

This chapter introduces basic concepts for mapping an image into an image, typically used for improving image quality or for purposes defined by a more complex context of a computer vision process.

2.1 Point, Local, and Global Operators

When recording image data outdoors (a common case for computer vision), there are often particular challenges compared to indoor recording, such as difficulties with lighting, motion blur, or sudden changes in scenes. Figure 2.1 shows images recorded in a car (for vision-based driver-assistance). An unwanted data is called *noise*. These are three examples of noise in this sense of “unwanted data”. In the first case we may aim at transforming the images such that the resulting images are “as taken at uniform illumination”. In the second case we could try to do some sharpening for removing the blur. In the third case we may aim at removing the noise.

This section provides time-efficient methods that you may consider for preprocessing your data prior to subsequent image analysis.

2.1.1 Gradation Functions

We transform an image I into an image I_{new} of the same size by mapping a grey level u at pixel location p in I by a *gradation function* g onto a grey level $v = g(u)$ at the same pixel location p in I_{new} . Because the change only depends on value u at location p , we also speak about a *point operator* defined by a gradation function g .

If the goal is that I_{new} satisfies some properties defined in terms of its histogram, then we speak about a *histogram transform*.

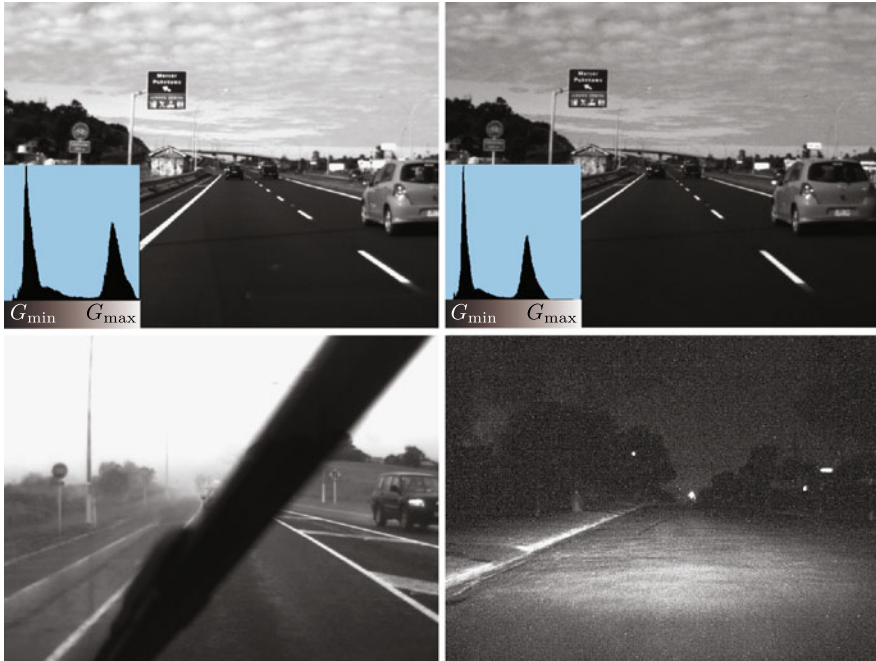


Fig. 2.1 *Top*: A pair of images SouthLeft and SouthRight taken time-synchronized but of different brightness; see the shown grey-level histograms. *Bottom left*: Blurring caused by rain in image Wiper. *Bottom right*: Noise in a scene Uphill recorded at night

Histogram Equalization We transform a scalar image I such that all grey levels appear equally often in the transformed image I_{new} . The goal is to achieve that

$$H_{I_{new}}(u) = \text{const} = \frac{N_{cols}N_{rows}}{G_{\max} + 1} \quad (2.1)$$

for all $u \in \{0, 1, \dots, G_{\max}\}$.

Unfortunately, this is not possible in general, due to the constraint that identical values in I can only map on the same value in I_{new} . For example, a binary image I cannot be mapped onto a histogram-equalized grey-level image I_{new} (even in the case if we would have a continuous binary image; but having digital images also contributes to excluding perfect equalization). The following transform is thus just an approximate solution towards the ideal goal.

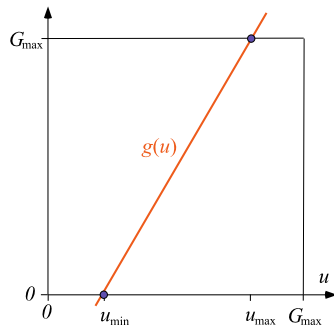
Given is an $N_{cols} \times N_{rows}$ scalar image I with absolute frequencies $H_I(u)$ for $0 \leq u \leq G_{\max}$. We transform I into an image I_{new} of the same size by mapping intensities u in I by the following gradation function g onto new intensities $v = g(u)$ in I_{new} :

$$g(u) = c_I(u) \cdot G_{\max} \quad (2.2)$$



Fig. 2.2 *Left:* Input image RagingBull (in the public domain) with histogram. *Right:* The same image after histogram equalization

Fig. 2.3 Graph of the gradation function for linear scaling, defined by being incident with points $(u_{\min}, 0)$ and (u_{\max}, G_{\max})



where c_I is the relative cumulative frequency function. Figure 2.2 illustrates such a *histogram equalization*.

It is not difficult to show the equalization property for the histogram transform, defined by (2.2), using the property that the cumulative relative frequency c_I is an increasing function. The relative histogram $h_I(u)$ corresponds to an estimate of a density function, $c_I(u)$ to an estimate of a probability distribution function, and $h_{I_{\text{new}}}(u)$ to an estimate of the uniform density function.

Linear Scaling Assume that an image I has positive histogram values in a limited interval only. The goal is that all values used in I are spread linearly onto the whole scale from 0 to G_{\max} . Let $u_{\min} = \min\{I(x, y) : (x, y) \in \Omega\}$, $u_{\max} = \max\{I(x, y) : (x, y) \in \Omega\}$, and

$$a = -u_{\min} \quad \text{and} \quad b = \frac{G_{\max}}{u_{\max} - u_{\min}} \quad (2.3)$$

$$g(u) = b(u + a) \quad (2.4)$$

As a result, pixels having the value u_{\min} in the image I now have the value 0 in the resulting image I_{new} , and pixels having the value u_{\max} in the image I now have the value G_{\max} in I_{new} . This is illustrated in Fig. 2.3. This figure can also serve as an

illustration when discussing the correctness of the histogram transform defined by (2.4).

Conditional Scaling As another example of a use of a gradation function, we want to map an image J into an image J_{new} , such that it has the same mean and the same variance as an already given image I . For this *conditional scaling*, let

$$a = \mu_J \cdot \frac{\sigma_I}{\sigma_J} - \mu_I \quad \text{and} \quad b = \frac{\sigma_J}{\sigma_I} \quad (2.5)$$

$$g(u) = b(u + a) \quad (2.6)$$

Now we map the grey level u at pixel p in J onto the new value $v = g(u)$ at the same pixel p in J_{new} . It is not difficult to show that $\mu_{J_{new}} = \mu_I$ and $\sigma_{J_{new}} = \sigma_I$. The performed normalization is the same as in (1.12), where we normalized data measures.

2.1.2 Local Operators

For a given $N_{cols} \times N_{rows}$ image I , we consider sliding windows W_p , each of size $(2k + 1) \times (2k + 1)$, where the reference point p is always at the centre of the window. The reference point moves into all possible pixel locations of I and so moves the window over the image. At these locations we perform a *local operation*; the result of the operation defines the new value at p . Thus, the input image I is transformed into a new image J .

Two Examples: Local Mean and Maximum For example, the local operation can be the *local mean*, $J(p) = \mu_{W_p(I)}$, with

$$\mu_{W_p(I)} = \frac{1}{(2k + 1)^2} \cdot \sum_{i=-k}^{+k} \sum_{j=-k}^{+k} I(x + i, y + j) \quad (2.7)$$

for $p = (x, y)$.

As another example, the local operation can be the calculation of the *local maximum*

$$J(p) = \max\{I(x + i, y + j) : -k \leq i \leq k \wedge -k \leq j \leq k\} \quad (2.8)$$

for $p = (x, y)$. See Fig. 2.4 for an example.

Windows centred at p and not completely contained in I , require a special “border-pixel strategy”; there is no general proposal for such a strategy. One option is to consider the same local operation just for a smaller window, which is possible for the two examples of local operations given above.



Fig. 2.4 *Top, left:* Original image Set1Seq1 with $N_{cols} = 640$. *Top, right:* Local maximum for $k = 3$. *Bottom, left:* Local minimum for $k = 5$. *Bottom, right:* Local operator using the 3×3 filter kernel shown in the middle of Fig. 2.5

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

 $/S$

1	0	-1
2	0	-2
1	0	-1

 $/9$

1	1	1
1	1	1
1	1	1

 $/9$

Fig. 2.5 *Left:* General representation for a 3×3 filter kernel. *Middle:* Filter kernel illustrated in Fig. 2.4, bottom, right, approximating a derivative in x -direction. *Right:* The filter kernel of a 3×3 box filter

Linear Operators and Convolution A *linear local operator* is defined by a *convolution* of an image I at $p = (x, y)$ with a *filter kernel* W ,

$$J(p) = I * W(p) = \frac{1}{S} \sum_{i=-k}^{+k} \sum_{j=-k}^{+k} w_{i,j} \cdot I(x+i, y+j) \quad (2.9)$$

with weights $w_{i,j} \in \mathbb{R}$ and a *scaling factor* $S > 0$. The arguments in (2.9) go out of Ω if p is close to the border of this image carrier. A theorem says that then you apply a modulo rule conceptually equivalent to a 2D periodic copying of the image I on Ω into the grid \mathbb{Z}^2 .

The array of $(2k+1) \times (2k+1)$ weights and scaling factor S define the filter kernel W . It is common to visualize filter kernels W of linear local operators as shown in Fig. 2.5.

Equation (2.7) is an example of such a linear local operator, known as a *box filter*. Here we have all weights equal to 1, and $S = (2k+1)^2$ is the sum of all those weights.

General View on Local Operators We summarize the properties of *local operators*:

1. Operations are limited to windows, typically of square and odd size $(2k + 1) \times (2k + 1)$; of course, with respect to *isotropy* (i.e. rotation invariance), approximately circular windows should be preferred instead, but rectangular windows are easier to use.
2. The window moves through the given image following a selected scan order (typically aiming at a complete scan, having any pixel at the reference position of the window at some stage).
3. There is no general rule how to deal with pixels close to the border of the image (where the window is not completely in the image anymore), but they should be processed as well.
4. The operation in the window should be the same at all locations, identifying the purpose of the local operator.
5. The results can either be used to replace values in place at the reference points in the input image I , defining a *sequential local operator* where new values propagate like a “wave” over the original values (windows of the local operator then contain the original data and already-processed pixel values), or resulting values are written into a second array, leaving the original image unaltered this way, defining a *parallel local operator*, so called due to the potential of implementing this kind of a local operator on specialized parallel hardware.

In case of $k = 0$ (i.e., the window is just a single pixel), we speak about a *point operator*. If k grows so that the whole picture is covered by the window, then it turns into a *global operator*. The 2D Fourier transform of an image is an example for a global transform.

Insert 2.1 (Zamperoni) *There is immense diversity of published proposals for image processing operators due to the diversity of image data and particular tasks in applications. For example, the book [R. Klette and P. Zamperoni. Handbook of Image Processing Operators. Wiley, Chichester, 1996] details many of the usual point, local, and global operators.*

The memory of Piero Zamperoni (1939–1998), an outstanding educator in pattern recognition, is honoured by the IAPR by issuing the Piero Zamperoni Best Student Paper Award at their biennial ICPR conferences.

2.1.3 Fourier Filtering

The inverse 2D DFT (see (1.23)) transforms a Fourier transform \mathbf{I} back from the frequency domain into the spatial domain. The inverse 2D DFT will lead to a real-valued function I as long as \mathbf{I} satisfies the symmetry property of (1.29). Thus, any change in the frequency domain is constrained by this.

Fourier Filtering The inverse 2D DFT can be read as follows: the complex numbers $\mathbf{I}(u, v)$ are the Fourier coefficients of I , defined for different frequencies u and v . Each Fourier coefficient is multiplied with a combination of sine and cosine functions (see the Eulerian formula (1.22)), and the sum of all those combinations forms the image I . In short, the image I is represented by basis functions being powers of roots of unity in the complex plane, and the Fourier coefficients specify this basis function representation.

This means that if we modify one of the Fourier coefficients (and its symmetric coefficient due to the constraint imposed by the symmetry property) before applying the inverse 2D DFT, then we obtain a modified function I .

For a linear transform of the image I , there are two options:

1. We modify the image data by a linear convolution

$$J(x, y) = (I * G)(x, y) = \sum_{i=0}^{N_{cols}-1} \sum_{j=0}^{N_{rows}-1} I(i, j) \cdot G(x - i, y - j) \quad (2.10)$$

in the spatial domain, where G is the filter kernel (also called the *convolution function*). Function J is the *filtered image*.

2. We modify the 2D DFT \mathbf{I} of I by multiplying the values in \mathbf{I} , position by position, with the corresponding complex numbers in \mathbf{G} [i.e., $\mathbf{I}(u, v) \cdot \mathbf{G}(u, v)$]. We denote this operation by $\mathbf{I} \circ \mathbf{G}$ (not to be confused with matrix multiplication). The resulting complex array is transformed by the inverse 2D DFT into the modified image J .

