Concise Computer Vision

Chapter-2

Noise :- An unwanted data is called noise

Histogram Equalization

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

Transforming an image I such that all the grey level appear equally often in the transformed image Inew is called histogram equalization.

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

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

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}:

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

here c_I is the relative cummulative frequencies

$h_I(u)$ → Estimate of Density function

$c_I(u)$ → Probablity Distribution function

$h_{I_{new}}(u)$ → Uniform density function

Linear Scaling :-

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

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

$$g(u) = b(u + a) \tag{2.4}$$

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

Conditional Scaling :-

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

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

$$g(u) = b(u + a) \tag{2.6}$$

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.

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

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

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.

Fourier Filtering

→ The inverse 2D DFT will lead to real valued functions I as long as I satisfies the symmetry.

- Inverse 2D DFT is defined as a complex number $I(u, v)$ are fourier coefficient of I defined for different frequencies u and v
- Each fourier coefficient is multiplied with combination of sin and cosine functions and sum of all those combinations forms image I

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

1. We modify the image data by a linear convolution

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

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*:

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

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.

Box Filter

$G(x,y) = \frac{1}{a}$ for (x,y) in a $(2k+1) \times (2k+1)$ window, centered at origin $(0,0)$, with $a = (2k+1)^2$

Outside this we have $G(x,y) = 0$

→ the 2D DFT for this filter is close to 1 for low frequencies and close to 0 for high frequencies thus this behaves as low pass filter

Integral Image

for a pixel location $p(x,y)$

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

This is sum of $I(i,j)$ at pixel locations $q=(i,j)$ that are neither below p nor right of p

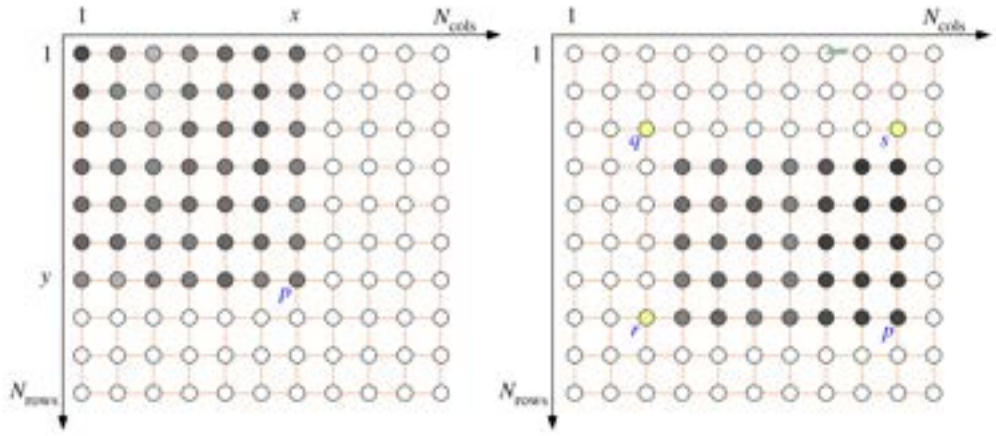


Fig. 2.8 *Left*: At $I_{int}(x, y)$ we have the sum of all the shown pixel values. *Right*: If an algorithm requires to use the sum of all pixel values in the shown rectangular window, then we only need to combine the values of the integral image in the four corners p, q, r, and t; see the text for the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

S_W (sum of pixel) for a window shown in right image defined by pixel p, q, r, s are

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

→ Number of operations for calculating the integral image in window of mxn is $m.(n-1)$ arithmatic operations.

