributed along the angular interval $[0, 2\pi[$. The function $\text{adist}(\varphi_1, \varphi_2)$ returns the angular distance between the arguments.

These channels form the desired response vector $\mathbf{u} = \begin{pmatrix} u_1 & u_2 & \dots & u_k \end{pmatrix}$.

6.5.4 Training

The network is trained on every fourth view, and evaluated on all views. The network responses are shown in figure 6.11.

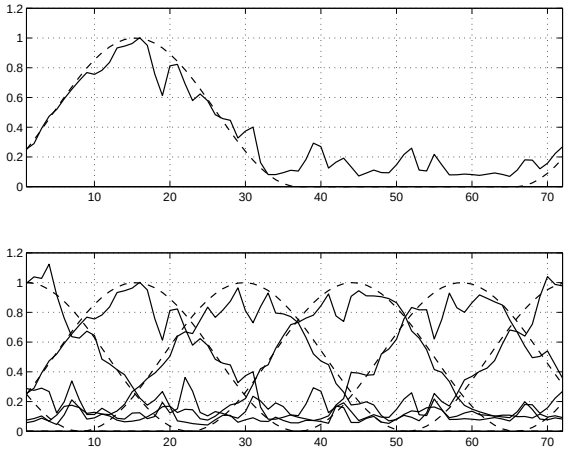


Figure 6.11: Network responses.
Top plot shows desired response no 2 (dashed), and the network output (solid). Bottom plot shows all five responses.

The individual response channels might not look too impressive. The final estimated angle however, takes the partially redundant data from three consecutive channels into account, and the resultant reconstruction is consequently much more robust to variations within the feature set (see chapter 4 for details on how to perform the reconstruction). The reconstructed view angle is plotted in figure 6.12. As can be seen, the error is always less than 15°.

The bottom plot of figure 6.12 shows the response magnitude, i.e. the estimated height of the peak. When the system lacks input features, this value will drop.

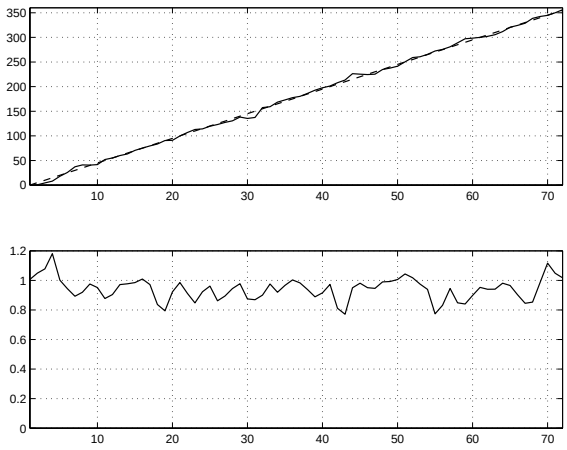


Figure 6.12: Reconstruction error.
Top plot shows desired view angle (dashed), and the network output (solid). The error is always less than 15°. *Bottom plot shows the response magnitude, which is an indication of confidence.*

The training time, using a fast iterative algorithm, for 5 response channels and 300 iterations is 51 seconds on a SUN Ultra-60 at 296 MHz. The training time is roughly proportional to the number of response channels, the average number of non-zero feature channels, and to the number of samples. If we set low feature values to zero, the training time will be reduced considerably, since the features are sparse and thus mostly close to zero anyway. Removal of low feature values normally do not have noticeable effects on accuracy.

6.5.5 Varied number of responses

The number of response channels can be tuned to the particular training situation to increase the robustness of the system. Figure 6.13 shows RMSE (Root Mean Square Error), MAE (Mean Absolute Error), and how often the absolute error has deviated further than 5°, as a function of the number of response channels K.

All estimates obtained lie well inside the 20° error limit.

The extreme case of $K = 18$ response channels in figure 6.13 corresponds to one channel for each example in the training set. As can be seen in the figure, the

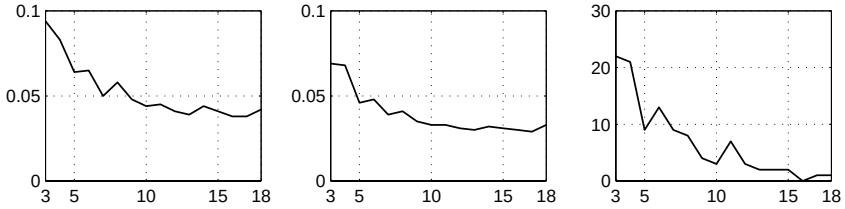


Figure 6.13: Reconstruction error as function of K.
Left to right: RMSE, MAE, and number of times error exceeds 5°

reconstruction is quite accurate for a much smaller number of response channels.

6.5.6 Varied sample density

A crucial aspect of the network is its ability to interpolate between the views it has been shown during the training phase. Since the responses are slightly overlapping, they can be said to define a *metric* or a local measure of distance to a set of prototypes. Provided that everything works as it should, we should have a gradual decrease in performance, as we decrease the number of views.

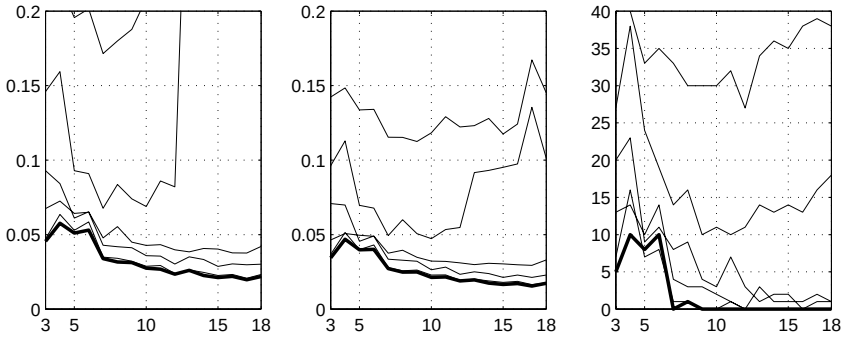


Figure 6.14: Varied sample density ($\omega = \pi/3$).
Left to right: RMSE, MAE, and number of times error exceeds 5°.
Thick: training on all views. Thin: 10°, 15°, 20°, 25°, *and* 30° *view distance.*

As can be seen in figure 6.14 the network performance is acceptable until the view distance becomes greater than 20°. The abrupt change at 25° is probably due to the fact that a rotation 25° moves the extent of a DIV response more than one grid distance away in the 32×32 feature array.

6.5.7 Continuous function mapping

As a comparison we will now illustrate the advantage with localised, partially redundant responses. Figure 6.15 shows the response after training with a contin-

uous response (the straight dashed lines in the plots). The response in these tests has been set to the view angle in radians plus a constant offset of 1.

First we should note that it is far from obvious that this training procedure should work at all. The fact that it does work is most likely due to the DIV features being a very good choice of inputs. As previously stated, a good feature should be localised in state-space. In the present setup this means that a feature should not reappear in several of the views, unless they are adjacent.

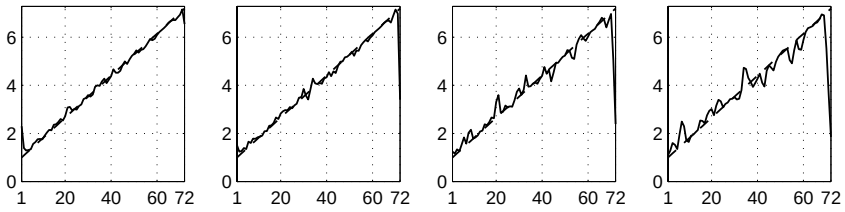


Figure 6.15: Continuous function mapping.
Left to right: 5°, 10°, 15°, *and* 20° *view distance.*
Dashed: desired response. Solid: network response.

We now compute the error measures previously used for these graphs as well:

Distance	5°	10°	15°	20°
RMSE	0.200026	0.324995	0.379816	0.501304
MAE	0.107731	0.12961	0.196327	0.276713
AE $> 5°$	31	28	33	40
AE $> 20°$	2	3	12	19

If we compare these numbers with the graphs in figure 6.14, we see that the error when the system is shown all views is now actually greater than the error with a 20° view distance in figure 6.14. As we reduce the number of samples it gets even worse.

The increased robustness with localised, distributed responses can be much attributed to the fact that the response is represented in a distributed form, shared over several channels at a time. Since the individual errors in the responses tend not to correlate, their influence will be reduced in the reconstruction. The same phenomenon is used in noise reduction through averaging.

6.5.8 Non-zero coefficient ratio

An important aspect of the associative networks is that the connections should be sparse. The degree of sparsity of the **C** matrix can be measured using the non-zero coefficient ratio r, which is defined as follows:

$$r = \frac{1}{HK} \sum_k \sum_h h(c_{hk}) \quad \text{where} \quad h(c_{hk}) = \begin{cases} 1 & \text{if } c_{hk} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.17}$$

Figure 6.16 (Left) shows a plot of the non-zero coefficient ratio against the number of responses K for three different values of the overlap, ω.[1] As we can see, the number of non-zero coefficients drops when the number of responses is increased. This phenomenon can be used to control the sparsity of an associative network.

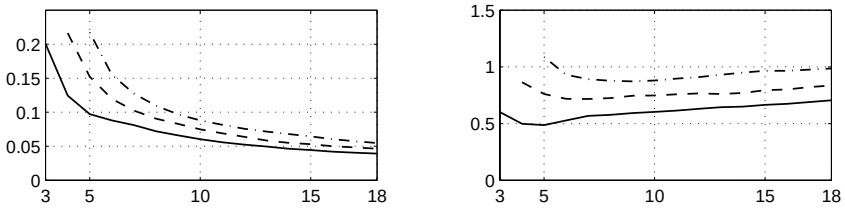


Figure 6.16: Non-zero ratio as function of K.
Left: r as a function of K Right: rK as function of K.
Solid, dashed, and dash-dotted lines correspond to $\omega = \pi/3$, $\pi/4$, and $\pi/5$ respectively.

If we multiply the non-zero ratio, r, with the number of responses, K, we obtain the plots in the right half of figure 6.16. As we can see, the relationship after an initial dip, is approximately linear. For the current experiment, we obtain the following relationships through least-squares estimation in the linear section of the plots:

Overlap	Relationship
$\pi/3$	$rK \approx 0.4810 + 0.0123K$
$\pi/4$	$rK \approx 0.6452 + 0.0100K$
$\pi/5$	$rK \approx 0.7508 + 0.0135K$

The reason why the product rK is not constant is probably the fact that the response channels are overlapping. As more response channels are added, individual features will tend to be used to model more than one response. Note that no significant change in the slope of the curves can be attributed to the overlap, ω. Rather, experiments on other objects indicate that the slope is object dependent. We can also note that the amount of non-zero coefficients seems to be linearly dependent on ω.