Interestingly, both options lead to identical results assuming that \mathbf{G} is the 2D DFT of G , due to the *convolution theorem*:

$$I * G \text{ equals the inverse 2D DFT of } \mathbf{I} \circ \mathbf{G} \quad (2.11)$$

Thus, either a convolution in the spatial domain or a position-by-position multiplication in the frequency domain produce identical filtered images. However, in the convolution case we miss the opportunity to design frequency-dependent filter functions in the frequency domain.

Steps of Fourier Filtering Given is an image I and a complex-valued filter function \mathbf{G} (which is satisfying the symmetry property of (1.29)) in the frequency domain. Apply an FFT program for doing the following; if required for the applied FFT program, first map the image I into a larger $2^n \times 2^n$ array:

1. Transform the image I into the frequency domain, into the complex-valued \mathbf{I} by using the FFT program.
2. Multiply the complex-valued \mathbf{I} , element by element, with the complex-valued filter function \mathbf{G} .
3. Transform the result back into the spatial domain by using the FFT program for the inverse DFT.

The filter function \mathbf{G} can be obtained as the Fourier transform of a filter kernel G in the spatial domain. It is common procedure to design filter functions \mathbf{G} directly in the frequency domain.

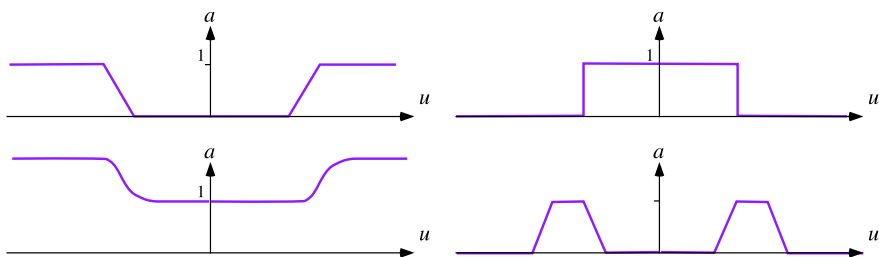


Fig. 2.6 1D profiles of rotation-symmetric filter functions. *Top:* A linear high-pass filter and an ideal low-pass filter. *Bottom:* An exponential high-emphasis filter and a linear band-pass filter

Example 2.1 The *box filter* is a linear convolution in the spatial domain. Its filter kernel is defined by the weights $G(x, y) = 1/a$ for (x, y) in a $(2k + 1) \times (2k + 1)$ window, centred at the origin $(0, 0)$, with $a = (2k + 1)^2$. Outside of this window we have that $G(x, y) = 0$.

The 2D DFT of this function G has amplitudes close to 1 for low frequencies, with a steep decrease in amplitudes towards zero for higher frequencies.

The Fourier transform \mathbf{G} of Example 2.1 is a typical *low-pass filter*: low frequencies are “allowed to pass” (because multiplied with values of amplitudes close to 1), but higher frequencies are “drastically reduced” (because multiplied with values of amplitude close to 0).

Design of Filter Functions The frequency domain is well suited for the design of filter functions. See Fig. 2.6. We may decide for a *high-pass filter* (e.g., for edge detection, or for visualizing details and for suppressing low frequencies), a *high-emphasis filter* (e.g., for enhancing contrast), a *band-pass filter* (for allowing only a certain range of frequencies “to pass”), or a filter that eliminates or enhances selected frequencies (under proper consideration of the symmetry constraint). The impact of a low-pass filter is a reduction of outliers and of contrast, i.e. a smoothing effect.

Attributes “linear”, “exponential”, or “ideal” of a filter function specify the way how the transition is defined from large amplitudes of the filter to low amplitudes. See Fig. 2.6 for examples of transitions. Low-pass and band-pass filtering of an image is illustrated in Fig. 2.7.

Besides the flexibility in designing filter functions, the availability of time-efficient 2D FFT algorithms is also an important argument for using a DFT-based filter instead of a global convolution. Local convolutions are normally more efficiently performed in the spatial domain by a local operator.

2.2 Three Procedural Components

This section introduces three procedural components that are commonly used in image processing programs, such as for local operators, but also when implementing particular image analysis or computer vision procedures.

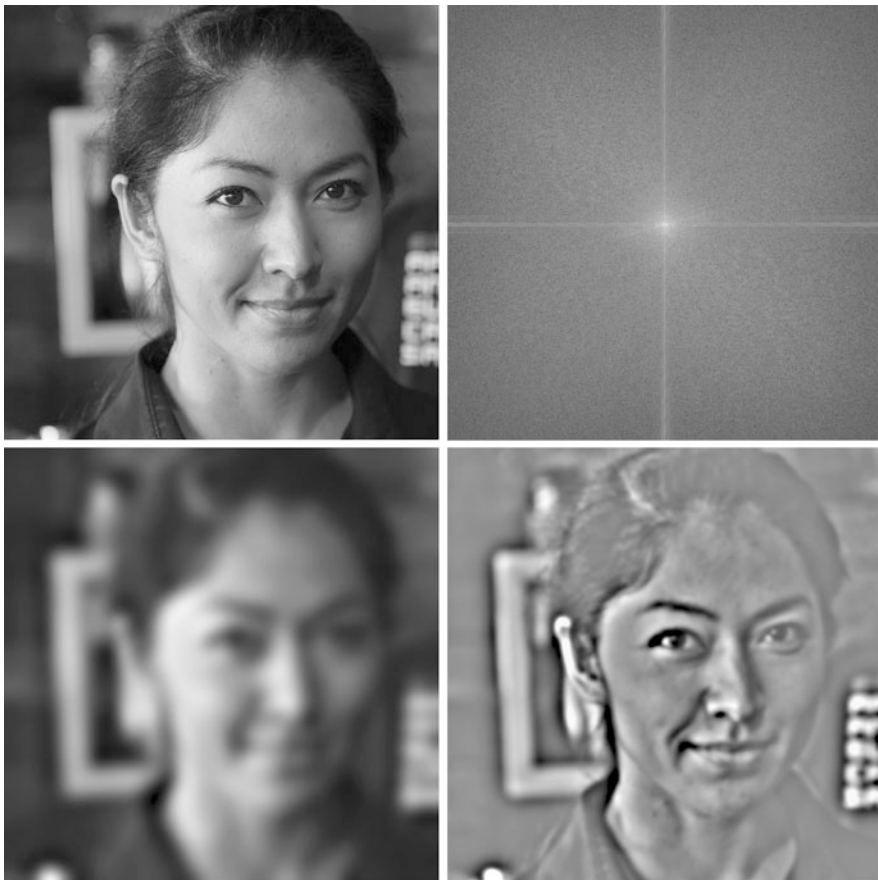


Fig. 2.7 *Upper row, left:* Intensity channel of image Emma, shown also in colour in Fig. 2.9. *Upper row, right:* Its spectrum, centred and with log-transform. *Lower row, left:* An ideal low-pass filtered Emma, showing a typical smoothing effect. *Lower row, right:* An exponential-band-pass filtered Emma, already showing more higher frequencies than lower frequencies

2.2.1 Integral Images

The calculation of an *integral image* I_{int} for a given image I is a common preprocessing step for speeding up operations on I which involve rectangular windows (e.g. for feature detection). “Integration” means adding small units together. In this case, the small units are the pixel values. For a pixel $p = (x, y)$, the *integral value*

$$I_{int}(p) = \sum_{1 \leq i \leq x \wedge 1 \leq j \leq y} I(i, j) \quad (2.12)$$

is the sum of all the values $I(i, j)$ at pixel locations $q = (i, j)$ that are neither below p nor right of p . See Fig. 2.8, left.

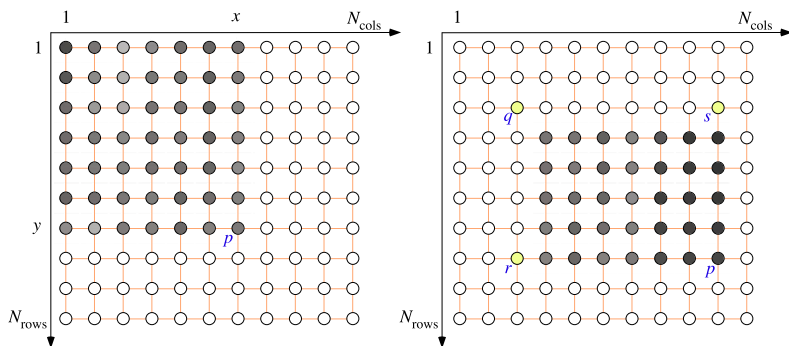


Fig. 2.8 *Left: At $I_{int}(x, y)$ we have the sum of all the shown pixel values. Right: If an algorithm requires to use the sum of all pixel values in the shown rectangular window, then we only need to combine the values of the integral image in the four corners p , q , r , and s ; see the text for the formula*

Insert 2.2 (The Introduction of Integral Images into Computer Vision) *Integral images have been introduced in the Computer Graphics literature in [F.C. Crow. Summed-area tables for texture mapping. Computer Graphics, vol. 18, pp. 207–212, 1984] and then popularized by [J.P. Lewis. Fast template matching. In Proc. Vision Interface, pp. 120–123, 1995] and [P. Viola and M. Jones. Robust real-time object detection. Int. J. Computer Vision, pp. 137–154, 2001] in the Computer Vision literature.*

Now consider a rectangular window W in an image defining four pixels p , q , r , and s , as illustrated in Fig. 2.8, right, with q , r , and s just one pixel away from W . The sum S_W of all pixel values in W is now simply defined by

$$S_W = I_{int}(p) - I_{int}(r) - I_{int}(s) + I_{int}(q) \quad (2.13)$$

We only have to perform one addition and two subtractions, independent upon the size of the rectangular window W . This will later (in this book) prove to be very handy for classifying objects shown in images.

Example 2.2 (Number of Operations with or Without Integral Image) Assume that we calculate the sum in an $n \times m$ window by using (2.13). We have three arithmetic operations, no matter what are the values of m or n .

Without an integral image, just by adding all the $m \cdot n$ numbers in the window, we have $m \cdot n - 1$ arithmetic operations.

If we also count the addressing arithmetic operations, for the sequential sliding window in the integral image, they are reduced to four ++ increments if we keep the addresses for pixels p , q , r , and s in address registers.

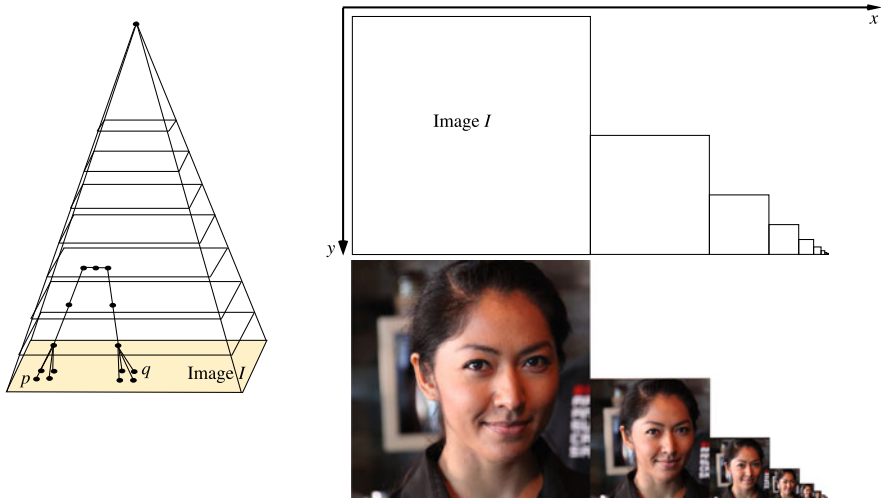


Fig. 2.9 Illustrations of picture pyramids. *Left*: A regular pyramid is the assumed model behind subsequent size reductions. *Left, top*: Sketch of pairwise disjoint arrays. *Left, bottom*: Example of layers for image Emma

Observation 2.1 After one preprocessing step for generating the integral image, any subsequent step, requiring to know the sum of pixel values in a rectangular window, only needs constant time, no matter what is the size of the window.

2.2.2 Regular Image Pyramids

A pyramid is a common data structure used for representing one input image I at different sizes. See Fig. 2.9. The original image is the base layer of the pyramid. Images of reduced sizes are considered to be subsequent layers in the pyramid.

Use of Scaling Factor 2 If scaling down by factor 2, as illustrated in Fig. 2.9, then all additional levels of the pyramid require less than one third of the space of the original image, according to the geometric series

$$1 + \frac{1}{2 \cdot 2} + \frac{1}{2^2 \cdot 2^2} + \frac{1}{2^3 \cdot 2^3} + \cdots < \frac{4}{3} \quad (2.14)$$

When reducing the size from one layer to the next layer of the pyramid, bottom-up, the mean was calculated for 2×2 pixels for generating the corresponding single pixel at the next layer. For avoiding spatial aliasing, it is also recommended to perform some *Gauss smoothing* (to be explained in the following section) prior to taking those means.

By creating a new pixel r in Layer $n + 1$ of the pyramid, defined by (say) four pixels p_1 , p_2 , p_3 , and p_4 at Layer n , we create new adjacencies (p_1, r) , (p_2, r) , (p_3, r) , and (p_4, r) , additionally to (say) 4-adjacency in Layer n , as illustrated in

Fig. 2.9, right. For going via adjacent pixels from pixel location p to pixel location q in image I , we now also have the option to go first up in the pyramid to some level, then a few steps sideward in this level, and again down to q . In general, this supports shorter connecting paths than only using 4-adjacency in the input image I .