→ If we count for the sequential sliding window in integral image, they are reduced to four ++ increments if we keep the addresses for pixel p, q, r, s in address registers

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.*

Regular Image Pyramid

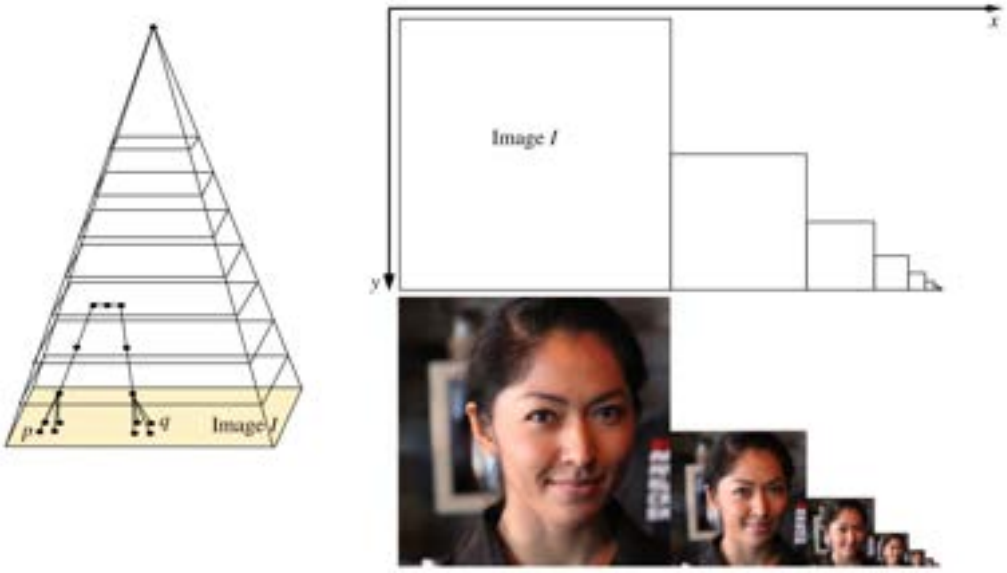


Fig. 2.9 Illustrations of picture pyramids. *Left*: A regular pyramid is the assumed model behind subsequent size reductions. *Left, top*: Sketch of pairwise disjoint arrays. *Left, bottom*: Example of layers for image Emma

→ Pyramid is a data structure used to represent images of different size. Here original image is the base of the pyramid. Image of reduced size are considered to be subsequent layers in the pyramid.

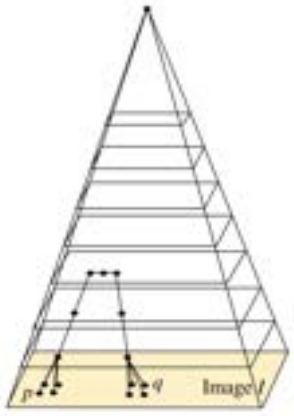
→ If we scale down by the factor 2 then all additional level of pyramid requires less than 1/3rd of the space of the original image

$$1 + \frac{1}{2 \cdot 2} + \frac{1}{2^2 \cdot 2^2} + \frac{1}{2^3 \cdot 2^3} + \frac{1}{2^4 \cdot 2^4} \cdots < \frac{4}{3}$$

→ For reducing the size from one layer to another mean is calculated for the 2×2 pixel to generate pixel at next upper layer in pyramid.

→ By creating new pixel r in layer (n+1) of the pyramid defined by 4 pixels p_1, p_2, p_3 and p_4 at layer n, we create new adjacents

(p_1, n), (p_2, n), (p_3, n) and (p_4, n), additionally to 4-adjacency in layer n



→ For going via adjacent pixel p to q in image I. We can first go up in the pyramid and then the sidewards and then down in the q which supports shorter connection path than using only 4 adjacency in input image I

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

$$2^{n+1} - 2 \tag{2.15}$$

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.*

Classes of local Operator

1) Smoothing

Image smoothing aims at eliminating "outliers" in image value. Outliers are basically the noise in the image.

Box Filter → It is a simple smoothing filter which eliminates outlier.