6.5.9 Increased overlap

As we saw in section 6.5.7, a distributed representation improves the robustness of the network. The question is, will an even more distributed representation work still better? In order to find out we will now try other values of the overlap ω. Figure 6.17 shows plots corresponding to the ones in figure 6.14, but with $\omega = \pi/4$, and $\pi/5$. As can be seen, the improvement is negligible until the view distance exceeds 20°. Since increased overlap normally means more non-zero coefficients in

[1] For these plots the network has been trained using all 72 views. If fewer number of views are used, similar but increasingly noisy curves will be obtained.

the network (see section 6.5.8) we conclude that, at least for this relatively noise free data-set, overlaps larger than $\pi/3$ are not worth the cost.

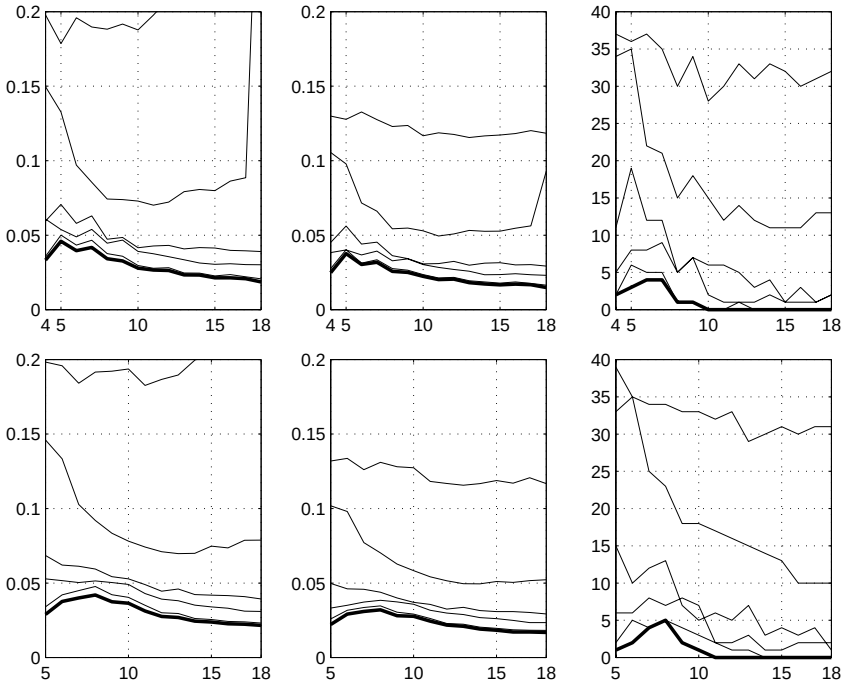


Figure 6.17: Varied sample density ($\omega = \pi/4$, and $\pi/5$).
Left to right: RMSE, MAE, and number of times error exceeds 5°.
Top: $\omega = \pi/4$. *Bottom* $\omega = \pi/5$.
Thick: training on all views. Thin: 10°, 15°, 20°, 25°, *and* 30° *view distance.*

6.5.10 Other COIL-100 objects

In order to show that the associative networks work on other objects than #60, we will now try a selection of other objects from the COIL-100 database. Just as we did with #60 in figure 6.13, we will vary the number of channels between 3 and 18, but keep the view distance fixed at 20°, and the overlap at $\omega = \pi/3$.

For many of the objects it is impossible to tell the views apart, even for a human viewer, since they look the same from several different views. The error measures for some of the objects that have sufficiently different views, are shown in figure 6.18.

The error measures for some of the objects that did not work are shown in figure 6.19 (top). These objects look very similar in different views, but there are de-facto differences. The results can thus be improved by showing the network all 72 views. The network will then be able to detect features that reappear in several views more reliably, and exclude them from the linkage vector, **C**. The

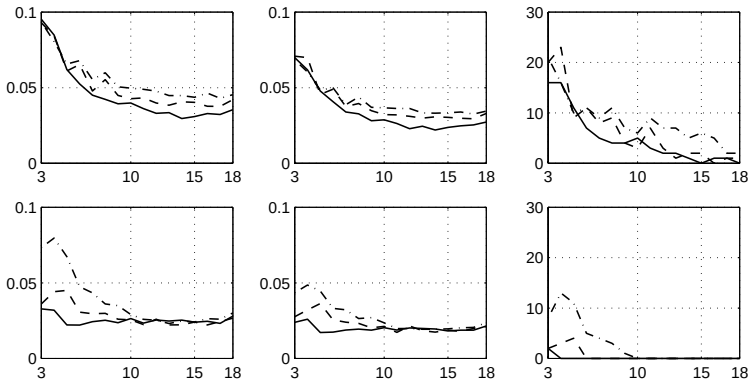


Figure 6.18: Reconstruction error class 1 and 2.
Top: Objects #3, #60, *and* #85, *plotted as solid, dashed, and dash-dotted.*
Bottom: Objects #14, #52, *and* #74, *plotted as solid, dashed, and dash-dotted.*
Left to right: RMSE, MAE, and number of times error exceeds 5°.

result when the network is shown all views for the same three objects is shown in figure 6.19 (bottom).

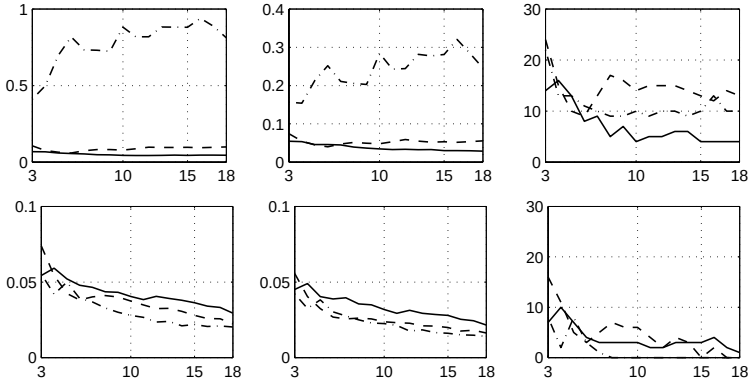


Figure 6.19: Reconstruction error class 3.
Objects #02, #10, *and* #68 *are plotted as solid, dashed, and dash-dotted.*
Top row: Training with 20° *view distance.*
Bottom row: Training on all 72 *views.*
Left to right: RMSE, MAE, and number of times error exceeds 5°.

Note that even though the examples shown here do work, a specification of the response as view angle is dangerous, as it is often an ill posed problem, especially for many synthetic objects.

6.5.11 Pruning of input features

We will now investigate the effect of setting input features below a certain threshold to zero. Figure 6.20 shows average estimated PDFs for the DIV features of object #60. As can be seen, the features are mostly zero.

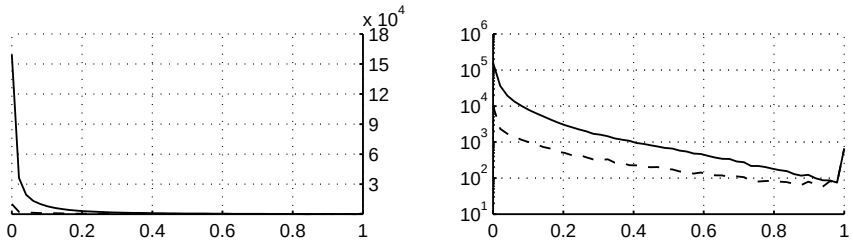


Figure 6.20: Estimated PDF of DIV features.
Left: Soft histograms of the DIV features for object #60*.*
Right: Logarithmic plot of same data.
Solid lines are average PDFs for all features, dashed lines are average PDFs for those features that at some point raise above 0.95*.*

We will now perform a *pruning* of the input features. i.e.

$$\hat{a}_k = \begin{cases} 0 & \text{if } a_k < \text{thr} \\ a_k & \text{otherwise} \end{cases}$$

We will vary the value of thr in the range $[0, 1]$, and look at some of the performance measures used before. The results of this experiment are plotted in figure 6.21.

As we can see, the training time can be much reduced by applying even a low pruning threshold. As mentioned earlier, the sparse matrix toolbox in MATLAB has been used for these experiments, and this merely tell us that the MATLAB routines work as they should. From the plots we can also see that the increase in mean absolute error (MAE) is relatively small until the threshold rises above 0.5. The number of outliers (AE $> 5°$) however rises more quickly, and thus pruning should be used with care if the processing to follow does not deal with outliers in some way.

The plots of the non-zero coefficient ratio (r) all exhibit an interesting peak. The reason why we have an initial rise in r is most likely that the feature functions will tend to get a more localised support through the pruning. Due to the improved localisation, more features are likely to be useful for modelling each response. However, when we raise the pruning threshold above the peak of a feature function,

the associative network will not see the feature at all, and thus no linkage coefficient will be generated.

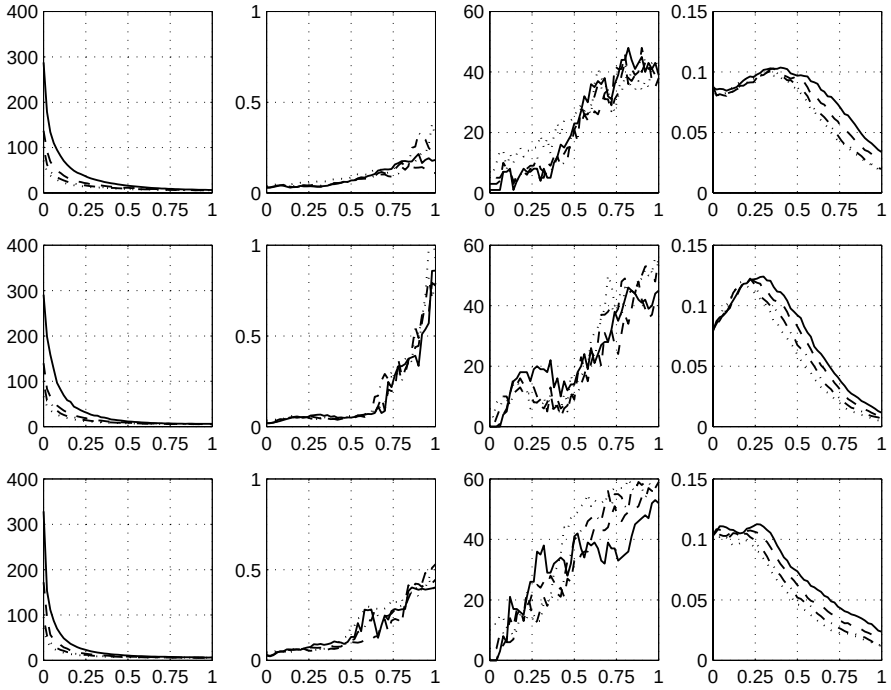


Figure 6.21: Result of pruning on objects #60, #03, and #85.
Left to right: Training time (sec), MAE, AE $> 5°$*, and r plotted against the pruning threshold.*
Top to bottom: results for objects #60*,* #03*, and* #85*.*
Line styles: Solid, dashed, dash-dot, and dotted correspond to $5°$*,* $10°$*,* $15°$*, and* $20°$ *view distance respectively.*

6.6 Experiments with views of a model car

We will now evaluate the associative networks on another data set, a model of a car rendered in a number of different views and scales. These views were generated during a master's thesis at CVL [57]. Each image in the dataset is a 256 RGB bitmap.