Example 2.3 (Longest Path in a Regular Pyramid of Scaling Factor 2) Assume an image I of size $2^n \times 2^n$ and a regular pyramid on top of this image created by using scaling factor 2.

For the longest path between two pixel locations, we consider p and q being diagonal corners in I . Using 4-adjacency in I , their distance to each other is $2^n - 1$ steps towards one side, and again $2^n - 1$ steps towards another side, no matter in which order we do those steps. Thus, the longest path in I , not using the pyramid, equals

$$2^{n+1} - 2 \quad (2.15)$$

This reduces to a path of length $2n$ when also using the adjacencies defined by the pyramid.

Observation 2.2 *Adjacencies in a pyramid reduce distances between pixels in an image significantly; this can be used when there is a need to send a “message” from one pixel to others.*

Pyramids can also be used for starting a computer vision procedure at first at one selected level in the data structure, and results are then refined by propagating them down in the pyramid to layers of higher resolution. We will discuss examples at some places in the book.

2.2.3 Scan Orders

The basic control structure of an image analysis program (not only for local operators, also, e.g. for component labelling) typically specifies a scan order for visiting all or some of the pixels.

Standard Scan Order and Variants Figure 2.10 illustrates not only the standard scan order, but also others that might be of interest under particular circumstances. Spiral or meander scans offer the opportunity that prior calculations are used at the next location of the sliding window, because only $2k + 1$ pixels enter the window, replacing $2k + 1$ “leaving” pixels.

Insert 2.3 (Hilbert, Peano, and Euclid) *In 1891, D. Hilbert (1862–1943), the major German mathematician, defined a curve filling completely the unit square, following Jordan’s initial definition of a curve. A finite number of repetitions of this construction, as illustrated in Fig. 2.11, leads to a Hilbert scan*

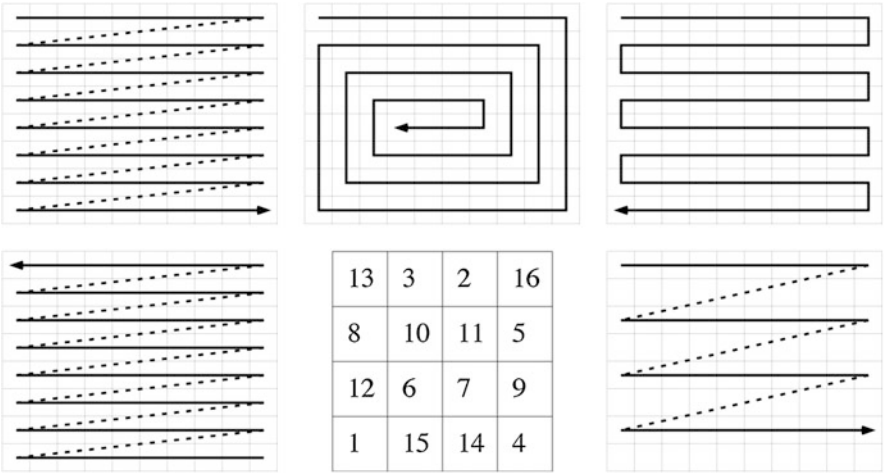


Fig. 2.10 Scan orders: standard (*upper left*), inward spiral (*upper middle*), meander (*upper right*), reverse standard (*lower left*), magic square (*lower middle*), and selective standard (as used in interlaced scanning), e.g. every second row (*lower right*)

in a grid of size $2^n \times 2^n$, not to be confused with the original curve defined by Hilbert in the Euclidean space. Hilbert's curve is a variant of a curve defined in 1890 by the Italian mathematician G. Peano (1858–1932) for the same purpose.

Euclid of Alexandria (about –300) was a Greek mathematician, known for his *Elements*, which was the standard work in Geometry until the 19th century.

A *magic square scan* (Fig. 2.10, bottom, middle, shows a simple 4×4 example) generates a pseudo-random access to pixels; in a *magic square*, numbers add up to the same sum in each row, in each column, and in forward and backward main diagonals. A *Hilbert scan* is another option to go towards pseudo-random access (or output, e.g. for generating a picture on a screen). See Fig. 2.11.

Hilbert Scan Fig. 2.11 specifies the Hilbert scan in a way that we enter the image at its north–west corner, and we leave it at its north–east corner. Let us denote the four corners of a $2^n \times 2^n$ picture by a, b, c, d , starting at the north–west corner and in clock-wise order. We assume a Hilbert scan $H_n(a, d, c, b)$, where we start at corner a , continue then with corner d , proceed to corner c , and terminate then at corner b .

$H_1(a, b, c, d)$ is a scan of a 2×2 image, where we just visit in the order a, b, c, d as shown.

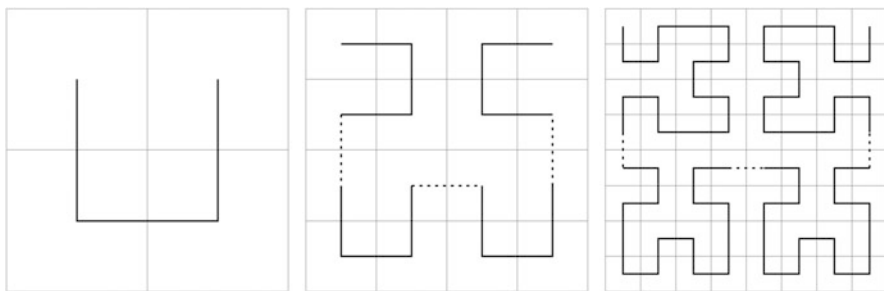


Fig. 2.11 Hilbert scans for 2×2 , 4×4 , or 8×8 images illustrating the recursive extension to larger images of size $2^n \times 2^n$

$H_{n+1}(a, d, c, b)$ is a scan where we start at the north-west corner; we perform $H_n(a, b, c, d)$, followed by one step down, then $H_n(a, d, c, b)$, followed by one step to the right, then (again) $H_n(a, d, c, b)$, followed by one step up, and finally $H_n(c, d, a, b)$, which takes us to the north-east corner of the $2^{n+1} \times 2^{n+1}$ image.

2.3 Classes of Local Operators

Local intensity patterns in one image can be considered to be “fairly” independent if they are at some distance to each other within the carrier Ω . Local operators make good use of this and are time-efficient and easy to implement on usual sequential and parallel hardware. Thus, not surprisingly, there is a large diversity of proposed local operators for different purposes. This section illustrates this diversity by only providing a few “popular” examples for four classes of local operators.

2.3.1 Smoothing

Image *smoothing* aims at eliminating “outliers” in image values considered to be noise in a given context.

Box Filter The $(2k + 1) \times (2k + 1)$ box filter, performing the local mean calculation as already defined in (2.7), is a simple option for image smoothing. It removes outliers, but it also reduces significantly the contrast $C(I)$ of an image I . Often it is sufficient to use just a 3×3 or 5×5 filter kernel. The local mean for larger kernel sizes can be conveniently calculated by using the integral image I_{int} of input image I .

Median Operator The *median* of $2n + 1$ values is the value that would appear in sorted order at position $n + 1$. For example, 4 is the median of the set $\{4, 7, 3, 1, 8, 7, 4, 5, 2, 3, 8\}$ because 4 is in position 6 in the sorted sequence 1, 2, 3, 3, 4, 4, 5, 7, 7, 8, 8.

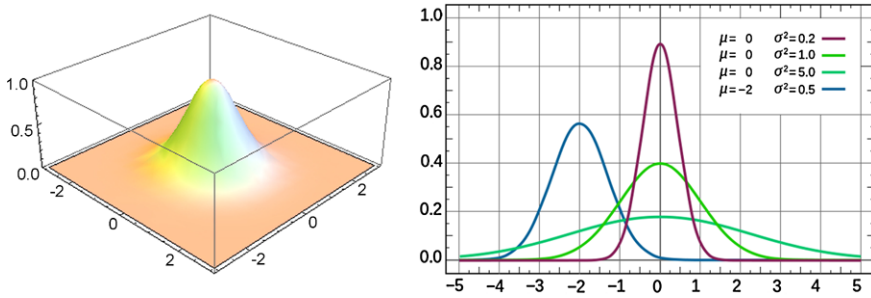


Fig. 2.12 *Left:* The 2D Gauss function for expected values $\mu_x = \mu_y = 0$. *Right:* Four examples of 1D Gauss functions for different expected values and different variances

The $(2k+1) \times (2k+1)$ median operator maps the median of a $(2k+1) \times (2k+1)$ window to the reference pixel p . It achieves the removal of outliers with only an insignificant change in image contrast $C(I)$.

Insert 2.4 *C.F. Gauss (1777–1855), a brilliant German mathematician working at Göttingen university, very well described in a novel “Measuring the World” by D. Kehlmann (original publication in German in 2005).*

Gauss Filter The Gauss filter is a local convolution with a filter kernel defined by samples of the 2D *Gauss function*. This function is the product of two 1D Gauss functions defined as follows:

$$\begin{aligned}
 G_{\sigma, \mu_x, \mu_y}(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x - \mu_x)^2 + (y - \mu_y)^2}{2\sigma^2}\right) \\
 &= \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{(x - \mu_x)^2}{2\sigma^2}} \cdot e^{-\frac{(y - \mu_y)^2}{2\sigma^2}}
 \end{aligned} \tag{2.16}$$

where (μ_x, μ_y) combines the expected values for x - and y -components, σ is the standard deviation (σ^2 is the variance), which is also called the *radius* of this function, and e is the Euler number.

The Gauss function is named after C.F. Gauss (see Insert 2.4). The Euler number is named after L. Euler; see Insert 1.3 and 1.22 for the *Eulerian formula*. Figure 2.12 illustrates the Gauss function. The standard deviation σ is also called the *scale*.

Observation 2.3 *The second line in (2.16) shows that a 2D Gauss filter can be realized by two subsequent 1D Gauss filters, one in horizontal and one in vertical direction.*

Fig. 2.13 Filter kernel for Gaussian smoothing defined by $k = 2$ and $s = 2$ (i.e. $\sigma = 1$)

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

/273

Centred Gauss Function By assuming a centred Gauss function (i.e. with zero means $\mu_x = \mu_y = 0$, as in Fig. 2.12, left), (2.16) simplifies to

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) = \frac{1}{\pi s} \cdot e^{-\frac{x^2}{2\sigma^2}} \cdot e^{-\frac{y^2}{2\sigma^2}} \quad (2.17)$$

Such a centred Gauss function is now sampled at $(2k + 1) \times (2k + 1)$ locations, with the window's reference pixel at the origin $(0, 0)$. This defines an important filter kernel for a local operator, parameterized by $\sigma > 0$ and $k \geq 1$. We will later use it for defining *differences of Gaussians* (DoGs) and the *scale space*.

Figure 2.13 shows a sampled filter kernel for $\sigma = 1$. Following the *three-sigma rule* in statistics, G_σ is sampled by a kernel of size $6\sigma - 1$.

For an input image I , let

$$L(x, y, \sigma) = [I * G_\sigma](x, y) \quad (2.18)$$

be a local convolution with function G_σ with $\sigma > 0$. For implementation, we sample G_σ symmetrically to the origin at $w \times w$ grid positions for defining the filter kernel, where w is the nearest odd integer to $6\sigma - 1$.

Gaussian Scale Space For a scaling factor $a > 1$, we can step from the smoothed image $L(x, y, \sigma)$ to $L(x, y, a\sigma)$. By using repeatedly scales $a^n \cdot \sigma$, for an initial scale σ and $n = 0, 1, \dots, m$, we create a set of subsequent *layers* of a *Gaussian scale space*. See Fig. 2.14 for an example.

In this book, the layers in a scale space are all of identical size $N_{cols} \times N_{rows}$. For implementation efficiency, some authors suggested to reduce this size by a factor of 2 for any doubling of the used scale σ , thus creating *octaves* of blurred images. The blurred images in one octave remain at constant size until the next doubling of σ occurs, and the size is then again reduced by factor of 2. This is an implementation detail, and we do not use octaves in the discussion of scale spaces in this book.

Sigma Filter This filter is just an example of a simple but often useful local operator for noise removal. For an example of a result, see Fig. 2.15. Again, we discuss this local operator for $(2k + 1) \times (2k + 1)$ windows $W_p(I)$ with $k \geq 1$. We use a parameter $\sigma > 0$, considered to be an approximation of the image acquisition noise of image I (for example, σ equals about 50 if $G_{\max} = 255$). Suggesting a parallel local operator, resulting values are forming a new picture J as follows:



Fig. 2.14 Smoothed versions of the image Set1Seq1 (shown in Fig. 2.4, upper left) for $\sigma = 0.5$, $\sigma = 1$, $\sigma = 2$, $\sigma = 4$, $\sigma = 8$, and $\sigma = 16$, defining six layers of the Gaussian scale space

1. Calculate the histogram of window $W_p(I)$.
2. Calculate the mean μ of all values in the interval $[I(p) - \sigma, I(p) + \sigma]$.
3. Let $J(p) = \mu$.

In some cases, camera producers specify parameters for the expected noise of their CCD or CMOS sensor elements. The parameter σ could then be taken as 1.5 times the noise amplitude. Note that

$$\mu = \frac{1}{S} \cdot \sum_{u=I(p)-\sigma}^{I(p)+\sigma} u \cdot H(u) \quad (2.19)$$

where $H(u)$ denotes the histogram value of u for window $W_p(I)$ and scaling factor $S = H(I(p) - \sigma) + \dots + H(I(p) + \sigma)$.

Figure 2.15 illustrates the effects of a box filter, the median filter, and the sigma filter on a small image.