↳ One major disadvantage is that it reduces contrast as well

→ It is convinent to use 3x3 or 5x5 filter kernel

→ The local mean for the largest kernel size can be calculated using integral image Int of input image I

Median Filter:- 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)

Gauss filter

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:

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

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 second line of the above eqn indicate 2D gauss filter is two subsequent 1D gauss filter. One in horizontal direction and other one in vertical direction.

Centred Gauss function

$\mu_x = \mu_y = 0$ in eqn 2.16

$$G_{\sigma,0,0}(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2}{2\sigma^2}} e^{-\frac{y}{2\sigma^2}}$$

$$G_{\sigma,0,0}(x,y) = \frac{1}{\pi s} e^{-\frac{x^2}{2\sigma^2}} e^{-\frac{y}{2\sigma^2}}$$

Centered Gauss function of window $(2k+1) \times (2k+1)$ with window reference pixel at origin.

For an input image I, let

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

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

$L(x, y, \sigma) \Longrightarrow L(x, y, a^n \sigma)$

where $a > 1$

By repeatedly scaling $a^n \sigma$ and $n = 0, 1, \ldots m$ we create Gaussian scale space

Sigma Filter

→ used for noise Removal

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:

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

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

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

Sharpening

Producing 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 smoothened version $S(p)$ of $I(p)$. This residual is then added to the given image I

$$J(p) = I(p) + \lambda(I(p) - S(p))$$
$$= [1+\lambda] I(p) - \lambda S(p)$$

where $\lambda > 0$ is a scaling factor any smoothing operator pro image $S(p)$

Basic Edge Detectors

Discrete Derivatives

The derivative of a many function f in the continuous case is defined by convergence of differe quotients where Δ non-zero offse $\varepsilon \to 0$

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(\varepsilon)}{\varepsilon}$$

In case of two argument, partial derivatives

$$\frac{\partial f}{\partial y}(x,y) = f_y(x,y)$$

$$= \lim_{\varepsilon \to 0} \frac{f(x, y+\varepsilon) - f(x,y)}{\varepsilon}$$

In discrete grids ε is limited to 1 so we go for symmetric representation taking difference $\varepsilon = 1$ in both directions

The simplest symmetric difference quotient with respect to y is as follows

$$I_y(x,y) = \frac{I(x,y+\varepsilon) - I(x,y-\varepsilon)}{2\varepsilon}$$

$$= \frac{I(x,y+1) - I(x,y-1)}{2}$$

This is very noisi sensitive approximation of the first order derivative.

$$I_x(x,y) = \frac{I(x+1,y) - I(x-1,y)}{2}$$

Approximate magnitude of gradient is given by

$$\sqrt{I_x(x,y)^2 + I_y(x,y)^2}$$

$$\approx \|\text{grad } I(x,y)\|_2$$

Sobel Filter

Fig. 2.18 Filter kernels for the Sobel operator

−1	0	1	
−2	0	2	/1
−1	0	1	

−1	−2	−1	
0	0	0	/1
1	2	1	

Sobel filter approximates the two partial derivatives of image I by using the filter kernel shown above

These two mask are discrete version of simple gaussian convolution along rows and columns followed by derivative estimated by mask

For eg

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1, 0, 1 \end{bmatrix}$$

The value of vosel filter at pixel location (x,y) equals to

$$|S_x(x,y)| + |S_y(x,y)|$$

$$\approx \|\text{grad } I(x,y)\|,$$

— This operator is used to detect local minima thus it is used for edge detection

Canny Operator

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.

Laplacian

According to step edge model, edges are identified with zero crossing of second order derivative

Fig. 2.19 Three masks for approximate calculations of the Laplacian

$$\frac{1}{1}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \frac{1}{1}\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{1}\begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

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

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

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

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

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.