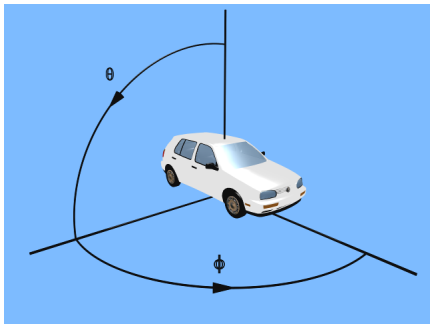


Figure 6.22: Different views of the model car.
Figure courtesy of Robert Söderberg [57].

The car is situated at the coordinate system origin, and the camera (always pointing at the origin), is moved along part of the surface of a sphere (see figure 6.22). The camera location is specified through the angular coordinates ϕ, and θ, which are defined as shown in the figure. The angle θ varies from 45° to 90° in 5° steps, while ϕ varies from 50° to −45° in 5° steps (see figure 6.23). This gives a total of $20 \times 10 = 200$ views.

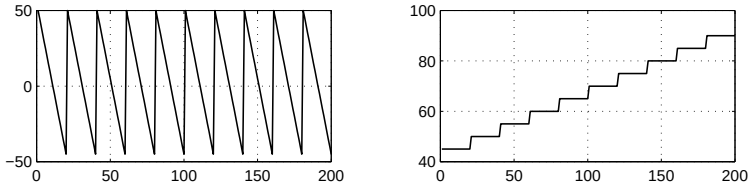


Figure 6.23: Variations of the angles ϕ and θ.
Left and Right: ϕ and θ as functions of the sample number.

The advantage with this data set is that the views define a two dimensional surface in state space, and not just a trajectory, as the views of the COIL-100 objects do.

6.6.1 Initial experiment

As a first test we define the responses using 10 channels for each response variable (see figure 6.24). Just like in the experiments with the COIL-100 objects, we use

DIV responses separated in four monopolar values as inputs. Again we go down to a 32×32 array, this corresponds to a spatial size of $\sigma = 11.1$ for the DIV operator.

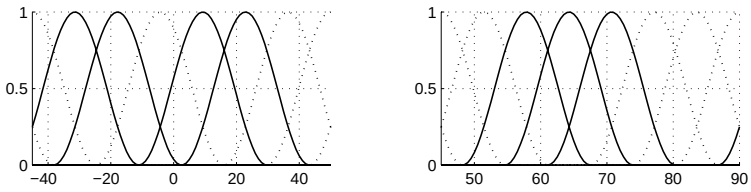


Figure 6.24: Channel encoding of the angles ϕ and θ.
Left and Right: ϕ and θ channels.
Most channels are plotted as dotted in order to ease their discrimination.

We thus have $N = 200$ samples, $K_1 + K_2 = 20$ responses, and $H = 4096$ features. If we train our network with these data, and evaluate on the same inputs, we obtain the results shown in figure 6.25. Note that the samples have been rearranged in the ϕ plot in order to ease the estimation of performance.

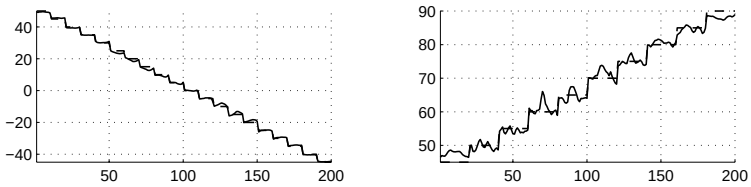


Figure 6.25: Reconstruction error for the angles ϕ and θ.
Left and Right: Reconstructed ϕ and θ responses.

We also calculate some of the performance measures used in the previous section. To ease comparison, RMSE and MAE are computed in radians:

Variable	r	RMSE	MAE	AE $> 5°$
ϕ	0.0629	0.0145	0.0112	0
θ	0.0621	0.0280	0.0220	1

As we can see, the non-zero coefficient ratio, r, is within the same range as in the COIL-100 experiments, and the average errors, RMSE and MAE, are both less than $1°$ for ϕ, and less than $2°$ for θ. The average errors are quite acceptable, however, the fact that the estimation of θ fails to estimate the correct view even though the training set equals the evaluation set is clearly unacceptable.

6.6.2 Covariant components

If we instead compute covariant components of the features, as suggested in section 6.1.4, we will end up with $H = 4096 \times (4096-1)/2 = 8386560$ features. This might sound like a lot of data, but we can get away with this by thresholding the features

before computing covariant combinations. The DIV responses normally lie in the range $[0, 1]$, and we now set all values below 0.1 to zero.

Before this operation we have a non-zero coefficient ratio of 0.4295 for the **A** matrix. Afterwards we have $r = 0.0644$. After computing covariant combinations we end up with a 8386560×200 matrix that is very sparse, $r = 0.0041$.

Using these inputs we obtain the results shown in figure 6.26. The performance measures are collected in the table below.

Variable	r	RMSE	MAE	AE $> 5°$
ϕ	0.0115	0.0021	0.0012	0
θ	0.0069	0.0074	0.0058	0

As can be seen, the use of covariant components greatly improves the ability of the network to discriminate between different views. However, they should be used with care, since they also make the feature functions much more localised, and thus reduce the interpolating ability of a network. Evaluation of the interpolating ability of an outer product network is left for future research.

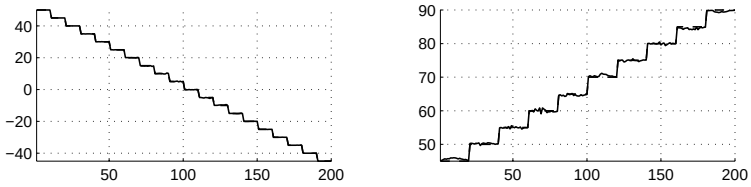


Figure 6.26: Reconstruction error for the angles ϕ and θ.
Left and Right: Reconstructed ϕ and θ responses.

Chapter 7

Sparse template matching

7.1 Product sums on sparse data

In chapter 2 we noted that biological vision systems seem to make use of sparse coding of their inputs. We also stated that sparse coding reduces template storage requirements, and improves matching performance. We are now going to examine these claims more thoroughly by comparing matching results on some different kinds of data.

A common artificial neuron model consists of a number of inputs and one output (see figure 7.1). The computation of the output is a weighted summation of the inputs, followed by a non-linear function. The computation performed by such an artificial neuron can be seen as a form of template matching, where the template consists of the weights, and the inputs are the data we compare with the template.

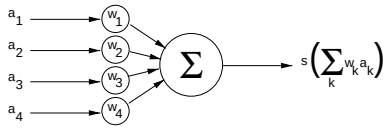


Figure 7.1: Simple neuron model.

The response of the neuron can be viewed as a degree of match. Ideally one would like to have the maximum response when the input equals the template, and a gradual decrease as they become more different. As we shall see, the representation of the inputs, a_k, matters a great deal for the performance of this kind of matching.

7.1.1 Intensity based matching

We will now try to recognise an image region containing a car using various representations of the inputs. We cut out a small region around the car in the intensity

image (see figure 7.2), and use it as a template. (This corresponds to the weights, w_k, in the neuron model in figure 7.1). We then try to match all possible cut outs of the intensity image with the template using a plain product sum (see equation 7.1). We have omitted the non-linear function s(m) (see figure 7.1) at this stage, since it in general is a monotonous function, and thus only of interest when the response is fed to another neuron.

$$\text{match} = \sum_k w_k a_k \tag{7.1}$$

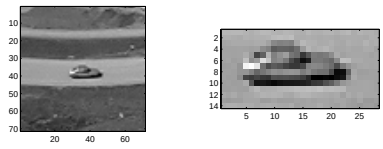


Figure 7.2: Intensity image and template.

The result of this matching is illustrated in figure 7.3 (left). As can be seen, product sums are very sensitive to the magnitude of the signal values. If we invert the intensity values in both image and template, we end up with the match in the next figure of the plot. This phenomenon (i.e. the fact that product sum matching results in high match values at locations with high values, regardless of how the template looks) is a serious problem.

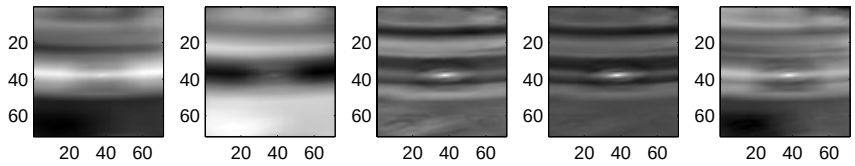


Figure 7.3: Intensity based matching.
Left to right: Product sum, Product sum with inverted intensity, Normalised product sum, Normalised product sum with inverted intensity, SSD (inverted colourmap).

One way to deal with this problem is to normalise the matching, i.e.

$$\text{match} = \frac{\sum_k w_k a_k}{\sum_k w_k \sum_k a_k} \tag{7.2}$$

In this way the intensity relative to neighbouring intensities will be important, instead of the actual values. This operation can be simplified somewhat by normalising the weights w_k beforehand, to ensure that they sum to 1. The division by $\sum_k a_k$ however, has to be performed in each matching, and thus adds an extra

computational cost. The normalised matching described here is a special case of *normalized convolution* [36, 61, 12], where the response strength equals the response certainty. The third and fourth images in figure 7.3 show the result of normalised product sum matching. As can be seen, this result is much better.

An other way to reduce the influence of high intensities is to subtract the average before matching:

$$\text{match} = \sum_k (w_k - m_k)(a_k - m_k) \quad \text{where} \quad m_k = \frac{w_k + a_k}{2} \tag{7.3}$$

This expression can be rewritten into $\sum_k -0.25(w_k - a_k)^2$. That is, we look at the squared difference of image and template values. If we remove the constant -0.25 we have arrived at an operation that is commonly known as *Sum of Squared Difference*, SSD.

$$\text{match} = \sum_k (w_k - a_k)^2 \tag{7.4}$$

The result of this operation is shown in figure 7.3 (right). Since good matches are represented by low values in this method, we have inverted the palette to aid comparison.

As can be seen in figure 7.3, the horizontal structure in the input image is also present in the response of all the intensity based matching techniques.

7.1.2 Difference of Gaussian based matching

As mentioned in chapter 2, the *centre-surround* cells (or M-type ganglion cells) of the mammalian retina perform an operation that is usually modelled as a *Difference of Gaussian* (DoG) computation. That is, each response is computed as a difference between two low-pass filtered versions of the input. From an evolutionary point of view, this computation must be to some kind of advantage to the organism. One possible advantage is, as we shall see, that it aids product sum matching.