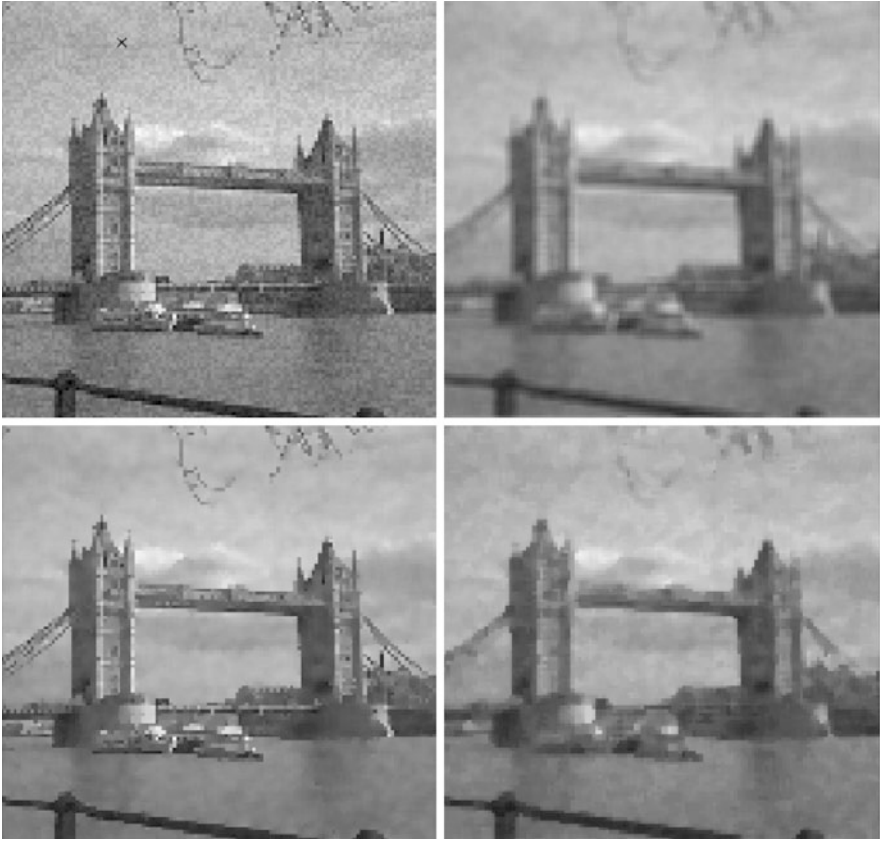


Fig. 2.15 Illustration of noise removal. *Upper left:* 128×128 input image with added uniform noise (± 15). *Upper right:* 3×3 box filter. *Lower left:* 3×3 sigma-filter with $\sigma = 30$. *Lower right:* 3×3 median filter

2.3.2 Sharpening

Sharpening aims at producing an enhanced image J by increasing the contrast of the given image I along edges, without adding too much noise within homogeneous regions in the image.

Unsharp Masking This local operator first produces a residual $R(p) = I(p) - S(p)$ with respect to a smoothed version $S(p)$ of $I(p)$. This residual is then added to the given image I :

$$\begin{aligned} J(p) &= I(p) + \lambda[I(p) - S(p)] \\ &= [1 + \lambda]I(p) - \lambda S(p) \end{aligned} \tag{2.20}$$



Fig. 2.16 Illustration of unsharp masking with $k = 3$ and $\lambda = 1.5$ in (2.20). *Upper left:* 512×512 blurred input image `Altar` (of the baroque altar in the church at Valenciana, Guanajuato). *Upper right:* Use of a median operator. *Lower left:* Use of a Gauss filter with $\sigma = 1$. *Lower right:* Use of a sigma filter with $\sigma = 25$

where $\lambda > 0$ is a scaling factor. Basically, any of the smoothing operators of Sect. 2.3.1 may be tried to produce the smoothed version $S(p)$. See Fig. 2.16 for three examples.

The size parameter k (i.e. the “radius”) of those operators controls the spatial distribution of the smoothing effect, and the parameter λ controls the influence of the correction signal $[I(p) - S(p)]$ on the final output. Thus, k and λ are the usual interactive control parameters for unsharp masking.

According to the second equation (2.20), the process is also qualitatively described by the equation

$$J(p) = I(p) - \lambda' S(p) \quad (2.21)$$

(for some $\lambda' > 0$), which saves some computing time. Instead of applying unsharp masking uniformly in the whole image I , we can also add some kind of local adaptivity, for example such that changes in homogeneous regions are suppressed.

2.3.3 Basic Edge Detectors

We describe simple edge detectors that follow the step-edge model, either by approximating first-order derivatives or by approximating second-order derivatives.

Discrete Derivatives The derivative of a unary function f in the continuous case is defined by the convergence of difference quotients where a nonzero offset ε approaches 0:

$$\frac{df}{dx}(x) = f'(x) = \lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f(x)}{\varepsilon} \quad (2.22)$$

The function f is differentiable at x if there is a limit for these difference quotients. In the case of functions with two arguments, we have partial derivatives, such as

$$\frac{\partial f}{\partial y}(x, y) = f_y(x, y) = \lim_{\varepsilon \rightarrow 0} \frac{f(x, y + \varepsilon) - f(x, y)}{\varepsilon} \quad (2.23)$$

with respect to y , and analogously with respect to x .

However, in the discrete grid we are limited by a smallest distance $\varepsilon = 1$ between pixel locations. Instead of just reducing the derivative in (2.23) to a difference quotient for $\varepsilon = 1$, we can also go for a symmetric representation taking the difference $\varepsilon = 1$ in both directions. The simplest symmetric difference quotient with respect to y is then as follows:

$$\begin{aligned} I_y(x, y) &= \frac{I(x, y + \varepsilon) - I(x, y - \varepsilon)}{2\varepsilon} \\ &= \frac{I(x, y + 1) - I(x, y - 1)}{2} \end{aligned} \quad (2.24)$$

where we decide for a symmetric difference for better balance. We cannot use any smaller ε without doing some subpixel-kind of interpolations.

Equation (2.24) defines a very noise-sensitive approximation of the first derivative. Let $I_x(x, y)$ be the corresponding simple approximation of $\frac{\partial I}{\partial x}(x, y)$. The resulting approximated magnitude of the gradient is then given by

$$\sqrt{I_x(x, y)^2 + I_y(x, y)^2} \approx \|\mathbf{grad} I(x, y)\|_2 \quad (2.25)$$

This value combines results of two linear local operators, one with a filter kernel representing I_x and one for I_y , shown in Fig. 2.17. The scaling factor 2 is in this case not the sum of the given weights in the kernel; the sum of the weights is zero. This corresponds to the fact that the derivative of a constant function equals zero.

Fig. 2.17 Filter kernels for differences as defined in (2.24)

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} / 2 \quad \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} / 2$$

Fig. 2.18 Filter kernels for the Sobel operator

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} / 1 \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} / 1$$

The result of a convolution with one of those kernels can be negative. Thus, I_x and I_y are not images in the sense that we also have negative values here, and also rational numbers, not only integer values in $\{0, 1, \dots, G_{\max}\}$. It is common to visualize discrete derivatives such as I_x and I_y by showing rounded integer values of $|I_x|$ and $|I_y|$.

Insert 2.5 (Origin of the Sobel Operator) *The Sobel operator was published in [I.E. Sobel. Camera models and machine perception. Stanford, Stanford Univ. Press, 1970, pp. 277–284].*

Sobel Operator The *Sobel operator* approximates the two partial derivatives of image I by using the filter kernels shown in Fig. 2.18. The convolution with the filter kernel approximating a derivative in the x -direction is shown in Fig. 2.4, bottom, right.

These two masks are discrete versions of simple Gaussian convolutions along rows or columns followed by derivative estimates described by masks in Fig. 2.17. For example,

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad (2.26)$$

The two masks in Fig. 2.18 define two local convolutions that calculate approximations S_x and S_y of the partial derivatives. The value of the Sobel operator at pixel location (x, y) equals

$$|S_x(x, y)| + |S_y(x, y)| \approx \|\mathbf{grad} I(x, y)\|_1 \quad (2.27)$$

This value is shown as grey level in the edge map defined by the Sobel operator. Of course, this can also be followed by a detection of local maximum of the values of the Sobel operator; this extension explains why the Sobel operator is also called an *edge detector*.

Insert 2.6 (Origin of the Canny Operator) *This operator was published in [J. Canny. A computational approach to edge detection. IEEE Trans. Pattern Analysis Machine Intelligence, vol. 8, pp. 679–698, 1986].*

Canny Operator The *Canny operator* maps a scalar image into a binary edge map of “thin” (i.e. having the width of one pixel only) edge segments. The output is not uniquely defined; it depends on two thresholds T_{low} and T_{high} with $0 < T_{low} < T_{high} < G_{max}$, not counting a fixed scale used for Gaussian smoothing.

Let I be already the smoothed input image, after applying a convolution with a Gauss function G_σ of scale $\sigma > 0$, for example $1 \leq \sigma \leq 2$.

We apply now a basic gradient estimator such as the Sobel operator, which provides, for any $p \in \Omega$, simple estimates for the partial derivatives I_x and I_y , allowing one to have estimates $g(p)$ for the magnitude $\|\mathbf{grad} I(p)\|_2$ of the gradient and estimates $\theta(p)$ for its direction $\text{atan2}(I_y, I_x)$. The estimates $\theta(p)$ are rounded to multiples of $\pi/4$ by taking $(\theta(p) + \pi/8)$ modulo $\pi/4$.

In a step of *non-maxima suppression* it is tested whether a value $g(p)$ is maximal in the (now rounded) direction $\theta(p)$. For example, if $\theta(p) = \pi/2$, i.e. the gradient direction at $p = (x, y)$ is downward, then $g(p)$ is compared against $g(x, y - 1)$ and $g(x, y + 1)$, the values above and below of p . If $g(p)$ is not larger than the values at both of those adjacent pixels, then $g(p)$ becomes 0.

In a final step of *edge following*, the paths of pixel locations p with $g(p) > T_{low}$ are traced, and pixels on such a path are marked as being edge pixels. Such a trace is initialized by a location p with $g(p) \geq T_{high}$.

When scanning Ω , say with a standard scan, left-to-right, top-down, and arriving at a (not yet marked) pixel p with $g(p) \geq T_{high}$, then

1. mark p as an edge pixel,
2. while there is a pixel location q in the 8-adjacency set of p with $g(q) > T_{low}$, mark this as being an edge pixel,
3. call q now p and go back to Step 2,
4. search for the next start pixel p until the end of Ω is reached.

By using two thresholds, this algorithm applies *hysteresis*: The following pixel q may not be as good as having a value above T_{high} , but it had at least one predecessor on the same path with a value above T_{high} ; thus, this “positive” history is used to support the decision at q , and we also accept $g(q) > T_{low}$ for continuation.

Insert 2.7 (Laplace) *P.S. Marquis de Laplace (1749–1827) was a French applied mathematician and theoretical physicist.*

Laplacian Following the step-edge model, edges are also identified with zero-crossings of second-order derivatives. Common (simple) discrete approximations of the Laplacian of an image I are defined by the filter kernels shown in Fig. 2.19.

Fig. 2.19 Three masks for approximate calculations of the Laplacian

0	1	0	
1	-4	1	1
0	1	0	

1	1	1	
1	-8	1	1
1	1	1	

-1	2	-1	
2	-4	2	1
-1	2	-1	

In the following example we derive the filter kernel given on the left as an example for operator discretization.

Example 2.4 For deriving the first mask in Fig. 2.19, assume that we map I into a matrix of first-order difference quotients

$$I_y(x, y) = \frac{I(x, y + 0.5) - I(x, y - 0.5)}{1} \\ = I(x, y + 0.5) - I(x, y - 0.5)$$

and then again into a matrix of second-order difference quotients

$$I_{yy}(x, y) = I_y(x, y + 0.5) - I_y(x, y - 0.5) \\ = [I(x, y + 1) - I(x, y)] - [I(x, y) - I(x, y - 1)] \\ = I(x, y + 1) + I(x, y - 1) - 2 \cdot I(x, y)$$

We do the same for approximating I_{xx} and add both difference quotients. This defines an approximation of $\nabla^2 I = \Delta I$, which coincides with the first mask in Fig. 2.19. Figure 2.20 illustrates a row profile of an image after applying this approximate Laplacian.

2.3.4 Basic Corner Detectors

A *corner* in an image I is given at a pixel p where two edges of different directions intersect; edges can be defined by the step-edge or the phase congruency model. See Fig. 2.21 for a general meaning of “corners” in images and Fig. 2.22 for an illustration of three corners when zooming into an image.