Corner

A corner in an image I is given at a pixel p where two edges of different direction intersect

Corner Detection using Hessian Matrix

Hessian Matrix

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

Eigen value λ_1 and λ_2 of hessian matrix helps in identifying the corner. If both eigen value is larg then we at corner. One large and one small then we are at step edge. If both small then we are at low contrat image

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

Insert 2.9 (Trace of a Matrix, Determinant, and Eigenvalues) *The trace* $\mathrm{Tr}(\mathbf{A})$ *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*

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

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 Detection by Harris and Stephens

Instead of having Hessian of original image I (second order derivative. We use first order derivative of smoothened image $L(\ldots\sigma)$ for $\sigma > 0$

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

Let at pixel p. The eigen value λ_1 and λ_2 of matrix G represent change in intensities in orthogonal direction. Instead of finding eigen value, we calculate cornerness measure

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

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

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

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.

FAST

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 then the use of the originally proposed decision tree.

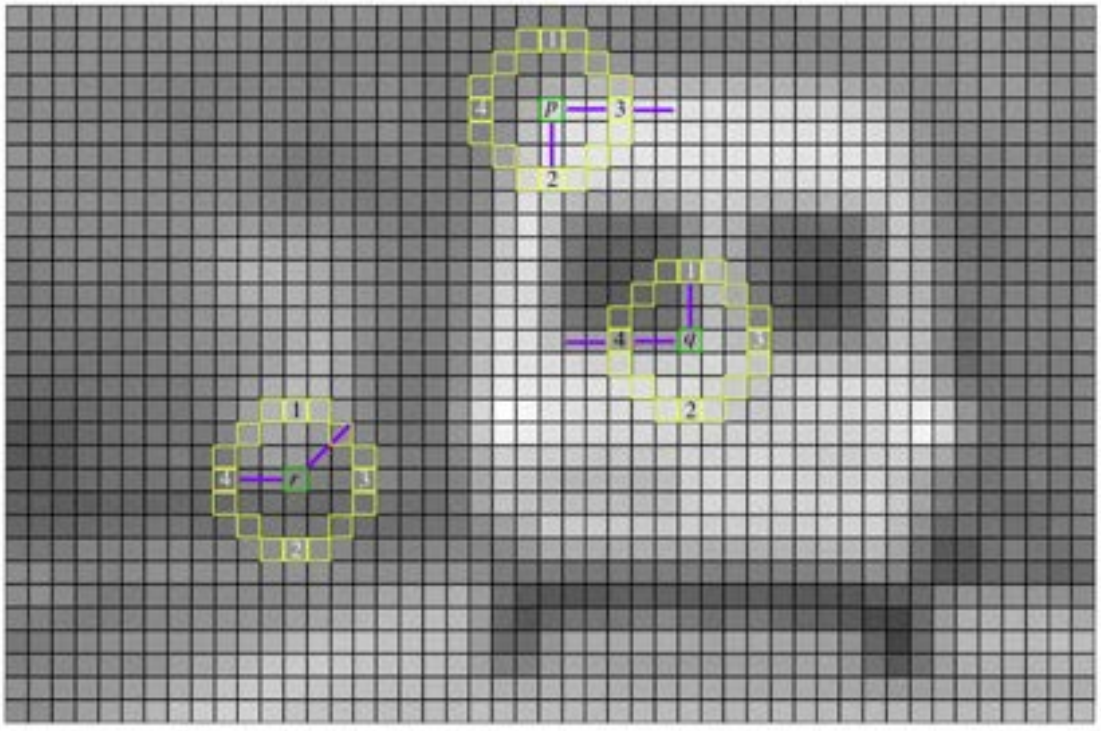


Fig. 2.22 Pixels p, q, and r are at intersections of edges; directions of those edges are indicated by the shown *blue lines*. The shown discrete circles (of 16 pixels) are used in the discussion of the FAST corner detector. Small window of image `Set1Seq1`

Illumination artefacts:-

They are different exposures, shadows, and reflections or vignetting pose problems. for computer vision problems.

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.

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 eg using sobel edge map as inputs for subsequent computer vision algorithm rather than using original image data

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

$$I(p) = S(p) + R(p) \tag{2.32}$$

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 $\{0, 1, \ldots, G_{max}\}$, 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-L^2 operator (not explained in this textbook).