Centre-surround cells are somewhat similar in function to SSD, since both look at deviations from the average intensity value. An important difference however is that centre-surround cells distinguish between locations above and below the average. The use of this kind of monopolar signals seems to be widespread in biological vision systems (see chapter 2), but is rarely found in machine vision. It has however been advocated in [20].

We can model the behaviour of the centre-surround cells with the following responses:

$$\begin{aligned} y_0(\mathbf{x}) &= (s * g_1)(\mathbf{x}) - (s * g_2)(\mathbf{x}) \\ y_1(\mathbf{x}) &= \max(0, y_0(\mathbf{x})) \\ y_2(\mathbf{x}) &= \max(0, -y_0(\mathbf{x})) \end{aligned} \tag{7.5}$$

Here g_1 is a broad-band Gaussian[1] kernel, while g_2 is mainly of low frequency. The resultant responses $y_1(\mathbf{x})$ and $y_2(\mathbf{x})$ are thus the positive and negative parts of a band-pass response. We now attempt the same kind of operation as in the intensity based matching: We compute DoG on the images and select templates in the region corresponding to the car location (see figure 7.4). Note that since we have two kinds of responses, we will have two templates, and two feature images to match.

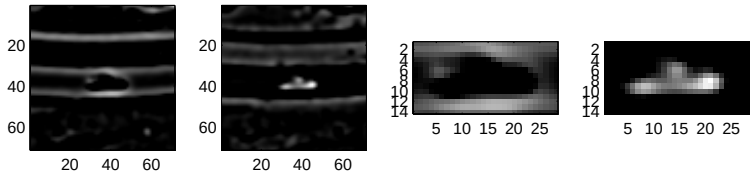


Figure 7.4: DoG responses and templates.
Left: $y_1(\mathbf{x})$ and $y_2(\mathbf{x})$ images. Right: the two resultant templates.

The result of the matching is shown in figure 7.5. The first two responses are the matching of individual templates with the corresponding images. The third response is their sum, and the fourth is their product. The use of a product can conceptually be seen as a requirement of match of both the positive *and* the negative templates.

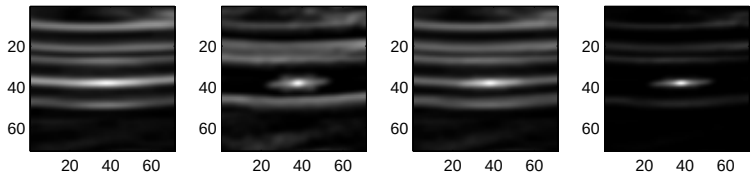


Figure 7.5: DoG based matching.
Left to right: Positive centre-surround matching, Negative centre-surround matching, sum of matches, product of matches.

As can be seen, the responses are more selective than those obtained from intensity based matching, especially when the two matches are combined as a product. However, the horizontal structure in the input image is also present in the responses at this level.

It is interesting to note that the normalisation (see equation 7.2) in a sense is similar to applying a difference of Gaussian operation before the matching. If we see the normalisation as a preprocessing step:

[1]For efficient implementation, Gaussian kernels should be separated into two one-dimensional kernels in the horizontal and vertical directions respectively.

$$r_0(\mathbf{x}) = (s * g_1)(\mathbf{x}) \qquad (7.6)$$
$$r_1(\mathbf{x}) = (s/r_0)(\mathbf{x})$$

which is followed by a plain product sum matching, we note that large values in the surrounding region will cause the response, $r_1(\mathbf{x})$, to drop. The same applies to the DoG computation in equation 7.5. However, DoG is linear, while the normalisation (see equation 7.6 above) is not.

7.1.3 Edge based matching

The next level of mammalian visual processing is usually modelled as Gabor type wavelets. The approximate Gabor wavelets in the mammalian visual processing exhibit variations in scale, orientation, and phase, but we will here settle for one scale, horizontal and vertical orientations, and the $\pm\pi/2$ phases.[2] For computational efficiency we will also replace the Gabor wavelets with differentiating Gaussians. A differentiating Gaussian filter in the x-direction can be separated into a Gaussian low-pass filter[3] $\mathrm{g_x}$ and a small differentiating kernel $\mathrm{d_x} = [\ -1 \quad 0 \quad 1\ ]$. To improve the robustness of the result we also want to low-pass filter the result with a Gaussian $\mathrm{g_y}$ in the y-direction. If we make the two low-pass filters $\mathrm{g_x}$ and $\mathrm{g_y}$ the same size, we can implement edge filtering in the x and y directions as a separable Gaussian low-pass filtering followed by a small differentiating convolution for each direction:

$$\begin{aligned} \mathrm{e_1}(\mathbf{x}) &= (\mathrm{s} * \mathrm{g_x} * \mathrm{d_x} * \mathrm{g_y})(\mathbf{x}) \\ \mathrm{e_2}(\mathbf{x}) &= (\mathrm{s} * \mathrm{g_y} * \mathrm{d_y} * \mathrm{g_x})(\mathbf{x}) \end{aligned}$$

$$\Downarrow$$

$$\begin{aligned} \mathrm{e_0}(\mathbf{x}) &= (\mathrm{s} * \mathrm{g_x} * \mathrm{g_y})(\mathbf{x}) \\ \mathrm{e_1}(\mathbf{x}) &= (\mathrm{e_0} * \mathrm{d_x})(\mathbf{x}) \\ \mathrm{e_2}(\mathbf{x}) &= (\mathrm{e_0} * \mathrm{d_y})(\mathbf{x}) \end{aligned}$$

The standard deviation parameter, σ, of the Gaussian filters can be used to adjust the scale of these edge filters. Figure 7.6 shows the positive and negative parts of $e_1(\mathbf{x})$ and $e_2(\mathbf{x})$ respectively, and figure 7.7 shows the corresponding templates.

The result of matching on this type of input is shown in figure 7.8. As can be seen in the rightmost image, product matching on this kind of inputs produce responses that are quite selective, yet they show a smooth transition from match to non-match.

Thus, we can see that sparse inputs improve the quality of the matching, at the cost of an extra preprocessing stage.

[2] One motivation for choosing $\pm\pi/2$ phases is that they are potentially *characteristic phases*, and thus likely to be stable over scale, see chapter 3.

[3] A Gaussian low-pass filter is a truncated and discretised representation of the function $\mathrm{g_x}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$.

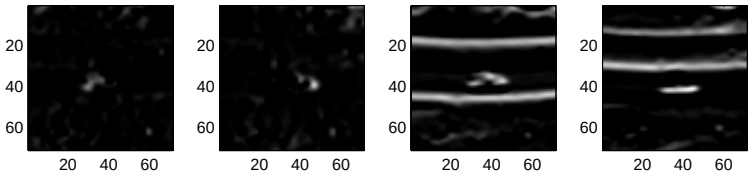


Figure 7.6: Edge responses.
Left to right: Positive horizontal, negative horizontal, positive vertical, negative vertical.

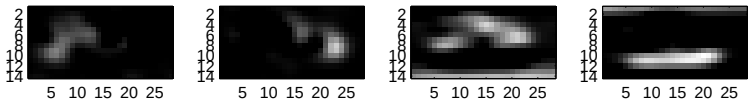


Figure 7.7: Edge templates.
Left to right: Positive horizontal, negative horizontal, positive vertical, negative vertical.

7.1.4 Sparse template matching

As mentioned in chapter 2, the higher levels of mammalian visual processing tend to employ increasingly sparse representations. As can be seen in figure 7.6, edge responses are quite sparse. We will now illustrate that sparse coding the inputs, not only improves the performance of the matching, but in many applications it also reduces the computational load.

The key idea of sparse template matching is illustrated in figure 7.9. In the intensity representation we have to verify the correspondence between image and template in all template positions, but at the sparse feature representation **we only have to compare those features that are active in the template**. The reason for this is that in a product sum match (see equation 7.1) the template coefficients, w_k, with large magnitudes will dominate the result completely. Thus, a *pruning*, or removal of low-magnitude coefficients in the templates will have little impact on the result.

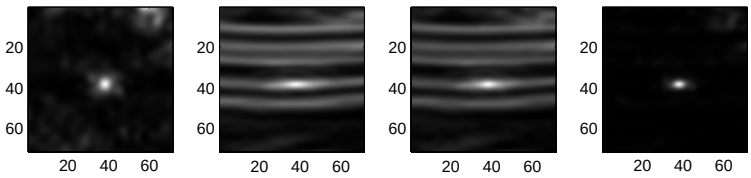


Figure 7.8: Edge based matching.
Left to right: Sum of horizontal responses, Sum of vertical responses, Sum of all responses, Product of horizontal and vertical responses.

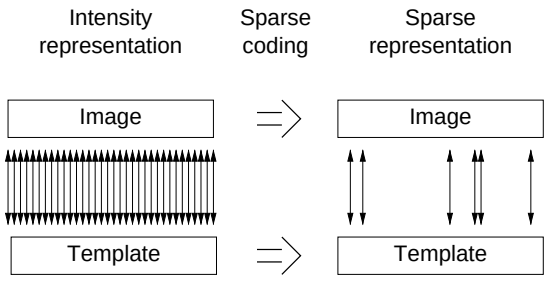


Figure 7.9: Sparse template matching.

In the next section we will demonstrate that the edge templates shown in figure 7.7 can be made much more sparse, in a way that will hardly affect the performance of the matching at all. The degree of sparsity is directly proportional to the total computational load of the matching. If we for instance only use 1/50 of the template locations, the matching will be 50 times faster.

Note however that the sparse template matching approach is only useful when the computational cost of sparse coding the image and template is smaller than the cost of performing the matching. This is definitely the case in applications where we want to match several templates to one image, since the sparse coded image will only have to be computed once. Most template-matching based systems of any degree of complexity will naturally require comparison of several templates to the image.

7.2 Sparse adaptive templates for fast matching

This section contains material from the paper "Sparse adaptive templates for fast matching"[4], which describes an application of sparse template matching. Some parts of the material has been left out, since it has already been covered in the preceding sections.

The paper describes a window matching technique for use in estimation of *ego-motion* (Ego-motion is the camera motion as computed from a sequence of camera images) in a system with a moving camera. A major issue for such systems is the real time requirement. We perform window matching by transforming image data into sparse features, and apply a computationally efficient matching technique in the sparse feature space. The gain in execution time for the matching is roughly 25 times compared to full window matching techniques such as SSD, MSD, and MAD, but the total execution time for the matching also involves computation of an edge image. The proposed matching technique is increasingly advantageous when the number of regions to keep track of increases, and when the size of the search window increases.

[4]The paper is currently under review.

7.2.1 Introduction

This paper describes a window matching technique that has been developed for the purpose of *ego-motion* estimation in the scope of the WITAS project [25]. Ego-motion is a well studied problem, and a common way to accomplish it is by means of keeping track of local image regions [33].