Insert 2.8 (Hesse and the Hessian Matrix) *The Hessian matrix is named after L.O. Hesse (1811–1874), a German mathematician.*

Corner Detection Using the Hessian Matrix Following the definition of a corner above, it is characterized by high curvature of intensity values. Accordingly, it can be identified by the eigenvalues λ_1 and λ_2 (see Insert 2.9) of the *Hessian matrix*

$$\mathbf{H}(p) = \begin{bmatrix} I_{xx}(p) & I_{xy}(p) \\ I_{xy}(p) & I_{yy}(p) \end{bmatrix} \quad (2.28)$$

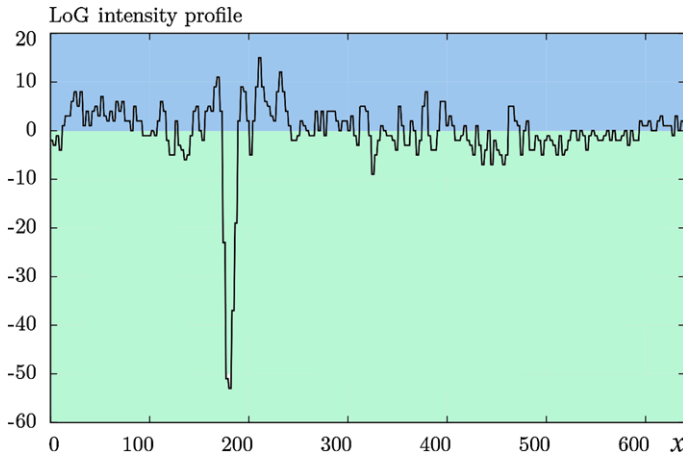
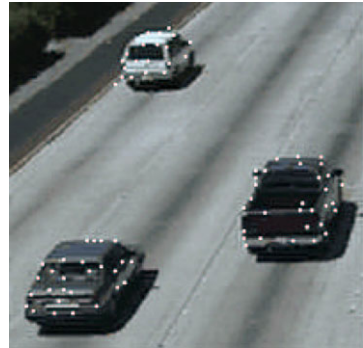


Fig. 2.20 Value profile of a row (close to the middle) in the resulting array when applying the Laplacian to a smoothed version of the image `Set1Seq1` (see Fig. 2.4, upper left) using scale $s = 2$. The steep global minimum appears between branches of a shrub

Fig. 2.21 Detected corners provide important information for localizing and understanding shapes in 3D scenes



at pixel location p . If the magnitude of both eigenvalues is “large”, then we are at a corner; one large and one small eigenvalue identifies a step edge, and two small eigenvalues identify a low-contrast region.

Insert 2.9 (Trace of a Matrix, Determinant, and Eigenvalues) *The trace $\text{Tr}(\mathbf{A})$ of an $n \times n$ matrix $\mathbf{A} = (a_{ij})$ is the sum $\sum_{i=1}^n a_{ii}$ of its (main) diagonal elements. The determinant of a 2×2 matrix $\mathbf{A} = (a_{ij})$ is given by*

$$\det(\mathbf{A}) = a_{11}a_{22} - a_{12}a_{21}$$

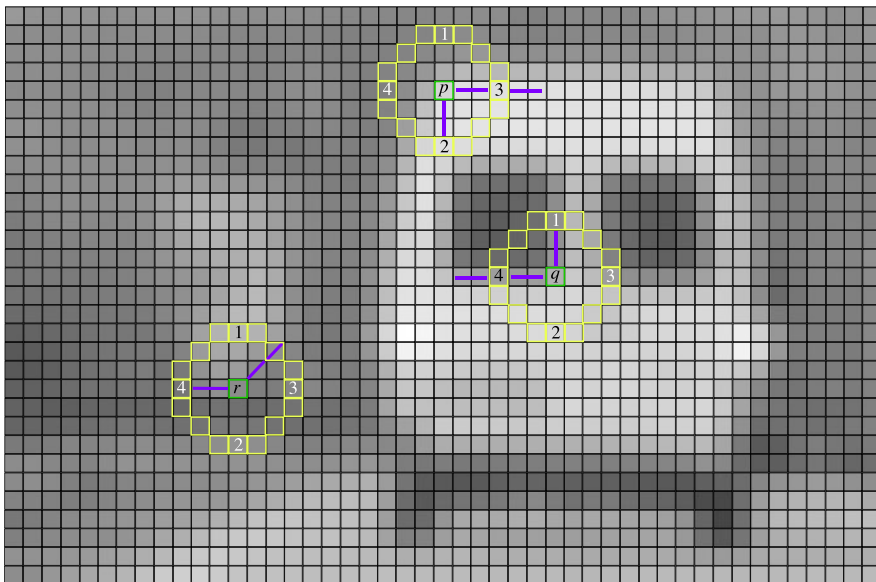


Fig. 2.22 Pixels p , q , and r are at intersections of edges; directions of those edges are indicated by the shown *blue lines*. The shown discrete circles (of 16 pixels) are used in the discussion of the FAST corner detector. Small window of image Set1Seq1

The determinant of a 3×3 matrix $\mathbf{A} = (a_{ij})$ is given by

$$\det(\mathbf{A}) = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}.$$

The eigenvalues of an $n \times n$ matrix \mathbf{A} are the n solutions of its characteristic polynomial $\det(\mathbf{A} - \lambda \mathbf{I}) = 0$, where \mathbf{I} is the $n \times n$ identity matrix, and \det denotes the determinant.

Eigenvalues are real numbers for a real-valued matrix \mathbf{A} . They can be used for modelling stability of solutions of a linear equational system defined by a matrix \mathbf{A} .

The determinant of a square matrix is equal to the product of its eigenvalues, and the trace is equal to the sum of its eigenvalues.

Corner Detector by Harris and Stephens This corner detection method is known as the *Harris detector*. Rather than considering the Hessian of the original image I (i.e. second-order derivatives), we use the first-order derivatives of the smoothed version $L(., ., \sigma)$, as defined in (2.18), for some $\sigma > 0$. Let

$$\mathbf{G}(p, \sigma) = \begin{bmatrix} L_x^2(p, \sigma) & L_x(p, \sigma)L_y(p, \sigma) \\ L_x(p, \sigma)L_y(p, \sigma) & L_y^2(p, \sigma) \end{bmatrix} \quad (2.29)$$

at pixel location p . The eigenvalues λ_1 and λ_2 of the matrix \mathbf{G} represent changes in the intensities in orthogonal directions in the image I . Instead of calculating those eigenvalues, we consider the *cornerness measure*

$$\mathcal{H}(p, \sigma, a) = \det(\mathbf{G}) - a \cdot \text{Tr}(\mathbf{G}) \quad (2.30)$$

for a small parameter $a > 0$ (e.g. $a = 1/25$). Due to the general properties of eigenvalues, we have that

$$\mathcal{H}(p, \sigma, \lambda) = \lambda_1 \lambda_2 - a \cdot (\lambda_1 + \lambda_2) \quad (2.31)$$

If we have one large and one small eigenvalue (such as on a step edge), then having also the trace in (2.30) ensures that the resulting value $\mathcal{H}(p, \sigma, a)$ remains reasonably small.

The cornerness measure \mathcal{H} was proposed in 1988 as a more time-efficient way in comparison to a calculation and analysis of eigenvalues. For results, see Fig. 2.23, left.

Insert 2.10 (Origin of the Harris Detector) *This method was published in [C. Harris and M. Stephens. A combined corner and edge detector. In Proc. Alvey Vision Conference, pp. 147–151, 1988].*

FAST Time constraints in today’s embedded vision (i.e. in “small” independent systems such as micro-robots or cameras in mini-multi-copters), define time-efficiency as an ongoing task. *Features from an accelerated segment test* FAST identify a corner by considering image values on a digital circle around the given pixel location p ; see Fig. 2.22 for 16 image values on a circle of radius $\rho = 3$.

Cornerness test: The value at the centre pixel needs to be darker (or brighter) compared to more than 8 (say, 11 for really identifying a corner and not just an irregular pixel on an otherwise straight edge) subsequent pixels on this circle and “similar” to the values of the remaining pixels on the circle.

For results, see Fig. 2.23, right.

Time Efficiency For being time efficient, we first compare the value at the centre pixel against the values at locations 1, 2, 3, and 4 in this order (see Fig. 2.22); only in cases where it still appears to be possible that the centre pixel passes the cornerness test, we continue with testing more pixels on the circle, such as between locations 1, 2, 3, and 4. The original FAST paper proposes to learn a decision tree for time optimization. The FAST detector in `OpenCV` (and also the one in `libCVD`) applies SIMD instructions for concurrent comparisons, which is faster than the use of the originally proposed decision tree.

Non-maxima Suppression FAST also applies non-maxima suppression for keeping numbers of detected corners reasonably small. For example, for a detected corner, we can calculate the maximum difference T between the value at the centre

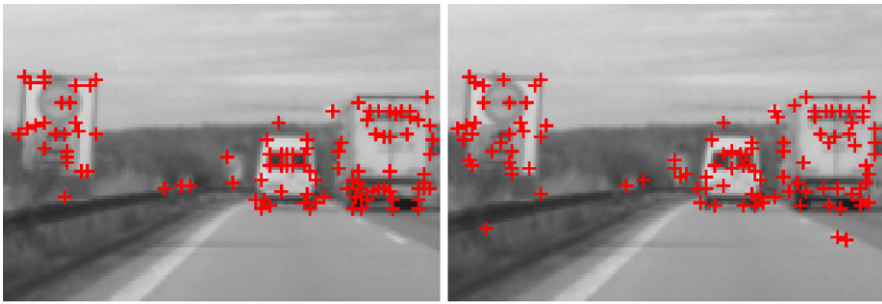


Fig. 2.23 Window of image *Set1Seq1*. *Left*: Detected corners using the Harris detector. *Right*: Corners detected by FAST

pixel and values on the discrete circle being classified as “darker” or “brighter” such that we still detect this corner. Non-maxima suppression deletes then in the order of differences T .

Insert 2.11 (Origin of FAST) *The paper* [E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In Proc. European Conf. Computer Vision, vol. 1, pp. 430–443, 2006] *defined FAST as a corner detector.*

2.3.5 Removal of Illumination Artefacts

Illumination artefacts such as differing exposures, shadows, reflections, or vignetting pose problems for computer vision algorithms. See Fig. 2.24 for examples.

Failure of Intensity Constancy Assumption Computer vision algorithms often rely on the *intensity constancy assumption* (ICA) that there is no change in the appearance of objects according to illumination between subsequent or time-synchronized recorded images. This assumption is actually violated when using real-world images, due to shadows, reflections, differing exposures, sensor noise, and so forth.

There are at least three different ways to deal with this problem. (1) We can transform input images such that illumination artefacts are reduced (e.g. mapping images into a uniform illumination model by removing shadows); there are proposals for this way but the success is still limited. (2) We can also attempt to enhance computer vision algorithms so that they do not rely on ICA, and examples for this option are discussed later in this book. (3) We can map input images into images containing still the “relevant” information for subsequent computer vision algorithms, without aiming at keeping those images visually equivalent to the original data, but at removing the impacts of varying illumination.



Fig. 2.24 Example images from real-world scenes (*black pixels* at borders are caused by image rectification, to be discussed later in the book). The pair of images *NorthLeft* and *NorthRight* in the *top row* show illumination differences between time-synchronized cameras when the exposures are bad. The *bottom-left image* *LightAndTrees* shows an example where trees can cause bad shadow effects. The *bottom-right image* *MainRoad* shows a night scene where head-lights cause large bright spots on the image

We discuss two methods for the third option. A first approach could be to use either histogram equalization or conditional scaling as defined before. Those methods map the whole image uniformly onto a normalized image, normalized with respect to a uniform grey-level histogram or constant mean and standard deviation, respectively. But those uniform transforms are not able to deal with the non-global nature of illumination artefacts.

For example, in vision-based driver assistance systems, there can be the “dancing light” from sunlight through trees, creating local illumination artefacts. See the bottom-left image in Fig. 2.24.

Using Edge Maps Local derivatives do not change when increasing image values by an additive constant. Local derivatives, gradients, or edge maps can be used to derive image representations that are less impacted by lighting variations.

For example, we may simply use Sobel edge maps as input for subsequent computer vision algorithms rather than the original image data. See Fig. 2.25 on the

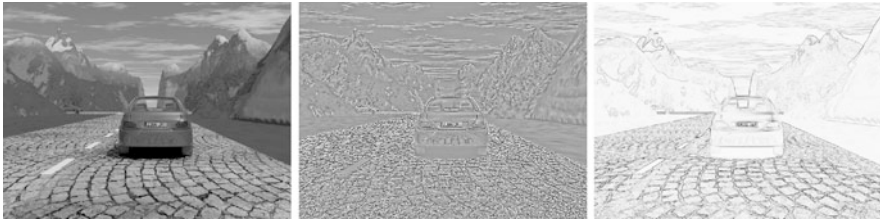


Fig. 2.25 Original image Set2Seq1 (*left*) has its residual image (*middle*), computed using TVL₂ (not discussed in this textbook) for smoothing, and the Sobel edge map (*right*) shown

right. The Sobel edge map is not a binary image, and also not modified due to particular heuristics (as it is the case for many other edge operators), just the “raw edge data”.

Insert 2.12 (Video Test Data for Computer Vision on the Net) *The shown synthetic image in Fig. 2.25 is taken from Set 2 of EISATS, available online at www.mi.auckland.ac.nz/EISATS. There are various test data available on the net for comparing computer vision algorithms on recorded image sequences. For current challenges, see also www.cvlibs.net/datasets/kitti/, the KITTI Vision Benchmark Suite, and the Heidelberg Robust Vision Challenge at ECCV 2012, see hci.iwr.uni-heidelberg.de/Static/challenge2012/.*

Another Use of Residuals with Respect to Smoothing Let I be an original image, assumed to have an additive decomposition

$$I(p) = S(p) + R(p) \quad (2.32)$$

for all pixel positions p , S denotes the smooth component of image I (as above when specifying sharpening), and R is again the residual image with respect to the smoothing operation which produced image S . The decomposition expressed in (2.32) is also referred to as the *structure-texture decomposition*, where the structure refers to the smooth component, and the texture to the residual.

The residual image is the difference between an input image and a smoothed version of itself. Values in the residual image can also be negative, and it might be useful to rescale it into the common range of $\{0, 1, \dots, G_{\max}\}$, for example when visualizing a residual image. Figure 2.25 shows an example of a residual image R with respect to smoothing when using a TV-L² operator (not explained in this textbook).