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

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

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.

LoG (Laplacian of Gaussian) Edge Detector

$$\nabla^2 (G_\sigma * I) = I * \nabla^2 G_\sigma$$

$*$ → convolution

This algorithm follows

$$D(f * H) = D(f) * H = f * D(H)$$

where D denotes the derivatives and f and H denotes the differentiable functions

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

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

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:

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

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

$$w = |x_1 - x_2| = 2\sqrt{2}\sigma \tag{2.38}$$

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

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

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 $3w = 8.485\ldots$) will be of size 9×9 (i.e., $k = 4$). For given images, it is of interest to compare results for $k = 4, 5, 6, \ldots$.

DoG (Difference of Gaussian)

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:

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

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

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

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 $a = 1.6$.

Embedded confidence

Calculated data which quantifies about the presence of feature in the data

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 θ(p);
3:     compute the confidence measure η(p);
4: end for
5: for every pixel p in image I do
6:     determine value ρ(p) in the cumulative distribution of gradient magnitudes;
7: end for
8: generate the ρη 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.

The Kovesi Algorithm

Gobar Wavelet

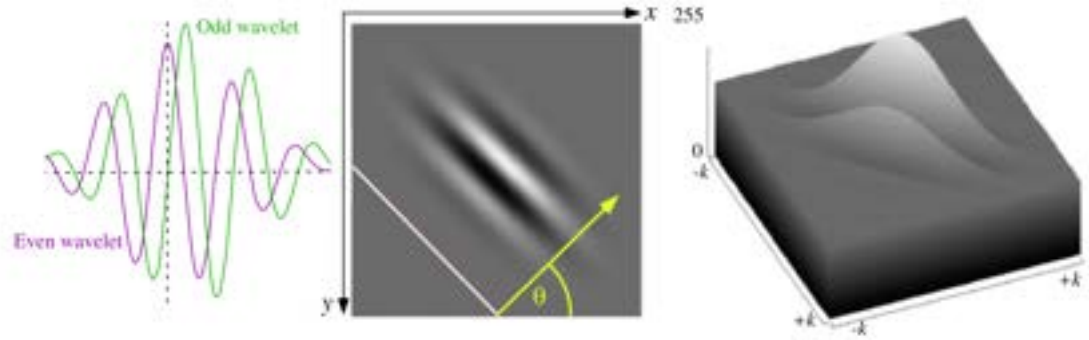


Fig. 2.35 *Left*: Two 1D cuts through an odd and an even Gabor wavelet. *Middle*: A grey-level representation of a square Gabor wavelet in a window of size $(2k + 1) \times (2k + 1)$ with direction θ, with its 3D surface plot (*right*)

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

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.

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

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

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

$$u = x\cos\theta + y\sin\theta \tag{2.48}$$

$$v = -x\sin\theta + y\cos\theta \tag{2.49}$$

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,

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

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

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

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

Preparing for the Algorithm The Kovesi 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 Kovesi 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

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

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

$$0 \le \mathscr{P}_{phase}(p) \le 1 \tag{2.54}$$

Select m_1 uniformly distributed directions $\theta_1, \ldots, \theta_{m_1}$ and m_2 scales $s_1, \ldots, s_{m_2}$ (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 $\mathscr{P}_i(p)$, $1 \le i \le m_1$, 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 $1 \le i \le m_1$ by

$$[X_i, Y_i]^\top = \mathscr{P}_i(p) \cdot \left[\sin(\theta_i), \cos(\theta_i)\right]^\top \tag{2.55}$$

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

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

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

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 $\mathscr{P}(p)$ that is proportional to the cosine of the phase deviation angles, which gives a "soft" response.

Given that $\mathscr{P}(p)$ 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

$$\mathscr{P}_{rev}(p) = \mathrm{pos}\big(1 - \arccos\big(\mathscr{P}(p)\big)\big) \tag{2.57}$$

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.