Region tracking is the problem of estimating the displacements of local image regions in a sequence of camera images. If we know the displacements of local image patches, we can estimate the change in perspective, and with the additional knowledge of some camera parameters, we can finally compute the camera motion.

The problem setup in ego-motion estimation provides us with a great deal of useful *a priori* information. Region tracking is a much simpler problem than general template matching, and correspondence in general. Some important assumptions that can be made are:

- Due to the inertial constraints of the real world, it is reasonable to assume that the displacements of image regions between two frames is limited, and that the change of scale is gradual.
- Illumination changes in the scene can be assumed to be gradual, and in the cases when they are sharp (for instance shadows) they will not change all of the scene at once.
- Not all regions in the scene exhibit the aperture problem.

We thus start our region tracking by selecting a set of neighbourhoods that do not exhibit the aperture problem (more on this later). Some representation of the selected neighbourhoods is stored as *templates*, and these templates are used as long as possible. When the mismatch between template and scene becomes too large (for instance due to occlusion, illumination changes or scale change), a new template is selected.

Region tracking is often performed by *block matching* methods, such as SSD[5] (Sum of Squared Difference of the pixels in both regions), for instance in video coding applications [46]. We will instead make a transformation of the image data, and then apply a less computationally intensive matching technique.

7.2.2 Sparse coding

Sparse coding is a way to transform data in such a way that patterns are easy to detect. Typically inputs and outputs are represented in a *channel representation* [22]. That is, signals are limited, and positive. A non-zero channel value signifies something, while a zero response means "no information". In our case the input is an intensity image. Here each pixel can be viewed as a channel, and a non-zero response indicates that light has hit the corresponding sensor.

David Field [13] discusses sparse coding of natural images in depth, and contrasts it to compact coding techniques such as PCA. The goal of compact coding

[5] An equivalent term is MSD (Mean Squared Difference). Another similar method is called MAD (Mean Absolute Difference).

is to minimise the number of output nodes, by concentrating the signal entropy in a small number of nodes. Sparse coding, on the other hand tries to concentrate the information content in the *active* nodes. The total number of nodes is allowed to remain the same as in the input, or even to increase.

There are several proposed optimisation schemes to find sparse transforms for a given set of input data, for instance [14, 48, 5]. The mammalian retina and primary visual cortex also seem to make use of a similar optimisation technique [13].

The result of sparse coding is a representation of the input as a combination of a few commonly occurring sub-patterns or *independent components*. For natural images, these sub-patterns consist of line and edge segments in a number of scales [48, 5], and this is our motivation for choosing edge detection as a first filtering of our input. Other kinds of sparse data, such as the divergence and rotational symmetries computed in [34] could of course also be used by the template matcher.

7.2.3 Edge images

The sparse features we will use are edge filter responses. To compute an edge image is fairly straight-forward, and it can be made computationally efficient if we use separable differentiating Gaussian filters (see section 7.1.3). The following computations are needed:

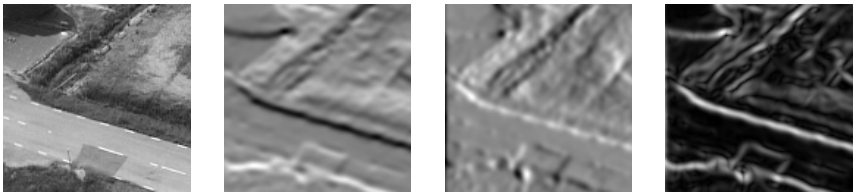
$$\begin{aligned} \mathrm{e}_0(\mathbf{x}) &= (\mathrm{s} * \mathrm{g}_\mathrm{x} * \mathrm{g}_\mathrm{y})(\mathbf{x}) \\ \mathrm{e}_1(\mathbf{x}) &= (\mathrm{e}_0 * \mathrm{d}_\mathrm{x})(\mathbf{x}) \\ \mathrm{e}_2(\mathbf{x}) &= (\mathrm{e}_0 * \mathrm{d}_\mathrm{y})(\mathbf{x}) \end{aligned}$$

The standard deviation parameter, σ, of the Gaussian filters can be used to adjust the scale of these edge filters. Throughout this paper $\sigma = 2.0$ has been used. The resultant Gaussian filter has 15 real coefficients, and thus our edge filtering involves a total of $15 + 15 + 2 + 2 = 34$ coefficients.

The sum of the magnitudes of the edge responses, $\mathrm{e}(\mathbf{x}) = \mathrm{abs}(\mathrm{e}_1(\mathbf{x}))+\mathrm{abs}(\mathrm{e}_2(\mathbf{x}))$, is plotted in figure 7.10. This plot illustrates an important property of the information representation in the edge images. As we can see, only a small fraction of the image positions will have large magnitudes. If the matching of this kind of data is performed as a product sum, a small fraction of the template coefficients will have a dominating influence on the result. Thus, a *pruning*, or removal of low-magnitude coefficients in the templates will have little impact on the result.

7.2.4 Template construction

Edge-image based matching is nothing new, but it has traditionally involved an additional distance map computation [18]. The purpose of the distance map computation is to obtain *metric*—a measure of how close we are to an edge in each image position. By summing the distances to edges in all positions indicated as edges in the template, we can obtain a measure that increases smoothly as we move away from the location of the best match.



$\mathrm{s}(\mathbf{x})$ $\mathrm{e}_1(\mathbf{x})$ $\mathrm{e}_2(\mathbf{x})$ $\mathrm{e}(\mathbf{x})$

Figure 7.10: Edge image construction.
e_1 *and* e_2 *have a colour map which displays positive values as bright, and negative values as dark.*

However, an edge image computed at low frequency already has a *local* distance metric: The highest values are always found at the edge location, while the response magnitude slowly decreases as we move away from the edge. Thus, we can avoid the distance map computation by using low frequency edge responses instead.

An other difference in our approach is that we will not perform the matching on the compact feature map $\mathrm{e}(\mathbf{x})$, but instead on four *sparse* feature maps, or *feature channels*. When combining $\mathrm{e}_1(\mathbf{x})$ with $\mathrm{e}_2(\mathbf{x})$ to form $\mathrm{e}(\mathbf{x})$ we introduce unnecessary confusion, and remove information that could aid the matching. Our sparse feature maps are instead constructed as the positive and negative parts of $\mathrm{e}_1(\mathbf{x})$ and $\mathrm{e}_2(\mathbf{x})$:

$$\begin{aligned}
\mathrm{c}_1(\mathbf{x}) &= \max(0, \mathrm{e}_1(\mathbf{x})) \\
\mathrm{c}_2(\mathbf{x}) &= \max(0, -\mathrm{e}_1(\mathbf{x})) \\
\mathrm{c}_3(\mathbf{x}) &= \max(0, \mathrm{e}_2(\mathbf{x})) \\
\mathrm{c}_4(\mathbf{x}) &= \max(0, -\mathrm{e}_2(\mathbf{x}))
\end{aligned}$$

Here $\mathrm{c}_1(\mathbf{x})$ and $\mathrm{c}_2(\mathbf{x})$ signify dark-to-bright and bright-to-dark edges in the x-direction respectively, while $\mathrm{c}_3(\mathbf{x})$ and $\mathrm{c}_4(\mathbf{x})$ signify edges in the y-direction.

To illustrate the sparse template construction, we start from a region in the scene that does not have the aperture problem (see figure 7.11).

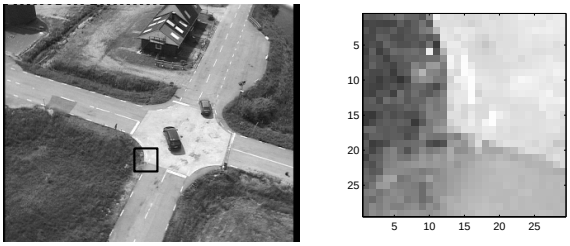


Figure 7.11: A region without the aperture problem.

Such a region can be found by first constructing a tensor image $\mathbf{T}(\mathbf{x})$:

$$\mathbf{T}(\mathbf{x}) = \begin{pmatrix} \mathrm{e}_1(\mathbf{x}) \\ \mathrm{e}_2(\mathbf{x}) \end{pmatrix} \cdot \begin{pmatrix} \mathrm{e}_1(\mathbf{x}) & \mathrm{e}_2(\mathbf{x}) \end{pmatrix}$$

We then average the tensor components in local image regions (this can be implemented as a separable low-pass filtering) and select points with two large eigenvalues [43]. This is equivalent to the Harris corner detector [28].

For 2×2 matrices, the eigenvalues λ_1 and λ_2 (with $\lambda_1 \geq \lambda_2$) can be computed as:

$$\begin{aligned} a &= \sqrt{\|\mathbf{T}\|^2 + 2\det|\mathbf{T}|} \\ b &= \sqrt{\mathrm{abs}(\|\mathbf{T}\|^2 - 2\det|\mathbf{T}|)} \\ \lambda_1 &= (a+b)/2 \\ \lambda_2 &= (a-b)/2 \end{aligned}$$

Since we know that $\lambda_1 \geq \lambda_2$, points with two large eigenvalues can easily be found by looking at local peaks in a λ_2 image.

The positive and negative parts of the edge filter responses for the region shown in figure 7.11 are shown in the top row of figure 7.12. The first step in the sparsification of the templates is a directed *non-max-suppression* in each of the templates. That is, we loop over the two vertical templates, and set all positions that are smaller than either the neighbour above or below, to zero. The same procedure is performed on the horizontal templates, but here the left and right neighbours are checked.[6]

The directed non-max-suppression operation can be seen as a crude approximation of the lateral inhibition mechanisms found in biological vision systems.

A less crude, but more computationally demanding way to obtain this result could be to detect *characteristic phase* as was done in chapter 3.

To control the sparseness of the templates, the remaining non-zero positions are sorted according to magnitude, and those belonging to the largest percentile are kept. In this process the two vertical templates are sorted together, as are the two horizontal. We thus know that after the sparsification $N_1 + N_2 = N_3 + N_4$ (where N_k is the number of remaining coefficients in template k).

The bottom row of figure 7.12 shows the templates after directed non-max-suppression and removal of everything except the top 2% percentiles of the horizontal and vertical templates respectively. As can be seen, the operation will produce a template in which the spatial extent of the ridges are kept, even for very sparse templates.

This degree of sparsity works fine for smaller templates as well [7], and results in a matching that is roughly 25 times faster than SSD. The total execution time for the

[6] Note that non-max-suppression destroys the local metric in the templates only. There still are continuous variations in the edge responses we match the templates with, and this is what gives us continuous responses.

[7] The limit seems to be about 15×15 for 2% templates. This implies a total of $5 + 5 = 10$ coefficients, less is apparently not enough to describe the shape of the ridges.