A smoothing filter can be processed in multiple iterations, using the following scheme:

$$\begin{aligned} S^{(0)} &= I \\ S^{(n)} &= S(S^{(n-1)}) \quad \text{for } n > 0 \\ R^{(n)} &= I - S^{(n)} \end{aligned} \tag{2.33}$$

The iteration number n defines the applied residual filter. When a 3×3 box filter is used iteratively n times, then it is approximately identical to a Gauss filter of radius $n + 1$.

The appropriateness of different concepts needs to be tested for given classes of input images. The iteration scheme (2.33) is useful for such tests.

2.4 Advanced Edge Detectors

This section discusses step-edge detectors that combine multiple approaches into one algorithm, such as combining edge-detection with pre- or post-processing into one optimized procedure. We also address the phase-congruency model for defining edges by discussing the Kovesi operator.

2.4.1 LoG and DoG, and Their Scale Spaces

The *Laplacian of Gaussian* (LoG) and the *difference of Gaussians* (DoG) are very important basic image transforms, as we will see later at several places in the book.

Insert 2.13 (Origin of the LoG Edge Detector) *The origin of the Laplacian of Gaussian (LoG) edge detector is the publication [D. Marr and E. Hildreth. Theory of edge detection. Proc. Royal Society London, Series B, Biological Sciences, vol. 207, pp. 187–217, 1980]. For this reason, it is also known as the Marr–Hildreth algorithm.*

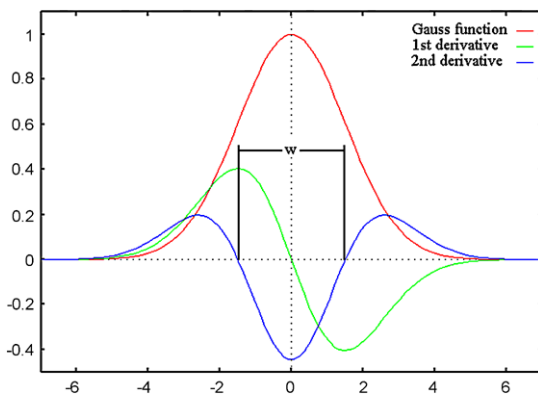
LoG Edge Detector Applying the Laplacian for a Gauss-filtered image can be done in one step of convolution, based on the theorem

$$\nabla^2(G_\sigma * I) = I * \nabla^2 G_\sigma \tag{2.34}$$

where $*$ denotes the convolution of two functions, and I is assumed (for showing this theorem) to be twice differentiable. The theorem follows directly when applying twice the following *general rule of convolutions*:

$$D(F * H) = D(F) * H = F * D(H) \tag{2.35}$$

Fig. 2.26 The 2D Gauss function is rotationally symmetric with respect to the origin $(0, 0)$; it suffices that we show cuts through the function graph of G and its subsequent derivatives



where D denotes a derivative, and F and H are differentiable functions. We have:

Observation 2.4 For calculating the Laplacian of a Gauss-filtered image, we only have to perform one convolution with $\nabla^2 G_\sigma$.

The filter kernel for $\nabla^2 G_\sigma$ is not limited to be a 3×3 kernel as shown in Fig. 2.19. Because the Gauss function is given as a continuous function, we can actually calculate the exact Laplacian of this function. For the first partial derivative with respect to x , we obtain that

$$\frac{\partial G_\sigma}{\partial x}(x, y) = -\frac{x}{2\pi\sigma^4} e^{-(x^2+y^2)/2\sigma^2} \quad (2.36)$$

and the corresponding result for the first partial derivative with respect to y . We repeat the derivative for x and y and obtain the LoG as follows:

$$\nabla^2 G_\sigma(x, y) = \frac{1}{2\pi\sigma^4} \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^2} \right) e^{-(x^2+y^2)/2\sigma^2} \quad (2.37)$$

See Fig. 2.26. The LoG is also known as the *Mexican hat function*. In fact, it is an “inverted Mexican hat”. The zero-crossings define the edges.

Advice on Sampling the LoG Kernel Now we sample this Laplacian into a $(2k+1) \times (2k+1)$ filter kernel for an appropriate value of k . But what is an appropriate value for k ? We start with estimating the standard deviation σ for the given class of input images, and an appropriate value of k follows from this.

The parameter w is defined by zero-crossings of $\nabla^2 G_\sigma(x, y)$; see Fig. 2.26. Consider $\nabla^2 G_\sigma(x, y) = 0$ and, for example, $y = 0$. We obtain that we have both zero-crossings defined by $x^2 = 2\sigma^2$, namely at $x_1 = -\sqrt{2}\sigma$ and at $x_2 = +\sqrt{2}\sigma$. Thus, we have that

$$w = |x_1 - x_2| = 2\sqrt{2}\sigma \quad (2.38)$$

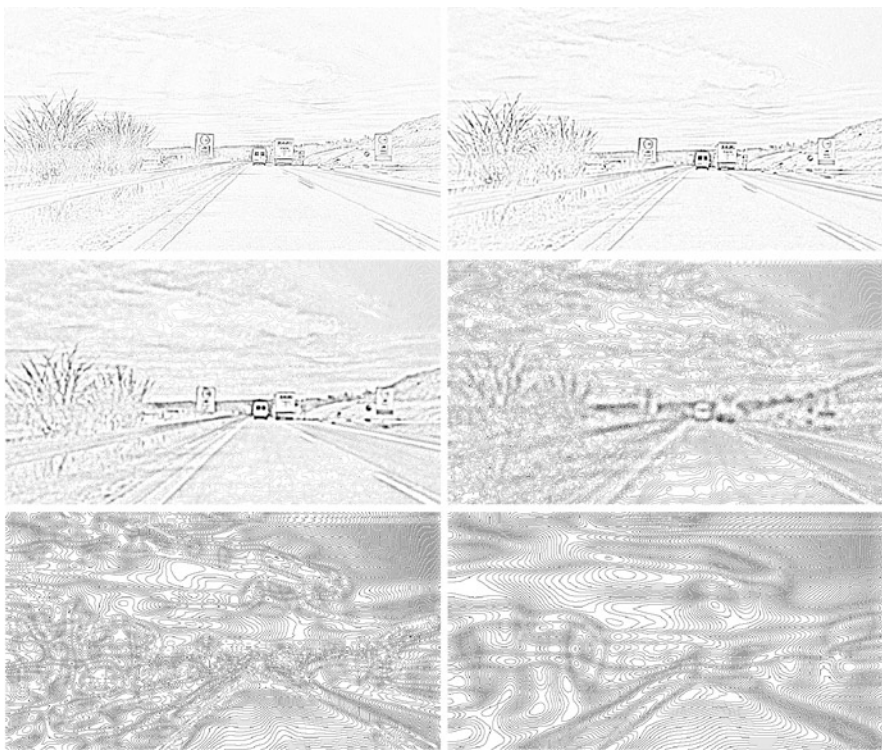


Fig. 2.27 Laplacians of the images shown in Fig. 2.13, representing six layers in the LoG scale space of the image Set1Seq1

For representing the Mexican hat function properly by samples, it is proposed to use a window size of $3w \times 3w = 6\sqrt{2}\sigma \times 6\sqrt{2}\sigma$. In conclusion we have that

$$2k + 1 \times 2k + 1 = \text{ceil}(6\sqrt{2}\sigma) \times \text{ceil}(6\sqrt{2}\sigma) \quad (2.39)$$

where ceil denotes the ceiling function (i.e. the smallest integer equal to or larger than the argument).

The value of σ needs to be estimated for the given image data. Smoothing a digital image with a very “narrow” (i.e. $\sigma < 1$) Gauss function does not make much sense. So, let us consider $\sigma \geq 1$. The smallest kernel (for $\sigma = 1$, thus $3w = 8.485\dots$) will be of size 9×9 (i.e., $k = 4$). For given images, it is of interest to compare results for $k = 4, 5, 6, \dots$

LoG Scale Space Figure 2.13 shows six layers of the Gaussian scale space for the image Set1Seq1. We calculate the Laplacians of those six layers and show the resulting images (i.e. the absolute values of results) in Fig. 2.27; linear scaling was applied to all the images for making the intensity patterns visible. This is an example of a *LoG scale space*. As in a Gaussian scale space, each layer is defined by the

scale σ , the used standard deviation in the Gauss function, and we can generate subsequent layers when starting at an initial scale σ and using subsequent scales $a^n \cdot \sigma$ for $a > 1$ and $n = 0, 1, \dots, m$.

Difference of Gaussians (DoG) The *difference of Gaussians* (DoG) operator is a common approximation of the LoG operator, justified by reduced run time. Equation (2.17) defined a centred (i.e. zero-mean) Gauss function G_σ .

The DoG is defined by an initial scale σ and a scaling factor $a > 1$ as follows:

$$D_{\sigma,a}(x, y) = L(x, y, \sigma) - L(x, y, a\sigma) \quad (2.40)$$

It is the difference between a blurred copy of image I and an even more blurred copy of I . As for LoG, edges (following the step-edge model) are detected at zero-crossings.

Regarding a relation between LoG and DoG, we have that

$$\nabla^2 G_\sigma(x, y) \approx \frac{G_{a\sigma}(x, y) - G_\sigma(x, y)}{(a - 1)\sigma^2} \quad (2.41)$$

with $a = 1.6$ as a recommended parameter for approximation. Due to this approximate identity, DoGs are used in general as time-efficient approximations of LoGs.

DoG Scale Space Different scales σ produce layers $D_{\sigma,a}$ in the *DoG scale space*. See Fig. 2.28 for a comparison of three layers in LoG and DoG scale space, using scaling factor $a = 1.6$.

Insert 2.14 (Origins of Scale Space Studies) *Multi-scale image representations are a well-developed theory in computer vision, with manifold applications. Following the LoG studies by Marr and Hildreth (see Insert 2.13), P.J. Burt introduced Gaussian pyramids while working in A. Rosenfeld's group at College Park; see [P. J. Burt. Fast filter transform for image processing. Computer Graphics Image Processing, vol. 16, pp. 20–51, 1981].*

See also [J.L. Crowley. A representation for visual information. Carnegie-Mellon University, Robotics Institute, CMU-RI-TR-82-07, 1981] and [A.P. Witkin. Scale-space filtering. In Proc. Int. Joint Conf. Artificial Intelligence, pp. 1019–1022, 1983] for early publications on Gaussian pyramids, typically created in increments by factor $a = 2$, and resulting blurred images of varying size were called octaves.

Arbitrary scaling factors $a > 1$ were later introduced into scale-space theory; see, for example, [T. Lindeberg. Scale-Space Theory in Computer Vision. Kluwer Academic Publishers, 1994] and [J.L. Crowley and A.C. Sanderson. Multiple resolution representation and probabilistic matching of 2-D grey-scale shape. IEEE Trans. Pattern Analysis Machine Intelligence, vol. 9, pp. 113–121, 1987].



Fig. 2.28 LoG (left) and DoG (right) layers of image Set1Seq1 are generated for $\sigma = 0.5$ and $a_n = 1.6^n$ for $n = 0, \dots, 5$, and the figure shows results for $n = 1, n = 3$, and $n = 5$

2.4.2 Embedded Confidence

A *confidence measure* is quantified information derived from calculated data, to be used for deciding about the existence of a particular feature; if the calculated data match the underlying model of the feature detector reasonably well, then this should correspond to high values of the measure.

Insert 2.15 (Origin of the Meer–Georgescu Algorithm) *This algorithm has been published in [P. Meer and B. Georgescu. Edge detection with embedded confidence. IEEE Trans. Pattern Analysis Machine Intelligence, vol. 23, pp. 1351–1365, 2001].*

The Meer–Georgescu Algorithm The *Meer–Georgescu algorithm* detects edges while applying a confidence measure based on the assumption of the validity of the step-edge model.

```

1: for every pixel  $p$  in image  $I$  do
2:   estimate gradient magnitude  $g(p)$  and edge direction  $\theta(p)$ ;
3:   compute the confidence measure  $\eta(p)$ ;
4: end for
5: for every pixel  $p$  in image  $I$  do
6:   determine value  $\rho(p)$  in the cumulative distribution of gradient magnitudes;
7: end for
8: generate the  $\rho\eta$  diagram for image  $I$ ;
9: perform non-maxima suppression;
10: perform hysteresis thresholding;

```

Fig. 2.29 Meer–Georgescu algorithm for edge detection

Four parameters are considered in this method. For an estimated gradient vector $\mathbf{g}(p) = \nabla I(x, y)$ at a pixel location $p = (x, y)$, these are the estimated gradient magnitude $g(p) = \|\mathbf{g}(p)\|_2$, the estimated gradient direction $\theta(p)$, an edge confidence value $\eta(p)$, and the percentile ρ_k of the cumulative gradient magnitude distribution. We specify those values below, to be used in the Meer–Georgescu algorithm shown in Fig. 2.29.

Insert 2.16 (Transpose of a Matrix) *The transpose \mathbf{W}^\top of a matrix \mathbf{W} is obtained by mirroring elements about the main diagonal, and $\mathbf{W}^\top = \mathbf{W}$ if \mathbf{W} is symmetric with respect to the main diagonal.*

Let \mathbf{A} be a matrix representation of the $(2k + 1) \times (2k + 1)$ window centred at the current pixel location p in input image I . Let

$$\mathbf{W} = \mathbf{s}\mathbf{d}^\top \quad (2.42)$$

be a $(2k + 1) \times (2k + 1)$ matrix of weights, obtained as the product of two vectors $\mathbf{d} = [d_1, \dots, d_{2k+1}]$ and $\mathbf{s} = [s_1, \dots, s_{2k+1}]$, where

1. both are unit vectors in the L_1 -norm, i.e. $|d_1| + \dots + |d_{2k+1}| = 1$ and $|s_1| + \dots + |s_{2k+1}| = 1$,
2. \mathbf{d} is an asymmetric vector, i.e. $d_1 = -d_{2k+1}$, $d_2 = -d_{2k}$, \dots , $d_{k+1} = 0$, which represents differentiation of one row of matrix \mathbf{A} , and
3. \mathbf{s} is a symmetric vector, i.e. $s_1 = s_{2k+1} \leq s_2 = s_{2k} \leq \dots \leq s_{k+1}$, which represents smoothing in one column of a matrix \mathbf{A} .

For example, asymmetric $\mathbf{d} = [-0.125, -0.25, 0, 0.25, 0.125]^\top$ and symmetric $\mathbf{s} = [0.0625, 0.25, 0.375, 0.25, 0.0625]^\top$ define a 5×5 matrix \mathbf{W} .

Let \mathbf{a}_i be the i th row of Matrix \mathbf{A} . By using

$$d_1 = \text{Tr}(\mathbf{W}\mathbf{A}) = \text{Tr}(\mathbf{s}\mathbf{d}^\top \mathbf{A}) \quad (2.43)$$

$$d_2 = \text{Tr}(\mathbf{W}^\top \mathbf{A}) = \mathbf{s}^\top \mathbf{A}\mathbf{d} = \sum_{i=1}^{2k+1} s_i (\mathbf{d}^\top \mathbf{a}_i) \quad (2.44)$$

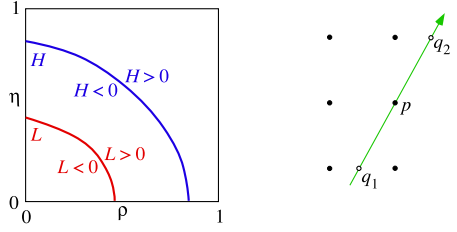


Fig. 2.30 *Left:* Illustration of curves L and H in a $\rho\eta$ diagram; each separates the square into points with positive L or H , or negative L or H signs. For a (ρ, η) point on a curve, we have $L(\rho, \eta) = 0$ or $H(\rho, \eta) = 0$. *Right:* A 3×3 neighbourhood of pixel location p and virtual neighbours q_1 and q_2 in estimated gradient direction

we obtain the first two parameters used in the algorithm:

$$g(p) = \sqrt{d_1^2 + d_2^2} \quad \text{and} \quad \theta(p) = \arctan\left(\frac{d_1}{d_2}\right) \quad (2.45)$$

Let \mathbf{A}_{ideal} be a $(2k + 1) \times (2k + 1)$ matrix representing a template of an ideal step edge having the gradient direction $\theta(p)$. The value $\eta(p) = |\text{Tr}(\mathbf{A}_{ideal}^\top \mathbf{A})|$ specifies the used confidence measure. The values in \mathbf{A} and \mathbf{A}_{ideal} are normalized such that $0 \leq \eta(p) \leq 1$, with $\eta(p) = 1$ in case of a perfect match with the ideal step edge.

Let $g_{[1]} < \dots < g_{[k]} < \dots < g_{[N]}$ be the ordered list of distinct (rounded) gradient-magnitudes in image I , with cumulative distribution values (i.e. probabilities)

$$\rho_k = \text{Prob}[g \leq g_{[k]}] \quad (2.46)$$

for $1 \leq k \leq N$. For a given pixel in I , assume that $g_{[k]}$ is the closest real to its edge magnitude $g(p)$; then we have the percentile $\rho(p) = \rho_k$.

Altogether, for each pixel p , we have a percentile $\rho(p)$ and a confidence $\eta(p)$ between 0 and 1. These values $\rho(p)$ and $\eta(p)$ for any pixel in I define a 2D $\rho\eta$ -diagram for image I . See Fig. 2.30, left.

We consider curves in the $\rho\theta$ space given in implicit form, such as $L(\rho, \theta) = 0$. For example, this can be just a vertical line passing the square, or an elliptical arc. Figure 2.30, left, illustrates two curves L and H . Such a curve separates the square into points having positive or negative signs with respect to the curve and into the set of points where the curve equals zero. Now we have all the tools together for describing the decision process.

Non-maxima Suppression For the current pixel p , determine virtual neighbours q_1 and q_2 in estimated gradient direction (see Fig. 2.30, right) and their ρ and η values by interpolation using values at adjacent pixel locations.

A pixel location p describes with respect to a curve X in $\rho\theta$ space a *maximum* if both virtual neighbours q_1 and q_2 have a negative sign for X . We suppress non-maxima in Step 9 of the algorithm by using a selected curve X for this step; the remaining pixels are the candidates for the edge map.

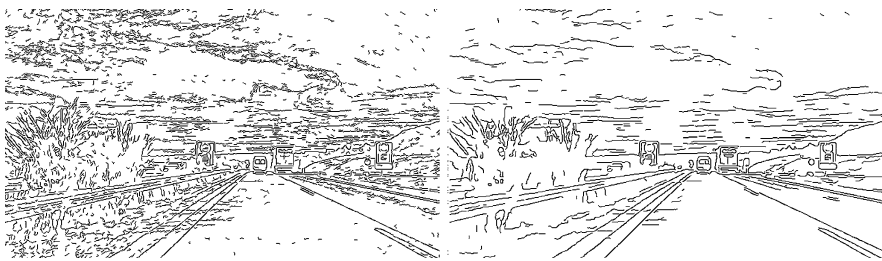


Fig. 2.31 Resultant images when using the Meer–Georgescu algorithm for a larger (*left*) or smaller (*right*) filter kernel defined by parameter k . Compare with Fig. 2.32, where the same input image Set1Seq1 has been used (shown in Fig. 2.4, top, left)

Hysteresis Thresholding *Hysteresis thresholding* is a general technique to decide in a process based on previously obtained results. In this algorithm, hysteresis thresholding in Step 10 is based on having two curves L and H in the $\rho\theta$ space, called the two *hysteresis thresholds*; see Fig. 2.30, left. Those curves are also allowed to intersect in general.

We pass through all pixels in I in Step 9. At pixel p we have values ρ and θ . It stays on the edge map if (a) $L(\rho, \eta) > 0$ and $H(\rho, \eta) \geq 0$ or (b) it is adjacent to a pixel in the edge map and satisfies $L(\rho, \eta) \cdot H(\rho, \eta) < 0$. The second condition (b) describes the hysteresis thresholding process; it is applied recursively.

This edge detection method can be a Canny operator if the two hysteresis thresholds are vertical lines, and a confidence-only detector if the two lines are horizontal. Figure 2.31 illustrates images resulting from an application of the Meer–Georgescu algorithm.

2.4.3 The Kovesei Algorithm

Figure 2.32 illustrates results of four different edge detectors on the same night-vision image, recorded for vision-based driver assistance purposes. The two edge maps on the top are derived from phase congruency; the two at the bottom by applying the step-edge model.

Differences between step-edge operators and phase-based operators are even better visible for a simple synthetic input image as in Fig. 2.33. Following its underlying model, a gradient-based operator such as the Canny operator identifies edge pixels defined by maxima of gradient magnitudes, resulting in double responses around the sphere and a confused torus boundary in Fig. 2.34, left. We present the algorithm used for generating the result in Fig. 2.34, right.

Gabor Wavelets For a local analysis of frequency components, it is convenient not to use wave patterns that run uniformly through the whole $(2k + 1) \times (2k + 1)$ window (as illustrated in Fig. 1.15) but rather *wavelets*, such as *Gabor wavelets*, which are sine or cosine waves modulated by a Gauss function of some scale σ and



Fig. 2.32 The phase-congruency model versus the step-edge model on the image *Set1Seq1*, shown in Fig. 2.4, top, left. *Upper row*: Results of the *Kovesi operator*, which is based on the phase-congruency model, using program `phasecongmono.m` (see link in Insert 2.18) on the left and the next most recent code `phasecong3.m` on the right, both with default parameters. *Lower left*: The Sobel operator follows the step-edge model, using OpenCV's `Sobel()` with x order 1, y order 0, and aperture 3. *Lower right*: The Canny operator is another implementation for the step-edge model using `Canny()` with minimum threshold 150 and maximum threshold 200

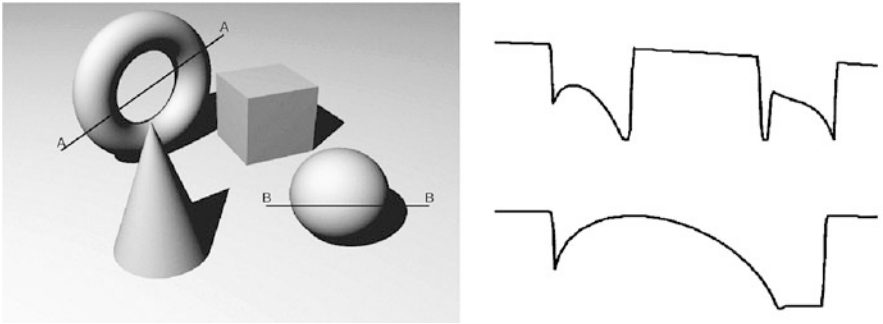


Fig. 2.33 Left: Synthetic input image. Right: Intensity profiles along Section A–A (top) and Section B–B (bottom)

thus of decreasing amplitudes around a centre point. See Fig. 2.35. The image in the middle shows *stripes* that are orthogonal to a defining rotation angle θ .

There are *odd* and *even* Gabor wavelets. An odd wavelet is generated from a sine wave, thus having the value 0 at the origin. An even wavelet is generated from a cosine wave, thus having its maximum at the origin.

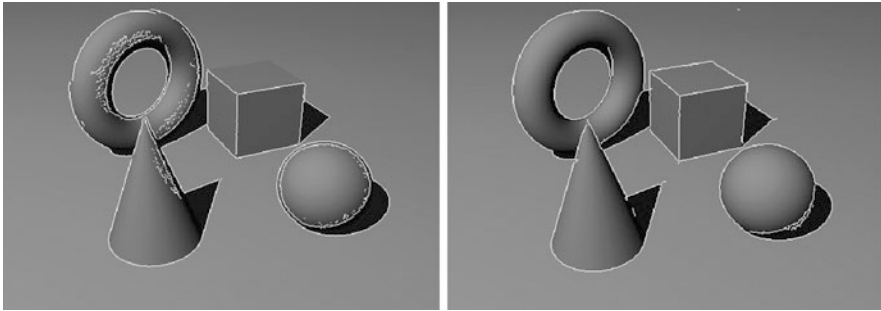


Fig. 2.34 *Left:* Edges detected by the Canny operator. *Right:* Edges detected by the Kovess algorithm

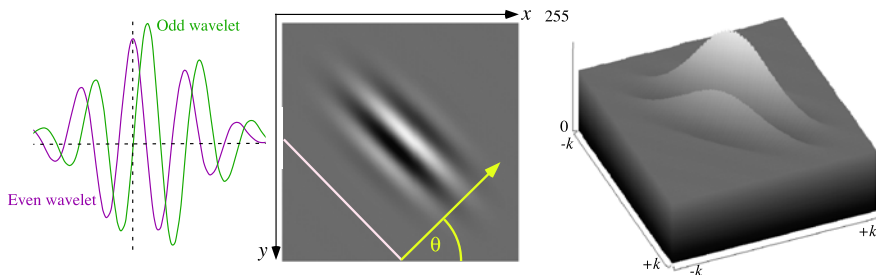


Fig. 2.35 *Left:* Two 1D cuts through an odd and an even Gabor wavelet. *Middle:* A grey-level representation of a square Gabor wavelet in a window of size $(2k + 1) \times (2k + 1)$ with direction θ , with its 3D surface plot (*right*)

Insert 2.17 (Gabor) *The Hungarian born D. Gabor (1900–1979) was an electrical engineer and physicist. He worked in Great Britain and received in 1971 the Nobel Prize in Physics for inventing holography.*

For a formal definition of Gabor wavelets, we first recall the definition of the Gauss function:

$$G_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (2.47)$$

Furthermore, we map the coordinates x and y in the image into rotated coordinates

$$u = x \cos \theta + y \sin \theta \quad (2.48)$$

$$v = -x \sin \theta + y \cos \theta \quad (2.49)$$

where θ is orthogonal to the stripes in the Gabor wavelets; see Fig. 2.35, middle. Now consider also a phase-offset $\psi \geq 0$, wavelength $\lambda > 0$ of the sinusoidal factor, and a spatial aspect ratio $\gamma > 0$. Then, altogether,

$$g_{\text{even}}(x, y) = G_{\sigma}(u, \gamma v) \cdot \cos\left(2\pi \frac{u}{\lambda} + \psi\right) \quad (2.50)$$

$$g_{\text{odd}}(x, y) = G_{\sigma}(u, \gamma v) \cdot \sin\left(2\pi \frac{u}{\lambda} + \psi\right) \quad (2.51)$$

define one *Gabor pair* where sine and cosine functions are modulated by the same Gauss function. The pair can also be combined into one complex number using

$$\begin{aligned} g_{\text{pair}}(x, y) &= g_{\text{even}}(x, y) + \sqrt{-1} \cdot g_{\text{odd}}(x, y) \\ &= G_{\sigma}(u, \gamma v) \cdot \exp\left(2\pi \frac{u}{\lambda} + \psi\right) \end{aligned} \quad (2.52)$$

Insert 2.18 (Origin of the Kovesei Algorithm) *This algorithm has been published in [P.D. Kovesei. A dimensionless measure of edge significance from phase congruency calculated via wavelets. In Proc. New Zealand Conf. Image Vision Computing, pp. 87–94, 1993]. See also sources provided on www.csse.uwa.edu.au/~pk/Research/MatlabFns/index.html#phasecong. The program is very fast and routinely applied to images of size 2000×2000 or more; it applies actually log-Gabor wavelets instead of Gabor wavelets for better operator response and for better time efficiency.*

Preparing for the Algorithm The Kovesei algorithm applies a set of n square Gabor pairs, centred at the current pixel location $p = (x, y)$. Figure 2.36 illustrates such a set for $n = 40$ by illustrating only one function (say, the odd wavelet) for each pair; the Kovesei algorithm uses 24 pairs as default.