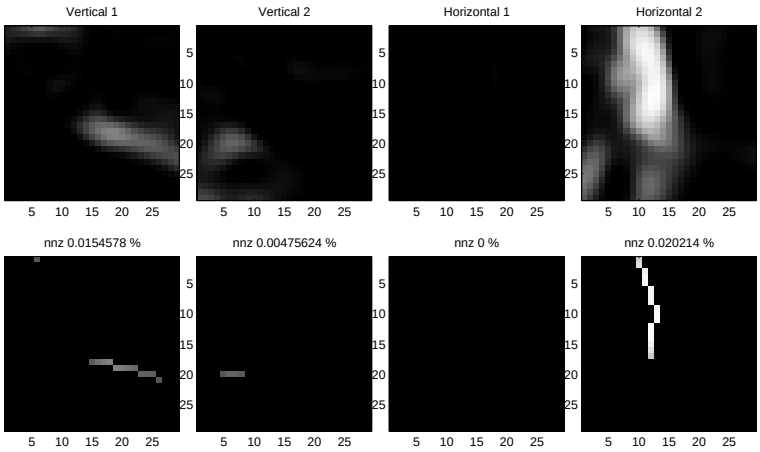


Figure 7.12: Sparsification of a template.

window matching algorithm should of course also include the edge-image creation. However, an important thing to note here is that the execution time for sparse template matching is not as sensitive to the size of the search window as SSD. Nor is the number of windows to match as important, and this fact is especially important for estimation of perspective changes, where more displacement estimates mean more robust estimations.

7.2.5 Sparse template matching

From e_1 and e_2 we extract the template regions t_1 and t_2. The corresponding *template channels* are denoted $\mathrm{tc}_1 \dots \mathrm{tc}_4$. To formulate the matching procedure we also add a conceptual index n, that loops over the template coefficients left after the sparsification. We can now write the product-sum matching as:

$$\begin{aligned} \mathrm{m_v} &= \sum_{n=1}^{N_1} \mathrm{tc}_1(\mathbf{x}_n)\mathrm{c}_1(\mathbf{x}_0+\mathbf{x}_n) + \sum_{n=1}^{N_2} \mathrm{tc}_2(\mathbf{x}_n)\mathrm{c}_2(\mathbf{x}_0+\mathbf{x}_n) \\ &= \sum_{n=1}^{N_1+N_2} \max(0, \mathrm{t}_1(\mathbf{x}_n)\mathrm{e}_1(\mathbf{x}_0+\mathbf{x}_n)) \quad \text{and:} \end{aligned} \tag{7.7}$$

$$\mathrm{m_h} = \sum_{n=1}^{N_3+N_4} \max(0, \mathrm{t}_2(\mathbf{x}_n)\mathrm{e}_2(\mathbf{x}_0+\mathbf{x}_n)) \tag{7.8}$$

Since the product sum can be written using e_1, and e_2, we do not have to store the positive, and negative parts of the edge responses separately. The separation is only conceptual.

The only thing that now makes this operation different from a plain product sum is the max operation in equations 7.7 and 7.8. The meaning of this max

operation would be that *mismatches are not ranked* (see figure 7.13). This is a characteristic property of monopolar product-sum matching.

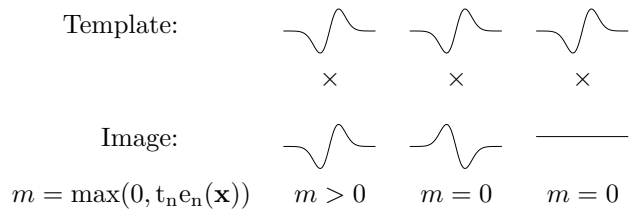


Figure 7.13: Edge image matching.

It turns out however, that the removal of the max operation hardly affects the response near a peak at all. In order to save some time, we will thus compute the matches as:

$$\mathrm{m_v} = \max(0, \sum_{n=1}^{N_3+N_4} \mathrm{t}_1(\mathbf{x}_n)\mathrm{e}_1(\mathbf{x}_0 + \mathbf{x}_n)) \tag{7.9}$$

$$\mathrm{m_h} = \max(0, \sum_{n=1}^{N_3+N_4} \mathrm{t}_2(\mathbf{x}_n)\mathrm{e}_2(\mathbf{x}_0 + \mathbf{x}_n)) \tag{7.10}$$

That is, we move the max operation outside the sum. The compound match can now be computed as:

$$\mathrm{match} = \mathrm{m_v} \cdot \mathrm{m_h} \tag{7.11}$$

This compound match requires a high degree of match for features in both vertical and horizontal direction. As we can see, the max operations in equations 7.9 and 7.10 are needed, since large mismatches in both horizontal and vertical directions would otherwise result in a high total match.

The result of this matching technique with the templates in the bottom row of figure 7.12 is shown in figure 7.14. The leftmost image in the figure shows the frame in which the matching has been performed (not the same as the one from which the template was extracted), and the centre and right images show the $\mathrm{m_v}$ and $\mathrm{m_h}$ responses respectively.

The compound matching result is shown in figure 7.15 together with the response from a full edge-image product sum, and SSD. The SSD result has a colourmap with low values shown as bright in order to aid comparison. Details of this figure are shown in figure 7.16. This will hopefully illustrate that sparse template matching produces results that are at least as useful as SSD. Both methods produce results that gradually increase near the peak–a sign of graceful degradation. The smoothness of the peak in figure 7.16 is controlled by the σ parameter of the initial low-pass filters $\mathrm{g_x}$ and $\mathrm{g_y}$. The figures in this paper all use $\sigma = 2.0$.

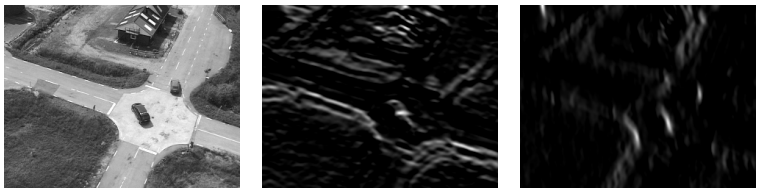


Figure 7.14: Matches in vertical and horizontal directions.

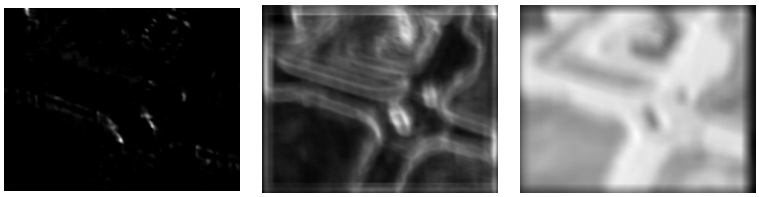


Figure 7.15: Sparse template matching response compared with edge-image product sum, and SSD.

In a region tracking situation, we would like to keep our templates as long as possible, since each time we select a new template we introduce an error. If we do not use sub-pixel resolution coordinates, this error is in the range $[-0.5, 0.5]$ pixel in each dimension. Graceful degradation is very useful here, since we can use the degree of match as an indication of when to select a new template. In the current implementation new templates are selected when the degree of match deviates too much from the expected value, i.e.:

$$\text{Keep template if} \quad \frac{\text{match}(\mathrm{T}, \mathrm{I})}{\text{match}(\mathrm{T}, \mathrm{T})} \in \left[\epsilon, \frac{1}{\epsilon}\right] \tag{7.12}$$

Where $\epsilon = 0.9$ or similar. Since $\text{match}(\mathrm{T}, \mathrm{T})$ is constant for each template, we could simply divide all template coefficients with this value beforehand to simplify the comparison. The normalisation described here is an approximation of the normalised product sum described in section 7.1.1. Ideally we should divide the

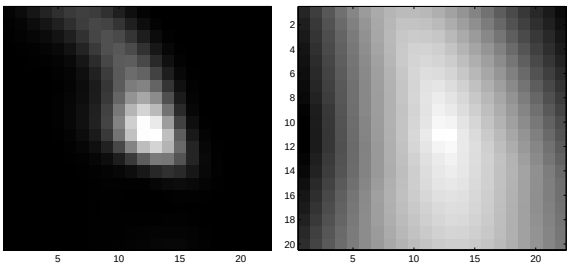


Figure 7.16: Detail of figure 7.15. Sparse template matching and SSD responses.

match by the coefficient sums of the template and the image region, but since the image content varies, this is computationally expensive.

The gradual increase near the peak in the response image implies that we could use the response image to find the displacement with sub-pixel resolution. This has been done, for instance in [43] and [18]. See appendix B for a description of how this is done.

7.2.6 Sigmoid-like function

A common problem with product sums on edge images is that very sharp edges will get very high responses, and will thus tend to dominate the result completely. This problem can be dealt with by using normalised product sums (see section 7.1.1), but these have the disadvantage of increasing the computational load. It is also dealt with to some extent by the directed non-max-suppression treatment of the templates (see section 7.2.4), since this will tend to make the shape of the edge ridges more important than their actual values.

An approach that is less computationally demanding than the normalised product sums is to apply a sigmoid-like function on the edge image responses before they are used. An example of such a function that has been found to be satisfactory is:

$$\begin{aligned} \mathrm{n}(\mathbf{x}) &= \mathrm{e}_1(\mathbf{x})^2 + \mathrm{e}_2(\mathbf{x})^2 \\ \mathrm{f}_1(\mathbf{x}) &= \mathrm{sign}(\mathrm{e}_1(\mathbf{x}))\mathrm{e}_1(\mathbf{x})^2/(500 + \mathrm{n}(\mathbf{x})) \\ \mathrm{f}_2(\mathbf{x}) &= \mathrm{sign}(\mathrm{e}_2(\mathbf{x}))\mathrm{e}_2(\mathbf{x})^2/(500 + \mathrm{n}(\mathbf{x})) \end{aligned}$$

The parameter 500 in the denominator applies to images in the range $[0 \ldots 255]$, and Gaussian filters g_x and g_y that sum to 1.

The main advantage with this function is that it preserves the symmetrical shape of the matching response near the peaks (see figure 7.16), contrary to most non-linear functions operating on e_1 and e_2 separately.

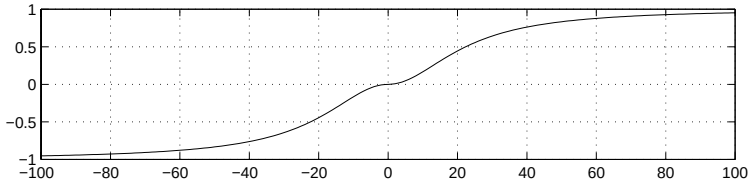


Figure 7.17: Sigmoid-like function ($\mathrm{f}_1(\mathrm{e}_1)$ for $\mathrm{e}_2 = 0$).

The sigmoid-like function can be seen as a soft threshold of the edge data. I.e. once the edge response is above a certain threshold it does not matter much how much above the threshold it is (see figure 7.17). Since the variations in edge image magnitude will become smaller when a sigmoid is applied, the decision threshold for changing template will also have to be raised (see equation 7.12).

One reason for not using a sigmoid-like function is the small amount of computational overhead it adds. In the current implementation this is approximately 20% of the time needed to compute the edge responses. An advantage, compared to the directed non-max-suppression approach is that non-max-suppression seems to degrade the performance of subsequent sub-pixel-resolution methods slightly.

Of course there is also the option to use both a sigmoid-like function and directed non-max-suppression. If a fast stable window matcher without sub-pixel accuracy is sought, this is the way to go.

Chapter 8

Future research directions

8.1 End notes

As can be seen by browsing this thesis, this is a work in progress. At the moment, there are two main directions in which the work is planned to progress.

8.1.1 Ego-motion estimation

The first area that will be investigated is development and expansion of ego-motion estimation methods. This work will build on the sparse template matching technique described in chapter 7, as well as on a preliminary study of camera parameter updates [15].

8.1.2 Associative networks

The second area to investigate is augmentation of the associative networks described in chapter 6.

Temporal continuity and contextual information

The first area that will be addressed here is addition of temporal continuity constraints and contextual information. The networks described in this thesis are all static[1], and for more complex problems this is not a realistic approach.

Intermediate level network responses and place cells

The second issue to address within the area of associative networks is generation of intermediate level network responses. For more complex problems this will be a necessity. Intermediate responses should probably have a more relaxed specification than those at the final level. They should however be guided by a need to

[1] They are static in the sense that they do not take the temporal order of things into account.

model specific system states, and thus amongst others, the approaches described in [3] and [38] will be investigated further.

Appendices

A Some theorems concerning $\cos^2()$ channels

We will now have look at the values of a set of channels, and derive some general results concerning channel encoding. We will use the same notations as earlier, i.e. the first active channel will be called k, and the number of active channels will be called N.

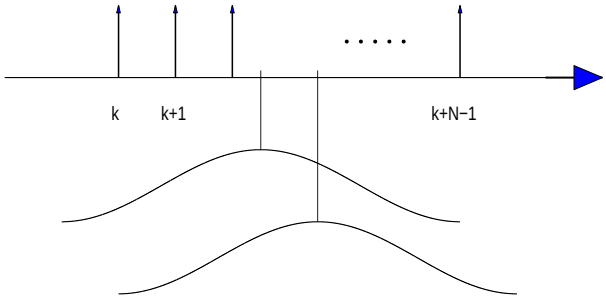


Figure A.1: Valid range.
Valid range for local reconstruction.

Theorem A.1 *If the channels $\psi_k(s)$ through $\psi_{k+N-1}(s)$ can be activated by a single scalar, the scalar has to lie within the range $k + N - 1 - \frac{\pi}{2\omega} \leq s \leq k + \frac{\pi}{2\omega}$.*

The validity of this statement is motivated by figure A.1. The lowest scalar value that could have activated the channels $\psi_k(s)$ through $\psi_{k+N-1}(s)$ is one for which the envelope function $\psi_s(x)$ is just above zero at position $x = k + N - 1$. This happens when the function just about touches the point $k+N-1$. Since the function extends $\frac{\pi}{2\omega}$ in both directions from s, this happens when $s \geq k+N-1-\frac{\pi}{2\omega}$.

In the same manner we can see that the highest channel value occurs when the envelope function only just touches the point k. This leads to an upper limit of $s \leq k + \frac{\pi}{2\omega}$.

Note 1 If we want to check consecutive groups of N channel values for a solution, we should set one of the range limits to a strict inequality, to avoid duplicate solutions when the channel value is exactly on the boundary of two intervals.
Note 2 If $\omega \neq \frac{\pi}{n}$ where $n \in \mathcal{N}_+$ we will have two types of intervals (see theorem A.2). For these cases, every other solution interval will involve N_l channel values, and N_h respectively. Since a large support solution is more robust, it is preferable, the valid interval for small support solutions should thus be restricted to the range between two large support regions. i.e. $\frac{\pi}{2\omega} - 1 \leq s \leq N - \frac{\pi}{2\omega}$.

Theorem A.2 *The number of channels N activated by one scalar is related to the overlap ω according to $\frac{\pi}{N+1} \leq \omega < \frac{\pi}{N-1}$.*

If we look at the definition of the envelope function (equation 4.1) we see that its non-zero interval is $n + \frac{\pi}{2\omega} < s < n - \frac{\pi}{2\omega}$. The length of this interval is $\frac{\pi}{\omega}$. Since the start and the end of the interval has the function value zero, the worst case is to let the interval start at an integer position, and end at an integer position. In this case, the number of non-zero channel values becomes $N \geq \frac{\pi}{\omega} - 1$. The best case is when the range is expanded an infinitesimal amount in each direction. This leads to two more non-zero values, and the (now strict) inequality $N < \frac{\pi}{\omega} + 1$.

This gives the following restriction on N:

$$\frac{\pi}{\omega} - 1 \leq N < \frac{\pi}{\omega} + 1 \tag{A.1}$$

This can be rewritten as a restriction of ω in a number of steps:

$$\frac{\pi}{\omega} - 1 \leq N < \frac{\pi}{\omega} + 1 \tag{A.2}$$

$$-1 - N \leq -\frac{\pi}{\omega} < 1 - N \tag{A.3}$$

$$N + 1 \geq \frac{\pi}{\omega} > N - 1 \tag{A.4}$$

$$\frac{1}{N+1} \leq \frac{\omega}{\pi} < \frac{1}{N-1} \tag{A.5}$$

$$\frac{\pi}{N+1} \leq \omega < \frac{\pi}{N-1} \tag{A.6}$$

And we have arrived at the expression stated in theorem A.2.

Theorem A.3 *The sum of a channel value vector $\begin{pmatrix} \psi_1(s) & \psi_2(s) & \dots \end{pmatrix}$ does not depend on the value of s if $\omega = \pi/N$, where N is an integer $N \geq 2$.*

To prove this theorem, we first define a complex valued function as:

$$\boldsymbol{v}_k(x) = e^{\boldsymbol{i} 2\omega(x-k)} \tag{A.7}$$

With the properties:

$$\mathrm{Re}\left[\boldsymbol{v}_k(x)\right] = \cos(2\omega(x-k)) \tag{A.8}$$
$$\mathrm{Im}\left[\boldsymbol{v}_k(x)\right] = \sin(2\omega(x-k)) \tag{A.9}$$

The envelope function can now be rewritten as:

$$\begin{aligned}\psi_k(s) &= \cos^2(\omega(s-k)) = 0.5 + 0.5\cos(2\omega(s-k)) = \\ &= 0.5 + 0.5\mathrm{Re}\left[\mathbf{v}_k(s)\right]\end{aligned}$$

We can now rewrite the sum of a set of channel values:

$$\sum_{n=0}^{N-1}\psi_{k+n}(s) = \frac{N}{2} + \frac{1}{2}\mathrm{Re}\left[\sum_{n=0}^{N-1}\mathbf{v}_{k+n}(s)\right] \tag{A.10}$$

The complex sum in this expression can be rewritten as:

$$\sum_{n=0}^{N-1}\mathbf{v}_{k+n}(s) = \sum_{n=0}^{N-1}e^{\boldsymbol{i}2\omega(x-k-n)} = e^{\boldsymbol{i}2\omega(x-k)}\sum_{n=0}^{N-1}\left(e^{-\boldsymbol{i}2\omega}\right)^n \tag{A.11}$$

for $e^{-\boldsymbol{i}2\omega} \neq 1$[1] this geometric sum can be written as:

$$\sum_{n=0}^{N-1}\left(e^{-\boldsymbol{i}2\omega}\right)^n = \frac{1-e^{-\boldsymbol{i}2\omega N}}{1-e^{-\boldsymbol{i}2\omega}} \tag{A.12}$$

The numerator of this expression is zero exactly when $\omega = n\pi/N$, for integers n and N, $N \geq 2$. Due to the constraint in theorem A.2, we can see that n must be 1, and thus $\omega = \pi/N$. From this follows that the exponential sum in equation A.10 will become zero for these values as well.

We can now reformulate equation A.10 as:

$$\sum_{n=0}^{N-1}\psi_{k+n}(s) = \frac{N}{2} \quad \text{for } \omega = \pi/N \text{ where } N \in \mathbb{N}, \text{ and } N \geq 2. \tag{A.13}$$

Theorem A.4 *The sum of a squared channel value vector* $\begin{pmatrix} \psi_1^2(s) & \psi_2^2(s) & \dots \end{pmatrix}$ *does not depend on the value of* s *if* $\omega = \pi/N$*, where* N *is an integer* $N \geq 3$*.*

The proof of this theorem is similar to the proof of theorem A.3.

We first rewrite the squared envelope function as:

[1] In practice this does not happen, since this is equivalent to $\omega \neq n\pi$ where $n \in \mathbb{N}$, and such solutions gives a basis that is under-complete.

$$\psi_k^2(s) = \cos^4(\omega(s-k)) = \frac{3}{8} + \frac{1}{2}\cos(2\omega(s-k)) + \frac{1}{8}\cos(4\omega(s-k))$$

We then rewrite the sum of squares of our channel values:

$$\sum_{n=0}^{N-1} \psi_{k+n}^2(s) = \frac{3N}{8} + \frac{1}{2}\mathrm{Re}\left[\sum_{n=0}^{N-1} \mathbf{v}_{k+n}(s)\right] + \frac{1}{8}\mathrm{Re}\left[\sum_{n=0}^{N-1} \mathbf{v}_{k+n}^2(s)\right] \tag{A.14}$$

The first complex sum in this expression is zero for $\omega = \pi/N$, where N is an integer $N \geq 2$ (see equations A.11 and A.12).

The second sum can be written as:

$$\sum_{n=0}^{N-1} \mathbf{v}_{k+n}^2(s) = \sum_{n=0}^{N-1} e^{\boldsymbol{i}4\omega(x-k-n)} = e^{\boldsymbol{i}4\omega(x-k)} \sum_{n=0}^{N-1} \left(e^{-\boldsymbol{i}4\omega}\right)^n \tag{A.15}$$

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:

$$\sum_{n=0}^{N-1} \left(e^{-\boldsymbol{i}4\omega}\right)^n = \frac{1 - e^{-\boldsymbol{i}4\omega N}}{1 - e^{-\boldsymbol{i}4\omega}} \tag{A.16}$$

The numerator of this expression is zero exactly when $\omega = \frac{n\pi}{2N}$, for integers n, and N, but again we can see that $n = 2$ must hold, according to theorem A.2. Thus we again have $\omega = \frac{\pi}{N}$, and the constraints on equation A.16 further restricts this to $N \geq 3$.

We can now reformulate equation A.14 as:

$$\sum_{n=0}^{N-1} \psi_{k+n}^2(s) = \frac{3N}{8} \quad \text{for } \omega = \pi/N \text{ where } N \in \mathbb{N}, \text{ and } N \geq 3. \tag{A.17}$$

[2]In effect this excludes the solutions $N = 1$, and $N = 2$.