The convolution with each Gabor pair defines one complex number. The obtained n complex numbers have amplitude r_h and phase α_h .

Equation (1.33) defines an ideal phase congruency measure. For cases where the sum $\sum_{h=1}^n r_h$ becomes very small, it is convenient to add a small positive number ε to the denominator, such as $\varepsilon = 0.01$. There is also noise in the image, typically uniform. Let $T > 0$ be the sum of all noise responses over all AC components (which can be estimated for given images). Assuming constant noise, we simply subtract the noise component and have

$$\mathcal{P}_{\text{phase}}(p) = \frac{\text{pos}(\|z\|_2 - T)}{\sum_{h=1}^n r_h + \varepsilon} \quad (2.53)$$

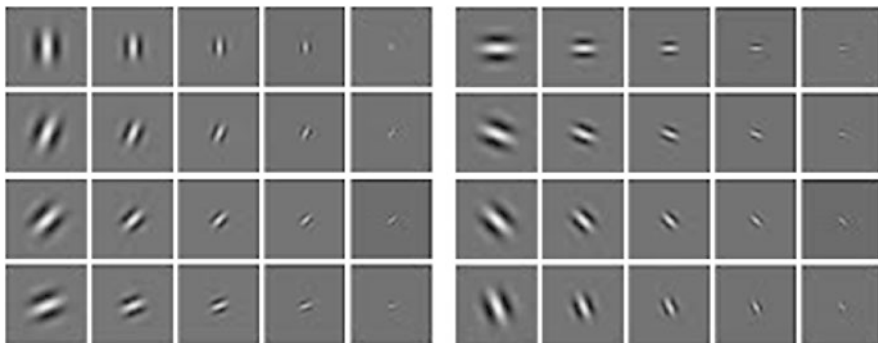


Fig. 2.36 Illustration of a set of Gabor functions to be used for detecting phase congruency at a pixel location

where the function pos returns the argument if positive and 0 otherwise. We have that

$$0 \leq \mathcal{P}_{\text{phase}}(p) \leq 1 \quad (2.54)$$

Select m_1 uniformly distributed directions $\theta_1, \dots, \theta_{m_1}$ and m_2 scales s_1, \dots, s_{m_2} (for example, $m_1 = 6$ and $m_2 = 4$). For specifying the set of $m_1 \cdot m_2$ Gabor wavelets, select the smallest scale (e.g. equal to 3) and a scaling factor between successive scales (say, equal to 2.1). The convolution with those Gabor wavelets can be done more time-efficiently in the frequency domain than in the spatial domain. If in the spatial domain, then the size $(2k + 1) \times (2k + 1)$ of the convolution kernel should be such that $2k + 1$ is about three times the wavelength of the filter.

Processing at One Pixel Now we have all together for analysing phase congruency at the given pixel location $p = (x, y)$:

1. Apply at p the set of convolution masks of $n = m_1 \cdot m_2$ Gabor pairs producing n complex numbers (r_h, α_h) .
2. Calculate the phase congruency measures $\mathcal{P}_i(p)$, $1 \leq i \leq m_1$, as defined in (2.53), but by only using the m_2 complex numbers (r_h, α_h) defined for direction θ_i by m_2 scales.
3. Calculate the directional components X_i and Y_i for $1 \leq i \leq m_1$ by

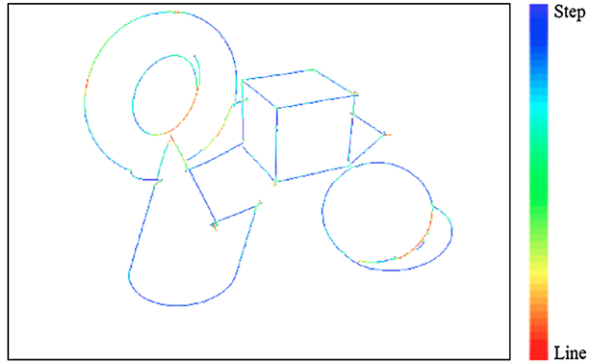
$$[X_i, Y_i]^T = \mathcal{P}_i(p) \cdot [\sin(\theta_i), \cos(\theta_i)]^T \quad (2.55)$$

4. For the resulting *covariance matrix* of directional components,

$$\begin{bmatrix} \sum_{i=1}^{m_1} X_i^2 & \sum_{i=1}^{m_1} X_i Y_i \\ \sum_{i=1}^{m_1} X_i Y_i & \sum_{i=1}^{m_1} Y_i^2 \end{bmatrix} \quad (2.56)$$

calculate the eigenvalues λ_1 and λ_2 ; let $\lambda_1 \geq \lambda_2$. (This matrix corresponds to the 2×2 Hessian matrix of second-order derivatives; for L.O. Hesse, see Insert 2.8.)

Fig. 2.37 Colour-coded results when classifying detected feature points in a scale between “Step” and “Line”, using the colour key shown on the *right*. For the original image, see Fig. 2.33, left



The magnitude of λ_1 indicates the significance of a local feature (an edge, corner, or another local feature); if λ_2 is also of large magnitude, then we have a corner; the principle axis corresponds with the direction of the local feature.

Detection of Edge Pixels After applying the procedure above for all $p \in \Omega$, we have an array of λ_1 values, called the *raw result* of the algorithm. All values below a chosen cut-off threshold (say, 0.5) can be ignored.

We perform non-maxima suppression in this array (possibly combined with hysteresis thresholding, similar to the Meer–Georgescu algorithm), i.e. set to zero all values that do not define a local maximum in their (say) 8-neighbourhood.

All the pixels having non-zero values after the non-maxima suppression are the identified edge pixels.

Besides technical parameters which can be kept constant for all processed images (e.g. the chosen Gabor pairs, parameter ε for eliminating instability of the denominator, or the cut-off threshold), the algorithm only depends on the parameter T used in (2.53), and even this parameter can be estimated from the expected noise in the processed images.

Equation (2.53) gives a measure $\mathcal{P}(p)$ that is proportional to the cosine of the phase deviation angles, which gives a “soft” response.

Given that $\mathcal{P}(p)$ represents a weighted sum of the cosines of the phase deviation angles, taking the arc cosine gives us a weighted sum of the phase deviation angles. A suggested revision of the phase deviation measure is then given by

$$\mathcal{P}_{rev}(p) = \text{pos}(1 - \arccos(\mathcal{P}(p))) \quad (2.57)$$

with function pos as defined above.

Classification into Edge or Line Pixels Having two eigenvalues as results for each pixel, these two values can also be used for classifying a detected feature. See Fig. 2.37 for an example.

2.5 Exercises

2.5.1 Programming Exercises

Exercise 2.1 (Variations of Histogram Equalization) The book [R. Klette and P. Zamperoni: *Handbook of Image Processing Operators*. Wiley, Chichester, 1996] discusses variations of histogram transforms, in particular variations of histogram equalization

$$g_{\text{equal}}^{(r)}(u) = \frac{G_{\max}}{Q} \sum_{w=0}^u h_I(w)^r \quad \text{with } Q = \sum_{w=0}^{G_{\max}} h_I(w)^r$$

Use noisy (scalar) input pictures (of your choice) and apply the sigma filter prior to histogram equalization. Verify by your own experiments the following statements:

A stronger or weaker equalization can be obtained by adjusting the exponent $r \geq 0$. The resultant histogram is uniformly (as good as possible) distributed for $r = 1$. For $r > 1$, sparse grey values of the original picture will occur more often than in the equalized picture. For $r = 0$, we have about (not exactly!) the identical transform. A weaker equalization in comparison to $r = 1$ is obtained for $r < 1$.

Visualize results by using *2D histograms* where one axis is defined by r and the other axis, as usual, by grey levels; show those 2D histograms either by means of a 2D grey-level image or as a 3D surface plot.

Exercise 2.2 (Developing an Edge Detector by Combining Different Strategies) Within an edge detector we can apply one or several of the following strategies:

1. An edge pixel should define a local maximum when applying an operator (such as the Sobel operator) that approximates the magnitude of the gradient ∇I .
2. After applying the LoG filter, the resulting arrays of positive and negative values need to be analysed with respect to zero-crossings (i.e. pixel locations p where the LoG result is about zero, and there are both positive and negative LoG values at locations adjacent to p).
3. The discussed operators are modelled with respect to derivatives in x - or y -directions only. The consideration of directional derivatives is a further option; for example, derivatives in directions of multiples of 45° .
4. More heuristics can be applied for edge detection: an edge pixel should be adjacent to other edge pixels.
5. Finally, when having a sequences of edge pixels, then we are interested in extracting “thin arcs” rather than having “thick edges”.

The task in this programming exercise is to design your own edge detector that combines at least two different strategies as listed above. For example, verify the presence of edge pixels by tests using both first-order and second-order derivatives. As a second example, apply a first-order derivative operator together with a test for adjacent edge pixels. As a third example, extend a first-order derivative operator by directional derivatives in more than just two directions. Go for one of those three examples or design your own combination of strategies.

Exercise 2.3 (Amplitudes and Phases of Local Fourier Transforms) Define two $(2k + 1) \times (2k + 1)$ local operators, one for amplitudes and one for phases, mapping an input image I into the *amplitude image* \mathcal{M} and *phase image* \mathcal{P} defined as follows:

Perform the 2D DFT on the current $(2k + 1) \times (2k + 1)$ input window, centred at pixel location p . For the resulting $(2k + 1)^2 - 1$ complex-valued AC coefficients, calculate a value $\mathcal{M}(p)$ representing the percentage of amplitudes at high-frequency locations compared to the total sum of all $(2k + 1)^2 - 1$ amplitudes and the phase-congruency measure $\mathcal{P}(p)$ as defined in (2.53).

Visualize \mathcal{M} and $\mathcal{P}(p)$ as grey-level images and compare with edges in the input image I . For doing so, select an edge operator, thresholds for edge map, amplitude image, and phase image and quantify the numbers of pixels being in the thresholded edge and amplitude image versus numbers of pixels being in the thresholded edge and phase image.

Exercise 2.4 (Residual Images with Respect to Smoothing) Use a 3×3 box filter recursively (up to 30 iterations) for generating residual images with respect to smoothing. Compare with residual images when smoothing with a Gauss filter of size $(2k + 1) \times (2k + 1)$ for $k = 1, \dots, 15$. Discuss the general relationship between recursively repeated box filters and a Gauss filter of the corresponding radius. Actually, what is the corresponding radius?

2.5.2 Non-programming Exercises

Exercise 2.5 Linear local operators are those that can be defined by a convolution. Classify the following whether they are linear operators or not: box, median, histogram equalization, sigma filter, Gauss filter, and LoG.

Exercise 2.6 Equalization of colour pictures is an interesting area of research. Discuss why the following approach is expected to be imperfect: do histogram equalization for all three colour (e.g. RGB) channels separately; use the resulting scalar pictures as colour channels for the resulting image.

Exercise 2.7 Prove that conditional scaling correctly generated an image J that has the mean and variance identical to those corresponding values of the image I used for normalization.

Exercise 2.8 Specify exactly how the integral image can be used for minimizing run time for a box filter of large kernel size.

Exercise 2.9 Following Example 2.4, what could be a filter kernel for the quadratic variation (instead of the one derived for the Laplace operator)?

Exercise 2.10 Prove that Sobel masks are of the form \mathbf{ds}^\top and \mathbf{sd}^\top for 3D vectors \mathbf{s} and \mathbf{d} that satisfy the assumptions of the Meer–Georgescu algorithm for edge detection.

Exercise 2.11 The sigma filter replaces $I(p)$ by $J(p)$ as defined in (2.19). The procedure uses the histogram $H(u)$ computed for values u in the window $W_p(I)$ that belong to the interval $[I(p) - \sigma, I(p) + \sigma]$. Alternatively, a direct computation can be applied:

$$J(p) = \frac{\sum_{q \in Z_{p,\sigma}} I(q)}{|Z_{p,\sigma}|} \quad (2.58)$$

where $Z_{p,\sigma} = \{q \in W_p(I) : I(p) - \sigma \leq I(q) \leq I(p) + \sigma\}$. Analyse possible advantages of this approach for small windows.

Exercise 2.12 Sketch (as in Fig. 2.6) filter curves in the frequency domain that might be called an “exponential low-emphasis filter” and “ideal band-pass filter”.

<http://www.springer.com/978-1-4471-6319-0>

Concise Computer Vision

An Introduction into Theory and Algorithms

Klette, R.

2014, XVIII, 429 p. 298 illus., 229 illus. in color.,

Softcover

ISBN: 978-1-4471-6319-0