B Sub-pixel peak location from a feature image

If the signal within a feature image varies slowly, the location of a peak in the image is easily found by looking for the point with the highest match.

We can refine the peak location by looking at the ratios of the peak value, and its eight immediate neighbours (see figure B.1). In order to do so however, we must first assume a local signal model.

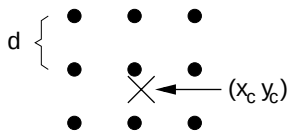


Figure B.1: Finding a peak with sub-pixel accuracy.

The model we will use is that the response, $\mathrm{m}(x, y)$ is proportional to the Euclidean distance to the peak i.e.

$$\mathrm{m}^2(x, y) = M + \alpha(x - x_c)^2 + \alpha(y - y_c)^2 \tag{B.1}$$

where M is an unknown offset, α is the proportionality constant, and $(x_c,\ y_c)$ is the sought peak location.

This gives us the following expression:

$$\mathrm{m}^2(x, y) = \begin{pmatrix} x^2 + y^2 & -2x & -2y & 1 \end{pmatrix} \begin{pmatrix} \alpha & \alpha x_c & \alpha y_c & M + \alpha(x_c^2 + y_c^2) \end{pmatrix}^t \tag{B.2}$$

Assuming a pixel distance of d as in figure B.1, and a coordinate origin at the centre pixel location, we get the following equation system:

$$\underbrace{\begin{pmatrix} \mathrm{m}^2(-d,-d) \\ \mathrm{m}^2(-d,0) \\ \mathrm{m}^2(-d,d) \\ \mathrm{m}^2(0,-d) \\ \mathrm{m}^2(0,0) \\ \mathrm{m}^2(0,d) \\ \mathrm{m}^2(d,-d) \\ \mathrm{m}^2(d,0) \\ \mathrm{m}^2(d,d) \end{pmatrix}}_{\mathbf{m}} = \underbrace{\begin{pmatrix} 2d^2 & 2d & 2d & 1 \\ d^2 & 2d & 0 & 1 \\ 2d^2 & 2d & -2d & 1 \\ d^2 & 0 & 2d & 1 \\ 0 & 0 & 0 & 1 \\ d^2 & 0 & -2d & 1 \\ 2d^2 & -2d & 2d & 1 \\ d^2 & -2d & 0 & 1 \\ 2d^2 & -2d & -2d & 1 \end{pmatrix}}_{\mathbf{W}} \underbrace{\begin{pmatrix} \alpha \\ \alpha x_c \\ \alpha y_c \\ M + \alpha(x_c^2 + y_c^2) \end{pmatrix}}_{\mathbf{p}} \tag{B.3}$$

With the solution:

$$\mathbf{p} = \mathbf{W}^* \mathbf{m} \tag{B.4}$$

Where $\mathbf{W}^*$ is the *pseudoinverse* of $\mathbf{W}$. The advantage with this approach is that $\mathbf{W}^*$ stays constant as long as d does. For $d = 1$, we get

$$\mathbf{W}^* = \frac{1}{36} \begin{pmatrix} 6 & -3 & 6 & -3 & -12 & -3 & 6 & -3 & 6 \\ 3 & 3 & 3 & 0 & 0 & 0 & -3 & -3 & -3 \\ 3 & 0 & -3 & 3 & 0 & -3 & 3 & 0 & -3 \\ -4 & 8 & -4 & 8 & 20 & 8 & -4 & 8 & -4 \end{pmatrix}$$

Finally, we can compute the sub-pixel offset $(x_c,\ y_c)$ as:

$$\begin{pmatrix} x_c \\ y_c \end{pmatrix} = \frac{1}{p_1} \begin{pmatrix} p_2 \\ p_3 \end{pmatrix} \tag{B.5}$$

Note that we actually only need the first three rows of $\mathbf{W}^*$.

An other interesting observation is that p_2 and p_3 can be identified as first order moments in the x and y directions respectively.

Bibliography

[1] M. Adlers. *Topics in Sparse Least Squares Problems*. PhD thesis, Linköping University, Linköping, Sweden, Dept. of Mathematics, 2000. Dissertation No. 634.

[2] J. A. Anderson. *The Handbook of Brain Theory and Neural Networks*, chapter Associative Networks. MIT Press, 1995. M. A. Arbib, Ed.

[3] T. Andersson. Learning in a Reactive Robotic Architecture. Lic. Thesis LiU-Tek-Lic-2000:13, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, March 2000. Thesis No. 817, ISBN 91-7219-694-7.

[4] M. F. Bear, B. W. Connors, and M. A. Paradiso. *Neuroscience: Exploring the Brain*. Williams & Wilkins, 1996. ISBN 0-683-00488-3.

[5] A. J. Bell and T. J. Sejnowski. Edges are the 'independent components' of natural scenes. *Advances in Neural Information Processing Systems*, 9, 1996.

[6] A. Blake. *The Handbook of Brain Theory and Neural Networks*, chapter Active Vision, pages 61–63. MIT Press, 1995. M. A. Arbib, Ed.

[7] 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.

[8] 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.

[9] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):255–274, November 1986.

[10] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Trans. on Information Theory*, 36(5):961–1005, September 1990.

[11] 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, Amsterdam, 2000.

[12] G. Farnebäck. Spatial Domain Methods for Orientation and Velocity Estimation. Lic. Thesis LiU-Tek-Lic-1999:13, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, March 1999. Thesis No. 755, ISBN 91-7219-441-3.

[13] D. J. Field. What is the goal of sensory coding? *Neural Computation*, 1994.

[14] F. Földiák. Forming sparse representations by local anti-hebbian learning. *Biological Cybernetics*, 1990.

[15] 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.

[16] 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.

[17] P.-E. Forssén and B. Johansson. Fractal coding by means of local feature histograms. Report LiTH-ISY-R-2295, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, September 2000.

[18] A. Giachetti. Matching techniques to compute image motion. *IVC*, 18(3):247–260, February 2000.

[19] G. H. Granlund. Personal communication.

[20] 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.

[21] G. H. Granlund. The complexity of vision. *Signal Processing*, 74(1):101–126, April 1999. Invited paper.

[22] 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.

[23] G. H. Granlund and H. Knutsson. *Signal Processing for Computer Vision*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9530-1.

[24] G. Granlund. Does Vision Inevitably Have to be Active? In *Proceedings of SCIA99, the 11th Scandinavian Conference on Image Analysis*, Kangerlussuaq, Greenland, June 7–11 1999. Also as Technical Report LiTH-ISY-R-2247.

[25] 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.

[26] R. M. Gray. Dithered quantizers. *IEEE Transactions on Information Theory*, 39(3):805–812, 1993.

[27] 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.

[28] C. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, September 1988.

[29] S. Haykin. *Neural Networks–A comprehensive foundation*. Prentice Hall, 2nd edition, 1999. ISBN 0-13-273350-1.

[30] D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.

[31] 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.

[32] I. P. Howard and B. J. Rogers. *Binocular Vision and Stereopsis*. Oxford Psychology Series, 29. Oxford University Press, New York, 1995.

[33] M. Irani, B. Rousso, and S. Peleg. Recovery of ego-motion using region alignment. *IEEE Trans. on PAMI*, pages 268–272, March 1997.

[34] 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.

[35] H. Knutsson, M. Andersson, and J. Wiklund. Advanced Filter Design. In *Proceedings of the11th Scandinavian Conference on Image Analysis*, Greenland, June 1999. SCIA. Also as report LiTH-ISY-R-2142.

[36] H. Knutsson and C.-F. Westin. Normalized and Differential Convolution: Methods for Interpolation and Filtering of Incomplete and Uncertain Data. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 515–523, New York City, USA, June 1993. IEEE.

[37] P. Kovesi. Image features from phase congruency. Tech. Report 95/4, University of Western Australia, Dept. of CS, 1995.

[38] 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.

[39] T. Lindeberg. *Scale-space Theory in Computer Vision*. Kluwer Academic Publishers, 1994. ISBN 0792394186.

[40] 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.

[41] M. L. Minsky and S. Papert. *Perceptrons*. M.I.T. Press, Cambridge, Mass., 1969.

[42] 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.

[43] A. Moe. Passive Aircraft Altitude Estimation using Computer Vision. Lic. Thesis LiU-Tek-Lic-2000:43, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden, September 2000. Thesis No. 847, ISBN 91-7219-827-3.

[44] M. C. Morrone, J. R. Ross, and R. A. Owens. Mach bands are phase dependent. *Nature*, 324:250–253, 1986.

[45] S. K. Nayar, S. A. Nene, and H. Murase. Real-Time 100 Object Recognition System. In *Proc. of ARPA Image Understanding Workshop*, 1996.

[46] K. N. Ngan, T. Meier, and D. Chai. *Advanced Video Coding: Principles and Techniques*. Elsevier Science B.V., 1999.

[47] 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.

[48] B. A. Olshausen and D. J. Field. Emergence of simple cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

[49] 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.

[50] M. J. L. Orr. Introduction to radial basis function networks. Technical report, Centre for Cognitive Science, University of Edinburgh, Edinburgh EH8 9LW, Scotland, 1996.

[51] R. Palm, H. Hellendoorn, and D. Driankov. *Model Based Fuzzy Control*. Springer-Verlag, Berlin, 1996. ISBN 3-540-61471-0.

[52] E. Parzen. On estimation of probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[53] P. Perona and J. Malik. Detecting and localizing edges composed of steps, peaks and roofs. In *Proceedings of ICCV*, pages 52–57, 1990.

[54] D. Reisfeld. The constrained phase congruency feature detector: simultaneous localization, classification, and scale determination. *Pattern Recognition letters*, 17(11):1161–1169, 1996.

[55] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27:642–669, 1956.

[56] I. Sobel. Camera models and machine perception. Technical Report AIM-21, Stanford Artificial Intelligence Laboratory, Palo Alto, California, 1970.

[57] R. Söderberg. View Dependent Recognition of Objects. Master's thesis, Linköping University. In print.

[58] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. International Thomson Publishing Inc., 1999. ISBN 0-534-95393-X.

[59] 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.

[60] H. Wässle and B. B. Boycott. Functional architecture of the mammalian retina. *Physiological Reviews*, pages 447–480, 1991.

[61] C.-F. Westin. *A Tensor Framework for Multidimensional Signal Processing*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, 1994. Dissertation No 348, ISBN 91-7871-421-4.

[62] WITAS web page: http://www.ida.liu.se/ext/witas/.

[63] A. Witkin. Scale-space filtering. In *8th Int. Joint Conf. Artificial Intelligence*, pages 1019–1022, Karlsruhe, 1983.

[64] M.-H. Yang, D. Roth, and N. Ahuja. Learning to Recognize 3D Objects with SNoW. In *Proceedings of the 6th European Conference on Computer Vision*, volume I, pages 439–454, Dublin, Ireland, June 2000.